Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра мережевих технологій

Магістерська робота

Освітній ступінь: магістр

На тему: «Побудова багаторівневого веб-застосунку на хмарній платформі для платформи пошуку роботи»
Текстова частина до дипломної роботи за спеціальністю «Комп'ютерні науки» 122

Виконав: студент 2 року навчання
Спеціальності
122 Комп'ютерні науки
Гурін Валентин Ігорович
Керівник: Черкасов Д.І.

Рецензент
Магістерська робота захищена
з оцінкою_____
Секретар ЕК С.А. Мелещенко

«___» _____2025

Київ 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережевих технологій

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
доктор техн. наук, декан ФІ
_____…
(підпис)
“____”_____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу
студенту Гуріну Валентину
факультету інформатики 2 курсу магістерської програми

ТЕМА: Побудова багаторівневого веб-застосунку на хмарній платформі для платформи пошуку роботи.

Вихідні дані:
Зміст ТЧ до курсової роботи:

Вступ
Анотація
1. Огляд існуючих рішень і технологій для побудови застосунку.
2. Програмна реалізація
3. Автоматизоване розгортання, CI/CD інтеграція
Висновки
Список використаної літератури та електронних ресурсів
Додатки

Дата видачі “_____” _____202_ р.
Керівник _____ Завдання отримано _____

Тема: Побудова багаторівневого веб-застосунку на хмарній платформі для платформи пошуку роботи.

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи		
2.	Вивчення предметної області		
3.	Огляд засобів реалізації		
4.	Вивчення технологій		
5.	Написання першої частини курсової роботи		
6.	Написання другої частини курсової роботи		
7.	Написання третьої частини курсової роботи		
8.	Написання висновків курсової роботи		
9.	Перегляд змісту роботи з керівником		
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника		
11.	Створення презентації		
12.	Захист роботи		

Студент Гурін В.І.

Керівник Черкасов Д.І.

“____” _____

Анотація

Робота присвячена розробці та проєктуванню багаторівневого веб-застосунку для платформи пошуку роботи. Основна увага зосереджена на створенні надійної та масштабованої веб-інфраструктури, розгорнутої на хмарній платформі Amazon Web Services (AWS). Це включає розробку фронтенд-частини на Next.js, бекенд API на Nest.js з використанням GraphQL, та системи управління даними на базі PostgreSQL, з розглядом можливостей масштабування через архітектури Data Warehouse, такі як Amazon Redshift.

У роботі детально аналізується проблематика сучасних платформ пошуку роботи та обґрунтовується вибір технологічного стеку для ключових компонентів системи: фронтенду, API, бази даних, інфраструктури розгортання, безпеки та комунікацій. Додатково розглядається інтеграція модуля машинного навчання як допоміжного інтелектуального шару, що використовує AWS SageMaker для задач бінарної класифікації, AWS Data Wrangler для підготовки даних та відповідні механізми розгортання й MLOps. Запропонована архітектура спрямована на створення високопродуктивного, користувацько-орієнтованого та технологічно гнучкого рішення у сфері онлайн-рекрутингу.

Вступ

Ринок праці за останнє десятиріччя остаточно перемістився у цифрову площину: розміщення вакансій, подання резюме та комунікація між кандидатами й роботодавцями відбуваються переважно через спеціалізовані онлайн-системи. За прогнозами, у 2025 р. кількість активних вакансій, доступних у мережі одночасно, перевищить 70 млн, тоді як аудиторія шукачів налічуватиме понад 300 млн користувачів. Такий масштаб породжує інформаційне перевантаження: кандидати витрачають надмірний час на пошук релевантних пропозицій, а роботодавці — на обробку великого обсягу нерелевантних відгуків, що знижує ефективність підбору персоналу та збільшує операційні витрати.

Існуючі сервіси здебільшого покладаються на просте текстове зіставлення й базову фільтрацію, що не враховує поведінкову історію користувача, динаміку професійного розвитку та приховані семантичні зв’язки між навичками й вимогами. У результаті коефіцієнт конверсії «перегляд → відгук» залишається на рівні 2–3 %, а час очікування першої релевантної відповіді часто перевищує тиждень. Водночас масштаб даних та пікові навантаження висувають жорсткі вимоги до продуктивності й горизонтального масштабування систем. Таким чином, нагальним є створення багаторівневих веб-застосунків нового покоління, які поєднують швидкий інтерфейс, гнучкий програмний API та потужну систему управління даними, доповнену інтелектуальними алгоритмами персоналізації.

Об’єкт дослідження: процес розробки, розгортання та супроводу багаторівневого веб-застосунку для платформи пошуку роботи, що інтегрує інтелектуальні рекомендаційні системи на основі машинного навчання.

Мета дослідження: розробити та описати архітектуру та програмну реалізацію багаторівневого веб-застосунку для платформи пошуку роботи, що використовує моделі машинного навчання для надання релевантних рекомендацій, забезпечує автоматизоване розгортання та управління життєвим циклом моделей за допомогою MLOps підходів.

Робота складається з трьох розділів:

- У **першому розділі** проводиться детальний аналіз проблематики онлайн-пошуку роботи, огляд існуючих рішень та їхніх обмежень. Також здійснюється огляд та обґрунтування вибору технологій, сервісів та інструментів, що використовуються для розробки всіх компонентів інтелектуальної платформи пошуку роботи
- У **другому розділі** детально описується процес проєктування та програмної реалізації багаторівневого веб-застосунку і впровадження MLOps практик для розгортання моделей машинного навчання.
- У **третьому розділі** розглядаються питання автоматизованого розгортання застосунку.

