

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Платформа для автоматизації процесів найму за
допомогою ШІ»**

Виконав студент групи ІПЗп-21-2
ЗАЄЦЬ Андрій Юрійович

Керівник роботи:
ЧИЖМОТРЯ Олена Геннадіївна

Рецензент:
ЄФРЕМОВ Юрій Миколайович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЗАЄЦЬ Андрій Юрійович**

Керівник роботи: **ЧИЖМОТРЯ Олена Геннадіївна**

Тема роботи: **«Платформа для автоматизації процесів найму за допомогою ШІ»,**

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **Об'єктом дослідження є використання інформаційних технологій для автоматизації процесів найму працівників. Предметом дослідження є використання технологій розробки веб-застосунків для автоматизації процесів найму працівників за допомогою штучного інтелекту.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О. Г.	27.02.2025р.	27.02.2025р.
2	Чижмотря О. Г.	06.03.2025р.	06.03.2025р.
3	Чижмотря О. Г.	24.04.2025р.	24.04.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного комплексу та пояснювальної записки. Пояснювальна записка до випускної роботи містить 99 сторінок, включає 15 ілюстрацій, 33 таблиці, 1 додаток та 30 використаних літературних джерел. Система включає модулі для управління компаніями, вакансіями, кандидатами, етапами найму, моделями ШІ, відстеження прогресу найму, а також абсолютно усіма процесами.

Метою роботи є створення платформи для автоматизації процесів найму із застосуванням ШІ, що забезпечує повний цикл рекрутингу: від генерації та публікації вакансій і парсингу резюме до проведення AI-інтерв'ю та аналітики взаємодії кандидатів у реальному часі. Система реалізована у клієнт-серверній архітектурі (ReactTS, Node.js/TypeScript) з використанням PostgreSQL, SQS та інфраструктури AWS (ECS/Fargate/EC2) [11], підтримує модулі керування вакансіями, кандидатами, компаніями та користувачами, нотифікації й інтеграції з зовнішніми рекрутинговими майданчиками, а також побудована у монорепозіторії з використанням UI-компонентів shadcn [18].

У роботі поставлені основні завдання на розробку системи. Також було проведено порівняльний аналіз аналогічних систем. Було обрано клієнт-серверну архітектуру [20]. Приведені основні сценарії використання системи, логіка взаємодії між модулями системи, діаграма класів та алгоритми роботи ключових модулів, спроектована база даних з використанням PostgreSQL. Клієнтська частина розроблена за допомогою фреймворку ReactTS, а серверна частина мовою програмування TS в середовищі Node.js.

Ключові слова: NODE.JS, FRONT END, BACK END, AI, SQS, PostgreSQL, AI, AWS, REACT, SHADCN, FARGATTE, EC2, ECS, MONOREPO.

					ІПЗп.КР.Б – 121 – 25 - ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Платформа для автоматизації процесів найму за допомогою ІІІ Пояснювальна записка	Лім.	Арк.	Аркушів
Розроб.		Заєць А.Ю.						
Керівник.		Чижмoтpя О.Г					3	99
Рецензент		Єфремов Ю.М.						
Зав. каф.		Вакалюк Т.А						
						Житомирська політехніка, група ІПЗп-21-2		

ABSTRACT

The bachelor's final qualification work consists of a software package and an explanatory note. The explanatory note to the final work contains 99 pages, includes 15 illustrations, 33 tables, 1 appendix and 30 references. The system includes modules for managing companies, vacancies, candidates, recruitment stages, AI models, tracking recruitment progress, and absolutely all processes.

The goal of the project is to create a platform for automating recruitment processes using AI, which provides a full recruitment cycle: from generating and publishing job openings and parsing resumes to conducting AI interviews and analyzing candidate interactions in real time. The system is implemented in a client-server architecture (ReactTS, Node.js/TypeScript) using PostgreSQL, SQS, and AWS infrastructure (ECS/Fargate/EC2) [11], supports modules for managing vacancies, candidates, companies, and users, notifications, and integration with external recruitment platforms, and is built in a monorepository using shadcn UI components [18].

The main tasks for developing the system are set out in the paper. A comparative analysis of similar systems was also carried out. A client-server architecture was chosen [20]. The main scenarios for using the system, the logic of interaction between system modules, a class diagram, and algorithms for key modules are presented, and a database was designed using PostgreSQL. The client part was developed using the ReactTS framework, and the server part was developed using the TS programming language in the Node.js environment.

Keywords: NODE.JS, FRONT END, BACK END, AI, SQS, PostgreSQL, AI, AWS, REACT, SHADCN, FARGATTE, EC2, ECS, MONOREPO.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ, ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ	9
1.1. Визначення та постановка задачі	9
1.2. Аналіз аналогів програмного продукту	10
1.3. Визначення архітектури програмного забезпечення	14
1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення	16
Висновки до першого розділу.....	19
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	21
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	21
2.2. Розробка бази даних системи.....	31
2.3. Проєктування та реалізація алгоритмів роботи системи	43
Висновки до другого розділу	51
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ	52
3.1. Порядок встановлення та налаштування параметрів системи	52
3.2. Структура інтерфейсу.....	53
3.3. Тестування роботи програмного комплексу	58
Висновки до третього розділу	61
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63
ДОДАТКИ.....	66

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AI - штучний інтелект

JS - Мова програмування «JavaScript»

TS - Мова програмування «TypeScript»

C# - Мова програмування «C#»

ООП - об'єктно орієнтовний підхід

ПЗ - Програмне забезпечення

БД - База даних

API - Application programming interface, дослівно: Інтерфейс прикладного програмування.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми цієї кваліфікаційної роботи зумовлена стрімким зростанням попиту на висококваліфікованих ІТ-фахівців, що є характерною ознакою сучасного ринку праці. З огляду на велику кількість кандидатів і потребу у швидкому, ефективному та об'єктивному підборі персоналу, компанії стикаються з необхідністю оптимізувати свої рекрутингові процеси. У цьому контексті особливо важливою є роль сучасних технологій, які дозволяють автоматизувати рутинні завдання, забезпечуючи тим самим більшу точність і швидкість прийняття рішень у процесі найму. Рішення цієї проблеми через розробку платформи для автоматизації процесів рекрутингу, зокрема за допомогою штучного інтелекту, є надзвичайно актуальним, оскільки такі системи здатні не лише полегшити роботу рекрутерів, але й забезпечити високий рівень стандартів і прозорості в усіх етапах взаємодії з кандидатами. У результаті, застосування таких технологій дозволить підвищити якість відбору кандидатів, скоротити час на процес найму та зробити його більш ефективним і орієнтованим на дані.

Метою випускної кваліфікаційної роботи бакалавра є розробка платформи для автоматизації процесів найму з використанням штучного інтелекту, що забезпечує ефективне керування вакансіями, кандидатами та інтерв'ю, підвищує швидкість відбору та покращує якість прийняття рішень у рекрутингу.

Для реалізації поставленої мети потрібно виконати наступні **завдання**:

- Провести огляд існуючих систем автоматизації рекрутингу та AI-рішень у HR.
- Обрати клієнт-серверну архітектуру та інструментальні засоби для розробки платформи.
- Спроектувати базу даних та моделі домену для вакансій, кандидатів, інтерв'ю та компаній.
- Розробити модулі керування вакансіями, кандидатами, компаніями та користувачами з ролями й правами доступу.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

- Реалізувати механізми імпорту/парсингу резюме та автоматизованого AI-інтерв'ю з фіксацією результатів.
- Інтегрувати зовнішні рекрутингові майданчики та канали подачі кандидатів.
- Забезпечити аналітику та моніторинг взаємодій кандидатів для підтримки прийняття рішень.
- Провести тестування, оцінити показники продуктивності та підготувати рекомендації щодо розгортання.

Об'єктом дослідження є процес розробки та впровадження системи для автоматизованого найму, що передбачає централізоване керування вакансіями, автоматичну оцінку кандидатів за допомогою AI, динамічне коригування критеріїв відбору та оцінку результативності впроваджених алгоритмів.

Предметом дослідження є створення системи та моделі для автоматизованого найму IT-фахівців, яка включатиме можливості аналізу резюме, оцінки інтерв'ю, інтеграції з AI-інструментами та HR-платформами, а також формування рекомендацій для покращення процесу підбору персоналу.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ, ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Визначення та постановка задачі

Випускна кваліфікаційна робота бакалавра передбачає розробку платформи для автоматизації процесів найму за допомогою ШІ. Платформа має забезпечити ефективне керування вакансіями, кандидатами, інтерв'ю та аналітикою взаємодій, скорочувати час підбору персоналу та підвищувати якість прийняття рішень у HR-процесах.

Сучасний ринок праці вимагає від компаній швидкого та якісного опрацювання великої кількості кандидатів, мультиканальної комунікації та прийняття рішень на основі даних. Для цього необхідна система, яка дозволяє автоматизувати весь цикл рекрутингу: від формування опису вакансії та її публікації на зовнішніх майданчиках до прийому та парсингу резюме, проведення AI-інтерв'ю, фіксації результатів і подальшої аналітики. Застосування методів штучного інтелекту дає змогу суттєво зменшити ручну рутину, стандартизувати оцінювання та забезпечити прозорий процес взаємодії між рекрутерами, менеджерами й кандидатами.

Платформа складається з кількох ключових модулів. Модуль вакансій забезпечує створення, редагування та публікацію вакансій, включно з AI-асистованою генерацією описів та вимог, а також інтеграцією із зовнішніми джерелами кандидатів. Модуль кандидатів підтримує імпорт і парсинг резюме, прив'язку до вакансій і стадій відбору, ведення історії взаємодій та автоматизованих рішень. Модуль інтерв'ю на базі ШІ дозволяє проводити AI-інтерв'ю та телефонні дзвінки з фіксацією транскриптів і ключових метрик поведінки кандидата. Модуль компаній і користувачів забезпечує керування організаціями, ролями та правами доступу. Модуль аналітики надає метрики джерел кандидатів, воронки відбору, продуктивності етапів та якості взаємодії, що підтримує ухвалення рішень у реальному часі. Система сповіщень інформує користувачів про ключові події процесу найму.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

Розробка платформи передбачає кілька етапів. Спочатку визначаються цілі, сценарії використання та вимоги до функціональності для різних ролей (рекрутер, HR-менеджер, адміністратор). Далі проєктуються доменна модель і база даних, які описують сутності вакансій, кандидатів, стадій відбору, інтерв'ю та історії комунікацій. На наступному етапі розробляється архітектура [21] клієнтської та серверної частин, включно з механізмами автентифікації та авторизації користувачів і інтеграціями з зовнішніми сервісами. Після цього реалізуються базові модулі (вакансії, кандидати, інтерв'ю на базі ШІ, аналітика), налаштовуються черги та фонові обробка подій, інтегруються сповіщення. Завершальний етап включає тестування функціональності, перевірку продуктивності та надійності, підготовку рекомендацій щодо розгортання і масштабування системи.

Результатом реалізації поставленого завдання стане платформа для автоматизації процесів найму із застосуванням ШІ, яка дозволяє компаніям оперативно формувати та публікувати вакансії, ефективно опрацьовувати резюме, проводити AI-інтерв'ю, отримувати аналітику у реальному часі та керувати доступом користувачів відповідно до ролей, забезпечуючи прозорий та відтворюваний процес рекрутингу.

1.2. Аналіз аналогів програмного продукту

У сучасному HR-технологічному ландшафті існує низка рішень, що автоматизують окремі або всі етапи найму: від пошуку кандидатів і публікації вакансій до AI-інтерв'ю та аналітики. Для визначення оптимальних функціональних рішень та позиціонування платформи автоматизації найму за допомогою ШІ розглянемо три відомі системи-аналоги [29]: HireVue, Eightfold AI та Greenhouse.

HireVue - платформа, що спеціалізується на автоматизації інтерв'ю та ранжуванні кандидатів. Ключова перевага - використання ШІ для аналізу відео- та асинхронних інтерв'ю, що скорочує час скринінгу та забезпечує масштабованість процесу без додаткового навантаження на рекрутерів (рис. 1.1). Водночас висока

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

вартість та залежність від якості записів/підключення можуть знижувати точність оцінювання та збільшувати операційні ризики для кандидатів з нестабільним інтернетом.

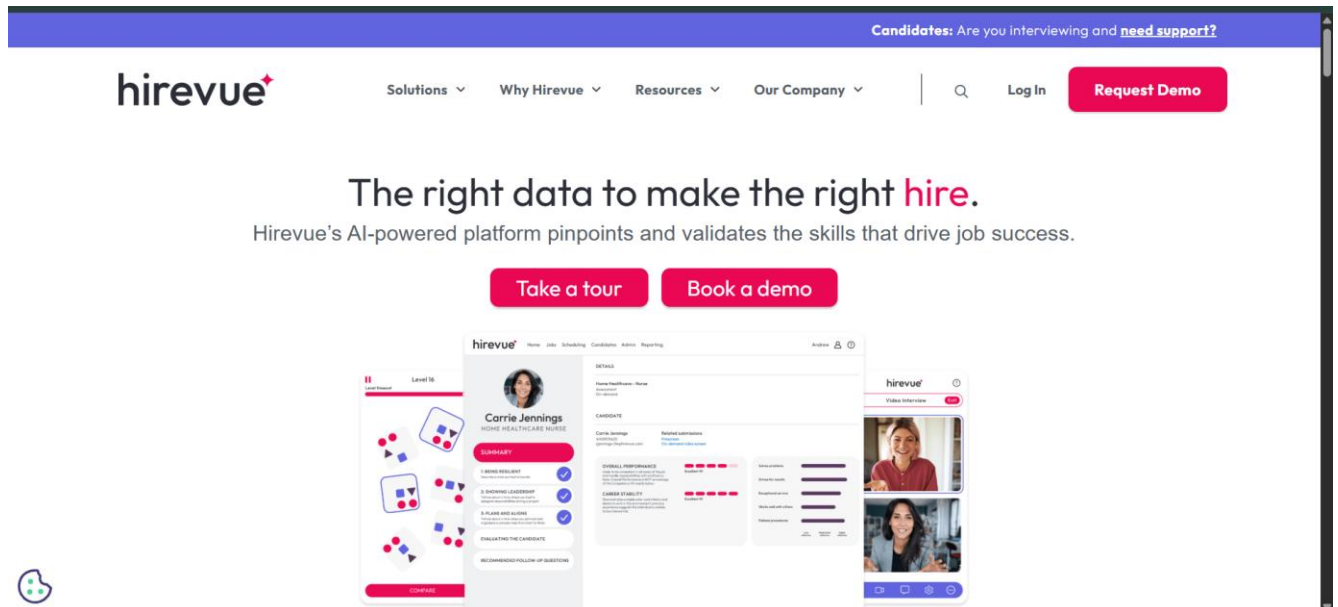


Рис. 1.1 - Головна сторінка сайту платформи “hirevue”

Eightfold AI - “talent intelligence” платформа, що застосовує машинне навчання для побудови профілів кандидатів, їхнього семантичного зіставлення з вакансіями та прогнозування успішності. Сильні сторони - точний пошук, рекомендації, робота з великими даними та підсвітка потенціалу кандидата. Водночас інтеграція з чинними HR-системами та підтримка якості моделей вимагають значних ресурсів, а складність налаштувань може збільшувати час впровадження (рис. 1.2).

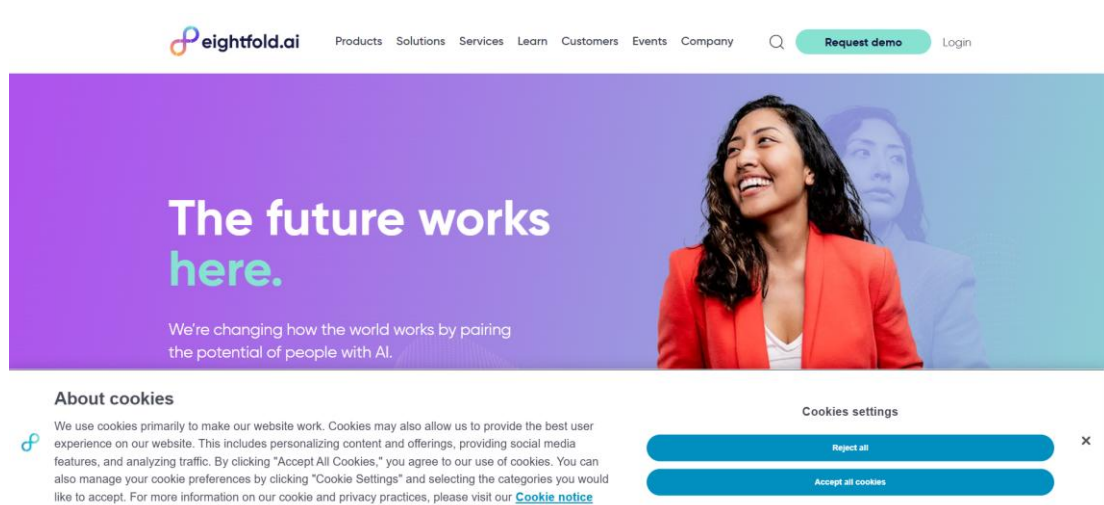


Рис. 1.2 - Головна сторінка сайту платформи “Eightfold AI”

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

Greenhouse, один із найпопулярніших ATS, який пропонує повний конвеєр найму: вакансії, етапи, співбесіди, аналітика, інтеграції з кар'єрними сайтами та інструментами оцінювання (рис. 1.3). Сильні сторони - зріла екосистема інтеграцій, ролі/права, масштабованість і зрозумілий UI. Слабкі - модель підписки, залежність від інтернету та те, що модульні AI-функції зазвичай реалізуються через сторонні інтеграції, а не “з коробки”.

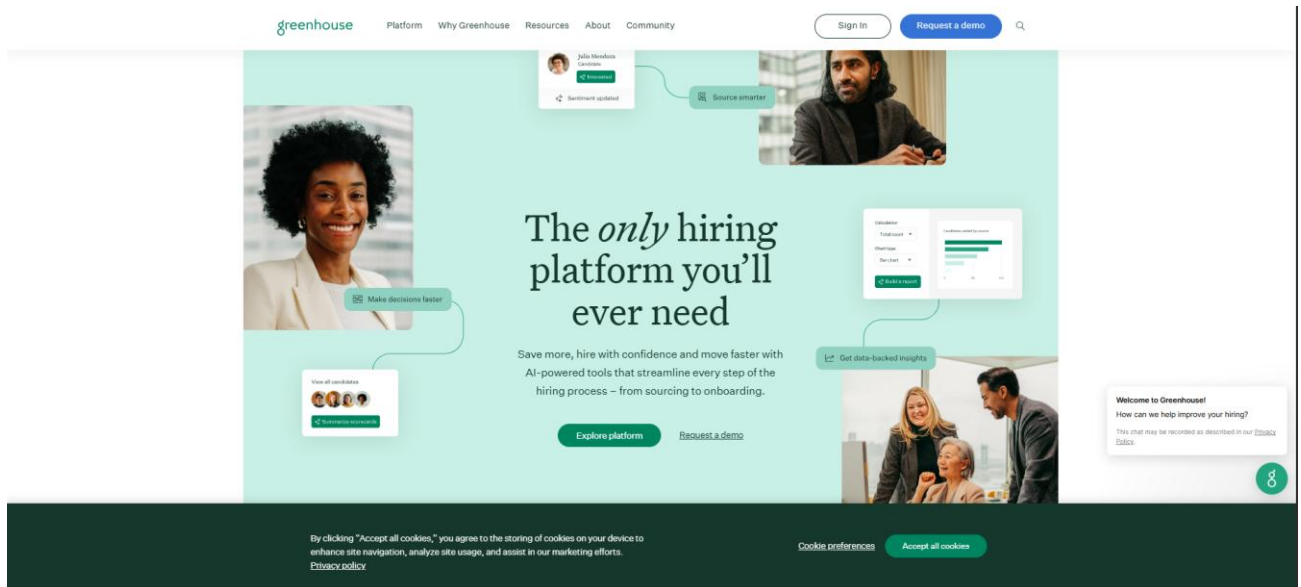


Рис. 1.3 - Головна сторінка сайту програмного забезпечення для рекрутингу ATS “Greenhouse”

Виходячи з виявлених переваг та недоліків, сформуємо порівняльну таблицю функціональних можливостей (табл. 1.1), з акцентом на ті складові, які критичні для нашої платформи: AI-інтерв'ю, парсинг резюме, аналітика, інтеграції та підтримка повного циклу.

Таблиця 1.1

Функціональні можливості

Функціонал	HireVue	Eightfold AI	Greenhouse
Фокус продукту	AI-інтерв'ю, відеоскринінг	Talent intelligence, підбір і рекомендації	ATS: вакансії, етапи, інтеграції
AI-інтерв'ю/дзвінки	Так (сильна сторона)	Опціонально, через інтеграції	Опціонально, через інтеграції
Парсинг/обробка резюме	Базово/через інтеграції	Так (семантика, ML)	Так (через інтеграції/партнерів)

Продовження таблиці 1.1

Аналітика кандидатів/воронки	Так, з акцентом на етап інтерв'ю	Так, глибока ML-аналітика	Так, стандартна ATS-аналітика
Публікація вакансій	Через інтеграції	Через інтеграції	Так (широка екосистема)
Ролі/права доступу	Підтримуються	Підтримуються	Розвинені (enterprise-рівень)
Масштабованість	Висока	Висока	Висока
Сильні сторони	Швидкий відеоскринінг, економія часу	Точні рекомендації, робота з big data	Повний ATS, інтеграції, UX
Слабкі сторони	Вартість, залежність від якості відео/мережі	Складність інтеграції, ML-утримання	Платна підписка, AI часто через сторонні сервіси
Цільова аудиторія	Команди з великим потоком інтерв'ю	Компанії, що потребують розумного підбору	Організації, що прагнуть зрілого ATS

Кожна з платформ має чітку спеціалізацію: HireVue - пришвидшення та стандартизація інтерв'ю, Eightfold AI - інтелектуальний підбір і прогнозування, Greenhouse - процесний "хребет" найму з багатою екосистемою. Наша платформа поєднує сильні сторони цих підходів: забезпечує AI-підтримку інтерв'ю та аналізу кандидатів, надає зручний процесний контур (вакансії → кандидати → етапи) та інтеграції для залучення кандидатів і публікації вакансій, а також пропонує вбудовану аналітику взаємодій у реальному часі. Такий синтез дозволяє зменшити ручні операції, скоротити час закриття позицій і покращити обґрунтованість рішень у рекрутингу.

1.3. Визначення архітектури програмного забезпечення

Для розробки платформи автоматизації процесів найму із застосуванням ШІ обрано клієнт–серверну архітектуру, у якій клієнтська частина взаємодіє з сервером для отримання, обробки та збереження даних (рис. 1.4). Такий підхід забезпечує масштабованість, модульність, можливість незалежного розвитку підсистем та подальше нарощування функціональності.