Розділ 1. Огляд існуючих рішень і технологій для побудови застосунку

1.1. Огляд ринку

Наведено глибокий аналітичний огляд шести провідних світових платформ-роботошукачів (LinkedIn, Indeed, Jooble, Glassdoor, Talent.com, ZipRecruiter) крізь призму масштабу, якості даних, соціальної взаємодії, зрілості ML-персоналізації та бізнес-моделі. Особливу увагу приділено «больовим точкам», що залишаються невирішеними: дублікати вакансій, обмежена explainable-рекомендація, відсутність цілісного графового скорингу та слабка інтеграція з хмарними ML-серверлес-сервісами. На підставі порівняльної таблиці сформульовано системні прогалини та конкретні напрями покращення, які ляжуть в основу розроблюваної платформи. Центральною інновацією пропонується модуль рекомендацій на AWS SageMaker (Studio + Data Wrangler + Canvas) із графовими ембеддингами, behavioural re-ranking та explainability-шаром.

Платформа	**Масштаб / аудиторія**	**Сильні сторони**	**Виявлені недоліки**	**Науково-практичні напрями удосконалення**

LinkedIn	1,1 млрд профілів (січень 2025) LinkedIn	Потужний соціальний граф, бренд-сторінки, групи, контент-стрічка	Висока конкуренція, дорогі рекрутингові фільтри, ML-рекомендації «чорний ящик»	1) Графові ембеддинги + GNN-скоринг «кандидат ↔ вакансія». 2) Explainable AI (SHAP-пояснення) для рекрутерів.
Indeed	≈ 600 млн унікальних відвідувачів /міс (2024) Investopedia	Найбільший індекс вакансій, швидкий відгук, PPC-монетизація	«Шумні» дані, дублікати, відсутня соціальна взаємодія	1) Probabilistic record-linkage для дедуплікації. 2) Behavioural re-ranking – RL-policy, що перетасовує видачу за історією кліків.
Jooble	90 млн відвідувань /міс, 67	Глобальне покриття, e-mail alerts,	Немає резюме-бази, брендингу, інтерактиву	1) Профілі компаній з knowledge-graph. 2) Matching API

	країн Similarweb	мультимовність		для ATS з двобічним ML-скорингом.
Glassdoor	55 млн унікальних відвідувачів /міс (Q4 2024) Similarweb	Відгуки й зарплатна аналітика, employer branding	Дані само-звітувань містять упередження, вузький ринок вакансій	1) Bayesian debiasing зарплатних оцінок. 2) Комбінування crowd-ratings з вакансійним графом для рекомендацій «компанія + кар'єрний шлях».
Talent.com	30 млн вакансій / 78 країн (2024) Similarweb	Aggressive ML-R&D; фреймворк DL-рекомендацій (+8,6 % CTR) Similarweb	Фокус на агрегованих даних без розширеного профілю користувача	1) Fine-tuning трансформерів на enriched-profile features. 2) Explainable counter-factual вакансій.

ZipRecruiter	≈ 36 млн резюме, 12 млн вакансій (2024) Similarweb	One-click apply, ML-“Phil” recommender	Приватний граф, обмежений міжнародний рельс	1) Federated learning для крос-платформних фіч. 2) Trust-layer з амбрелами навичок ESCO/ONET.

Таблиця 1.

Висновок. Ринок демонструє незадоволену потребу у сервісі, який поєднує масштаб агрегатора з прозорою ML-рекомендацією й елементами соціального графу, при цьому без надмірного paywall-бар’єру.

1.2. Компоненти користувацького інтерфейсу та інструментарій розробки

Ефективний та привабливий користувацький інтерфейс (UI) є ключовим фактором успіху будь-якого веб-застосунку, особливо для платформи пошуку роботи, де користувачі проводять значний час, взаємодіючи з великими обсягами інформації. Цей підрозділ присвячений аналізу інструментів та технологій, що використовуються для створення клієнтської частини системи.

1.2.1 Засоби для збірки та оптимізації клієнтського коду

Процес збірки клієнтської частини веб-застосунку відіграє вирішальну роль у забезпеченні його продуктивності, надійності та зручності підтримки. Основні завдання, які вирішують інструменти збірки, включають:

- Управління залежностями та бандлінг: Сучасні веб-додатки складаються з численних модулів та бібліотек. Збирачі об'єднують ці файли (JavaScript, CSS, зображення) в оптимізовані пакети (бандли), зменшуючи кількість HTTP-запитів та прискорюючи завантаження сторінок.
- Транспіляція та поліфілізація: Для забезпечення сумісності з різними браузерами, код, написаний з використанням новітніх стандартів JavaScript (ECMAScript), перетворюється (транспілюється) у старіші версії. Поліфіли додають функціональність, відсутню у старих браузерах.
- Мініфікація та оптимізація коду: Видалення зайвих символів (пробілів, коментарів), скорочення імен змінних, усунення "мертвого" коду (tree shaking) дозволяє значно зменшити розмір фінальних файлів, що позитивно впливає на швидкість завантаження.
- Робота з різними типами активів: Обробка та оптимізація CSS (препроцесори як SASS/LESS, PostCSS), зображень (стиснення, конвертація у формати типу WebP), шрифтів та інших ресурсів.
- Гаряче оновлення модулів (HMR - Hot Module Replacement): Важлива функція для розробки, що дозволяє бачити зміни в коді миттєво у браузері без повного перезавантаження сторінки.

На ринку існує декілька популярних інструментів для збірки фронтенд-проєктів. Розглянемо два з них: Webpack та Vite.

- Webpack: Протягом тривалого часу Webpack був стандартом де-факто для бандлінгу JavaScript-застосунків. Це надзвичайно потужний та гнучкий інструмент з величезною екосистемою плагінів та завантажувачів (loaders), що дозволяє налаштувати процес збірки під будь-які потреби. Однак, його конфігурація може бути складною, а час збірки, особливо для великих проєктів, іноді стає значним.
- Vite: Більш сучасний інструмент, що швидко набирає популярність завдяки своєму інноваційному підходу. Vite використовує нативні ES-модулі (ESM) браузера під час розробки, що забезпечує миттєвий запуск сервера розробки та надзвичайно швидке гаряче оновлення модулів (HMR). Для продакшн-збірки Vite використовує Rollup, який відомий своєю ефективністю у створенні оптимізованих бандлів. Vite пропонує простішу конфігурацію "з коробки" для популярних фреймворків.