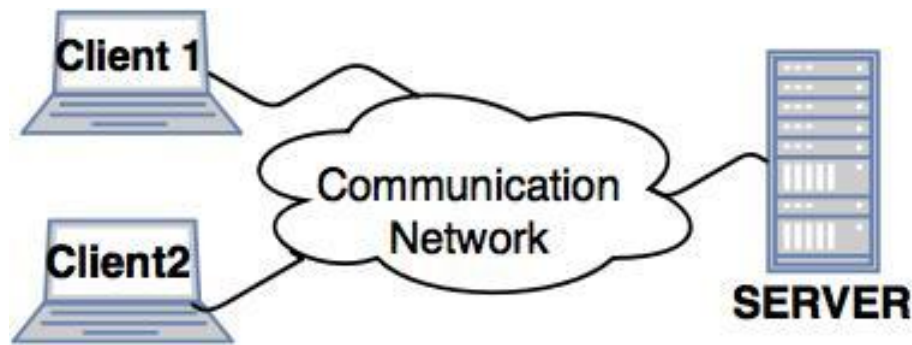


Рис. 1.4 - Схематичне зображення клієнт-серверної архітектури платформи автоматизації найму

Клієнтська частина реалізована у вигляді SPA на основі React TypeScript. Вона відповідає за інтерфейс користувача, сценарії створення та редагування вакансій, роботу з кандидатами, запуск і перегляд результатів AI-інтерв'ю, а також за відображення аналітики та сповіщень. Взаємодія з сервером відбувається через типобезпечний застосунковий інтерфейс tRPC, що мінімізує дублювання контрактів між фронтендом і бекендом і підвищує надійність змін. Аутентифікація та авторизація користувачів здійснюються через керування ролями та дозволами в організаціях, інтегроване у клієнтську частину через відповідні гард-компоненти та контексти доступу.

Серверна частина побудована на Node.js/TypeScript і організована як набір модулів у монорепозіторії (Nx), що полегшує повторне використання коду та чітко розмежовує домени (вакансії, кандидати, інтерв'ю, компанії, нотифікації, інтеграції). Вона реалізує бізнес-логіку керування вакансіями і кандидатами, обробку резюме (парсинг і збереження у сховище), проведення AI-інтерв'ю та запис транскриптів, аналітичні агрегати й інтерфейси для інтеграцій із зовнішніми

майданчиками подачі кандидатів. Для фонових задач і інтеграцій із зовнішніми подієвими джерелами використовуються вебхуки та черги повідомлень, що дозволяє розвантажити синхронні запити та підвищити відмовостійкість.

Сховище даних реалізовано на PostgreSQL [5]. Схема бази даних включає сутності вакансій, кандидатів, зв'язків “кандидат → позиція → стадія”, інтерв'ю/дзвінків, шаблонів комунікацій та журналів взаємодій. Доступ до БД інкапсульований через ORM/квері-білдери з чітким розмежуванням шарів читання/запису, що спрощує супровід та покращує тестованість. Файлові об'єкти (резюме, артефакти інтерв'ю) зберігаються у віддаленому об'єктному сховищі, доступ до якого організовано через пресайн-URL для безпечного та ефективного завантаження/відвантаження з клієнта.

Архітектурний стиль серверної частини дотримується принципів розділення відповідальностей: транспортний шар (tRPC-роутери), шар сервісів (бізнес-логіка), шар доступу до даних (репозиторії/ORM). На клієнті використовується компонентний підхід та хуки для керування станом і побічними ефектами, а також гард-компоненти для застосування політик доступу на рівні інтерфейсу. Для інтеграцій зі сторонніми сервісами (майданчики вакансій, аналітика, нотифікації) передбачений окремий інтеграційний шар, що спрощує заміну або розширення підрядників.

У реалізації застосовано низку патернів проектування. Патерн MVC відображено у фронтенд-композиції (розділення даних, представлення та керування подіями) та в бекенд-шарах (транспорт/сервіси/дані), що спрощує підтримку та тестування. Патерн Singleton використовується для створення одинарних екземплярів клієнтів зовнішніх сервісів (підключення до сховища файлів, аналітики, AI-сервісів), забезпечуючи централізоване керування конфігурацією та повторне використання з'єднань. Патерн Factory застосовується для побудови специфічних об'єктів домену й адаптерів інтеграцій (наприклад, генератори AI-описів вакансій або різні провайдери прийому резюме), що ізолює місця зміни та зменшує зв'язність. Інверсія керування (IoC) реалізована через явну

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

ін'єкцію залежностей у сервіси й адаптери, що дозволяє підмінювати реалізації під час тестування та полегшує конфігурацію середовищ.

Для реального часу використовується подієва модель із вебхуками та нотіфікаціями (оновлення статусів інтерв'ю/дзвінків, надходження нових кандидатів, зміни етапів відбору). Аналітичні модулі агрегують події взаємодії кандидатів та інтерв'ю, забезпечуючи оперативне оновлення показників у клієнті. Взаємодія з великими файлами (резюме, транскрипти) організована асинхронно, що дозволяє не блокувати користувацькі сценарії та покращує UX.

Обрана архітектура забезпечує масштабованість (горизонтальне масштабування серверних компонентів і окремих скейлінг фонових воркерів), модульність (чітке розділення доменів у монорепозіторії) [30], незалежність релізів (ізоляція інтеграцій і фонових задач) та спрощену підтримку (уніфіковані контракти між клієнтом і сервером через типобезпечні інтерфейси). Завдяки чітко визначеним межах відповідальності кожен компонент можна розробляти та тестувати окремо, що знижує ризики регресій і прискорює впровадження нових можливостей.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення

Для розробки платформи автоматизації процесів найму із застосуванням ШІ було обрано інструментальні засоби, що відповідають вимогам високої продуктивності, масштабованості, типобезпеки та зручності в розробці та обслуговуванні. Вибір технологій був обумовлений необхідністю забезпечити ефективне управління вакансіями, обробку резюме, проведення AI-інтерв'ю та інтеграцію з зовнішніми рекрутинговими майданчиками.

Для реалізації серверної частини було обрано Node.js з TypeScript, оскільки ця технологічна комбінація надає зручні інструменти для розробки масштабованих веб-застосунків із чітко організованою структурою та типобезпекою на рівні компіляції. TypeScript також включає функціональність для реалізації безпечного зберігання даних та роботи з базою даних через типізовані інтерфейси. Для

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

зберігання даних використано PostgreSQL, яка забезпечує високу надійність, швидкість обробки складних запитів, підтримку JSON-полів для метаданих та оптимальну підтримку реляційної моделі даних з розширеними можливостями аналітики.

Для побудови інтерактивного інтерфейсу було обрано React з TypeScript, який дозволяє організувати компонентну архітектуру з типобезпечним керуванням станом. React забезпечує гнучкість у створенні динамічних інтерфейсів для роботи з вакансіями, кандидатами та AI-інтерв'ю, а також підтримує взаємодію з сервером через типобезпечний tRPC API [4]. tRPC архітектура дозволяє забезпечити ефективну комунікацію між клієнтською та серверною частинами без дублювання типів, дозволяючи користувачам здійснювати запити до системи для створення вакансій, обробки кандидатів та проведення інтерв'ю.

Також для інтеграції з AI-сервісами було використано Retell [8] для проведення телефонних інтерв'ю та Langfuse для трекінгу AI-взаємодій. Ці сервіси забезпечують безпечну передачу даних та зручну інтеграцію з веб-застосунками для автоматизації процесу інтерв'ю. Для роботи з файлами резюме та метаданими використовується AWS S3 через пресайн-URL, що забезпечує безпечну передачу великих файлів та ефективне зберігання.

У результаті вибору цих інструментів було отримано систему, яка забезпечує високий рівень безпеки даних, ефективне управління вакансіями та кандидатами, автоматизацію інтерв'ю та інтеграцію з AI-сервісами. Вибір Node.js/TypeScript, PostgreSQL та tRPC дозволяє зберегти гнучкість і масштабованість платформи, що є важливим для її подальшого розвитку та розширення функціональності.

Для забезпечення високої продуктивності системи було використано принципи розподілу навантаження через налаштування черг повідомлень SQS, кешування та оптимізації запитів до бази даних. Крім того, при виборі інструментів було враховано їх популярність та активну підтримку спільнотою розробників, що дозволяє забезпечити стабільну роботу системи в умовах змінних вимог та швидкого розвитку AI-технологій.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

Також для забезпечення зручності користувачів була реалізована система ролей та прав доступу через Clerk [7], що дозволяє легко керувати користувачами, організаціями та правами доступу до різних функцій системи. Це дає змогу адміністраторам швидко налаштовувати доступ, змінювати права користувачів та відслідковувати активність через зручний інтерфейс. Використання монорепозиторію (Nx) дозволяє забезпечити швидкий розвиток додаткових функціональних можливостей та підтримку нових AI-сервісів або інтеграцій за потреби.

Створимо таблицю, що порівнює PostgreSQL з іншими популярними СУБД та вказує, чому саме PostgreSQL була обрана для нашої платформи автоматизації найму (табл. 1.2), а не MongoDB чи MySQL.

Порівняння баз даних дозволяє зрозуміти переваги кожної технології в контексті використання в HR-платформах, де важливими є швидкість роботи зі структурованими даними, можливості масштабування, підтримка JSON для метаданих та надійність системи в умовах зростаючих навантажень. Ми вибрали PostgreSQL, оскільки вона поєднує переваги реляційних баз даних з високою продуктивністю, розширеною підтримкою JSON та надійною безпекою, що важливо для нашої платформи.

Таблиця 1.2

Порівняння баз даних для використання в проєкті

Критерій	MongoDB	MySQL	PostgreSQL
Тип СУБД	Документна (NoSQL)	Реляційна (SQL)	Реляційна (SQL)
Структура даних	JSON-документи	Таблична, базова підтримка JSON	Таблична, підтримка JSONB
Масштабованість	Горизонтальна	Вертикальна	Горизонтальна і вертикальна

Продуктивність на складних запитах	Висока на неструктурованих даних	Середня для складних аналітичних запитів	Висока продуктивність на складних аналітичних запитах
Робота з JSON	Основний формат зберігання	Базова підтримка	JSONB - оптимізований для запитів та зберігання
Підтримка типів даних	Обмежена	Стандартна	Розширена (UUID, масиви, спеціальні типи)
Безпека даних	Вимагає додаткової конфігурації	Базова безпека	Розширені можливості шифрування та аудиту
Резервне копіювання	Потребує налаштування	Вбудована підтримка	Вбудована підтримка реплікації та резервного копіювання

Виходячи з наявних переваг та недоліків обраних СУБД, PostgreSQL була обрана для нашої платформи автоматизації найму. Вона має відмінне поєднання стабільності, продуктивності та надійності для обробки транзакційних даних у реляційних структурах з підтримкою JSON для метаданих кандидатів та аналітики. PostgreSQL підтримує SQL-запити високої складності, що зручно для розробки та інтеграції з TypeScript, а також має високу продуктивність [26] для операцій CRUD та аналітичних запитів, що важливо для великого обсягу кандидатів та вакансій у нашій платформі. Крім того, вона забезпечує гарну масштабованість, розширену безпеку та підтримку JSONB для зберігання метаданих інтерв'ю та аналітики, що робить її ідеальним вибором для проєкту з використанням AI.

Висновки до першого розділу

У першому розділі було визначено основні технологічні підходи та архітектурні рішення для розробки платформи автоматизації процесів найму із застосуванням ІІІ. Здійснено аналіз аналогічних платформ для автоматизації

рекрутингу, таких як HireVue, Eightfold AI та Greenhouse, що дозволило виявити їхні переваги та недоліки. Це дозволило сформулювати вимоги до платформи, зокрема щодо AI-інтерв'ю, автоматизації обробки резюме, аналітики взаємодій та інтеграції з зовнішніми сервісами.

На основі цього аналізу було обрано оптимальну архітектуру для проєкту, яка ґрунтується на клієнт-серверній моделі з монорепозіторієм, де серверна частина відповідає за обробку вакансій, управління кандидатами, AI-інтерв'ю та інтеграцію з зовнішніми сервісами, а клієнтська частина взаємодіє із сервером через типобезпечний tRPC API. Також обрано PostgreSQL як СУБД, що забезпечує високу продуктивність, надійність, підтримку JSON та масштабованість для збереження даних про вакансії, кандидатів та аналітику взаємодій.

Також розглянуто основні інструментальні засоби та програмні технології для реалізації системи, зокрема Node.js/TypeScript для серверної частини та React/TypeScript для клієнтської частини. Це дозволить створити ефективну платформу для автоматизації найму, забезпечуючи її високою продуктивністю, типобезпекою, масштабованістю та гнучкістю для подальшого розвитку та інтеграції нових AI-технологій.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

У цьому розділі наведено огляд варіантів використання та описано об'єктно-орієнтовану структуру платформи для автоматизації процесів найму за допомогою ІІІ. Система повинна відповідати функціональним і нефункціональним вимогам, серед яких ключовими є зручність користування, ефективне управління даними, підтримка AI-процесів (парсинг резюме, AI-інтерв'ю), інтеграції із зовнішніми майданчиками та аналітика в реальному часі.

Система поділяється на кілька логічних модулів. Клієнтський модуль (SPA) надає інтерфейси для створення вакансій, управління кандидатами, проведення AI-інтерв'ю та перегляду аналітики. Адміністративний модуль дозволяє керувати компаніями/клієнтами, правами доступу, інтеграціями та глобальними налаштуваннями. Модуль керування ролями/дозволами забезпечує диференційований доступ (Member/Manager/Admin/Super Admin) до функцій системи.

У системі передбачено основні ролі користувачів: користувач, менеджер, адміністратор та супер-адміністратор. Користувач (Recruiter/Member): створення та редагування власних вакансій, створення кандидатів, завантаження резюме, проведення AI-інтерв'ю, перегляд своїх кандидатів і комунікацій. Менеджер (Manager): розширений перегляд кандидатів і вакансій у межах організації, затвердження/редагування, робота з клієнтами. Адміністратор (Admin): управління клієнтами/організаціями, доступ до аналітики, глобальні налаштування, можливе редагування всіх вакансій (за політикою доступу). Супер-адміністратор (Super Admin): повні права, аналітика взаємодій, доступ до інтеграцій і керування планами/фічами.

Кожен модуль платформи має свою чітку мету і взаємодіє з іншими через відкриті API [10] та сервіси. Клієнтський модуль забезпечує інтерактивний досвід

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

для рекрутерів та менеджерів, дозволяючи їм швидко створювати вакансії, парсити резюме, проводити інтерв'ю за допомогою AI та отримувати статистичні дані по кожному етапу найму.

Бізнес-вимоги. Потрібно забезпечити безперервний цикл рекрутингу, який охоплює створення та публікацію вакансій, збір і парсинг резюме, проведення AI-інтерв'ю, комунікації та аналітику, а також надати інструменти для багаторольової взаємодії рекрутера, менеджера й адміністратора з ролями та правами на рівні організації.

Вимоги зовнішніх користувачів (рекрутер/користувач). Користувач має мати можливість створювати та редагувати вакансії, завантажувати резюме, створювати кандидатів, запускати AI-інтерв'ю, переглядати транскрипти й історію взаємодій, а також отримувати базову аналітику щодо власних вакансій і кандидатів.

Вимоги зовнішніх користувачів (менеджер). Менеджер повинен мати можливість переглядати статуси найму в межах команди або організації, втручатися у процес на ключових етапах через апрув етапів і зміну власника вакансії, працювати з клієнтами та відстежувати воронку найму.

Характеристика об'єкта комп'ютеризації. Об'єктом є платформа рекрутингу для компаній, у яких кілька команд здійснюють багатостадійний відбір кандидатів, і система має забезпечувати AI-допомогу у формуванні вакансій, парсинг резюме, AI-інтерв'ю включно з телефонними дзвінками, ведення історії комунікацій та аудиту, а також збір аналітики щодо джерел кандидатів і конверсій на етапах, при цьому інтерфейс має бути адаптивним [27], підтримувати пошук і фільтри та надавати доступ відповідно до ролей і дозволів.

Функціональні вимоги. Система повинна дозволяти створювати, редагувати, публікувати і архівувати вакансії; приймати/парсити резюме та формувати профілі кандидатів; керувати стадіями і статусами у воронці найму; проводити AI-інтерв'ю і зберігати транскрипти; збирати та відображати аналітику; керувати клієнтами/компаніями та правами доступу користувачів у межах організації (табл. 2.1).

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Функціональні вимоги до системи

Етап	Опис
Рольовий доступ (Member/Manager/Admin/Super Admin)	Розмежування функціоналу й даних за ролями та дозволами; контроль CRUD-операцій по вакансіях, кандидатах, клієнтах, інтеграціях та аналітиці.
Керування вакансіями	Створення/редагування/архівація; AI-генерація описів; налаштування етапів відбору; публікація на зовнішніх майданчиках; версіонування ключових полів.
Прийом і парсинг резюме	Завантаження одиночних/масових файлів; парсинг структурованих полів; збереження файлів у сховище; формування профілю кандидата.
Керування кандидатами	Прив'язка до вакансій і стадій; зміна статусів; нотатки; історія комунікацій; видалення/відновлення; пошук і фільтри.
AI-інтерв'ю/дзвінки	Ініціація AI-дзвінків/інтерв'ю; запис транскриптів; збір поведінкових метрик; прикріплення результатів до кандидата/стадії.
Аналітика	Воронки, конверсії, джерела кандидатів, активність, якість інтерв'ю; дашборди для менеджерів/адмінів; експорт звітів.
Керування клієнтами/компаніями	Створення/редагування клієнтів; прив'язка вакансій; видимість по організації; доступ за дозволами.
Сповіщення	Подієві нотифікації (нові відгуки, зміни статусів, результати інтерв'ю); підписки; поштові/внутрішні повідомлення.
Інтеграції	Публікація вакансій, прийом відгуків (вебхуки/форми); зберігання у S3; аналітика/трекінг; AI-провайдери.
Аудит і безпека	Логи доступів/ключових дій; політики видимості; шифрування трафіку; валідація даних; контроль розмірів/типів файлів.

Робота системи починається з аутентифікації та застосування політик доступу. Після авторизації користувач створює вакансії (у т.ч. з AI-генерацією), приймає та парсить резюме, керує стадіями у воронці, призначає або запускає AI-

інтерв'ю, отримує транскрипти й метрики, формує рішення і комунікує з кандидатами. Менеджери й адміністратори мають розширені повноваження для контролю процесів, управління клієнтами, інтеграціями, аналітикою й правами доступу.

Нефункціональні вимоги визначають якісні характеристики системи, які впливають на її ефективність, зручність використання, надійність та масштабованість. Вони забезпечують коректну роботу системи в реальних умовах експлуатації і безпосередньо не пов'язані з виконанням функціональних вимог, однак критично важливі для забезпечення стабільної та високопродуктивної роботи програмного забезпечення.

Юзабіліті визначається інтуїтивно зрозумілим інтерфейсом, де час освоєння базового функціоналу не повинен перевищувати 30 хвилин для рекрутера. Платформа має бути доступною та зручною для користувачів як на мобільних пристроях, так і на десктопах, що забезпечить гнучкість у виборі робочого середовища. Надійність системи повинна бути забезпечена безвідмовною роботою протягом не менше ніж 30 діб, з максимальною кількістю критичних помилок не більше 1 на 1000 запитів під високим навантаженням.

Продуктивність системи повинна гарантувати обробку одночасних запитів від не менше ніж 100 активних користувачів, що важливо для забезпечення безперебійної роботи при великій кількості користувачів. Крім того, платформа повинна мати можливість асинхронної обробки складних завдань, таких як парсинг та завантаження даних, щоб основні функції не блокувалися.

Масштабованість платформи має підтримувати горизонтальне масштабування сервісів для обробки збільшеного навантаження. Також важливою є можливість незалежного масштабування фонових воркерів і черг, що дозволить ефективно обробляти великі обсяги даних без деградації якості користувацького досвіду (UX).

Щодо безпеки, система повинна забезпечувати контроль доступу за ролями та дозволами, щоб обмежити доступ до чутливої інформації. Крім того, необхідне

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

безпечне зберігання файлів резюме, шифрування всіх даних та аудит ключових дій для забезпечення належного рівня захисту. Шифрування трафіку є необхідним для запобігання несанкціонованому доступу до даних під час їх передачі через мережу.

Системні вимоги для цієї платформи включають використання сучасних браузерів, таких як Chrome, Firefox, або Edge, з підтримкою останніх версій стандартів JavaScript та HTML. Програмне забезпечення має бути сумісним із операційними системами Windows, macOS або Linux. Мінімальна конфігурація клієнтського пристрою повинна включати 4 ГБ оперативної пам'яті та стабільне мережеве підключення для роботи з файлами та дзвінками.

Серверна частина платформи повинна працювати на середовищі Node.js для забезпечення високої продуктивності та гнучкості, а база даних PostgreSQL забезпечує надійне зберігання даних. Для зберігання файлів слід використовувати об'єктне сховище з пресайн-URL для підвищення безпеки та доступності файлів. Черви повідомлень та фонові процеси повинні підтримувати горизонтальне масштабування для забезпечення обробки збільшеного навантаження.

Інтеграція з зовнішніми майданчиками для публікацій та подачі вакансій є важливою для забезпечення швидкої публікації оголошень. Додатково, система повинна мати інтеграцію з AI-сервісами для автоматизації інтерв'ю та аналітики подій для збору та аналізу даних, що дозволяє вдосконалювати користувацький досвід та ефективність платформи.

Таким чином, дотримання зазначених нефункціональних вимог і системних характеристик дозволяє забезпечити стабільну роботу платформи для автоматизації процесів найму, забезпечуючи гнучкість, безпеку та ефективність в умовах зростання навантаження.

Нижче наведено типові варіанти використання системи для різних ролей. Вони відповідають основним бізнес-сценаріям: керування вакансіями, обробка резюме, AI-інтерв'ю, аналітика та адміністративні операції (рис. 2.1).

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

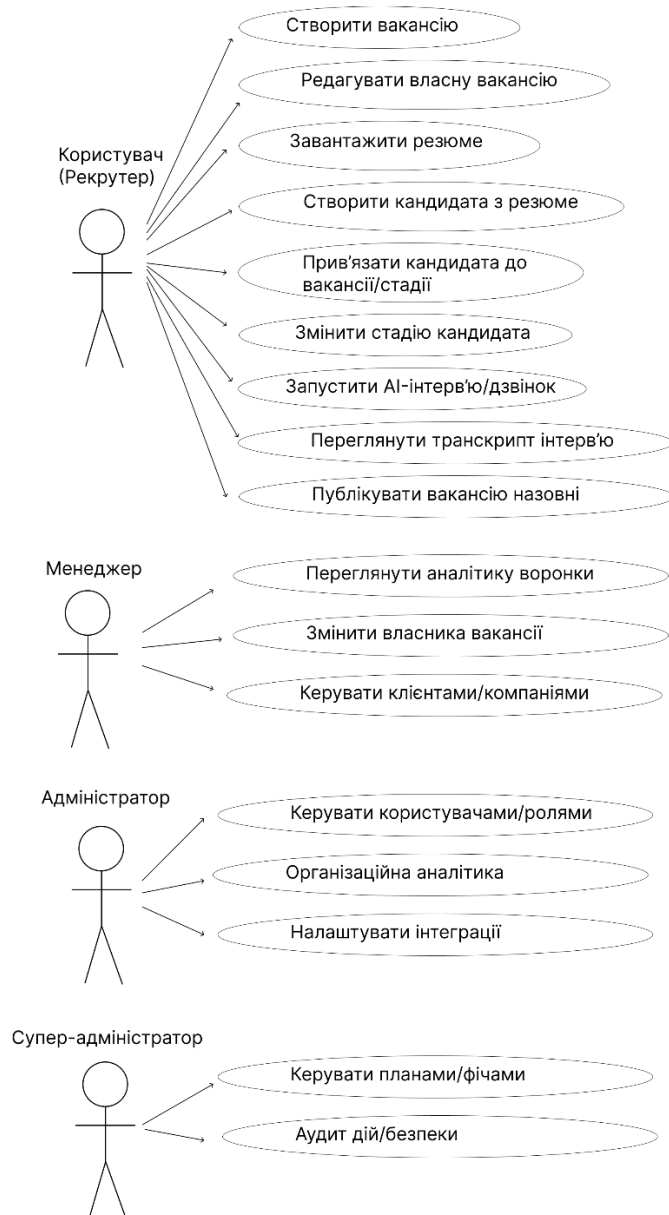


Рис. 2.1 - Діаграма варіантів використання

Опис варіантів використання наведено у таблицях (табл. 2.2 - 2.8). Кожна таблиця детально ілюструє окремий сценарій взаємодії користувачів із платформою автоматизації найму відповідно до їхніх ролей (Користувач/Рекрутер, Менеджер, Адміністратор, Супер-адміністратор). Це дозволяє чітко окреслити функціональні можливості, доступні кожному типу користувача, а також межі доступу до даних і операцій у межах організації.

Подані таблиці показують, як ролі взаємодіють із ключовими підсистемами платформи: керування вакансіями (створення, редагування, публікація), обробка та парсинг резюме (формування профілів кандидатів), керування воронкою та

комунікаціями (стадії, нотатки, історія взаємодій), AI-інтерв'ю/дзвінки (запуск, транскрипти, метрики), аналітика (джерела, конверсії, ефективність етапів), керування клієнтами/компаніями, а також адміністрування прав доступу й інтеграцій. Така структура слугує основою для організації інтерфейсу й навігації під потреби кожної ролі та спрощує подальшу інтеграцію, супровід і розвиток системи.

Таблиця 2.2

Опис прецеденту «Створення вакансії»

Назва варіанту використання	Створення вакансії
Основні актори	Користувач (Recruiter/Member), Менеджер, Адмін
Короткий опис	Користувач створює вакансію, за потреби використовує AI-асистента для описів і вимог; налаштовує етапи відбору; публікує позицію; система сповіщає команду про нову вакансію.

Цей сценарій забезпечує швидкий старт процесу найму: AI допомагає скоротити час на підготовку якісного опису, а попередньо налаштовані етапи воронки стандартизують процес відбору. Менеджер може затвердити/уточнити налаштування, а адміністратор - забезпечити відповідність політикам організації.

Таблиця 2.3

Опис прецеденту «Подача резюме/створення кандидата»

Назва варіанту використання	Подача резюме/створення кандидата
Основні актори	Користувач (Рекрутер), Менеджер
Короткий опис	Завантаження одного чи декількох резюме; парсинг даних (ПІБ, контакти, досвід, навички); автоматичне створення профілю кандидата й прив'язка до вакансії/стадії; збереження файлів у сховище.

Сценарій суттєво мінімізує ручне введення даних, зменшує помилки та прискорює первинний скринінг. Масове завантаження дозволяє швидко обробляти великі потоки відгуків, а збереження оригінальних CV гарантує відтворюваність даних у майбутньому.

Таблиця 2.4

Опис преценденту «Керування кандидатами та етапами»

Назва варіанту використання	Керування кандидатами та етапами
Основні актори	Користувач (Рекрутер), Менеджер
Короткий опис	Зміна стадій, фіксація нотаток, відправлення шаблонних листів, оновлення статусів; усі дії потрапляють до історії взаємодій з можливістю аудиту.

Цей сценарій забезпечує прозорий рух кандидата по воронці, централізує комунікації та підвищує керованість процесу в межах всієї організації. Менеджер має розширений огляд і може втручатися в критичних точках, забезпечуючи SLA та узгоджені рішення.

Таблиця 2.5

Опис преценденту «Проведення AI-інтерв'ю/дзвінка»

Назва варіанту використання	Проведення AI-інтерв'ю/дзвінка
Основні актори	Користувач (Рекрутер), Кандидат
Короткий опис	Ініціація AI-дзвінка/інтерв'ю; запис транскрипту та поведінкових метрик (кліки, увага, технічні параметри сесії); прикріплення результатів до кандидата/стадії.

Цей сценарій дозволяє рекрутерам значно зменшити навантаження, автоматизуючи початкові етапи інтерв'ю, що забезпечує стандартизований підхід до оцінки кандидатів. AI-система не тільки фіксує транскрипти розмови, але й аналізує поведінку кандидата, таких як реакції на запитання, інтонацію, час відповіді та інші метрики, що можуть бути важливими для прийняття рішення. Зібрані дані, разом з результатами інтерв'ю, прикріплюються до профілю кандидата, що дозволяє менеджерам мати швидкий доступ до всієї необхідної інформації для оцінки кандидатів та прийняття обґрунтованих рішень. Завдяки можливості проводити інтерв'ю в автоматизованому режимі, система зменшує ймовірність людських помилок та упередженості, що забезпечує більш справедливий та об'єктивний процес відбору кандидатів.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 2.6

Опис прецеденту «Публікація вакансій на зовнішніх майданчиках»

Назва варіанту використання	Публікація вакансій
Основні актори	Користувач (Рекрутер), Менеджер, Адміністратор
Короткий опис	Вакансія публікується через інтеграції; відгуки приймаються через форми/вебхуки; система автоматично створює кандидатів і зберігає їхні резюме.

Сценарій розширює охоплення кандидатів без дублювання роботи, забезпечує єдине місце істини для всіх відгуків і скорочує час від подачі до появи кандидата у воронці.

Таблиця 2.7

Опис прецеденту «Перегляд аналітики»

Назва варіанту використання	Перегляд аналітики воронки та джерел
Основні актори	Менеджер, Адміністратор, Супер-адміністратор
Короткий опис	Перегляд джерел кандидатів, конверсій по стадіях, часу на етапах, активності під час інтерв'ю; формування звітів і експорт даних.

Цей сценарій підтримує керування на основі даних: дозволяє виявляти вузькі місця процесу, оптимізувати етапи, перерозподіляти навантаження й покращувати ефективність каналів залучення.

Таблиця 2.8

Опис прецеденту «Управління клієнтами/користувачами та правами доступу»

Назва варіанту використання	Управління клієнтами/компаніями й правами доступу
Основні актори	Адміністратор, Супер-адміністратор
Короткий опис	Створення/редагування клієнтів; керування користувачами організації; призначення ролей і дозволів; аудит доступів та ключових дій.

Сценарій гарантує безпеку та відповідність політикам: доступ надається за принципом мінімально необхідних прав, а аудит забезпечує відстежуваність змін і контроль за діями користувачів.

Щоб ефективно реалізувати функціональні можливості, внутрішня структура системи організується за доменними компонентами (класи/сервіси/адаптери), що відповідають за конкретні підсистеми (табл. 2.9). Це забезпечує розширюваність і підтримуваність коду, а також мінімізує зв'язність між модулями.

Таблиця 2.9

Опис основних класів і компонентів системи

Клас	Опис
Position (Вакансія)	Атрибути, статуси, публічність, етапи відбору, зв'язки з клієнтом/організацією, інтеграційні ідентифікатори.
Candidate (Кандидат)	Профіль, контактні дані, навички, прив'язки до вакансій/стадій, статуси, посилання на резюме.
Resume (CV)	Файл у сховищі, метадані, витягнуті поля (освіта, досвід, навички), попередній скоринг.
Stage/Workflow (Етап/Воронка)	Конфігурація етапів; правила переходів; SLA; автоматизовані дії/сповіщення.
Interview	Дані AI-інтерв'ю/дзвінка; транскрипти; поведінкові метрики; результати.
Client/Company	Дані клієнта/компанії, реквізити, пов'язані вакансії, умови співпраці.
User	Облікові записи; організації; налаштування; аудити входів.
Role/Permission	Ролі та дозволи; політики видимості даних; мапінг на дії в UI/API.
CommunicationHistory	Історія листування/дзвінків/подій; статуси дзвінків; вкладення.
Notification	Події/тригери; канали (внутрішні/пошта); шаблони повідомлень.

Analytics	Агрегації по джерелах, етапах, конверсіях; дашборди; експорт.
IntegrationAdapter	Адаптери до зовнішніх майданчиків, AI-провайдерів, файлового сховища.
PositionService	Бізнес-логіка вакансій: CRUD, AI-генерація, публікації, права доступу.
CandidateService	Бізнес-логіка кандидатів: створення з резюме, стадії, комунікації.
InterviewService	Ініціація/управління AI-інтерв'ю; збереження транскриптів/метрик.
AnalyticsService	Збір подій; побудова аналітики; формування звітів і вітрин.

Завдяки такій структурі система зберігає чітке розмежування відповідальностей, підтримує швидке додавання нових інтеграцій і AI-функцій, а також забезпечує масштабованість і зручність супроводу в умовах росту навантаження та вимог бізнесу.

2.2. Розробка бази даних системи

Для зберігання даних у платформі автоматизації процесів найму було спроектовано реляційну базу даних (PostgreSQL), що забезпечує високу надійність і продуктивність при обробці великих обсягів інформації. Модель даних побудовано з урахуванням нормалізації, чіткого поділу відповідальностей між сутностями, масштабованості (для забезпечення росту обсягів кандидатів/вакансій), підтримки історії взаємодій і аналітики.

Структура БД охоплює ключові аспекти HR-процесу: організації/користувачі/права доступу, вакансії та їхні етапи, кандидати й резюме, інтерв'ю/дзвінки, історію комунікацій, клієнтів/компанії, нотифікації, інтеграції та метрики.

Нижче подано зведену таблицю (табл. 2.10) з описом основних таблиць бази даних.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

Структура бази даних

Назва таблиці	Призначення
organizations	Облік організацій (компаній), у межах яких працюють користувачі та відкриваються вакансії.
users	Облік користувачів платформи, їх базові дані, зв'язок із організаціями.
org_memberships	Членства користувачів в організаціях, їх ролі/дозволи.
roles	Довідник ролей (Користувач/Рекрутер, Менеджер, Адміністратор, Супер-адміністратор).
permissions	Довідник дозволів (читання/створення/редагування вакансій, кандидатів, клієнтів тощо).
role_permissions	Зв'язок ролей із дозволами (матриця доступу).
clients	Дані клієнтів/компаній-замовників, від імені яких відкриваються вакансії.
positions	Вакансії: опис, статус, параметри публікації, налаштування, зв'язки з організацією/клієнтом.
stages	Шаблони/етапи воронки найму (наприклад, "Screening", "Interview", "Offer").
position_stages	Відображення етапів на конкретну вакансію (послідовність, SLA, прапорці).
candidates	Кандидати: персональні дані, контактна інформація, базові атрибути.
candidate_positions	Зв'язок кандидат ↔ вакансія з поточним етапом, статусом, датами переходів.
resumes	Резюме (CV): файл, метадані, результати парсингу.
interviews	Інтерв'ю/дзвінки: параметри AI-інтерв'ю, транскрипти, результати, метрики.
communication_history	Журнал взаємодій (листи, дзвінки, коментарі) з прив'язкою до кандидата/вакансії.
notifications	Системні нотифікації (кому, коли, про що).
integrations	Налаштування інтеграцій (зовнішні майданчики/сервіси), ключі/параметри.
feature_plans	Тарифні плани та пов'язані фічі (доступність можливостей за планами).
analytics_events	Події для аналітики (джерела кандидатів, конверсії, активність інтерв'ю).

Надалі детально наведено приклади таблиць і ключових полів, які відображають критичні частини домену рекрутингу.

Для початку розглянемо таблицю organizations (табл. 2.11). Вона зберігає дані про організації (компанії), у межах яких створюються вакансії, ведеться воронка найму та керуються користувачі. Служить кореневою сутністю для ізоляції даних (multi-tenant).

Таблиця 2.11

Опис таблиці organizations

Поле	Тип	Призначення
id	uuid	Ідентифікатор організації
name	text	Назва організації
created_at	timestamp	Дата створення запису
updated_at	timestamp	Дата останнього оновлення

Таблиця users (табл. 2.12) репрезентує облікові записи користувачів платформи. Персональні дані зберігаються мінімально, розширені атрибути доступу - через членства в організаціях.

Таблиця 2.12

Опис таблиці users

Поле	Тип	Призначення
id	uuid	Ідентифікатор користувача
email	text	Електронна пошта
full_name	text	Повне ім'я
is_active	boolean	Ознака активності облікового запису
created_at	timestamp	Дата створення запису
updated_at	timestamp	Дата останнього оновлення

Таблиця org_memberships (табл. 2.13) фіксує приналежність користувача до організації з вказанням ролі. Базова точка для політик доступу на рівні тенанта. Кожен запис у таблиці визначає зв'язок між користувачем, організацією та його роллю, що дозволяє гнучко управляти правами доступу на основі ролей у межах конкретної організації. Ця таблиця є важливим елементом для реалізації

авторизації та аутентифікації користувачів в системі, що дозволяє обмежувати або надавати доступ до різних функцій платформи відповідно до ролі в організації.

Таблиця 2.13

Опис таблиці org_memberships

Поле	Тип	Призначення
id	uuid	Ідентифікатор членства
org_id	uuid (FK→organizations.id)	Посилання на організацію
user_id	uuid (FK→users.id)	Посилання на користувача
role_id	uuid (FK→roles.id)	Роль користувача в організації
created_at	timestamp	Дата створення запису

Таблиця roles (табл. 2.14) - довідник ролей у системі. Використовується для призначення прав через матрицю дозволів.

Таблиця 2.14

Опис таблиці roles

Поле	Тип	Призначення
id	uuid	Ідентифікатор ролі
code	text	Код ролі (member, manager, admin, super_admin)
name	text	Локалізована назва ролі

Таблиця permissions (табл. 2.15) це довідник атомарних дозволів (читання, створення, редагування, видалення) у розрізі доменів (вакансії, кандидати, клієнти тощо).

Таблиця 2.15

Опис таблиці permissions

Поле	Тип	Призначення
id	uuid	Ідентифікатор дозволу
code	text	Код дозволу (напр., positions.write_own)
description	text	Опис призначення дозволу

Таблиця `role_permissions` реалізує відношення “багато-до-багатьох” між ролями та дозволами; на її основі формується матриця доступу [28] (табл. 2.16).

Таблиця 2.16

Опис таблиці `role_permissions`

Поле	Тип	Призначення
<code>role_id</code>	<code>uuid (FK→roles.id)</code>	Посилання на роль
<code>permission_id</code>	<code>uuid (FK→permissions.id)</code>	Посилання на дозвіл

Таблиця `clients` (табл. 2.17) містить відомості про клієнтів/компанії-замовників (B2B), від імені яких відкриваються вакансії. Дозволяє групувати позиції та будувати аналітику по замовнику.

Таблиця 2.17

Опис таблиці `clients`

Поле	Тип	Призначення
<code>id</code>	<code>uuid</code>	Ідентифікатор клієнта
<code>org_id</code>	<code>uuid (FK→organizations.id)</code>	Організація-власник запису
<code>name</code>	<code>text</code>	Назва компанії/клієнта
<code>country_id</code>	<code>integer</code>	Країна (за довідником)
<code>created_at</code>	<code>timestamp</code>	Дата створення запису

Таблиця `positions` (табл. 2.18) є центральною сутністю процесу найму - вакансією. Вона зберігає ключові параметри опису вакансії, такі як її назва, детальний опис, статус та публічність, а також прив'язку до організації та клієнта, що допомагає визначити, хто є замовником цієї вакансії. Важливою частиною є збереження зовнішніх ідентифікаторів публікацій, що дозволяє інтегрувати вакансії з іншими системами, зокрема для відслідковування їхнього поширення на зовнішніх платформах. Поле `status` вказує на поточний стан вакансії, який може бути активним (`active`), чернеткою (`draft`) або архівованим (`archived`), що допомагає контролювати її життєвий цикл. Параметр `is_public` визначає, чи є вакансія доступною для широкого загалу, що впливає на її видимість на платформі або в

інших публічних ресурсах. Зберігання цих даних дає змогу здійснювати гнучке керування вакансіями та взаємодіями з ними в межах платформи.

Таблиця 2.18

Опис таблиці positions

Поле	Тип	Призначення
id	uuid	Ідентифікатор вакансії
org_id	uuid (FK→organizations.id)	Організація
client_id	uuid (FK→clients.id)	Клієнт/замовник (необов'язково)
title	text	Назва позиції
description	text	Опис (може бути AI-згенерований)
status	text	Статус (active/draft/archived)
is_public	boolean	Публічність позиції
author_id	text/uuid	Власник (автор) позиції
external_id	text	Зовнішній ідентифікатор публікації
created_at	timestamp	Дата створення
updated_at	timestamp	Дата оновлення

Таблиця stages (табл. 2.19) це довідник етапів воронки найму (шаблони), які можуть застосовуватись до різних вакансій. Вона забезпечує повторне використання та стандартизацію процесу.

Таблиця 2.19

Опис таблиці stages

Поле	Тип	Призначення
id	uuid	Ідентифікатор етапу
name	text	Назва
description	text	Опис/призначення
order_index	integer	Порядок у шаблоні

Таблиця position_stages накладає шаблон етапів на конкретну вакансію, зберігаючи порядок, SLA та інші специфічні налаштування воронки саме для цієї позиції (табл. 2.20).

Таблиця 2.20

Опис таблиці position_stages

Поле	Тип	Призначення
id	uuid	Ідентифікатор запису
position_id	uuid (FK→positions.id)	Посилання на вакансію
stage_id	uuid (FK→stages.id)	Посилання на шаблон етапу
order_index	integer	Порядок етапу для вакансії
sla_hours	integer	Очікуваний термін проходження етапу (години)

Таблиця candidates зберігає дані про кандидатів. Мінімальний профіль (контакти) доповнюється даними з резюме, інтерв'ю та історії комунікацій (табл. 2.21).

Таблиця 2.21

Опис таблиці candidates

Поле	Тип	Призначення
id	uuid	Ідентифікатор кандидата
org_id	uuid (FK→organizations.id)	Організація
full_name	text	ПІБ
email	text	Електронна пошта
phone	text	Телефон
created_by	text/uuid	Хто створив/імпортував
created_at	timestamp	Дата створення

Таблиця candidate_positions фіксує участь кандидата у відборі на конкретну вакансію, зберігає поточний етап, статус просування та часові мітки переходів (табл. 2.22). Вона дозволяє відстежувати рух кандидата через різні етапи найму, зберігаючи інформацію про його статус у процесі, наприклад, чи знаходиться він на етапі відбору, чи отримав пропозицію або був прийнятий на роботу.

Таблиця 2.22

Опис таблиці candidate_positions

Поле	Тип	Призначення
id	uuid	Ідентифікатор зв'язку
candidate_id	uuid (FK→candidates.id)	Кандидат
position_id	uuid (FK→positions.id)	Вакансія
current_stage_id	uuid (FK→position_stages.id)	Поточний етап у воронці позиції
status	text	Статус (in_progress/rejected/offered/hired)
updated_at	timestamp	Дата останнього оновлення

Таблиця resumes управляє файлами резюме та метаданими парсингу; великі файли зберігаються в об'єктному сховищі, у БД - лише посилання та структуровані поля (табл. 2.23).

Таблиця 2.23

Опис таблиці resumes

Поле	Тип	Призначення
id	uuid	Ідентифікатор резюме
candidate_id	uuid (FK→candidates.id)	Кандидат
filename	text	Назва файлу
object_path	text	Шлях у сховищі
parsed_json	jsonb	Результати парсингу (освіта, досвід, навички)
created_at	timestamp	Дата завантаження

Таблиця interviews зберігає дані AI-інтерв'ю/дзвінків: посилання на сесію, транскрипти, поведінкові метрики та підсумкові оцінки; прив'язується до кандидата і позиції (табл. 2.24). Вона дозволяє детально зберігати результати інтерв'ю, включаючи транскрипти відповідей та поведінкові показники, такі як кліки, фокус уваги, пристрій та географічні дані, що дозволяє аналізувати ефективність взаємодії з кандидатами. Ця таблиця є важливою для подальшого

оцінювання кандидатів, порівняння їхніх результатів та прийняття рішень на основі отриманих даних.

Таблиця 2.24

Опис таблиці interviews

Поле	Тип	Призначення
id	uuid	Ідентифікатор інтерв'ю
candidate_id	uuid (FK→candidates.id)	Кандидат
position_id	uuid (FK→positions.id)	Вакансія
link	text	Посилання/ідентифікатор сесії
transcript	text/jsonb	Транскрипт розмови/відповідей
metrics	jsonb	Метрики
status	text	Статус (started/finished/analyzed)
created_at	timestamp	Початок інтерв'ю
finished_at	timestamp	Завершення інтерв'ю

Таблиця communication_history (табл. 2.25) - це універсальний журнал комунікацій, який агрегує листи, дзвінки, коментарі; забезпечує прозорість і відтворюваність взаємодій у процесі найму. Вона зберігає деталі про кожну взаємодію з кандидатом, включаючи джерело комунікації, її тип і зміст, що дає змогу точно відслідковувати історію комунікацій. Завдяки цьому, усі взаємодії, будь то електронні листи, телефонні дзвінки чи системні повідомлення, можуть бути легко знайдені та проаналізовані. Це дозволяє забезпечити ефективну комунікацію між усіма сторонами процесу найму, мінімізуючи можливість помилок і сприяючи кращій координації між учасниками. Таблиця також дозволяє здійснювати детальний аналіз взаємодій за часом, що може бути корисно для виявлення проблемних етапів або покращення процесів комунікації. Зберігання зовнішніх ідентифікаторів, таких як ID дзвінків, дозволяє інтегрувати комунікаційні дані з іншими системами та сервісами для подальшого автоматизованого оброблення та аналітики.