Порівняльна таблиця Webpack та Vite:

Характеристика	**Webpack**	**Vite**
Основний підхід	Бандлінг всіх модулів перед запуском	Використання нативних ES-модулів під час розробки

Швидкість HMR	Відносно повільна для великих проєктів	Дуже швидка, практично миттєва
Час запуску сервера	Може бути значним для великих проєктів	Майже миттєвий
Конфігурація	Складна, потребує детального налаштування	Проста, багато налаштувань "з коробки"
Продакшн збірка	Власний механізм	Використовує Rollup під капотом
Спільнота/Екосистема	Дуже велика, зріла	Швидко зростаюча, активна

Таблиця 2.

Нижче наведений графік кількості скачувань із npm:

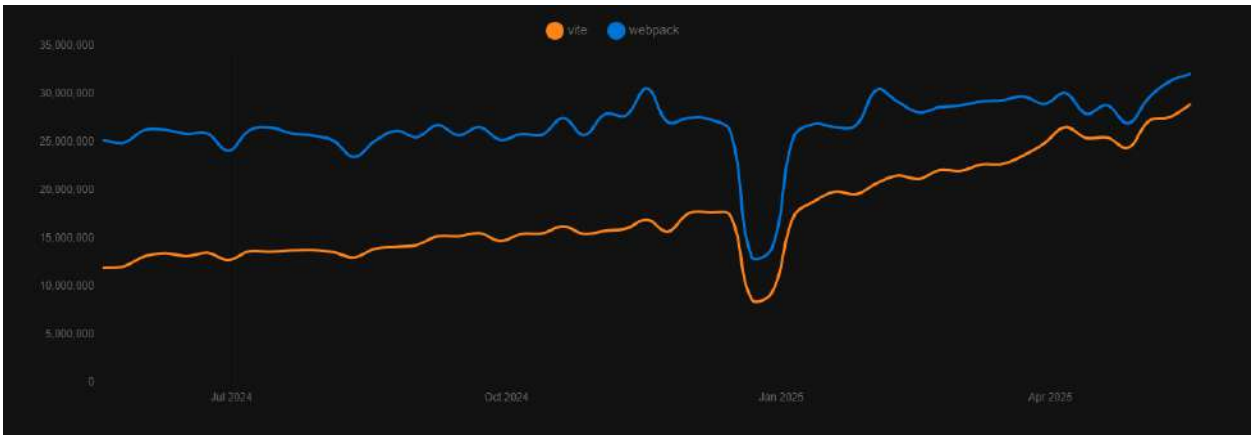

Рис. 1. Графік порівняння кількість завантажень з npm Webpack та Vite за останній рік. Джерело: npmtrends.com.

Зважаючи на переваги у швидкості розробки, простоті конфігурації та використанні сучасних підходів, для реалізації клієнтської частини даного веб-застосунку Next.js (який під капотом може використовувати Webpack з оптимізаціями або експериментальні збирачі типу Turbopack) є обґрунтованим вибором, оскільки він надає вже оптимізований процес збірки. Якщо б розробка велася на "чистому" React, Vite міг би бути привабливішою альтернативою для швидкого старту.

1.2.2 Технології для створення інтерактивних користувацьких інтерфейсів

Для розробки динамічних та інтерактивних користувацьких інтерфейсів сучасні веб-розробники мають у своєму розпорядженні декілька потужних JavaScript-бібліотек та фреймворків. Серед найпопулярніших можна виділити React.js, Vue.js та Angular.

- Angular (від Google): Це комплексний MVC/MVVM фреймворк, що надає повний набір інструментів для створення великих корпоративних

застосунків. Він має чітку структуру, використовує TypeScript "з коробки" та пропонує потужні можливості, такі як вбудована система маршрутизації, управління формами, HTTP-клієнт та інструменти для тестування. Однак, Angular має відносно високий поріг входження та може бути надлишковим для деяких проєктів [5].

- Vue.js: Прогресивний JavaScript-фреймворк, відомий своєю простотою інтеграції, гнучкістю та низьким порогом входження. Vue.js дозволяє поступово впроваджувати свої компоненти у вже існуючі проєкти. Він має детальну документацію та активну спільноту. Vue.js може бути чудовим вибором для проєктів різного масштабу, від малих до досить великих [6].
- React.js (від Meta): Це JavaScript-бібліотека для створення користувацьких інтерфейсів, що базується на компонентному підході. React фокусується на представленні (view layer) і може легко інтегруватися з іншими бібліотеками для управління станом, маршрутизації тощо. Основні переваги React [7]:
    - Декларативний підхід: Розробники описують, як має виглядати інтерфейс для певного стану, а React бере на себе оновлення DOM.
    - Компонентна архітектура: Дозволяє створювати незалежні компоненти, що перевикористовуються, спрощуючи розробку та підтримку.
    - Віртуальний DOM (Virtual DOM): Забезпечує високу продуктивність шляхом мінімізації прямих маніпуляцій з реальним DOM.

- Велика екосистема: Існує величезна кількість готових бібліотек, інструментів та навчальних матеріалів.
- Next.js: Популярний фреймворк на базі React, що додає можливості серверного рендерингу, статичної генерації, оптимізації та багато іншого, значно спрощуючи створення сучасних веб-застосунків.