Опис таблиці communication_history

Поле	Тип	Призначення
id	uuid	Ідентифікатор запису
candidate_id	uuid (FK→candidates.id)	Кандидат
position_id	uuid (FK→positions.id)	Вакансія
origin	text	Джерело (email/phone/system)
subject	text	Тема/тип взаємодії
message_text	text	Зміст повідомлення/замітки
message_id	text	Зовнішній ідентифікатор (ID дзвінка тощо)
status	text	Статус (для дзвінків: in_progress/succeeded/failed)
created_at	timestamp	Дата/час фіксації

Таблиця notifications (табл. 2.26) - це модель системних сповіщень. Дозволяє інформувати користувачів про події у процесі найму та фіксувати доставку/перегляд. Кожне сповіщення містить деталі про подію, такі як тип (наприклад, зміна етапу вакансії) та додаткову інформацію у вигляді JSON. Це дозволяє забезпечити ефективний контроль за повідомленнями та їхньою доставкою користувачам.

Таблиця 2.26

Опис таблиці notifications

Поле	Тип	Призначення
id	uuid	Ідентифікатор нотифікації
org_id	uuid	Організація
user_id	uuid	Отримувач
type	text	Тип події
payload	jsonb	Деталі події
created_at	timestamp	Дата створення

Таблиця integrations (табл. 2.27) це конфігурації інтеграцій із зовнішніми сервісами (публікація вакансій, прийом відгуків, аналітика). Зберігає ключі, параметри та стан.

Таблиця 2.27

Опис таблиці integrations

Поле	Тип	Призначення
id	uuid	Ідентифікатор
org_id	uuid (FK→organizations.id)	Організація
provider	text	Провайдер
settings	jsonb	Ключі та конфігурація
is_active	boolean	Ознака активності інтеграції

Таблиця feature_plans підтримує тарифні плани та доступні фічі. Використовується для ввімкнення/вимкнення можливостей на рівні компанії (табл. 2.28).

Таблиця 2.28

Опис таблиці feature_plans

Поле	Тип	Призначення
id	uuid	Ідентифікатор плану/зв'язку
plan_name	text	Назва плану
feature_code	text	Код фічі
description	text	Опис фічі/права

Таблиця analytics_events (табл. 2.29) зберігає аналітичні події для побудови дашбордів (джерела кандидатів, конверсії, активність на інтерв'ю, час на етапах тощо). Вона дозволяє детально відстежувати різноманітні події, що відбуваються протягом процесу найму, і використовувати ці дані для побудови звітів та аналітики. За допомогою цієї таблиці можна аналізувати, з яких джерел приходять кандидати, як вони просуваються через етапи відбору, а також вимірювати ефективність інтерв'ю. Зберігання цих даних у структурованому форматі дозволяє зручно обробляти великі обсяги інформації та отримувати інсайти для оптимізації процесів найму.

Опис таблиці analytics_events

Поле	Тип	Призначення
id	uuid	Ідентифікатор події
org_id	uuid	Організація
position_id	uuid	Вакансія (необов'язково)
candidate_id	uuid	Кандидат (необов'язково)
event_type	text	Тип події
payload	jsonb	Деталі події
created_at	timestamp	Час події

Діаграма бази даних (рис. 2.2) ілюструє зв'язки між основними таблицями, які формують повний цикл найму: “організація → вакансії → етапи → кандидати → рух у воронці → інтерв'ю → комунікації → аналітика”, а також адміністративні підсистеми: “користувачі/ролі/дозволи”, “клієнти/інтеграції”, “нотифікації/події”.

Така структура забезпечує цілісність і узгодженість даних, використовуючи зовнішні ключі для зв'язку між таблицями та нормалізацію для зменшення дублювання даних. Вона також підтримує масштабованість, оскільки великі сутності, такі як кандидати, резюме та події, виділені в окремі таблиці, що дозволяє легко масштабувати систему без зниження продуктивності. Це спрощує аналітику, оскільки дані зберігаються у форматі jsonb для метаданих інтерв'ю та подій, а чітка структура зв'язків забезпечує швидкий доступ до інформації про рух кандидатів по воронці відбору. Крім того, така модель гарантує безпеку та контроль доступу, оскільки ізоляція на рівні організацій, ролей та дозволів дозволяє чітко обмежити доступ до чутливих даних, залежно від прав користувача.

Це дозволяє не лише забезпечити ефективну роботу платформи, але й легко адаптувати систему до змін в процесах найму або вимогах бізнесу, забезпечуючи її гнучкість та надійність у довгостроковій перспективі. Також така архітектура забезпечує високу швидкість обробки запитів і дозволяє ефективно працювати з великими обсягами даних, що є критично важливим для масштабованих платформ найму.

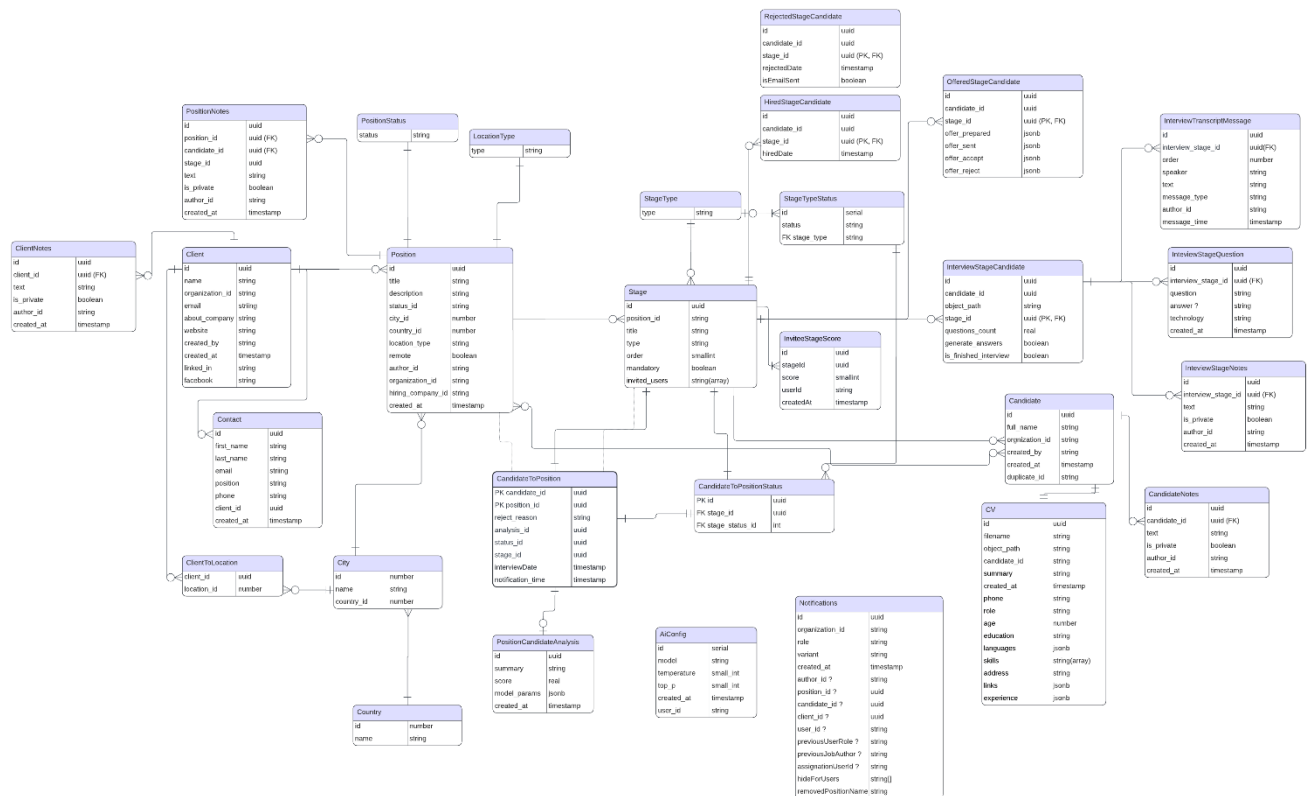


Рис. 2.2 - Діаграма бази даних системи

Таке проєктування бази даних забезпечує ефективний доступ до інформації, сумісність із ORM-інструментами [6], можливість масштабування системи для обробки зростаючого обсягу даних та підвищення ефективності аналітичних запитів. База даних дає змогу зберігати усі дані, необхідні для прозорого, відтворюваного та вимірюваного процесу найму з підтримкою AI-функціоналу.

2.3. Проєктування та реалізація алгоритмів роботи системи

Платформа автоматизації найму реалізує узгоджені алгоритми для всього рекрутингового циклу: автентифікація та рольова авторизація, створення і публікація вакансій з AI-асистуванням, прийом і парсинг резюме, керування воронкою відбору та історією комунікацій, проведення AI-інтерв'ю/дзвінків із фіксацією транскриптів і метрик, обробка зовнішніх відгуків через вебхуки та збір аналітики в режимі, наближеному до реального часу. Після авторизації користувач потрапляє у свій робочий простір організації; набір дозволених дій визначається роллю і наданими дозволами, які перевіряються як на клієнті, так і на сервері. Інтерфейс поділено на модулі вакансій, кандидатів, інтерв'ю та аналітики; для

адаптації під мобільні пристрої використовуються відповідні UI-шаблони. Далі подано ключові алгоритми з короткими ілюстраціями коду.

Алгоритм авторизації та перевірки доступу побудований на сесії користувача і контексті запиту, що містить ідентифікатор організації та перелік дозволів. На сервері контекст надає методи для примусової перевірки авторизованості та наявності дозволів; на клієнті відображення розділів і дій додатково захищається гард-компонентами. Це забезпечує єдину модель контролю доступу та узгодженість з UI. (лістинг 2.1).

```
return {
  auth: auth as AuthObject & { orgPermissions: OrgPermission[] },
  ip,
  authRequired() {
    if (!auth.userId || !auth.orgId) {
      throw new TRPCError({ code: 'UNAUTHORIZED' });
    }
  },
  isAuthenticated() {
    return !!auth.userId || !!auth.orgId;
  },
  hasOneOfPermissions(permissions: OrgPermission[]) {
    return permissions.some((permission) => permissionSet.has(permission));
  },
  hasPermission(permission: OrgPermission) {
    return permissionSet.has(permission);
  },
};
```

Лістинг 2.1 - Перевірка автентифікації та дозволів

Алгоритм створення вакансії поєднує класичний CRUD і AI-асистування. Користувач формує чернетку, за потреби застосовує генерацію опису, задає етапи воронки, зберігає позицію й може відразу опублікувати її на зовнішньому майданчику. При публікації використовується інтеграційний адаптер, який створює або оновлює запис на стороні сервісу та повертає зовнішній ідентифікатор; цей ідентифікатор далі використовується для прийому відгуків через вебхуки.

Алгоритм прийому та парсингу резюме складається з двох етапів: спочатку система створює кандидата і запис про CV, далі повертає пресайн-URL для безпечного завантаження файлу у сховище безпосередньо з клієнта. Після завершення завантаження бекенд оновлює шлях до об'єкта і ставить задачу на

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

парсинг, що заповнює структуровані поля профілю кандидата. Приклад запити пресайн-URL та зв'язування з вакансією (лістинг 2.2).

```
export async function requestCvUpload(ctx: TrpcContext, args: RequestCvUploadArgs) {
  const { positionId, filenames, tel, email } = args;
  const { auth } = ctx;
  const [positionDto] = await db
    .select({
      position: position,
    })
    .from(position)
    .where(equal(position.id, positionId));

  if (!positionDto.position.isPublic) {
    ctx.authRequired();
    throwIfNoMatchingPermissions(ctx, [OrgPermissionValue.OrgCreateCandidate]);
  }

  if (!positionDto) throw new Error('Position not found');

  return await db.transaction(async (tx) => {
    let orgId = auth.orgId;
    const userId = auth.userId ?? positionDto.position.authorId;

    logger.info('Getting presign url');
    const { presignedUrl, objectPath } = await presignUrl({
      orgId: auth.orgId,
      fileKey: cvId + ext,
      candidateId,
      positionId,
    });

    logger.info('Updating cv record with candidate CV data');
    await tx.update(cv).set({ objectPath }).where(equal(cv.id, cvId));

    return { presignedUrl, fileId: cvId };
  });
}
```

Лістинг 2.2 - Підготовка до завантаження резюме

Створення кандидата під час CV-флоу виконується транзакційно: спочатку запис кандидата, потім CV і повернення ідентифікаторів для подальших операцій.

Це гарантує цілісність зв'язку «кандидат ↔ CV» (лістинг 2.3).

```
export function createCandidateForCv(orgId: string, createdBy: string, filename: string) {
  return db.transaction(async (tx) => {
    const [{ id: candidateId }] = await tx
      .insert(candidate)
      .values({
        organizationId: orgId,
        createdBy,
      })
      .returning({
        id: candidate.id,
      });
  });
}
```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const [{ id: cvId }] = await tx
  .insert(cv)
  .values({
    candidateId,
    filename,
  })
  .returning({
    id: cv.id,
  });

return {
  candidateId,
  cvId,
};
});
}

```

Лістинг 2.3 - Транзакційне створення кандидата та CV

Алгоритм прийому зовнішніх відгуків через вебхук дозволяє автоматично створювати кандидатів для опублікованих вакансій. Сервіс приймає форму з резюме, визначає публікацію, створює кандидата, отримує пресайн-дані та відвантажує файл у сховище, одночасно прокидаючи метадані для подальшого AI-інтерв'ю (лістинг 2.4).

```

webhooksRouter.post('/position/apply', upload.single('resume'), async (req, res) => {
  const { orgId, positionId, fullName, email, phone, linkedIn } = req.body as {
    orgId: string;
    positionId: string;
    fullName: string;
    email: string;
    phone: string;
    linkedIn?: string;
  };
  const file = req.file as Express.Multer.File;

  if (!orgId || !positionId || !file) return res.status(404).send();
  const [findPosition] = await db
    .select({ authorId: position.authorId, })
    .from(position)
    .where(and(eq(position.id, positionId), eq(position.organizationId, orgId)));
  if (!findPosition) return res.status(404).send();

  const fileBuffer = file.buffer;
  const originalname = file.originalname;
  const filename = originalname.endsWith('.pdf') ? originalname : `${Date.now()}.pdf`;
  const { candidateId, cvId } = await createCandidateForCv(
    positionId,
    orgId,
    findPosition.authorId,
    filename
  );
});

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

```
const { presignedUrl, objectPath } = await presignUrl({
  orgId,
  fileKey: `${cvId}.pdf`,
  candidateId,
  positionId,
  generateaiinterview: 'yes',
  fullName,
  email,
  phoneNumber: phone,
  linkedIn,
});
```

Лістинг 2.4 - Прийом публічної подачі на позицію

Алгоритм AI-інтерв'ю забезпечує як телефонні, так і веб-сесії з фіксацією транскриптів та поведінкових метрик кандидата. Після завершення сесії система обчислює геолокацію за IP, збагачує метрики і записує їх до етапу кандидата у воронці. Це забезпечує відтворюваність результатів інтерв'ю, зіставлення контексту та подальший аналіз (лістинг 2.5).

```
export async function recordCheatStats(args: UpdateCheatStatsArgs) {
  const { copiedMessageCount, unfocusCount, devToolsWereOpen, retellLink, analytics } = args;
  let location: string | undefined;
  let flagUrl: string | undefined;
  try {
    if (analytics?.userIp) {
      const res = await fetch(`https://ipwho.is/${analytics.userIp}`);
      if (res.ok) {
        const j = await res.json();
        if (j?.success !== false) {
          const city = j?.city as string | undefined;
          const region = j?.region as string | undefined;
          const country = j?.country as string | undefined;
          location = [city, region, country].filter(Boolean).join(', ') || undefined;
          flagUrl = (j?.flag?.img as string | undefined) || undefined;
        }
      }
    } catch (e) { logger.error(e); }
    return await db.transaction(async (tx) => {
      const [result] = await tx
        .select({
          candidateId: retellInterview.candidateId,
          stageId: candidateToPosition.stageId,
        })
        .from(retellInterview)
        .innerJoin( candidateToPosition, (
          eq(candidateToPosition.positionId, retellInterview.positionId),
          eq(candidateToPosition.candidateId, retellInterview.candidateId)
        )
        )
        .where(and(eq(retellInterview.link, retellLink), isNull(retellInterview.deletedAt)));
      return tx
        .update(interviewStageCandidate)
        .set({
          messageCopiedCount: copiedMessageCount,
```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

```

pageUnfocusCount: unfocusCount,
devToolsTriggered: devToolsWereOpen,
analytics: {...analytics, location, flagUrl, },
})
.where(
  and(
    eq(interviewStageCandidate.candidateId, result.candidateId),
    eq(interviewStageCandidate.stageId, result.stageId)
  ));});}

```

Лістинг 2.5 - Збагачення метрик AI-інтерв'ю

Контроль доступу та матриця прав

Контроль доступу та матриця прав у системі реалізовані через дозволи на рівні організації. Кожна дія на сервері перевіряється відповідним пермішеном, що гарантує, що користувачі можуть виконувати лише ті дії, які дозволені їхніми правами доступу. На клієнтській стороні розділи, кнопки та інші елементи інтерфейсу відображаються лише в тому випадку, якщо користувач має відповідні права для взаємодії з ними, що дозволяє забезпечити чітку організацію доступу до функцій.

Це забезпечує прозорий розподіл повноважень між різними типами користувачів, такими як Користувач, Менеджер, Адміністратор і Супер-адміністратор, та гарантує, що кожен користувач має доступ тільки до тих даних і функцій, що відповідають його ролі в системі. Узагальнення рівнів доступу для основних функцій наведено в матриці прав, де для кожної функції зазначено рівень доступу: 0 - немає доступу, 1 - доступ лише до власних даних, 2 - повний доступ у межах організації, що чітко корелює з перевітками дозволів у коді.

Така система (табл. 2.30) дозволяє гнучко управляти доступом на всіх етапах роботи з платформою, забезпечуючи як безпеку, так і зручність використання. Це також дозволяє легко масштабувати систему і адаптувати її до різних вимог безпеки та бізнес-логіки, додаючи нові ролі або змінюючи дозволи для існуючих користувачів. Крім того, контроль доступу за допомогою матриці прав сприяє ефективному моніторингу та аудиту діяльності користувачів, забезпечуючи відповідність політикам безпеки та знижуючи ризик несанкціонованого доступу до чутливої інформації.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

Матриця доступу до функцій системи

Функція	Користувач (Рекрутер)	Менеджер	Адміністратор	Супер-адміністратор
Перегляд вакансій	1	2	2	2
Створення/редагування власних вакансій	1	2	2	2
Зміна власника вакансії	1	2	2	2
Публікація вакансій на зовнішніх майданчиках	1	2	2	2
Перегляд кандидатів	1	2	2	2
Створення кандидатів/завантаження резюме	1	2	2	2
Керування воронкою (стадії/статуси)	1	2	2	2
Проведення AI-інтерв'ю, перегляд транскриптів	1	2	2	2
Перегляд аналітики	1	2	2	2
Керування клієнтами/компаніями	1	2	2	2
Керування користувачами/ролями/дозволами	0	1	2	2
Керування інтеграціями	0	1	2	2
Керування планами/фічами	0	0	1	2

Основні функції, доступні кожному типу користувача, чітко розмежовані, щоб забезпечити оптимальний рівень доступу та контролю. Наприклад, рекрутер може лише переглядати вакансії та працювати з власними кандидатами, тоді як менеджер має більший доступ, зокрема до створення та редагування вакансій, а адміністратор і супер-адміністратор мають повний контроль над усіма аспектами платформи, включаючи керування користувачами, ролями та інтеграціями. Кожен рівень доступу відповідає конкретним вимогам щодо управління даними та функціями, що дозволяє забезпечити ефективне адміністрування та безпеку. Така матриця прав доступу дає змогу гнучко керувати доступом до різних функцій

системи, забезпечуючи чітке розмежування відповідальності та повноважень серед користувачів.

Діаграма активності (рис. 2.3) наочно ілюструє, як різні ролі користувачів взаємодіють з платформою для автоматизації процесів найму. Кожна роль - рекрутер, менеджер, адміністратор і супер-адміністратор - має власний набір доступних функцій, що забезпечує відповідний рівень доступу до вакансій, кандидатів, аналітики та управління правами доступу. Діаграма показує, як кожен користувач взаємодіє з системою в межах своїх дозволених функцій, таких як створення і редагування вакансій, перегляд кандидатів, управління воронкою відбору або керування клієнтами та інтеграціями. Це дозволяє чітко розрізняти ролі, визначати їх обов'язки та забезпечує ефективне управління правами доступу в межах організації, гарантуючи прозорість і контроль на всіх етапах процесу найму.