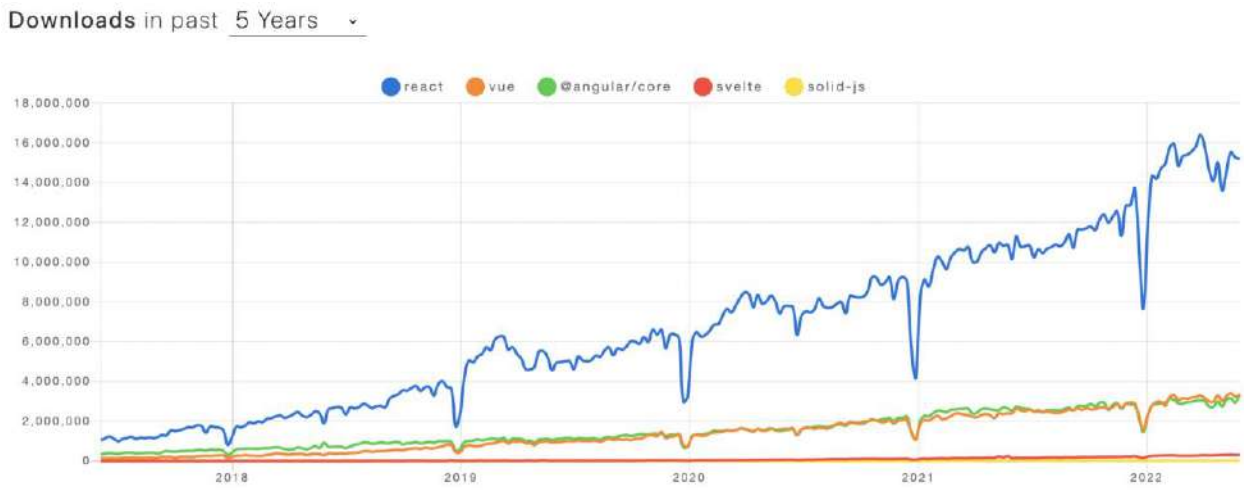

Рис. 2. Діаграма, що показує частку ринку за даними State of JS для React.js, Angular та Vue.js.

Для даного проєкту, який передбачає створення багатофункціональної платформи пошуку роботи, обрано React.js у поєднанні з фреймворком Next.js. Next.js надає потужну основу для створення швидких, SEO-дружніх та масштабованих веб-застосунків на React, включаючи оптимізовану збірку, маршрутизацію та різні стратегії рендерингу.

1.2.4 Бібліотеки готових UI-компонентів та стилізація

Використання готових бібліотек UI-компонентів та сучасних підходів до стилізації може значно прискорити розробку та забезпечити консистентний вигляд застосунку.

Бібліотеки компонентів (MUI, Ant Design, Chakra UI):

- **Material-UI (MUI):** Надає великий набір React-компонентів, що реалізують принципи Material Design від Google. Має розширені компоненти (MUI X) для таблиць даних, вибору дати/часу тощо. Добре документована та широко використовується.
- **Ant Design:** Пропонує величезну кількість високоякісних компонентів для корпоративних застосунків, з фокусом на продуктивність та зручність використання.
- **Chakra UI:** Надає набір доступних (accessibility-first), модульних та композиційних React-компонентів, які легко кастомізувати за допомогою пропсів стилів (style props).

Утилітарні CSS-фреймворки (Tailwind CSS):

- **Tailwind CSS:** На відміну від традиційних CSS-фреймворків (Bootstrap, Foundation), які надають готові компоненти, Tailwind CSS пропонує набір низькорівневих утилітарних класів, які можна комбінувати безпосередньо в HTML-розмітці для створення будь-якого дизайну. Це забезпечує високу гнучкість, запобігає "роздуванню" CSS та спрощує підтримку стилів [8].

Бібліотеки нестилізованих компонентів (Headless UI):

- **Headless UI:** Розроблена командою Tailwind CSS, ця бібліотека надає повністю нестилізовані, але функціональні та доступні компоненти (меню, списки, діалоги, вкладки тощо). Розробник отримує всю логіку та доступність, а зовнішній вигляд може повністю контролювати за допомогою Tailwind CSS або будь-якого іншого підходу до стилізації.

Для проєкту обрано комбінацію **Tailwind CSS** для стилізації та **Headless UI** для складних інтерактивних компонентів. Це дозволяє досягти унікального дизайну, зберігаючи при цьому високу швидкість розробки, доступність та легкість кастомізації.

1.3 Компоненти серверної частини та управління даними

Серверна частина (бекенд) є мозком веб-застосунку, відповідаючи за обробку бізнес-логіки, взаємодію з базами даних, автентифікацію, авторизацію та надання API для клієнтських застосунків.

1.3.1 Середовище виконання для серверної логіки (Node.js)

Node.js – це асинхронне, подієво-орієнтоване середовище виконання JavaScript, побудоване на рушії V8 від Google Chrome. Його використання для бекенду надає низку переваг [1]:

- **Висока продуктивність для I/O-bound операцій:** Завдяки неблокуючій моделі введення-виведення, Node.js ефективно обробляє велику кількість одночасних з'єднань, що є типовим для веб-серверів.
- **Використання JavaScript/TypeScript на всьому стеку:** Дозволяє розробникам використовувати одну мову (або її надбудову

TypeScript) як для фронтенду, так і для бекенду, що спрощує розробку та обмін кодом/знаннями в команді.

- **Велика екосистема (npm/pnpm):** Доступ до величезної кількості готових модулів та бібліотек для вирішення різноманітних завдань.
- **Активна спільнота:** Широка підтримка та велика кількість навчальних ресурсів.

Для даного проєкту обрано **Node.js** (рекомендована LTS версія) як середовище виконання для серверної частини.

1.3.2 Фреймворки для побудови API (Nest.js)

Хоча Node.js надає базові можливості для створення HTTP-серверів, для побудови складних та масштабованих API зазвичай використовуються фреймворки [11].