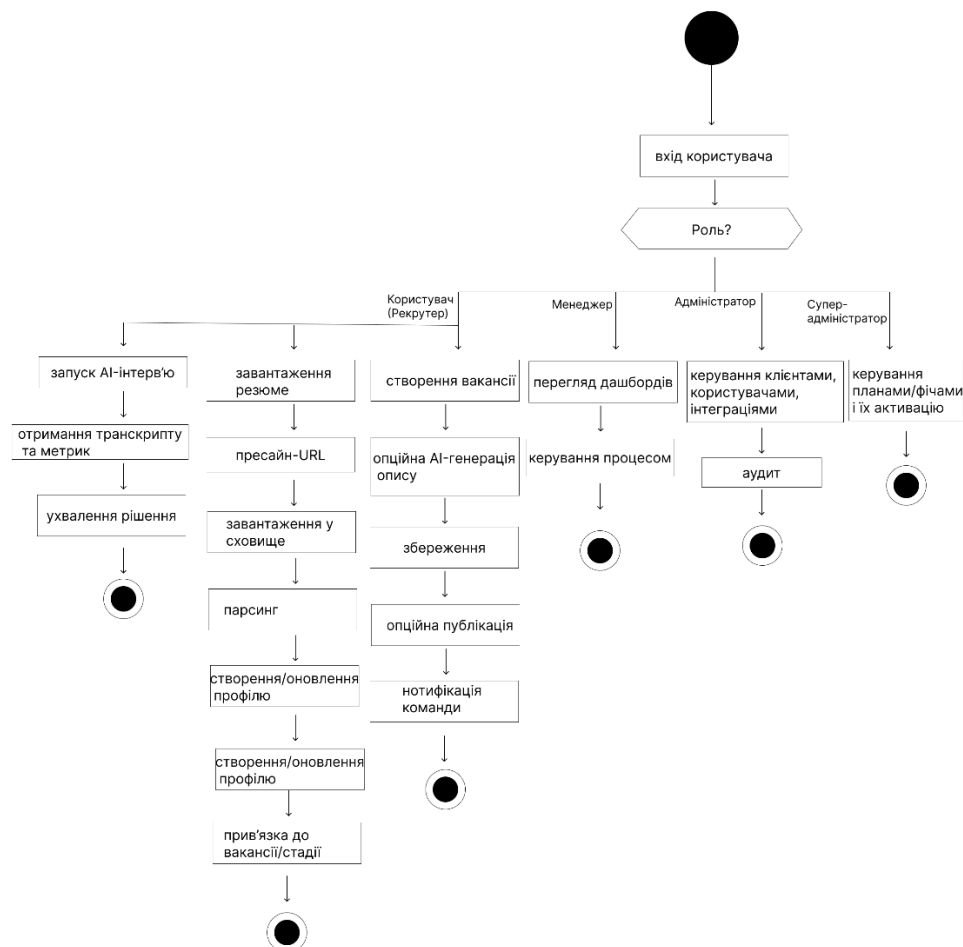


Рис. 2.3 - Діаграма активності

Висновки до другого розділу

У процесі проєктування платформи автоматизації найму було визначено варіанти використання для чотирьох ролей, сформовано об'єктно-орієнтовану структуру домену (вакансії, кандидати, етапи, інтерв'ю, комунікації, клієнти), а також спроектовано реляційну базу даних із підтримкою історії взаємодій, аналітики та інтеграцій. Реалізовано ключові алгоритми: рольову авторизацію, створення і публікацію вакансій з AI-асистуванням, безпечний прийом і парсинг резюме з пресайн-завантаженнями, керування воронкою відбору, проведення AI-інтерв'ю з транскриптами та метриками, обробку зовнішніх відгуків через вебхуки, нотифікації та збір аналітики. Запропонована архітектура забезпечує масштабованість, безпеку [24] і відтворюваність процесу, а також підтримує прийняття рішень на основі даних завдяки вбудованим дашбордам і подієвій моделі аналітики. Це дає змогу ефективно керувати вакансіями, кандидатами і командами рекрутингу та швидко розвивати систему шляхом підключення нових інтеграцій і AI-функцій.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ

3.1. Порядок встановлення та налаштування параметрів системи

Розроблена платформа автоматизації найму реалізована як монопозиторій на базі Nx [13] з використанням Node.js [1] для серверної частини та React з TypeScript для клієнтської частини. Встановлення та запуск системи здійснюється локально або на серверному середовищі з підтримкою Node.js, PostgreSQL та Docker.

Перед запуском необхідно встановити наступне програмне забезпечення:

- Node.js (версія 20 або новіша);
- Docker [12] та Docker Compose для локального середовища розробки;
- PostgreSQL (версія 15 або новіша) або використання Docker-контейнера;
- Git для клонування репозиторію;
- Infisical CLI для управління секретами.

Після підготовки середовища користувач має клонувати репозиторій системи з GitHub. Структура проєкту містить кілька основних каталогів: apps/ (додатки - web, api, api-private), libs/ (бібліотеки за доменами), tools/ (інфраструктурні скрипти) та конфігураційні файли.

Серверна частина розроблена за допомогою Node.js з використанням Express та tRPC. У конфігураційному файлі .env зазначаються параметри підключення до бази даних (лістинг 3.1).

```
DATABASE_URL=postgres://postgres:password@localhost:5436/makehire
```

Лістинг 3.1 - Параметри підключення до бази даних

Після цього необхідно встановити залежності проєкту за допомогою команди `npm install`. Для запуску локального середовища розробки використовується команда `npm run env:up` для підняття Docker-контейнерів з базою даних, потім `npm run db:migrate` для виконання міграцій бази даних, і нарешті `npm start` для запуску всіх сервісів. Веб-додаток буде доступний за адресою `http://localhost:4200`, а API - за адресою `http://localhost:3333`.

Клієнтська частина використовує React з TypeScript та TanStack Router [16] для навігації між сторінками.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

Перейдемо до налаштування бази даних. Після встановлення залежностей потрібно ініціалізувати базу даних. Структура бази даних відповідає таблицям, описаним у розділі 2.2. Для створення таблиць виконується команда `npm run db:migrate`. Для доступу до бази даних через веб-інтерфейс можна використовувати команду `npm run db:studio`.

Після налаштування серверної частини, клієнтської частини та бази даних, система готова до роботи. Користувачі можуть авторизовуватися у системі через Clerk, переглядати вакансії, керувати кандидатами, проводити AI-інтерв'ю та аналізувати результати найму.

Як альтернативу ручному налаштуванню середовища, можна використовувати Docker-контейнери, які дозволяють автоматизувати процес розгортання та забезпечують портативність конфігурацій. Для цього використовується `docker-compose.yml` файл, який налаштовує PostgreSQL та інші необхідні сервіси.

Отже, налаштування системи включає в себе підготовку середовища, встановлення необхідних залежностей, запуск Docker-контейнерів, налаштування бази даних і запуск всіх компонентів системи для обробки даних про найм та проведення AI-інтерв'ю.

3.2. Структура інтерфейсу

Інтерфейс розробленої платформи автоматизації найму спроектований з урахуванням принципів простоти та зручності використання, що забезпечує користувачам комфортну взаємодію як на десктопах, так і на мобільних пристроях. Всі елементи інтерфейсу взаємодіють між собою в єдиному веб-застосунку на основі React [2] та TypeScript [3], що забезпечує адаптивний дизайн [17].

Після переходу на сайт користувач потрапляє на сторінку авторизації, де може увійти до системи через Clerk (рис. 3.1). Для користувачів різних ролей - рекрутерів, менеджерів, адміністраторів та супер-адміністраторів - передбачено різний рівень доступу до функцій відповідно до матриці дозволів.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

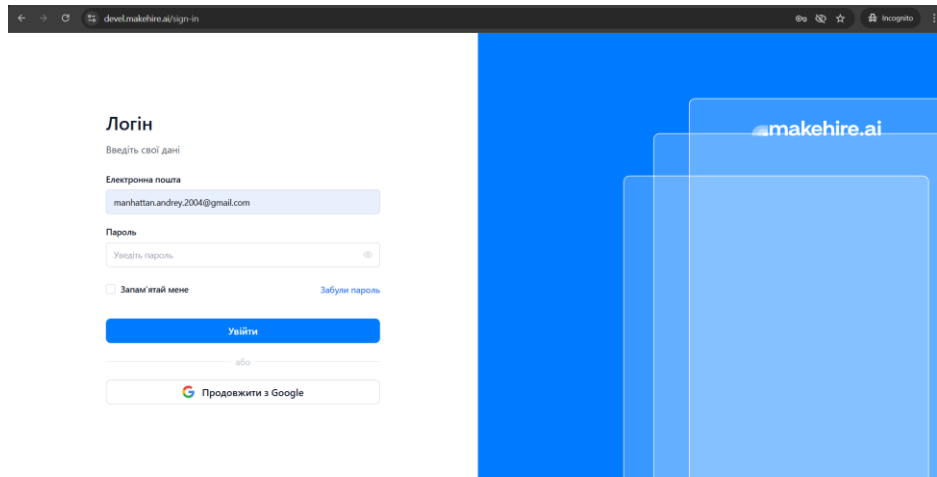


Рис. 3.1 - Сторінка входу

Після авторизації користувач потрапляє на головну сторінку з дашбордом, де розміщуються основні метрики найму, активні вакансії та статистика по кандидатах, що допомагає зразу орієнтуватися в поточному стані рекрутингових процесів. Ця сторінка також містить швидкі дії для створення нових вакансій та перегляду останніх кандидатів, що надає можливість користувачам швидко почати роботу з системою.

У розділі "Вакансії" користувач може переглядати всі доступні позиції, де всі вакансії розподілені за статусами, типами та іншими важливими характеристиками, що дозволяє детально ознайомитись з усіма активними пошуками. Каталог адаптований для зручності пошуку, даючи змогу користувачам швидко знайти потрібну вакансію завдяки системі фільтрації та сортування. На сторінці кожної вакансії представлені детальні характеристики, які включають опис, вимоги, етапи воронки та кандидатів, а також кнопки для редагування та публікації (рис. 3.2).

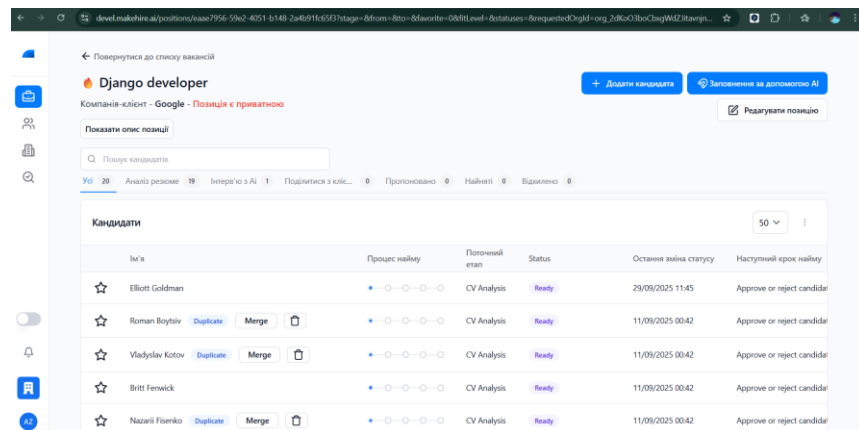


Рис. 3.2 - Сторінка вакансії

Після створення вакансії користувач може перейти до розділу "Кандидати", де відображаються всі претенденти, їх статуси, етапи воронки та результати AI-інтерв'ю. Користувач має можливість переглядати резюме, проводити інтерв'ю, додавати нотатки або переміщувати кандидатів між етапами (рис. 3.3). Крім того, система автоматично оновлює статуси кандидатів після кожної взаємодії, що дає користувачеві повний контроль над процесом відбору.

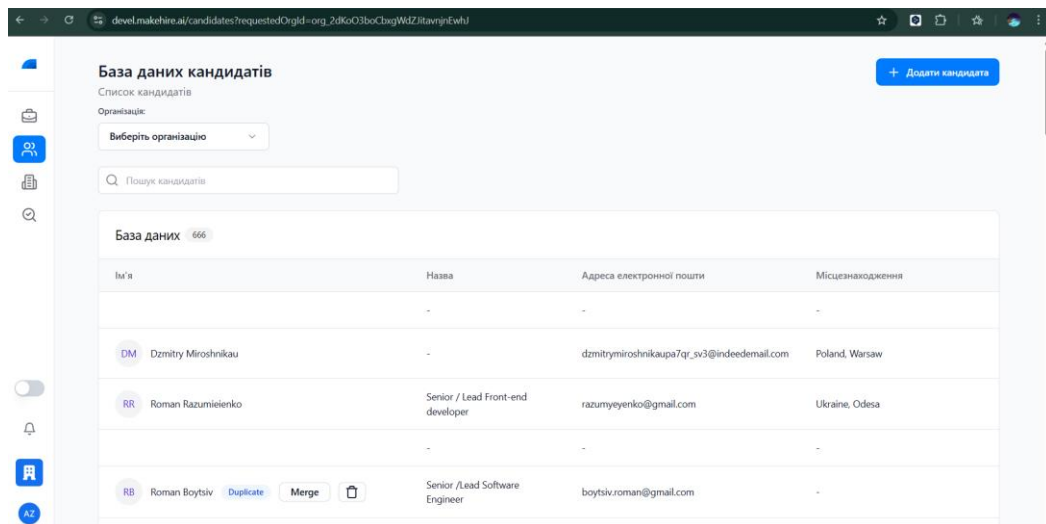


Рис. 3.3 - Сторінка кандидатів

Перейшовши до сторінки конкретного кандидата, користувач може переглядати детальну інформацію, включаючи резюме, результати AI-інтерв'ю, транскрипти дзвінків та аналітику поведінки під час інтерв'ю. Після аналізу, система дозволяє прийняти рішення щодо кандидата та перемістити його на наступний етап або відхилити (рис. 3.4).

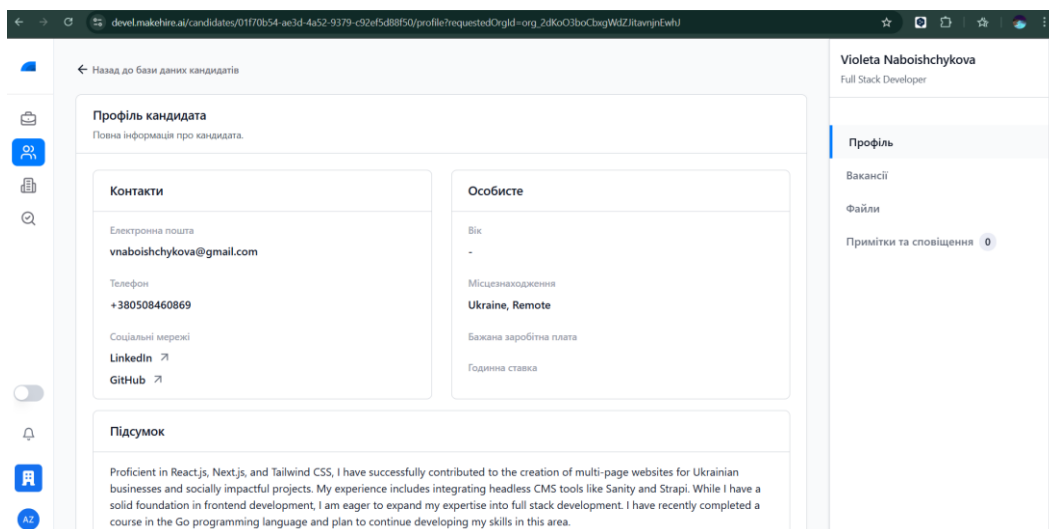


Рис. 3.4 - Профіль кандидата

Після того, як користувач переходить до проведення AI-інтерв'ю (рис. 3.5), він має можливість запустити автоматизоване інтерв'ю через Retell або Vari [9]. Система автоматично генерує питання на основі вакансії та проводить інтерв'ю, фіксуючи відповіді та поведінкові метрики кандидата.

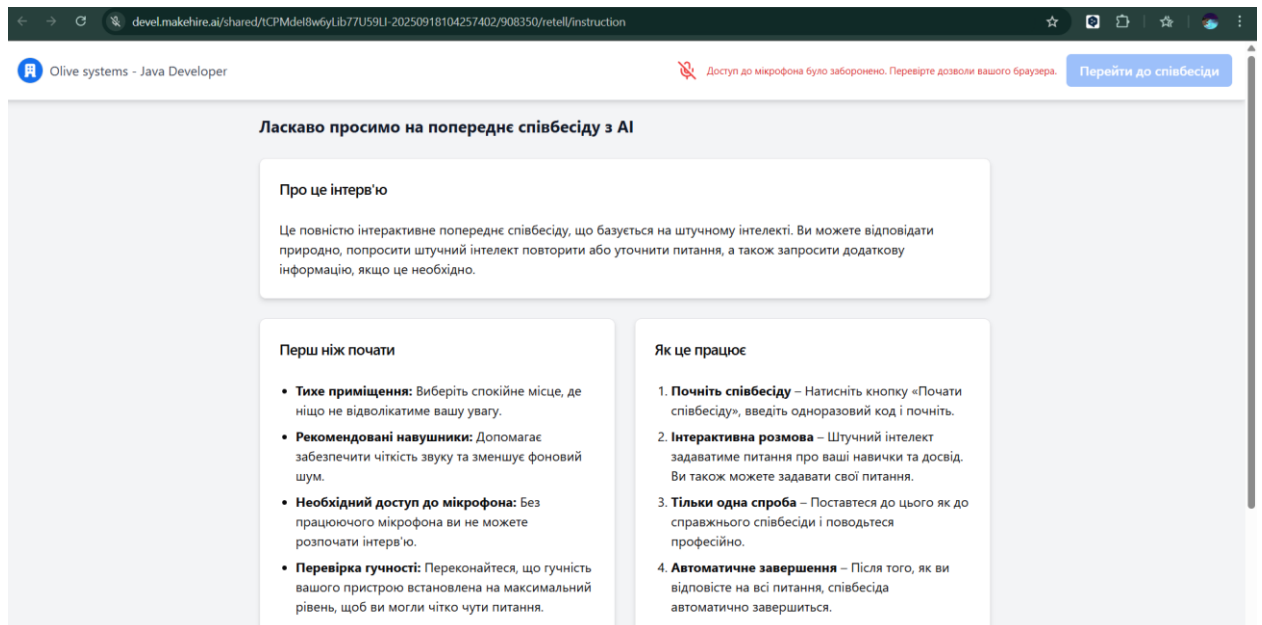


Рис. 3.5 - Сторінка AI-інтерв'ю

Після завершення інтерв'ю користувач отримує детальний звіт з транскриптом, оцінками та рекомендаціями. На сторінці показується аналіз відповідей, поведінкові метрики та можливість завантажити запис інтерв'ю (рис. 3.6).

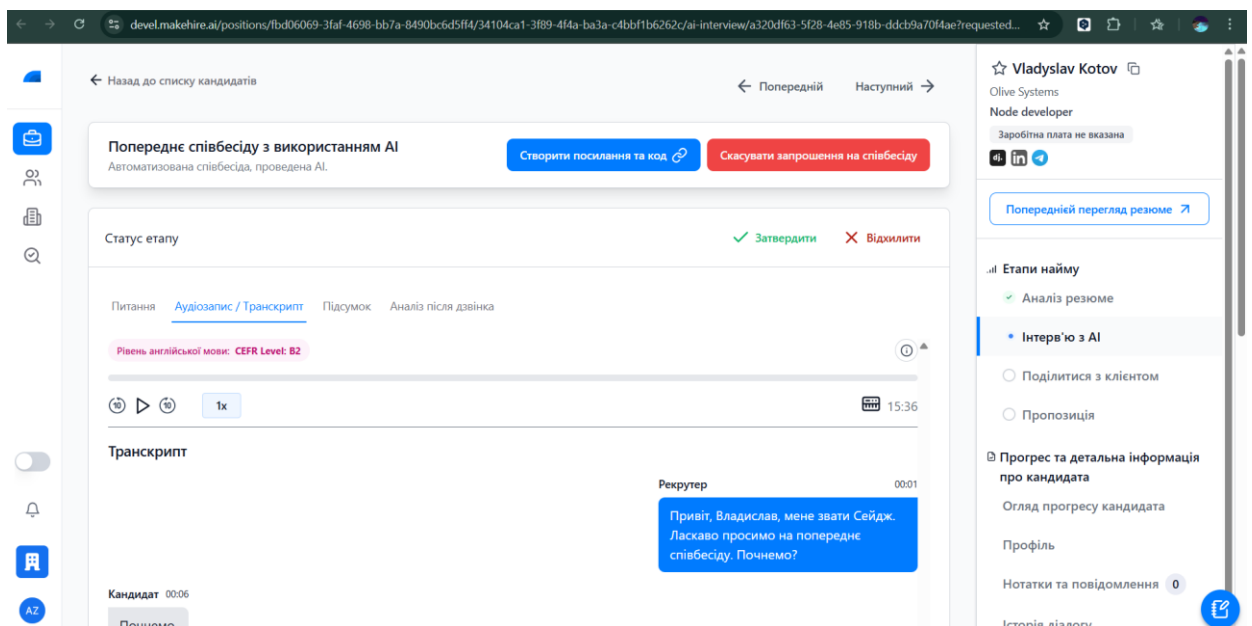


Рис. 3.6 - Результати AI-інтерв'ю

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

Інтерфейс системи підтримує адаптивну верстку, тому сторінки коректно відображаються на мобільних пристроях [19]. Всі елементи автоматично підлаштовуються під різні розміри екранів, що забезпечує зручність використання на будь-яких пристроях (рис. 3.7).

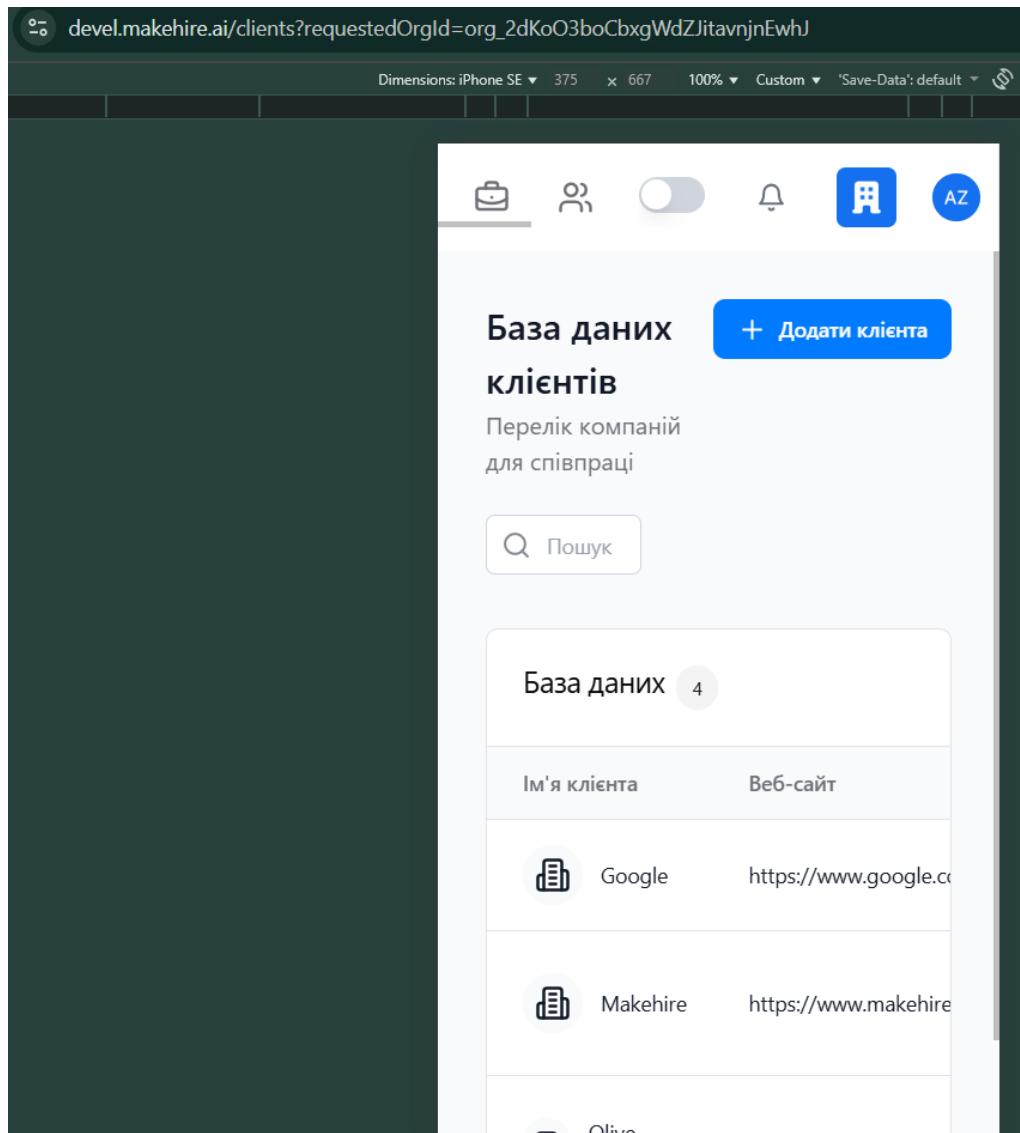


Рис. 3.7 - Адаптивний дизайн

Адмінам та менеджерам надано доступ до розділу "Клієнти", де вони можуть здійснювати управління компаніями-клієнтами, їх контактними особами та налаштуваннями. Сторінка для управління клієнтами дозволяє переглядати весь список клієнтів, додавати нових, редагувати існуючих або видаляти їх. Кожен клієнт має детальну інформацію про компанію, контактні дані та пов'язані вакансії.

На сторінці відображається таблиця зі списком всіх клієнтів, із можливістю фільтрації та пошуку за різними критеріями, такими як назва компанії, галузь або

статус. Також передбачена функціональність для редагування або додавання нового клієнта (рис. 3.8).

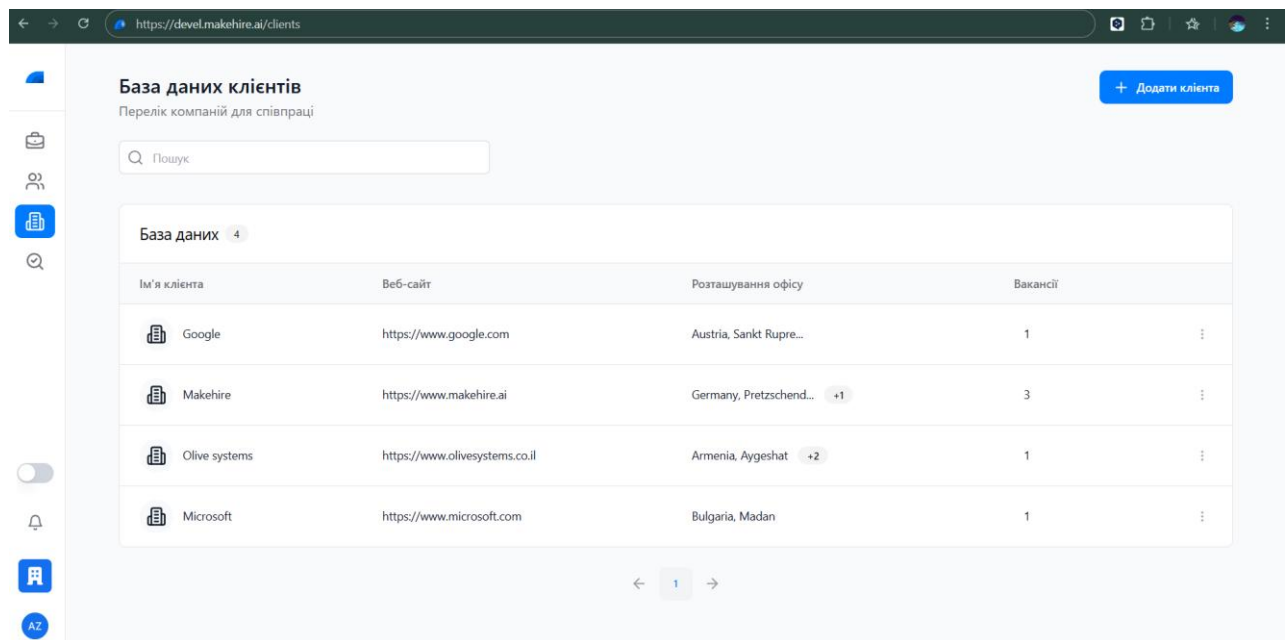


Рис. 3.8 - Сторінка управління клієнтами

Інтерфейс цієї сторінки розроблений таким чином, щоб адміністратори могли швидко знаходити потрібну інформацію, редагувати її та підтримувати актуальність даних у системі.

Таким чином, структура інтерфейсу [25] забезпечує простоту і зручність у використанні на всіх етапах взаємодії з користувачем, від авторизації до проведення AI-інтерв'ю та прийняття рішень щодо кандидатів.

3.3. Тестування роботи програмного комплексу

Тестування програмного комплексу платформи автоматизації найму є невід'ємним етапом життєвого циклу розробки ПЗ, що дозволяє переконатись у коректності реалізації функціоналу, стабільності роботи системи, а також зручності використання.

Основну увагу було приділено автоматизованому тестуванню через Jest [14] та Playwright [15], тестуванню функціоналу створення вакансій, управління кандидатами, перевірці прав доступу відповідно до ролей користувачів.

Також роботі AI-інтерв'ю та інтеграцій з зовнішніми сервісами (табл. 3.1).

У ході тестування було перевірено наступні сценарії:

Результати тестування основних сценаріїв

	1	2	3	4
Назва сценарію	Авторизація через Clerk	Створення вакансії з AI-асистуванням	Завантаження резюме та парсинг	Проведення AI-інтерв'ю
Передумови	Відсутній акаунт	Авторизований рекрутер	Створена вакансія	Кандидат з резюме
Кроки	Перейти на /sign-in, авторизуватися через Clerk	Перейти на /positions/new-ai, заповнити форму, згенерувати опис	Завантажити PDF резюме, перевірити парсинг	Запустити інтерв'ю через Retell/Vapi
Очікуваний результат	Користувача авторизовано та перенаправлено в дашборд	Вакансія створена з AI-описом	Кандидат створений, резюме розпарсено	Інтерв'ю проведено, транскрипт згенеровано
Фактичний результат	Успішно	Успішно	Успішно	Успішно
Коментар	-	AI-генерація працює стабільно	Парсинг PDF працює коректно	Транскрипт та метрики зберігаються

Під час тестування використовувались інструменти Jest для unit-тестування компонентів та сервісів, Playwright для end-to-end тестування [23] користувацьких сценаріїв, а також DevTools браузера для відстеження помилок у консолі та інспекції мережових запитів. Було виявлено кілька незначних проблем, таких як відсутність валідації в деяких полях форм та оптимізаційні помилки у запитах до бази даних, які були оперативно виправлені.

Також тестуванню піддавалися окремі компоненти системи, зокрема: система авторизації через Clerk, інтерфейс управління вакансіями (створення, редагування, публікація), логіка завантаження та парсингу резюме, інтеграція з AI-сервісами для генерації описів вакансій, проведення AI-інтерв'ю через Retell та Vapi, обробка

вебхуків від зовнішніх сервісів. Тестування кожної частини системи проводилося відповідно до принципів, зазначених у документації Jest та Playwright, що дозволяє оптимізувати процес тестування та забезпечити високу якість коду [22].

Результати тестування підтверджують, що система коректно працює на всіх етапах взаємодії користувача, від авторизації до проведення AI-інтерв'ю. Під час перевірки функціональності AI-інтеграцій система виявила високу стабільність при генерації описів вакансій та проведенні інтерв'ю через API, що також було підтверджено в тестах інтеграції з зовнішніми сервісами.

Важливим етапом також стало тестування PostgreSQL, що дозволило забезпечити правильне підключення до бази даних та безперервне збереження даних про вакансії, кандидатів та результати інтерв'ю, а також запитів до даних у реальному часі. Тестування бази даних також допомогло виявити деякі незначні оптимізаційні помилки у запитах, що були оперативно виправлені.

Тестування інтерфейсу було виконано із використанням Playwright, що дозволило автоматизувати процес перевірки користувацьких сценаріїв, що важливо для збереження стабільності при внесенні змін до основного коду та запуску нових версій застосунку.

Завдяки використанню React з TypeScript для відображення клієнтської частини, було легко здійснити тестування компонентів, що гарантує правильне відображення даних на всіх етапах взаємодії з користувачем. Це допомогло у виправленні помилок рендерингу, зокрема при створенні вакансій та управлінні кандидатами.

Усі критичні функції працюють відповідно до поставлених вимог, а результати тестування підтверджують готовність системи до експлуатації в реальних умовах. Тестування проводилося згідно з рекомендаціями з офіційної документації Jest та Playwright, а також загальними принципами тестування веб-додатків.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до третього розділу

У третьому розділі було здійснено розгортання та налаштування всіх компонентів розробленої платформи автоматизації найму. Зокрема, налаштовано серверну частину на базі Node.js з Express та tRPC, використано React з TypeScript для відображення клієнтської частини, а також налаштовано базу даних PostgreSQL для збереження всіх даних про вакансії, кандидатів, інтерв'ю та аналітику. Надано детальну інструкцію з інсталяції необхідних залежностей, конфігурації середовища через Docker та запуску системи.

Було розглянуто структуру інтерфейсу веб-застосунку, де для кожної з ролей (рекрутер, менеджер, адміністратор, супер-адміністратор) визначено доступний функціонал. Окремо представлені інтерфейси для авторизації через Clerk, управління вакансіями з AI-асистуванням, завантаження та парсингу резюме, проведення AI-інтерв'ю та аналізу результатів. Зазначено важливість адаптивності інтерфейсу, що гарантує коректне відображення сторінок як на великих екранах, так і на мобільних пристроях, що забезпечує зручність у взаємодії з системою.

Тестування системи включало перевірку основних функцій, таких як авторизація через Clerk, створення вакансій з AI-генерацією описів, завантаження та парсинг резюме, проведення AI-інтерв'ю через Retell та Vari, а також інтеграцію з зовнішніми сервісами через вебхуки. Було перевірено коректність роботи tRPC-запитів, генерацію транскриптів інтерв'ю та обробку аналітичних даних. Усі виявлені проблеми були оперативно усунуті, що підтверджує надійність та стабільність розробленої системи.

В результаті виконаних робіт система продемонструвала свою готовність до використання в реальному середовищі, забезпечуючи ефективне керування процесами найму, проведення AI-інтерв'ю та аналіз результатів відбору кандидатів. Тестування підтвердило, що всі функції працюють відповідно до вимог, що дозволяє рекомендувати систему до впровадження в реальних умовах.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було спроектовано та реалізовано платформу для автоматизації процесів найму за допомогою штучного інтелекту, що дозволяє ефективно керувати вакансіями, кандидатами, проводити AI-інтерв'ю та аналізувати результати відбору персоналу. Система реалізована на базі Node.js з використанням React та TypeScript для відображення клієнтської частини та tRPC для API.

Основну увагу було приділено розробці функціональних можливостей для кожної з ролей користувачів: рекрутера, менеджера, адміністратора та супер-адміністратора. Було спроектовано зручний інтерфейс для створення вакансій з AI-асистуванням, завантаження та парсингу резюме, проведення автоматизованих інтерв'ю та аналізу поведінкових метрик кандидатів. У результаті розробки була створена база даних PostgreSQL, що забезпечує ефективне зберігання інформації про вакансії, кандидатів, інтерв'ю та аналітику.

У ході роботи було проаналізовано та визначено основні вимоги до системи, зокрема до безпеки, продуктивності та зручності для користувача. Окремо було реалізовано систему авторизації через Clerk, що дозволяє користувачам безпечно входити в систему та отримувати доступ до функцій відповідно до їх ролей та дозволів організації.

Тестування системи показало, що усі основні функціональні можливості працюють стабільно та без помилок. Виявлені незначні недоліки були оперативно виправлені, що підтверджує готовність системи до використання в реальних умовах. Особливо успішно пройшли тести AI-інтеграцій для генерації описів вакансій та проведення автоматизованих інтерв'ю.

Таким чином, розроблена платформа автоматизації найму є функціональною, безпечною та зручною для користувачів. Система відповідає вимогам до ефективності, продуктивності та безпеки, інтегрує сучасні AI-технології для оптимізації процесів найму, що дозволяє рекомендувати її для впровадження в реальному бізнес-середовищі IT-компаній та рекрутингових агентств.

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js Documentation [Електронний ресурс] – Режим доступу: <https://nodejs.org/docs/>
2. React Documentation [Електронний ресурс] – Режим доступу: <https://react.dev/>
3. TypeScript Documentation [Електронний ресурс] – Режим доступу: <https://www.typescriptlang.org/docs/>
4. tRPC Documentation [Електронний ресурс] – Режим доступу: <https://trpc.io/docs>
5. PostgreSQL Documentation [Електронний ресурс] – Режим доступу: <https://www.postgresql.org/docs/>
6. Drizzle ORM Documentation [Електронний ресурс] – Режим доступу: <https://orm.drizzle.team/>
7. Clerk Authentication Documentation [Електронний ресурс] – Режим доступу: <https://clerk.com/docs>
8. Retell AI Documentation [Електронний ресурс] – Режим доступу: <https://docs.retellai.com/>
9. Vapi AI Documentation [Електронний ресурс] – Режим доступу: <https://docs.vapi.ai/>
10. OpenAI API Documentation [Електронний ресурс] – Режим доступу: <https://platform.openai.com/docs>
11. AWS SDK Documentation [Електронний ресурс] – Режим доступу: <https://docs.aws.amazon.com/sdk/>
12. Docker Documentation [Електронний ресурс] – Режим доступу: <https://docs.docker.com/>
13. Nx Documentation [Електронний ресурс] – Режим доступу: <https://nx.dev/>
14. Jest Testing Framework [Електронний ресурс] – Режим доступу: <https://jestjs.io/docs/>
15. Playwright Testing [Електронний ресурс] – Режим доступу: <https://playwright.dev/>

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

- 16.TanStack Router Documentation [Електронний ресурс] – Режим доступу:
<https://tanstack.com/router>
- 17.Tailwind CSS Documentation [Електронний ресурс] – Режим доступу:
<https://tailwindcss.com/docs>
- 18.Shadcn/ui Documentation [Електронний ресурс] – Режим доступу:
<https://ui.shadcn.com/>
- 19.Gamma, E., Helm, R., Johnson, R., & Vlissides, J.. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1995, 416 p. [Електронний ресурс] – Режим доступу:
[https://github.com/ansbilalgit/Books/blob/master/Erich%20Gamma%2C%20Richard%20Helm%2C%20Ralph%20Johnson%2C%20John%20M.%20Vlissides-Design%20Patterns_%20Elements%20of%20Reusable%20Object-Oriented%20Software%20%20-Addison-Wesley%20Professional%20\(1994\).pdf](https://github.com/ansbilalgit/Books/blob/master/Erich%20Gamma%2C%20Richard%20Helm%2C%20Ralph%20Johnson%2C%20John%20M.%20Vlissides-Design%20Patterns_%20Elements%20of%20Reusable%20Object-Oriented%20Software%20%20-Addison-Wesley%20Professional%20(1994).pdf)
- 20.Freeman, E., & Robson, E.. Head First Design Patterns. Sebastopol: O'Reilly Media, 2004, 688 p. [Електронний ресурс] – Режим доступу:
<https://github.com/ajitpal/BookBank/blob/master/%5BO%60Reilly.%20Head%20First%5D%20-%20Head%20First%20Design%20Patterns%20-%20%5BFreeman%5D.pdf>
- 21.Martin, R. C.. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River: Prentice Hall, 2017, 432 p. [Електронний ресурс] – Режим доступу: <https://raw.githubusercontent.com/sdcuik/Clean-Code-Collection-Books/master/Clean%20Architecture%20A%20Craftsman's%20Guide%20to%20Software%20Structure%20and%20Design.pdf>
- 22.Fowler, M.. Refactoring: Improving the Design of Existing Code. Boston: Addison-Wesley, 2018, 448 p. [Електронний ресурс] – Режим доступу:
<https://dl.ebooksworld.ir/motoman/Refactoring.Improving.the.Design.of.Existing.Code.2nd.edition.www.EBooksWorld.ir.pdf>

23. Software Testing Fundamentals [Електронний ресурс] – Режим доступу:
<https://softwaretestingfundamentals.com/>
24. OWASP Testing Guide v4 [Електронний ресурс] – Режим доступу:
<https://owasp.org/www-project-web-security-testing-guide/>
25. REST API Best Practices [Електронний ресурс] – Режим доступу:
<https://restfulapi.net/>
26. Web Performance Optimization [Електронний ресурс] – Режим доступу:
<https://web.dev/performance/>
27. Responsive Web Design Basics [Електронний ресурс] – Режим доступу:
<https://web.dev/responsive-web-design-basics/>
28. Role-Based Access Control (RBAC) in Web Applications [Електронний ресурс]
– Режим доступу: <https://www.digitalocean.com/community/tutorials>
29. AI in Recruitment: Trends and Applications [Електронний ресурс] – Режим
доступу: <https://www.hrtechnologist.com/articles/recruitment/ai-in-recruitment/>
30. Monorepo Architecture Best Practices [Електронний ресурс] – Режим доступу:
<https://monorepo.tools/>