- **Express.js:** Мінімалістичний та гнучкий веб-фреймворк, що став стандартом де-факто для багатьох Node.js проєктів. Він надає базовий набір інструментів для маршрутизації та обробки запитів, залишаючи багато рішень (структура проєкту, ORM, валідація) на розсуд розробника.
- **Nest.js:** Прогресивний Node.js фреймворк, побудований на TypeScript [2], що використовує багато концепцій з Angular (модулі, сервіси, декоратори, dependency injection). Nest.js пропонує чітку архітектуру "з коробки", що сприяє створенню добре структурованих, тестованих та масштабованих застосунків.
  - **Модульність:** Дозволяє розбивати застосунок на логічні функціональні блоки.

- **Підтримка TypeScript:** Повна інтеграція зі статичною типізацією.
- **Dependency Injection (DI):** Полегшує управління залежностями та тестування.
- **Інтеграція з різними технологіями:** Легко інтегрується з GraphQL (через Apollo Server), WebSocket, мікросервісними транспортами, ORM тощо.

Порівняння Express.js та Nest.js:

Аспект	**Express.js**	**Nest.js**
Рівень абстракції	Низький, мінімалістичний	Високий, надає структуру та багато вбудованих інструментів
TypeScript	Можлива інтеграція, але не "з коробки"	Основна мова, повна підтримка
Структура проєкту	Гнучка, визначається розробником	Чітко визначена, модульна архітектура
DI	Відсутня "з коробки"	Вбудована система

Поріг входження	Низький	Середній (особливо для тих, хто знайомий з Angular/Java Spring)
Масштабованість	Залежить від архітектури, створеної розробником	Сприяє побудові масштабованих систем

Таблиця 4.

Для проєкту обрано **Nest.js** через його переваги у структуруванні коду, вбудовану підтримку TypeScript, систему DI та легкість інтеграції з GraphQL, що є важливим для створення гнучкого API для платформи пошуку роботи.

1.3.3 Бібліотеки для валідації та трансформації даних

Валідація вхідних даних (наприклад, з HTTP-запитів) та трансформація даних між різними форматами є важливими аспектами забезпечення безпеки та коректності роботи API.

- **class-validator:** Бібліотека, що дозволяє використовувати декоратори для визначення правил валідації для класів TypeScript. Інтегрується з Nest.js для автоматичної валідації DTO (Data Transfer Objects).
- **class-transformer:** Бібліотека для трансформації простих JavaScript об'єктів у екземпляри класів (і навпаки), а також для

контролю за тим, які властивості об'єкта будуть серіалізовані/десеріалізовані. Часто використовується разом з class-validator.

Ці бібліотеки (class-validator та class-transformer) будуть використовуватися в Nest.js застосунку для забезпечення надійної валідації вхідних даних та їх коректної трансформації.

1.3.4 Системи управління базами даних (СУБД) та архітектура зберігання даних

Вибір адекватної системи управління базами даних та проєктування ефективної архітектури зберігання даних є критично важливими для забезпечення продуктивності, надійності та масштабованості будь-якого сучасного веб-застосунку. Для платформи пошуку роботи, що оперує різноманітними типами даних – від структурованої інформації про користувачів та вакансії до великих обсягів логів активності та даних для машинного навчання – необхідний комплексний підхід. Рішення щодо СУБД приймалося на основі аналізу вимог до транзакційності, консистентності, гнучкості схеми, можливостей масштабування та інтеграції з іншими компонентами технологічного стеку.

Огляд типів СУБД та їх релевантність для проєкту:

- **Реляційні системи управління базами даних (SQL СУБД):**
  Цей клас систем характеризується зберіганням даних у структурованих таблицях зі строго визначеними зв'язками між ними, підтримкою ACID-транзакцій (Atomicity, Consistency, Isolation, Durability) та використанням мови SQL для маніпуляції даними.

- **PostgreSQL:** Була обрана як основна реляційна СУБД для даного проєкту. Це потужна об'єктно-реляційна система з відкритим вихідним кодом, що здобула визнання завдяки своїй винятковій надійності, відповідності стандартам SQL, широким можливостям розширення та активній спільноті. Ключові переваги PostgreSQL, що зумовили її вибір:
  - **Надійність та ACID-транзакції:** Гарантують цілісність критично важливих даних, таких як інформація про користувачів, історія відгуків та взаємодій.
  - **Розширюваність:** Підтримка користувацьких типів даних, функцій, операторів та розширень. Для даного проєкту особливе значення має можливість інтеграції з розширеннями для роботи з векторними даними (наприклад, pgvector, хоча в поточній версії системи він не використовується, можливість його додавання є перевагою для майбутніх ML-задач).
  - **Робота з напівструктурованими даними:** Нативна підтримка типу даних JSONB дозволяє ефективно зберігати та індексувати гнучкі дані, такі як налаштування користувачів або динамічні атрибути профілів, без необхідності створення окремих таблиць для кожного поля.
  - **Повнотекстовий пошук:** Вбудовані потужні засоби для повнотекстового пошуку з урахуванням морфології різних мов.
  - **Зрілість та екосистема:** Велика кількість інструментів для адміністрування, моніторингу, резервного копіювання та

інтеграції з різними мовами програмування та фреймворками.

- **MySQL:** Інша популярна реляційна СУБД з відкритим кодом, яка також широко використовується у веб-розробці та могла б бути альтернативою, проте PostgreSQL часто вважається більш потужною для складних запитів та розширюваності.

- **Нереляційні системи управління базами даних (NoSQL СУБД):**

NoSQL системи пропонують альтернативні моделі даних (документні, ключ-значення, колонкові, графові) і часто оптимізовані для специфічних типів навантажень, горизонтального масштабування та гнучкості схеми.