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг роутеру webhooks.router.ts:

```

import { Router, json } from 'express';
import { logger } from '@makehire/shared-node/utils-logger';
import { Retell } from 'retell-sdk';
import {
  candidate,
  candidateToPosition,
  candidateToPositionStatus,
  cityModel,
  clientsModel,
  communicationHistory,
  communicationHistoryResponse,
  countryModel,
  cv,
  db,
  type InsertStageModel,
  interviewStageCandidate,
  interviewStageQuestion,
  interviewTranscriptMessage,
  position,
  retellInterview,
  stage,
  stageInterviewQuestion,
} from '@makehire/shared-node/data-access-drizzle';
import { and, asc, eq, isNull, or, type SQLWrapper } from 'drizzle-orm';
import {
  CallStatusValues,
  CommunicationTypeValue,
  getStageTypeStatus,
  MessageStatusValues,
  PositionStatusValue,
  StageTypeValues,
} from '@makehire/shared/utils-model';
import type {
  AiModel,
  Department,
  EmploymentType,
  LocationType,
  CompanyType,
  Domain,
  PositionLanguage,
  SalaryExpectations,
  DisconnectReason,
} from '@makehire/shared/utils-model';
import { apiEnv } from '@makehire/api/utils-env';
import { sendInterviewCompletedSlackMessage } from '@makehire/shared-node/utils-slack';
import { convertSecondsToTimeFormat } from '@makehire/api/feature-candidates';
import {
  analyzeInterview,
  postCallAnalysis,
} from '@makehire/shared-node/feature-analyze-interview';
import axios from 'axios';
import {

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    createCandidateForCv,
    presignUrl,
    createAiInterviewAgent,
  } from '@makehire/api/feature-positions';
import { eq as equal } from 'drizzle-orm/sql/expressions/conditions';
import multer from 'multer';
import { QuestionTypeValues } from '@makehire/shared/utls-types';
import { generateQuestionsGpt } from '@makehire/shared-node/feature-gen-questions';
import { insertCommHistoryRecord, updateCommHistoryRecord } from './utls/communication-history';

export const webhooksRouter = Router();

const upload = multer({
  storage: multer.memoryStorage(),
  limits: { fileSize: 15 * 1024 * 1024 },
});

webhooksRouter.post('/djinni/candidate', json(), async (req) => {
  try {
    const { cv_url, action, job_id, external_job_id, salary_min } = req.body as {
      cv_url: string;
      action: string;
      job_id: number | null;
      salary_min: number | undefined;
      external_job_id: string | null;
    };

    logger.info(`JOB_ID - ${job_id}`);
    logger.info(`EXTERNAL_JOB_ID - ${external_job_id}`);
    logger.info(`ACTION - ${action}`);
    logger.info(`CV_URL - ${cv_url}`);

    const djinniId = external_job_id || job_id;

    logger.info(`FINAL JOB_ID - ${djinniId}`);

    if (action === 'CANDIDATE_APPLY' && djinniId) {
      const [findPosition] = await db
        .select({
          positionId: position.id,
          orgId: position.organizationId,
          authorId: position.authorId,
          description: position.description,
          additionalInformation: position.additionalInformation,
          aiModel: position.aiModel,
        })
        .from(position)
        .where(eq(position.djinniId, Number(djinniId)));

      if (findPosition && findPosition.positionId && findPosition.orgId && findPosition.authorId) {
        const response = await axios.get(cv_url, { responseType: 'arraybuffer' });
        const body = Buffer.from(response.data);

        logger.info('Retrieved CV');

        const lastSlashIndex = cv_url.lastIndexOf('/');

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const filename = cv_url.endsWith('.pdf') ? cv_url.substring(lastSlashIndex + 1) : "";

const { candidateId, cvId } = await createCandidateForCv(
  findPosition.positionId,
  findPosition.orgId,
  findPosition.authorId,
  filename,
  undefined,
  undefined,
  undefined,
  salary_min
);

const { presignedUrl, objectPath } = await presignUrl({
  orgId: findPosition.orgId,
  fileKey: cvId + '.pdf',
  candidateId,
  positionId: findPosition.positionId,
});

await db.update(cv).set({ objectPath }).where(equal(cv.id, cvId));

const file = new File([body], filename, { type: 'application/pdf' });

const formData = new FormData();
Object.entries(presignedUrl.fields).forEach(([k, v]) => formData.append(k, v));
formData.append('Content-Type', file.type);
formData.append('x-amz-meta-positionId', findPosition.positionId);
formData.append('file', file);

await axios.post(presignedUrl.url, formData, {
  headers: {
    'Content-Type': 'multipart/form-data',
  },
  withCredentials: true,
});
}
} catch (e) {
  logger.error(e);
}
});

webhooksRouter.post('/aws/sns', json(), async (req, res) => {
  logger.info('AWS SNS webhook debug info');
  try {
    // For subscription confirmation
    if (req.body.Type === 'SubscriptionConfirmation') {
      logger.info('Received subscription confirmation request');
      const subscribeUrl = req.body.SubscribeURL;

      if (subscribeUrl) {
        logger.info(`Confirming subscription with URL: ${subscribeUrl}`);
        try {
          const response = await axios.get(subscribeUrl);

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (response.status === 200) {
        logger.info('SNS subscription confirmed successfully');
    } else {
        logger.error(`Failed to confirm subscription: ${response.status}`);
    }
} catch (error) {
    logger.error(
        `Error confirming subscription: ${
            error instanceof Error ? error.message : String(error)
        }`
    );
}

res.status(200).send('Ok');
return;
}

// Handle regular notification
if (req.body.Type === 'Notification') {
    try {
        const messageJson = JSON.parse(req.body.Message);

        if (messageJson.eventType && messageJson.mail && messageJson.mail.messageId) {
            await db
                .update(communicationHistory)
                .set({
                    status: messageJson.eventType,
                })
                .where(eq(communicationHistory.messageId, messageJson.mail.messageId));

            logger.info('Database - communicationHistory table record updated');
        } else {
            // Handle email replies
            const decodedEmail = messageJson;

            if (decodedEmail?.metadata?.context) {
                const substringToParse = 'In-Reply-To: <';
                const startIndex =
                    decodedEmail.metadata.context.indexOf(substringToParse) + substringToParse.length;

                if (startIndex > substringToParse.length) {
                    const endIndex = decodedEmail.metadata.context.indexOf('@', startIndex);
                    const responseToMessageId = decodedEmail.metadata.context.substring(
                        startIndex,
                        endIndex
                    );

                    logger.info('Response to message id');
                    logger.info(responseToMessageId);

                    const messageText = JSON.parse(decodedEmail.metadata.content || '{}');
                    logger.info('Message text');
                    logger.info(messageText);

                    await db.transaction(async (tx) => {

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const [originalMessage] = await tx
  .select()
  .from(communicationHistory)
  .where(eq(communicationHistory.messageId, responseToMessageId));

if (originalMessage?.id) {
  await tx.insert(communicationHistoryResponse).values({
    communicationHistoryId: originalMessage.id,
    text: 'Placeholder',
  });

  logger.info('Database - communicationResponse table record updated');
}
});
}
}
} catch (parseError) {
  logger.error(
    `Error parsing notification message: ${
      parseError instanceof Error ? parseError.message : String(parseError)
    }`
  );
}
} catch (error) {
  logger.error(`Webhook error: ${error instanceof Error ? error.message : String(error)}`);
}

logger.info('AWS SNS webhook end');
res.status(200).send('Ok');
});

webhooksRouter.post('/retell/call', json({ limit: '200mb' }), async (req, res) => {
  const { event, call } = req.body as {
    event: string;
    call: { call_id: string; transcript: string; disconnection_reason: DisconnectReason };
  };

  const disconnectionReason = call.disconnection_reason;

  const [prevRecord] = await db
    .select({
      positionId: communicationHistory.positionId,
      candidateId: communicationHistory.candidateId,
    })
    .from(communicationHistory)
    .where(eq(communicationHistory.messageId, call.call_id));

  const commonValues = {
    positionId: prevRecord.positionId,
    candidateId: prevRecord.candidateId,
    subject: 'Phone call',
    origin: CommunicationTypeValue.Phone,
    messageText: call.transcript,
    messageId: call.call_id,
  };

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

};

try {
    logger.info('retell/call webhook initiated');

    switch (event) {
        case 'call_started':
            logger.info('Call started:', call.call_id);

            if (prevRecord) {
                await updateCommHistoryRecord({
                    ...commonValues,
                    disconnectionReason,
                    status: CallStatusValues['Call in Progress'],
                });
            } else {
                await insertCommHistoryRecord({
                    ...commonValues,
                    disconnectionReason,
                    status: CallStatusValues['Call in Progress'],
                });
            }

            break;
        case 'call_ended':
            logger.info('Call ended:', call.call_id);

            if (prevRecord) {
                await updateCommHistoryRecord({
                    ...commonValues,
                    disconnectionReason,
                    status: CallStatusValues['Call succeeded'],
                });
            } else {
                await insertCommHistoryRecord({
                    ...commonValues,
                    disconnectionReason,
                    status: CallStatusValues['Call succeeded'],
                });
            }

            break;
        case 'call_analyzed':
            console.log('Call analyzed:', call.call_id);
            break;
    }

    res.status(204).send();
} catch (error) {
    if (prevRecord) {
        await updateCommHistoryRecord({
            ...commonValues,
            disconnectionReason,
            status: CallStatusValues['Server Error'],
        });
    } else {

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    await insertCommHistoryRecord({
      ...commonValues,
      disconnectionReason,
      status: CallStatusValues['Server Error'],
    });
  }

  logger.error(error);
}
});

webhooksRouter.post('/retell', json({ limit: '200mb' })), async (req, res) => {
  try {
    logger.info('/retell webhook call initiated');
    const { event, call } = req.body as {
      event: string;
      call: {
        call_id: string;
        disconnection_reason: DisconnectReason;
        call_status: string;
        call_analysis: { user_sentiment: string };
      };
    };

    const disconnectionReason = call.disconnection_reason;

    logger.info(`${event} - ${call.call_id}`);

    const [result] = await db
      .select({
        interviewStageId: interviewStageCandidate.id,
        organizationId: position.organizationId,
        positionTitle: position.title,
        candidateName: candidate.fullName,
        positionId: position.id,
        clientName: clientsModel.name,
        candidateId: candidate.id,
        stageId: stage.id,
      })
      .from(retellInterview)
      .leftJoin(candidateToPosition, eq(candidateToPosition.positionId, retellInterview.positionId))
      .leftJoin(position, eq(position.id, retellInterview.positionId))
      .leftJoin(clientsModel, eq(position.hiringCompanyId, clientsModel.id))
      .leftJoin(candidate, eq(candidate.id, retellInterview.candidateId))
      .leftJoin(stage, eq(stage.id, candidateToPosition.stageId))
      .leftJoin(interviewStageCandidate, eq(interviewStageCandidate.stageId, stage.id))
      .where(
        and(
          eq(candidateToPosition.candidateId, retellInterview.candidateId),
          eq(interviewStageCandidate.candidateId, retellInterview.candidateId),
          eq(stage.type, StageTypeValues['ai-interview']),
          eq(retellInterview.callId, call.call_id),
          isNull(retellInterview.deletedAt)
        )
      );
  }
};

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

logger.info('First request ended');

if (!result?.interviewStageId) {
  logger.info('Code 200 returning');
  return res.status(200).send();
}

const [findRes] = await db
  .select({
    isFinishedInterview: interviewStageCandidate.isFinishedInterview,
  })
  .from(interviewStageCandidate)
  .where(eq(interviewStageCandidate.id, result.interviewStageId));

logger.info('/findRes request ended');

if (findRes?.isFinishedInterview) {
  logger.info('Code 200');
  return res.status(200).send();
}

const client = new Retell({
  apiKey: apiEnv.RETELL_API_KEY,
});

logger.info('New retell client initialed');

const callResponse = await client.call.retrieve(call.call_id);

logger.info('call retrieved');

switch (event) {
  case 'call_analyzed': {
    logger.info('interview stage id' + result?.interviewStageId);

    await db
      .update(interviewStageCandidate)
      .set({
        isFinishedInterview: true,
      })
      .where(eq(interviewStageCandidate.id, result?.interviewStageId));

    logger.info('interview finished request ended');

    if (result?.interviewStageId && call.call_id) {
      const transcripts = callResponse?.transcript_object?.map((i, order) => ({
        order,
        interviewStageId: result.interviewStageId || '',
        messageType: i.role === 'agent' ? 'recruiter' : 'candidate',
        speaker: i.role === 'agent' ? 'A' : 'B',
        text: i.content,
        messageTime: convertSecondsToTimeFormat(i.words[0].start ?? 0),
      }));

      logger.info('transcripts count ' + transcripts?.length);
    }
  }
}

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

await db.transaction(async (tx) => {
  const { interviewStageId, organizationId } = result;

  logger.info('transaction started');

  if (transcripts) {
    logger.info('transaction present');
    await tx.insert(interviewTranscriptMessage).values(transcripts);

    const [statusResult] = await tx
      .select({
        positionDesc: position.description,
        positionKeyReq: position.additionalInformation,
        statusId: candidateToPosition.statusId,
      })
      .from(interviewStageCandidate)
      .innerJoin(stage, eq(stage.id, interviewStageCandidate.stageId))
      .innerJoin(
        candidateToPosition,
        eq(candidateToPosition.positionId, stage.positionId)
      )
      .innerJoin(position, eq(position.id, candidateToPosition.positionId))
      .where(
        and(
          eq(interviewStageCandidate.id, interviewStageId || ''),
          eq(candidateToPosition.candidateId, interviewStageCandidate.candidateId)
        )
      );

    logger.info('SELECT completed');

    await tx
      .update(candidateToPositionStatus)
      .set({
        stageStatusId: getStageTypeStatus('summarized'),
      })
      .where(eq(candidateToPositionStatus.id, statusResult.statusId));

    logger.info('Update completed');

    const dialog = transcripts.map((i) => i.speaker + ': ' + i.text);

    const resultAnalyze = await analyzeInterview({
      dialog,
      organizationId: organizationId ?? '',
      positionDesc: statusResult.positionKeyReq || statusResult.positionDesc,
      openAiApiKey: apiEnv.OPENAI_API_KEY,
    });

    logger.info('Interview analyzed');
    logger.info('Post call analyzing');

    const questionsResultStage = await tx
      .select({
        question: stageInterviewQuestion.question,
        questionType: stageInterviewQuestion.questionType,

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    })
    .from(stageInterviewQuestion)
    .where(
      and(
        eq(stageInterviewQuestion.stageId, result.stageId || ""),
        or(
          eq(stageInterviewQuestion.questionType, QuestionTypeValues['static']),
          eq(stageInterviewQuestion.questionType, QuestionTypeValues['stage'])
        )
      )
    )
    .orderBy(asc(stageInterviewQuestion.order), asc(stageInterviewQuestion.createdAt));

const questionsResults = await tx
  .select({
    question: interviewStageQuestion.question,
    questionType: interviewStageQuestion.questionType,
  })
  .from(interviewStageQuestion)
  .where(
    and(
      eq(interviewStageQuestion.interviewStageId, interviewStageId || ""),
      eq(interviewStageQuestion.questionType, QuestionTypeValues['personal'])
    )
  );

const questionsPersonalized = questionsResults.map((q) => q.question).filter(Boolean);

const questionsTechnical = questionsResultStage
  .filter((q) => q.questionType === QuestionTypeValues.stage)
  .map((q) => q.question)
  .filter(Boolean);

const questionsUserDefined = questionsResultStage
  .filter((q) => q.questionType === QuestionTypeValues.static)
  .map((q) => q.question)
  .filter(Boolean);

const postCallAnalyze = await postCallAnalysis({
  transcript: dialog,
  questionsPersonalized,
  questionsTechnical,
  questionsUserDefined,
  openAiApiKey: apiEnv.OPENAI_API_KEY,
});

logger.info('Post call analyzed');

await tx
  .update(interviewStageCandidate)
  .set({
    postCallAnalysis: postCallAnalyze.summary.replace(/[#*]/g, ""),
    aiSummary: resultAnalyze.recruiterSummary.replace(/[#*]/g, ""),
    aiSummaryForClient: resultAnalyze.clientSummary.replace(/[#*]/g, ""),
    aiScore: resultAnalyze.score,
    aiSkillScore: resultAnalyze.skills,
  });

```



```

    );

    logger.info('Status result select completed');

    await tx
      .update(candidateToPositionStatus)
      .set({
        stageStatusId: getStageTypeStatus('transcribing'),
      })
      .where(eq(candidateToPositionStatus.id, statusResult.statusId));

    logger.info('update completed');

    await tx
      .update(retellInterview)
      .set({
        isInterviewFinished: true,
        dateOfPassing: new Date(),
        disconnectReason: disconnectionReason,
        interviewAudioUrl: callResponse.recording_url,
      })
      .where(eq(retellInterview.callId, call.call_id));
  });
  logger.info('Update 2 completed');

  break;
}
}

return res.status(200).send();
} catch (e) {
  const { status, message } = req.body as { status: string; message: string };
  const callId = req.body?.call?.call_id;

  if (callId) {
    const [result] = await db
      .select({
        interviewStageId: interviewStageCandidate.id,
        organizationId: position.organizationId,
        positionTitle: position.title,
        candidateName: candidate.fullName,
        positionId: position.id,
        candidateId: candidate.id,
        stageId: stage.id,
      })
      .from(retellInterview)
      .leftJoin(
        candidateToPosition,
        eq(candidateToPosition.positionId, retellInterview.positionId)
      )
      .leftJoin(position, eq(position.id, retellInterview.positionId))
      .leftJoin(candidate, eq(candidate.id, retellInterview.candidateId))
      .leftJoin(stage, eq(stage.id, candidateToPosition.stageId))
      .leftJoin(interviewStageCandidate, eq(interviewStageCandidate.stageId, stage.id))
      .where(
        and(

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    eq(candidateToPosition.candidateId, retellInterview.candidateId),
    eq(interviewStageCandidate.candidateId, retellInterview.candidateId),
    eq(stage.type, StageTypeValues['ai-interview']),
    eq(retellInterview.callId, callId),
    isNull(retellInterview.deletedAt)
  )
);

await db.transaction(async (tx) => {
  const [statusResult] = await tx
    .select({
      positionDesc: position.description,
      positionKeyReq: position.additionalInformation,
      statusId: candidateToPosition.statusId,
    })
    .from(interviewStageCandidate)
    .innerJoin(stage, eq(stage.id, interviewStageCandidate.stageId))
    .innerJoin(candidateToPosition, eq(candidateToPosition.positionId, stage.positionId))
    .innerJoin(position, eq(position.id, candidateToPosition.positionId))
    .where(
      and(
        eq(interviewStageCandidate.id, result?.interviewStageId || ''),
        eq(candidateToPosition.candidateId, interviewStageCandidate.candidateId)
      )
    );

  logger.info('(error handler) Status result select completed');

  await tx
    .update(candidateToPositionStatus)
    .set({
      stageStatusId: getStageTypeStatus('error'),
    })
    .where(eq(candidateToPositionStatus.id, statusResult.statusId));

  logger.warn('call_id = ' + callId + 'error message: ' + message);
  logger.info('(error handler) update completed');

  await tx
    .update(retellInterview)
    .set({
      isInterviewFinished: true,
      dateOfPassing: new Date(),
      errorMessage: message,
    })
    .where(eq(retellInterview.callId, callId));
});

logger.error('Webhook retell error: ' + e);
logger.error(status);
logger.error(message);
return res.status(400).send();
}
});

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

webhooksRouter.post('/email', json(), async (req, res) => {
  const { type, data } = req.body as { type: string; data: { email_id: string } };

  logger.info('Email webhook - type value');
  logger.info(type);
  logger.info('Email ID');
  logger.info(data.email_id);

  if (
    ![
      'email.opened',
      'email.delivered',
      'email.delivery_delayed',
      'email.bounced',
      'email.failed',
      'email.complained',
      'email.unsubscribed',
    ].includes(type)
  ) {
    return res.status(200).send();
  }

  const statusByType = {
    'email.opened': MessageStatusValues.Open,
    'email.delivered': MessageStatusValues.Delivery,
    'email.delivery_delayed': MessageStatusValues.Delay,
    'email.bounced': MessageStatusValues.Bounced,
    'email.failed': MessageStatusValues.Failed,
    'email.complained': MessageStatusValues.Complaint,
    'email.clicked': MessageStatusValues.Clicked,
    'email.unsubscribed': MessageStatusValues.Unsubscribed,
  } as const;

  const targetStatus = statusByType[type as keyof typeof statusByType];

  const [existing] = await db
    .select({
      id: communicationHistory.id,
      status: communicationHistory.status,
    })
    .from(communicationHistory)
    .where(eq(communicationHistory.messageId, data.email_id))
    .limit(1);

  if (existing.status === targetStatus) {
    logger.info(`Status exist ${targetStatus} returning`);
    return res.status(200).send();
  }

  const commonWhereStatement: SQLWrapper[] = [eq(communicationHistory.messageId, data.email_id)];

  switch (type) {
    case 'email.opened': {
      logger.info('Email opened event');
      await db
        .update(communicationHistory)

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .set({
            status: MessageStatusValues.Open,
        })
        .where(and(...commonWhereStatement));
    return res.status(200).send();
}
case 'email.delivered': {
    logger.info('Email delivered event');
    await db
        .update(communicationHistory)
        .set({
            status: MessageStatusValues.Delivery,
        })
        .where(and(...commonWhereStatement));
    return res.status(200).send();
}
case 'email.delivery_delayed': {
    logger.info('Email delivery delayed event');
    await db
        .update(communicationHistory)
        .set({
            status: MessageStatusValues.Delay,
        })
        .where(and(...commonWhereStatement));
    return res.status(200).send();
}
case 'email.bounced': {
    logger.info('Email bounced');
    await db
        .update(communicationHistory)
        .set({
            status: MessageStatusValues.Bounced,
        })
        .where(and(...commonWhereStatement));
    return res.status(200).send();
}
case 'email.failed': {
    logger.info('Email failed to deliver');
    await db
        .update(communicationHistory)
        .set({
            status: MessageStatusValues.Failed,
        })
        .where(and(...commonWhereStatement));
    return res.status(200).send();
}
case 'email.complained': {
    logger.info('User complained to email');
    await db
        .update(communicationHistory)
        .set({
            status: MessageStatusValues.Complaint,
        })
        .where(and(...commonWhereStatement));
    return res.status(200).send();
}
}

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

case 'email.clicked': {
  logger.info('User complained to email');
  await db
    .update(communicationHistory)
    .set({
      status: MessageStatusValues.Clicked,
    })
    .where(and(...commonWhereStatement));
  return res.status(200).send();
}
case 'email.unsubscribed': {
  logger.info('Email delivery delayed event');
  await db
    .update(communicationHistory)
    .set({
      status: MessageStatusValues.Unsubscribed,
      isUnsubscribed: true,
    })
    .where(and(...commonWhereStatement));
  return res.status(200).send();
}
}

return res.status(200).send();
});

webhooksRouter.get('/positions/active', json(), async (req, res) => {
  const { orgId } = req.query as { orgId: string };

  if (!orgId) return res.status(404).send();

  const result = await db
    .select({
      id: position.id,
      title: position.title,
      description: position.description,
      additionalInfo: position.additionalInformation,
      createdAt: position.createdAt,
      cityName: cityModel.name,
      department: position.department,
      countryName: countryModel.name,
      locationType: position.locationType,
      employmentType: position.employmentType,
    })
    .from(position)
    .leftJoin(cityModel, eq(cityModel.id, position.cityId))
    .leftJoin(countryModel, eq(countryModel.id, position.countryId))
    .where(
      and(eq(position.organizationId, orgId), eq(position.statusId, PositionStatusValue.active))
    );

  return res.status(200).json(result);
});

webhooksRouter.post('/position/apply', upload.single('resume'), async (req, res) => {
  const { orgId, positionId, fullName, email, phone, linkedIn } = req.body as {

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

    orgId: string;
    positionId: string;
    fullName: string;
    email: string;
    phone: string;
    linkedIn?: string;
  };
  const file = req.file as Express.Multer.File;

  if (!orgId || !positionId || !file) return res.status(404).send();

  const [findPosition] = await db
    .select({
      authorId: position.authorId,
    })
    .from(position)
    .where(and(eq(position.id, positionId), eq(position.organizationId, orgId)));

  if (!findPosition) return res.status(404).send();

  const fileBuffer = file.buffer;
  const originalname = file.originalname;
  const filename = originalname.endsWith('.pdf') ? originalname : `${Date.now()}.pdf`;

  const { candidateId, cvId } = await createCandidateForCv(
    positionId,
    orgId,
    findPosition.authorId,
    filename
  );

  const { presignedUrl, objectPath } = await presignUrl({
    orgId,
    fileKey: `${cvId}.pdf`,
    candidateId,
    positionId,
    generateaiinterview: 'yes',
    fullName,
    email,
    phoneNumber: phone,
    linkedIn,
  });

  await db.update(cv).set({ objectPath }).where(equal(cv.id, cvId));

  const newFile = new File([fileBuffer], filename, { type: 'application/pdf' });

  const formData = new FormData();
  Object.entries(presignedUrl.fields).forEach(([key, value]) => {
    formData.append(key, value);
  });
  formData.append('Content-Type', newFile.type);
  formData.append('x-amz-meta-positionId', positionId);
  formData.append('x-amz-meta-generateaiinterview', 'yes');
  formData.append('x-amz-meta-fullname', fullName);
  formData.append('x-amz-meta-email', email);

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

```

formData.append('x-amz-meta-phone', phone);
if (linkedIn) formData.append('x-amz-meta-linkedin', linkedIn);
formData.append('file', newFile);

await axios.post(presignedUrl.url, formData, {
  headers: {
    'Content-Type': 'multipart/form-data',
  },
  withCredentials: true,
});

return res.status(200).json({ success: true });
});

webhooksRouter.post('/vapi', json({ limit: '500mb' }), async (req, res) => {
  const { message } = req.body as {
    message: {
      type: string;
      recordingUrl: string;
      call: { id: string };
      artifact: { messages: { role: string; message: string; secondsFromStart: number }[] };
    };
  };

  if (message.type !== 'end-of-call-report') return res.status(400).send();

  logger.info(JSON.stringify(message));

  const [result] = await db
    .select({
      interviewStageId: interviewStageCandidate.id,
      organizationId: position.organizationId,
    })
    .from(retellInterview)
    .leftJoin(candidateToPosition, eq(candidateToPosition.positionId, retellInterview.positionId))
    .leftJoin(position, eq(position.id, retellInterview.positionId))
    .leftJoin(stage, eq(stage.id, candidateToPosition.stageId))
    .leftJoin(interviewStageCandidate, eq(interviewStageCandidate.stageId, stage.id))
    .where(
      and(
        eq(candidateToPosition.candidateId, retellInterview.candidateId),
        eq(interviewStageCandidate.candidateId, retellInterview.candidateId),
        eq(stage.type, StageTypeValues['ai-interview']),
        eq(retellInterview.callId, message.call.id),
        isNull(retellInterview.deletedAt)
      )
    );

  if (!result?.interviewStageId) return res.status(400).send();

  const [findRes] = await db
    .select({
      isFinishedInterview: interviewStageCandidate.isFinishedInterview,
    })
    .from(interviewStageCandidate)
    .where(eq(interviewStageCandidate.id, result.interviewStageId));