- **MongoDB (документо-орієнтована):** Зберігає дані у вигляді JSON-подібних документів (BSON), що забезпечує гнучкість схеми. Може бути корисною для зберігання профілів з динамічною структурою або для швидкого прототипування. Однак, для основних транзакційних даних платформи перевага була надана реляційній моделі.
- **Redis (ключ-значення/in-memory):** Високопродуктивне сховище даних, що працює переважно в оперативній пам'яті. У даному проєкті Redis не використовується для основного кешування, але його можливості для реалізації сесій, лічильників або швидких черг є добре відомими і можуть розглядатися в майбутньому.
- **Apache Cassandra (колонко-орієнтована):** Розрахована на обробку величезних обсягів даних з високою доступністю та

відмовостійкістю. Може бути актуальною для зберігання масивних логів активності або часових рядів на дуже великих масштабах, але для поточних завдань проєкту є надлишковою.

- **Сховища даних (Data Warehouses, DWH):**

Ці системи спеціально розроблені для зберігання та аналізу великих обсягів історичних даних, оптимізовані для складних аналітичних запитів (OLAP), на відміну від операційних баз даних (OLTP), орієнтованих на транзакції.

- **Amazon Redshift:** Повністю кероване хмарне сховище даних петабайтного масштабу від AWS. Використовує колоночне зберігання та масивну паралельну обробку (MPP) для швидкого виконання аналітичних запитів. У контексті даного проєкту, Amazon Redshift розглядається як стратегічний компонент для майбутнього масштабування аналітичних можливостей та підготовки даних для більш складних ML-моделей. На поточному етапі, для аналітики та підготовки даних для SageMaker Canvas, використовується зв'язка Amazon S3 (як Data Lake) та Amazon Athena, але з ростом обсягів та складності аналітики, міграція або доповнення цієї схеми Redshift'ом є логічним кроком. Redshift може агрегувати дані з операційної бази даних PostgreSQL та з S3 Data Lake.
- **Google BigQuery, Snowflake:** Популярні альтернативні хмарні сховища даних з подібними можливостями.

Обрана архітектура зберігання даних для проєкту:

Зважаючи на вищевикладене, для даного проєкту було прийнято рішення про наступну архітектуру зберігання даних:

1. **Основна операційна база даних (OLTP):** В якості основної СУБД для зберігання всіх транзакційних та операційних даних (профілі користувачів, вакансії, компанії, відгуки, взаємодії тощо) обрано **PostgreSQL**. Для забезпечення керованості, надійності, автоматичного резервного копіювання та легкого масштабування, PostgreSQL розгортається на сервісі **Amazon RDS (Relational Database Service)**.

2. **Сховище сирих та проміжних даних (Data Lake): Amazon S3 (Simple Storage Service)** використовується як централізоване Data Lake. Сюди завантажуються:

    - Сирі дані зі сторонніх джерел (якщо такі є, наприклад, агрегатори вакансій).
    - Детальні логи активності користувачів на платформі.
    - Експортовані дані з операційної бази даних PostgreSQL для архівування або подальшої обробки.
    - Проміжні та фінальні датасети, підготовлені для тренування моделей машинного навчання.
    - Артефакти тренованих ML-моделей.

3. **Доступ до даних для аналітики та ML:** На поточному етапі, для виконання SQL-запитів до даних, що зберігаються в S3 Data Lake (наприклад, для ad-hoc аналізу або підготовки даних для SageMaker

Canvas), використовується сервіс **Amazon Athena**. Він дозволяє працювати з даними в S3 "на місці" без необхідності їх завантаження в окрему базу даних.

Рис. 3. Схема потоків даних: Amazon RDS for PostgreSQL (OLTP) -> AWS Glue (ETL) -> Amazon S3 (Data Lake) -> Amazon Athena (для запитів до S3) -> AWS SageMaker.

Такий багаторівневий підхід до зберігання даних, що поєднує надійність PostgreSQL для операційних завдань, гнучкість S3 Data Lake та можливості Athena (з перспективою Redshift) для аналітики та ML, забезпечує необхідний баланс між продуктивністю, вартістю та масштабованістю для платформи пошуку роботи.

1.3.5 Інструменти Object-Relational Mapping (ORM)

ORM спрощує взаємодію між об'єктно-орієнтованим кодом застосунку та реляційною базою даних, дозволяючи працювати з даними як з об'єктами.

- **Sequelize:** Популярна ORM для Node.js, що підтримує різні SQL-діалекти (PostgreSQL, MySQL, SQLite, MSSQL). Надає можливості для визначення моделей, асоціацій, виконання запитів, міграцій.
- **TypeORM:** Розроблена спеціально для TypeScript (хоча може використовуватися і з JavaScript), TypeORM підтримує як Active Record, так і Data Mapper патерни. Добре інтегрується з Nest.js та підтримує багато СУБД, включаючи PostgreSQL.
- **Prisma:** Сучасний ORM-інструментарій, що включає Prisma Client (автоматично генерований типізований клієнт для доступу до БД), Prisma Migrate (для міграцій схеми) та Prisma Studio (візуальний редактор даних). Відомий своєю простотою використання та фокусом на безпеці типів.

Порівняння TypeORM та Prisma:

Критерій	**TypeORM**	**Prisma**
Основний підхід	Побудова запитів через декоратори та методи репозиторіїв/менеджерів сутностей	Використання генерованого типізованого клієнта з виразним API запитів

Визначення схеми	Зазвичай через класи з декораторами	Через спеціальний файл схеми Prisma (schema.prisma)
Міграції	Вбудована система міграцій (генерація та виконання SQL)	Prisma Migrate (генерація та виконання SQL, відстеження історії міграцій)
Типізація	Дуже добра підтримка TypeScript	Відмінна, генерує повністю типізований клієнт
Складність	Може бути дещо складнішою для початківців через гнучкість	Вважається простішою для освоєння завдяки інтуїтивному API

Таблиця 5.

Для проєкту, що використовує Nest.js [11] та TypeScript [2], як **TypeORM**, так і **Prisma** є сильними кандидатами. Вибір може залежати від переваг команди та специфічних вимог до гнучкості запитів. У прикладі дипломної роботи згадується TypeORM, що може слугувати орієнтиром.

1.3.6 Платформи та сервіси для розгортання компонентів застосунку

Вибір відповідних платформ та сервісів для розгортання кожного компонента багаторівневого веб-застосунку є ключовим для забезпечення його надійності, масштабованості, продуктивності та легкості управління. Для даного проєкту було обрано комбінований підхід, що використовує переваги спеціалізованої платформи для фронтенду та потужної хмарної інфраструктури для бекенду, баз даних та машинного навчання.

1.3.6.1 Розгортання Фронтенд-частини на платформі Vercel

Vercel – це хмарна платформа, спеціально розроблена та оптимізована для розгортання сучасних фронтенд-застосунків та статичних сайтів. Враховуючи, що клієнтська частина даного проєкту розроблена на Next.js (фреймворк, створений командою Vercel), вибір цієї платформи є логічним та надає значні переваги.

- Оптимізація для Next.js та JavaScript-фреймворків: Vercel забезпечує найкращу продуктивність та набір функцій саме для проєктів на Next.js, включаючи підтримку серверного рендерингу (SSR), статичної генерації (SSG), інкрементальної статичної регенерації (ISR) та Edge Functions.
- Глобальна Edge Network: Vercel розгортає застосунки на глобальній мережі CDN-вузлів, що забезпечує мінімальні затримки для користувачів по всьому світу та високу швидкість завантаження контенту.
- Автоматичне масштабування та висока доступність: Платформа автоматично масштабує ресурси відповідно до навантаження,

забезпечуючи стабільну роботу застосунку навіть при пікових навантаженнях. Вбудоване балансування навантаження розподіляє трафік між доступними серверами.

- Інтеграція з Git та CI/CD: Vercel безшовно інтегрується з популярними Git-провайдерами (GitHub, GitLab, Bitbucket). Кожен пуш у репозиторій може автоматично запускати процес збірки та розгортання. Особливо корисною є функція створення "preview deployments" для кожної гілки або пул-реквесту, що дозволяє легко тестувати зміни в ізольованому середовищі перед їх злиттям в основну гілку та розгортанням на продакшн.
- Простота використання та Developer Experience: Vercel відомий своїм інтуїтивно зрозумілим інтерфейсом та мінімальними вимогами до конфігурації, що значно спрощує процес розгортання та управління фронтенд-застосунками.

Для даного проєкту, де фронтенд на Next.js є ключовим компонентом для взаємодії з користувачем, Vercel надає оптимальне середовище для його розгортання, забезпечуючи швидкість, надійність та зручність для команди розробників.

1.3.6.2 Розгортання Серверної частини, Баз Даних та ML-інфраструктури на Amazon Web Services (AWS)

Для всіх компонентів серверної частини, баз даних, зберігання даних для машинного навчання та самої ML-інфраструктури було обрано хмарну платформу Amazon Web Services (AWS). AWS надає найширший спектр зрілих та надійних сервісів, що дозволяють побудувати гнучку, масштабовану та безпечну інфраструктуру.

Обчислювальні ресурси для Бекенд API (Nest.js):

- AWS Fargate (з Amazon ECS або EKS): Обраний як основний сервіс для розгортання Nest.js API, запакованого в Docker-контейнери. Fargate є серверним обчислювальним рушієм, що дозволяє запускати контейнери без необхідності управляти базовими EC2-інстансами. Це забезпечує легке масштабування (як горизонтальне, так і вертикальне, залежно від налаштувань сервісу ECS/EKS), високу доступність та інтеграцію з іншими сервісами AWS, такими як Application Load Balancer.
- AWS Lambda: Використовується для реалізації окремих безсерверних функцій, наприклад, для обробки асинхронних завдань з черг SQS, для виконання деяких MLOps-автоматизацій (функції, що реагують на події EventBridge), або для розгортання ML-моделей для інференсу (як альтернатива постійно працюючим SageMaker Endpoints, якщо це економічно вигідніше для поточного навантаження).

Управління API:

- Amazon API Gateway: Слугує як єдина, керована точка входу для всіх запитів до бекенд API. API Gateway забезпечує маршрутизацію запитів до відповідних сервісів на Fargate або Lambda, автентифікацію та авторизацію (наприклад, через інтеграцію з Amazon Cognito), обмеження швидкості запитів (throttling), кешування відповідей (за потреби) та моніторинг API.

Керовані Бази Даних:

- Amazon RDS (Relational Database Service) for PostgreSQL: Надає повністю керовані екземпляри PostgreSQL. Це значно спрощує адміністрування бази даних, включаючи налаштування, резервне копіювання, застосування патчів, моніторинг та масштабування (як вертикальне, так і через репліки читання).

  Сховища Даних та Інфраструктура для ML:

- Amazon S3 (Simple Storage Service): Використовується як основне Data Lake для зберігання великих обсягів сирих даних, логів, проміжних результатів ETL-процесів, датасетів для тренування ML-моделей та артефактів самих моделей.
- Amazon Athena: Інтерактивний сервіс запитів, що дозволяє виконувати SQL-запити безпосередньо до даних в S3, що є ключовим для аналізу даних та їх підготовки для SageMaker Canvas без необхідності переміщення в окреме сховище.
- AWS Glue: Сервіс для реалізації ETL-процесів (вилучення, трансформація, завантаження даних), наприклад, для переміщення та підготовки даних з Amazon RDS в S3 Data Lake у форматі, оптимізованому для Athena та SageMaker.
- AWS SageMaker Canvas (та супутні сервіси SageMaker): Комплексна платформа для машинного навчання, де SageMaker Canvas використовується для AutoML-розробки моделей, SageMaker Model Registry для їх версіонування, а SageMaker Endpoints (або Lambda) для розгортання на інференс.