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if (findRes?.isFinishedInterview) return res.status(200).send();

logger.info('interview stage id' + result?.interviewStageId);

await db
  .update(interviewStageCandidate)
  .set({
    isFinishedInterview: true,
  })
  .where(eq(interviewStageCandidate.id, result?.interviewStageId));

const transcripts = message.artifact.messages
  .filter((i) => i.role !== 'system')
  .map((i, order) => ({
    order,
    interviewStageId: result.interviewStageId || '',
    messageType: i.role === 'bot' ? 'recruiter' : 'candidate',
    speaker: i.role === 'bot' ? 'A' : 'B',
    text: i.message,
    messageTime: convertSecondsToTimeFormat(i.secondsFromStart),
  }));

logger.info('transcripts count ' + transcripts?.length);

await db.transaction(async (tx) => {
  const { interviewStageId, organizationId } = result;

  if (transcripts) {
    await tx.insert(interviewTranscriptMessage).values(transcripts);

    const [statusResult] = await tx
      .select({
        positionDesc: position.description,
        positionKeyReq: position.additionalInformation,
        statusId: candidateToPosition.statusId,
      })
      .from(interviewStageCandidate)
      .innerJoin(stage, eq(stage.id, interviewStageCandidate.stageId))
      .innerJoin(candidateToPosition, eq(candidateToPosition.positionId, stage.positionId))
      .innerJoin(position, eq(position.id, candidateToPosition.positionId))
      .where(
        and(
          eq(interviewStageCandidate.id, interviewStageId || ''),
          eq(candidateToPosition.candidateId, interviewStageCandidate.candidateId)
        )
      );

    await tx
      .update(candidateToPositionStatus)
      .set({
        stageStatusId: getStageTypeStatus('transcribing'),
      })
      .where(eq(candidateToPositionStatus.id, statusResult.statusId));

    const dialog = transcripts.map((i) => i.speaker + ': ' + i.text);
  }
});

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const resultAnalyze = await analyzeInterview({
  dialog,
  organizationId: organizationId ?? "",
  positionDesc: statusResult.positionKeyReq || statusResult.positionDesc,
  openAiApiKey: apiEnv.OPENAI_API_KEY,
});

await tx
  .update(interviewStageCandidate)
  .set({
    aiSummary: resultAnalyze.recruiterSummary.replace(/[#*]/g, ""),
    aiSummaryForClient: resultAnalyze.clientSummary.replace(/[#*]/g, ""),
    aiScore: resultAnalyze.score,
    aiSkillScore: resultAnalyze.skills,
  })
  .where(eq(interviewStageCandidate.id, interviewStageId || ""));

await tx
  .update(candidateToPositionStatus)
  .set({
    stageStatusId: getStageTypeStatus('summarized'),
  })
  .where(eq(candidateToPositionStatus.id, statusResult.statusId));

await tx
  .update(retellInterview)
  .set({
    isInterviewFinished: true,
    dateOfPassing: new Date(),
    interviewAudioUrl: message.recordingUrl,
  })
  .where(eq(retellInterview.callId, message.call.id));
}
});

return res.status(200);
});

webhooksRouter.post('/create/position', json(), async (req, res) => {
  const input = req.body as {
    title: string;
    description: string;
    authorId: string;
    orgId: string;
    hiringCompanyId: string;
    department: Department;
    additionalInformation?: string;
    locationType?: LocationType;
    remote?: boolean;
    aiModel?: AiModel;
    companyType?: CompanyType;
    employmentType?: EmploymentType;
    domain?: Domain;
    isTestsTask?: boolean;
    coreSkills?: string[];
  };

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

secondarySkills?: string[];
overallExperiencesFrom?: number;
overallExperiencesTo?: number;
education?: boolean;
languages?: PositionLanguage;
salaryExpectations?: SalaryExpectations;
};

```

```

const stages = [
  {
    id: 'cv_analysis',
    title: 'CV Analysis',
    type: 'cv_analysis',
    generateQuestionsOnPositionCreation: false,
    questionsToGenerate: [
      {
        type: 'stage',
        count: 0,
      },
      {
        type: 'personal',
        count: 0,
      },
      {
        type: 'default',
        count: 0,
      },
      {
        type: 'static',
        count: 0,
      },
    ],
  },
  {
    id: 'stage-1750348083383',
    title: 'Ai-stage',
    type: 'ai-interview',
    generateQuestionsOnPositionCreation: true,
    questionsToGenerate: [
      {
        type: 'stage',
        count: 5,
      },
      {
        type: 'personal',
        count: -10,
      },
      {
        type: 'default',
        count: 0,
      },
      {
        type: 'static',
        count: 0,
      },
    ],
  },
];

```

```

],
generateAnswers: false,
staticQuestions: [],
},
{
  id: 'share',
  title: 'Share to client',
  type: 'share',
  generateQuestionsOnPositionCreation: false,
  questionsToGenerate: [
    {
      type: 'stage',
      count: 0,
    },
    {
      type: 'personal',
      count: 0,
    },
    {
      type: 'default',
      count: 0,
    },
    {
      type: 'static',
      count: 0,
    },
  ],
},
generateAnswers: false,
},
{
  id: 'offered',
  title: 'Offered',
  type: 'offered',
  generateQuestionsOnPositionCreation: false,
  questionsToGenerate: [
    {
      type: 'stage',
      count: 0,
    },
    {
      type: 'personal',
      count: 0,
    },
    {
      type: 'default',
      count: 0,
    },
    {
      type: 'static',
      count: 0,
    },
  ],
},
generateAnswers: false,
},
{
  id: 'hired',

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		

```

title: 'Hired',
type: 'hired',
generateQuestionsOnPositionCreation: false,
questionsToGenerate: [
  {
    type: 'stage',
    count: 0,
  },
  {
    type: 'personal',
    count: 0,
  },
  {
    type: 'default',
    count: 0,
  },
  {
    type: 'static',
    count: 0,
  },
],
generateAnswers: false,
},
];

const createdPos = await db.transaction(async (tx) => {
  const [createdPosition] = await tx
    .insert(position)
    .values({
      title: input.title,
      description: input.description.replace(/[#*]/g, ""),
      locationType: input?.locationType || 'remote',
      remote: input?.remote || false,
      isCreateDjinni: false,
      authorId: input.authorId,
      organizationId: input.orgId,
      mainAdditionalInformation: input?.additionalInformation,
      hiringCompanyId: input.hiringCompanyId || undefined,
      aiModel: input?.aiModel || 'gpt-4o',
      employmentType: input?.employmentType || 'full-time',
      domain: input?.domain,
      department: input.department,
      companyType: input?.companyType,
      isTestsTask: input?.isTestsTask ?? false,
      coreSkills: input?.coreSkills,
      secondarySkills: input?.secondarySkills,
      overallExperiences: input?.overallExperiencesFrom,
      overallExperiences2: input?.overallExperiencesTo,
      education: input?.education ?? false,
      languages: input?.languages,
      salaryExpectations: input?.salaryExpectations,
      type: 'aiDescGen',
    })
    .returning();

  const interviewStagesIndexes: number[] = [];

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const newStages = await Promise.all(
  stages.map(async (s, order) => {
    const isAiInterview = s.type === StageTypeValues['ai-interview'];
    const isInterview = s.type === StageTypeValues.interview;

    const agentId = isAiInterview
      ? await createAiInterviewAgent({
          positionTitle: createdPosition.title,
          organizationName: input.orgId,
          order,
        })
      : undefined;

    if (isInterview || isAiInterview) {
      interviewStagesIndexes.push(order);
    }

    return {
      ...s,
      id: undefined,
      agentId,
      positionId: createdPosition.id,
      order,
      questionsToGenerate: s.questionsToGenerate,
      mandatory: false,
      generateQuestionsOnPositionCreation: !s.generateQuestionsOnPositionCreation,
    };
  })
);

const insertedStages = await tx
  .insert(stage)
  .values(newStages as InsertStageModel[])
  .returning();

if (interviewStagesIndexes.length > 0) {
  for (const index of interviewStagesIndexes) {
    const interviewStage = insertedStages[index];
    const inputStage = stages[index];

    // if (inputStage.staticQuestions && inputStage.staticQuestions.length > 0) {
    //   await Promise.all(
    //     inputStage.staticQuestions.map(async (staticQuestion) => {
    //       return tx.insert(stageInterviewQuestion).values({
    //         stageId: insertedStages[index].id,
    //         question: staticQuestion.question,
    //         answer: staticQuestion.answer,
    //         technology: 'Static questions',
    //         questionType: QuestionTypeValues.static,
    //       });
    //     })
    //   );
    // }

    const isStageQuestionGenerationOn = inputStage.questionsToGenerate.findIndex(

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    (question) => question.type === QuestionTypeValues.stage
  );

  if (
    isStageQuestionGenerationOn === -1 &&
    inputStage.questionsToGenerate[isStageQuestionGenerationOn]['count'] <= 0
  )
    continue;

  const questions = await generateQuestionsGpt(
    {
      openAiApiKey: apiEnv.OPENAI_API_KEY,
      questionNumber: inputStage.questionsToGenerate[isStageQuestionGenerationOn]['count'],
      generateAnswers: inputStage.generateAnswers,
      skillList: createdPosition.coreSkills ?? [],
      experienceYears: createdPosition.overallExperiences ?? 0,
      jobTitle: createdPosition.title,
      questionsToAvoid: [],
    },
    createdPosition.aiModel
  );

  const questionValues = questions.flatMap((tech) => {
    return tech.questions.map((q, questionIndex) => ({
      ...q,
      stageId: interviewStage.id,
      questionType: QuestionTypeValues.stage,
      technology: tech.technology,
      order: questionIndex,
    }));
  });

  if (questionValues.length > 0)
    await tx.insert(stageInterviewQuestion).values(questionValues);
}

return createdPosition;
});

return res.status(200).json({ success: true, data: createdPos });
});

```

ЛІСТИНГ сервісу cv-restart-analysis.ts:

```

import {
  aiConfigTable,
  candidate,
  candidateToPosition,
  candidateToPositionStatus,
  cv,
  db,
  position,
  positionCandidateAnalysis,
} from '@makehire/shared-node/data-access-drizzle';

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import type { CvAnalysisRestartArgs } from '../schema/cv-analysis-restart.schema';
import type { SQLWrapper } from 'drizzle-orm';
import { ne, and, eq } from 'drizzle-orm';
import { getCvUploadBucketName } from '@makehire/shared-node/utlis-infra';
import { apiEnv } from '@makehire/api/utlis-env';
import type { AuthObject } from '@clerk/clerk-sdk-node';
import { s3Client } from '@makehire/api/utlis-aws';
import type { GetObjectCommandInput } from '@aws-sdk/client-s3';
import { DeleteObjectCommand, GetObjectCommand } from '@aws-sdk/client-s3';
import { pdfParseLegacy } from '@makehire/shared-node/feature-pdf-parser';
import { logger } from '@makehire/shared-node/utlis-logger';
import { PDFDocument } from 'pdf-lib';
import { load } from '@pspdfkit/nodejs';
import { createWorker } from 'tesseract.js';
import type { AiModel, Department, PositionLanguage } from '@makehire/shared/utlis-model';
import {
  CandidateStatusValue,
  getStageStatusValueIndex,
  getStageTypeStatus,
  StageStatusValue,
} from '@makehire/shared/utlis-model';
import {
  analyzeCvClaude,
  analyzeCvGpt,
  parseCvFields,
} from '@makehire/shared-node/feature-analyze-cv';

export async function requestCvAnalysisRestart(auth: AuthObject, args: CvAnalysisRestartArgs) {
  const { positionId, candidateId } = args;
  // we got cv and target candidate to position
  const [targetPosition] = await db
    .select()
    .from(candidateToPosition)
    .innerJoin(cv, eq(cv.candidateId, candidateId))
    .where(
      and(
        eq(candidateToPosition.candidateId, candidateId),
        eq(candidateToPosition.positionId, positionId)
      )
    );

  // updating status
  await db
    .update(candidateToPositionStatus)
    .set({
      stageStatusId: getStageTypeStatus(CandidateStatusValue.cv_analysis),
    })
    .where(eq(candidateToPositionStatus.id, targetPosition.candidate_to_position.statusId));

  await db
    .update(candidate)
    .set({
      createdAt: new Date(),
    })
    .where(eq(candidate.id, candidateId));

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

try {
  const candAnalysisId = tartgetPosition.candidate_to_position.analysisId ?? "";

  await db
    .update(positionCandidateAnalysis)
    .set({
      createdAt: new Date(),
    })
    .where(eq(positionCandidateAnalysis.id, candAnalysisId));
} catch ( ) {}

if (!tartgetPosition || !tartgetPosition.cv.objectPath) throw new Error('CV not found');
// get the cv
const key = tartgetPosition.cv.objectPath;
const bucket = getCvUploadBucketName(apiEnv.TARGET_ENV);

const commandInput: GetObjectCommandInput = { Bucket: bucket, Key: key };
const command = new GetObjectCommand(commandInput);
const resultCv = await s3Client.send(command);

if (!resultCv.Body) throw new Error('Body is undefined');
const body = await resultCv.Body.transformToByteArray();
const { Metadata } = resultCv;

// parse cv
let fullText = "";
try {
  logger.info('Try default parse');
  const docs = await pdfParseLegacy(Buffer.from(body));
  fullText = docs.pageContent;
  logger.info('Finish default parse');
} catch (error) {
  logger.info('Parse by OCR');
  const pdfDoc = await PDFDocument.load(Buffer.from(body));

  pdfDoc.getPages().forEach((page) => {
    const { width, height } = page.getSize();
    const newHeight = height + 40;

    page.setSize(width, newHeight);
  });

  const resizedPdfBuffer = await pdfDoc.save();
  const instance = await load({ document: resizedPdfBuffer });
  const pages = instance.getDocumentInfo().pages;

  const worker = await createWorker('eng');

  for (let pageIndex = 0; pageIndex < pages.length; pageIndex++) {
    const result = await instance.renderPage(pageIndex, {
      width: pages[pageIndex].width,
      height: pages[pageIndex].height,
    });

    const ret = await worker.recognize(Buffer.from(result));
    fullText += ret.data.text + '\n';
  }
}

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    await worker.terminate();
    await instance.close();
    fullText = fullText.replace('For Evaluation Purposes Only (remove with TRIAL key)', '');
    logger.info('Finish parse by OCR');
}

logger.info(fullText); // remove after debug
logger.info('Parsed PDF');

try {
    const query = db
        .select()
        .from(cv)
        .leftJoin(candidate, eq(candidate.id, cv.candidateId))
        .$dynamic();

    if (Metadata) {
        const metaObj = Metadata as Record<string, string>;

        if ('positionid' in metaObj) {
            const posId = metaObj['positionid'];
            query
                .leftJoin(
                    candidateToPosition,
                    and(
                        eq(candidateToPosition.candidateId, cv.candidateId),
                        eq(candidateToPosition.positionId, posId)
                    )
                )
                .leftJoin(position, eq(position.id, candidateToPosition.positionId));
        }
    }

    const [cvObject] = await query.where(eq(cv.objectPath, key));

    //TODO:remove later
    logger.info(cvObject);

    if (!cvObject || !cvObject.candidate?.id) {
        throw new Error('CV or candidate information not found');
    }

    if (!fullText || fullText.length <= 0) {
        logger.info('Parsed text is empty');
        if (Metadata) {
            const metaObj = Metadata as Record<string, string>;

            if ('positionid' in metaObj) {
                const posId = metaObj['positionid'];

                const [candidateToPos] = await db
                    .select({ statusId: candidateToPosition.statusId })
                    .from(candidateToPosition)
                    .where(

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        and(
            eq(candidateToPosition.candidateId, cvObject.candidate.id),
            eq(candidateToPosition.positionId, posId)
        )
    );

    await db
        .update(candidateToPositionStatus)
        .set({
            stageStatusId: getStageTypeStatus(CandidateStatusValue.error),
        })
        .where(eq(candidateToPositionStatus.id, candidateToPos.statusId));
    }

    logger.error('Could not process CV properly');
    throw new Error('CV could not be parced');
}
}

const parseCvPromise = parseCvFields({
    cvText: fullText,
    openAiApiKey: apiEnv.OPENAI_API_KEY,
});

logger.info('Starting cv analysis');
let analyzeCvPromise;
let customModelParams = false;

//TODO:rework it
if ('position' in cvObject) {
    const positionInfo = cvObject.position as {
        description: string;
        title: string;
        languages: PositionLanguage;
        department: Department;
        education: boolean;
        overallExperiences: number;
        overallExperiences2: number;
        coreSkills: string[];
        secondarySkills: string[];
        additionalInformation: string;
        aiModel: AiModel;
    };

    const [modelParams] = await db
        .select()
        .from(aiConfigTable)
        .where(eq(aiConfigTable.userId, cvObject.candidate.createdBy))
        .limit(1);

    if (modelParams) {
        customModelParams = true;
        logger.info('AI params override found', modelParams);
    }

    if (positionInfo.aiModel === 'claude-3-5-sonnet-latest') {

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

]);

if (parseResult.status === 'fulfilled' && analyzeResult.status === 'fulfilled') {
  const { value: parseResultData } = parseResult;
  const { value: analyzeResultData } = analyzeResult;

  logger.info('Finished CV analysis');

  await db
    .update(candidateToPositionStatus)
    .set({ stageStatusId: getStageStatusValueIndex(StageStatusValue.ready) })
    .where(eq(candidateToPositionStatus.id, tartgetPosition.candidate_to_position.statusId));

  const lastName = parseResultData.lastName || '';
  const fullName = (parseResultData.firstName.trim() + ' ' + lastName).trim();
  const email = parseResultData.email;

  if (email) {
    logger.info('Check for duplicate');

    const duplicateQuery = db
      .select({ id: candidate.id, email: cv.email })
      .from(cv)
      .leftJoin(candidate, eq(candidate.id, cv.candidateId))
      .$dynamic();

    const positionFilters: SQLWrapper[] = [];
    if ('candidate_to_position' in cvObject) {
      logger.info('Duplicate candidate from position');
      const ids = cvObject?.candidate_to_position as {
        candidateId: string;
        positionId: string;
      };

      if (ids.candidateId && ids.positionId) {
        duplicateQuery.leftJoin(
          candidateToPosition,
          eq(candidateToPosition.candidateId, cv.candidateId)
        );

        positionFilters.push(eq(candidateToPosition.positionId, ids.positionId));
      }

      const [findCvByEmail] = await duplicateQuery
        .where(
          and(
            ...positionFilters,
            eq(cv.email, email),
            eq(candidate.organizationId, cvObject?.candidate?.organizationId),
            ne(candidate.id, cvObject.candidate.id)
          )
        )
        .orderBy(cv.createdAt)
        .limit(1);

      if (findCvByEmail?.email && findCvByEmail?.id) {

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    logger.info('Duplicate cv');

    await db
      .update(candidate)
      .set({
        duplicateId: findCvByEmail.id,
      })
      .where(eq(candidate.id, cvObject.candidate.id));
  }
}

await db
  .update(candidate)
  .set({
    fullName,
  })
  .where(eq(candidate.id, cvObject.candidate.id));

if ('candidate_to_position' in cvObject) {
  logger.info(`Analyze params used: ${JSON.stringify(analyzeResultData.params)}`);

  const [analysisId] = await db
    .insert(positionCandidateAnalysis)
    .values({
      summary: analyzeResultData.result.summary.replace(/[#*]/g, " "),
      stringScore: analyzeResultData.result.score.toString(),
      modelParams: { ...analyzeResultData.params, hasOverrides: customModelParams },
    })
    .returning({ id: positionCandidateAnalysis.id });

  const ids = cvObject.candidate_to_position as { candidateId: string; positionId: string };

  const [stageStatusId] = await db
    .update(candidateToPosition)
    .set({
      analysisId: analysisId.id,
    })
    .where(
      and(
        eq(candidateToPosition.candidateId, ids.candidateId),
        eq(candidateToPosition.positionId, ids.positionId)
      )
    )
    .returning({ id: candidateToPosition.statusId });

  await db
    .update(candidateToPositionStatus)
    .set({
      stageStatusId: getStageTypeStatus('ready'),
    })
    .where(eq(candidateToPositionStatus.id, stageStatusId.id));
}

await db
  .update(cv)

```

					ІПЗп.КР.Б – 121 – 25 - ПЗ	Арк.
						97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

.set({
  objectPath: key,
  summary: parseResultData.summary,
  email: parseResultData.email,
  phone: parseResultData.phone,
  role: parseResultData.role,
  age: parseResultData.age,
  education: parseResultData.education,
  languages: parseResultData.languages,
  skills: parseResultData.skills,
  address: parseResultData.address,
  links: parseResultData.links,
  experience: parseResultData.experience,
})
.where(eq(cv.objectPath, key));
} else {
  if (parseResult.status === 'rejected') {
    logger.error(`Parse error: ${parseResult.reason}`);
  }
  if (analyzeResult.status === 'rejected') {
    logger.error(`Analyze error: ${analyzeResult.reason}`);
  }
  throw new Error('Error processing CV data with AI');
}

logger.info('Persisted CV analysis');
} catch (error) {
  logger.error('Failed to analyze ' + (error as Error).message, { error });
}

if (Metadata) {
  const metaObj = Metadata as Record<string, string>;

  if ('positionid' in metaObj) {
    const posId = metaObj['positionid'];

    const [result] = await db
      .select()
      .from(cv)
      .leftJoin(candidate, eq(candidate.id, cv.candidateId))
      .leftJoin(
        candidateToPosition,
        and(
          eq(candidateToPosition.candidateId, cv.candidateId),
          eq(candidateToPosition.positionId, posId)
        )
      )
      .leftJoin(position, eq(position.id, candidateToPosition.positionId))
      .where(eq(cv.objectPath, key));

    if (result?.candidate_to_position) {
      await db
        .update(candidateToPositionStatus)
        .set({
          stageStatusId: getStageTypeStatus(CandidateStatusValue.error),
        })
        .where(

```

```

        and(
            eq(candidateToPosition.candidateId, result.candidate?.id ?? ''),
            eq(candidateToPosition.positionId, posId)
        )
    );
}
}
}

console.log(error);
await s3Client.send(new DeleteObjectCommand(commandInput));
logger.info('Clean-up after error: Deleted CV from S3');
await db.delete(cv).where(eq(cv.objectPath, key));
logger.info('Clean-up after error: Deleted CV from DB');
throw new Error('Unexpected error: ' + (error as Error).message);
}
}

```

Лістинг утиліта filter.ts:

```

import { CandidateToPositionStatusSchema } from '@makehire/shared/utls-model';
import { z as zod } from 'zod';
import { fitLevelKeys } from './fit-levels';

export const CandidateToPositionFilterTypeSchema = zod.object({
    statuses: zod.set(CandidateToPositionStatusSchema).optional(),
    dateRange: zod
        .object({
            from: zod.date().optional(),
            to: zod.date().optional(),
        })
        .optional(),
    favorite: zod.boolean().optional(),
    fitLevel: zod.array(fitLevelKeys).optional(),
});

export const CandidateToPositionFilterSchema = zod.object({
    statuses: zod.array(CandidateToPositionStatusSchema).optional(),
    dateRange: zod
        .object({
            from: zod.string().optional(),
            to: zod.string().optional(),
        })
        .optional(),
    favorite: zod.boolean().optional(),
    fitLevel: zod.array(fitLevelKeys).optional(),
});

export type CandidateToPositionFilterType = zod.infer<typeof CandidateToPositionFilterTypeSchema>;

```