  Безпека та Управління Доступом:

- AWS IAM (Identity and Access Management): Забезпечує гранульований контроль доступу до всіх ресурсів AWS.
- Amazon Cognito: Для управління ідентифікацією та автентифікацією користувачів платформи.

Вибір AWS для бекенд-інфраструктури зумовлений широким набором сервісів, їхньою зрілістю, надійністю, можливостями масштабування та тісною інтеграцією між собою, що особливо важливо для реалізації комплексних рішень з компонентами машинного навчання.

1.3.7 Хмарні сервіси рекомендацій та MLOps

Інтеграція інтелектуальних функцій, таких як персоналізовані рекомендації та прогнозування поведінки користувачів, є ключовим аспектом створення конкурентоспроможної платформи пошуку роботи. Для реалізації цих функцій необхідно обрати відповідну платформу для машинного навчання (ML) та налаштувати процеси MLOps для управління життєвим циклом моделей.

Огляд та порівняння платформ для ML:

Платформа / Сервіс	Тип / Фокус	Простота використання (для Non-ML	Гнучкість / Контроль	Інтеграція з AWS	MLOps "з коробки"	Вартість

		експертів)				
AWS SageMaker Canvas	AutoML / Low-code ML	Висока (візуальний інтерфейс)	Обмежена, фокус на автоматизації	Нативна	Базові можливості, інтеграція з Model Registry	Залежить від часу використання та тренування
AWS SageMaker Studio (з Notebooks)	Повноцінна Data Science платформа (Custom Code)	Низька/Середня (потребує знань Python, ML)	Дуже висока, повний контроль	Нативна	SageMaker Pipelines, Experiments, Debugger	Залежить від інстансів та часу
Google Vertex AI (AutoM	Комплексна ML платформа	Висока (для AutoML), Середня	Висока (для Custom)	Обмежена	Vertex AI Pipelines	Залежить від використ

L Tables / Custom Training)	(AutoML та Custom)	(для Custom)			, Model Registry	ання сервісів
Azure Machine Learning (Automated ML / Designer / Notebooks)	Комплексна ML платформа (AutoML та Custom)	Висока (для AutoML/ Designer), Середня (для Notebooks)	Висока (для Notebooks)	Обмежена	Azure ML Pipelines, Model Registry	Залежить від використання сервісів
Сторонні SaaS AutoML платформи (H2O.ai, DataRo	Спеціалізовані AutoML рішення	Висока	Залежить від платформи, часто обмежена	Через API	Залежить від платформи	Зазвичай висока (підписка)

bot тощо)						

Таблиця 6.

Обґрунтування вибору AWS SageMaker Canvas:

Для даного проєкту, особливо на етапі розробки та впровадження перших версій ML-моделей (зокрема, для задачі бінарної класифікації – прогнозування прийняття запрошення кандидатом), **AWS SageMaker Canvas** було обрано як основний інструмент з наступних причин:

- **Швидкість прототипування та розробки:** SageMaker Canvas дозволяє значно прискорити цикл розробки ML-моделей завдяки своєму візуальному інтерфейсу та можливостям AutoML. Користувачі без глибоких навичок програмування ML можуть швидко імпортувати дані, автоматично підготувати їх, натренувати та оцінити декілька варіантів моделей.
- **Низький поріг входження:** Це робить ML-розробку доступнішою для ширшого кола спеціалістів у команді, не обмежуючись лише досвідченими дата-саєнтистами.
- **Інтеграція з екосистемою AWS:** SageMaker Canvas безшовно інтегрується з іншими сервісами AWS, такими як Amazon S3 (для зберігання датасетів та артефактів моделей), Amazon Athena (для доступу до даних в S3 Data Lake), SageMaker Model Registry (для

версіонування) та SageMaker Endpoints (для розгортання). Це спрощує побудову наскрізного MLOps-процесу в межах однієї платформи.

- **Автоматизація ключових етапів:** Canvas автоматизує багато рутинних завдань, таких як інженерія ознак (базовий рівень), вибір моделі та налаштування гіперпараметрів, що дозволяє зосередитися на бізнес-проблемі та інтерпретації результатів.
- **Інструменти для аналізу та інтерпретації:** Надає зрозумілі метрики продуктивності та візуалізацію впливу ознак (feature importance), що допомагає оцінити якість моделі та зрозуміти фактори, що впливають на прогнози.

Хоча SageMaker Studio надає значно більшу гнучкість та контроль для кастомної розробки складних моделей, для завдань, де швидкість виходу на ринок та простота використання є пріоритетом, SageMaker Canvas є дуже ефективним інструментом.

Передбачуваний робочий процес з SageMaker Canvas та MLOps:

- **Підготовка даних:** Дані з операційної бази даних (Amazon RDS PostgreSQL) та інших джерел агрегуються та обробляються за допомогою **AWS Glue ETL** і зберігаються в **Amazon S3 Data Lake**. **Amazon Athena** використовується для формування фінальних датасетів для Canvas.
- **Створення та тренування моделі в Canvas:** Датасети з S3 імпортуються в SageMaker Canvas, де відбувається візуальна підготовка даних, автоматичне тренування моделі та її оцінка.

- **Реєстрація та розгортання:** Успішна модель реєструється в **SageMaker Model Registry** та розгортається на **SageMaker Endpoint** для інференсу в реальному часі.
- **Автоматизація перетренування:** Для підтримки актуальності моделі, процеси її перетренування на нових даних автоматизуються. **AWS EventBridge** (або CloudWatch Events) може за розкладом запускати завдання перетренування в SageMaker Canvas. **GitHub Actions** можуть слугувати для ініціації цих процесів у відповідь на зміни в коді підготовки даних або за іншими бізнес-тригерами, викликаючи відповідні AWS API [14] [15].

1.4. Автоматизоване End-to-End тестування

Забезпечення високої якості та стабільності користувацького інтерфейсу є критично важливим аспектом розробки будь-якої сучасної веб-платформи. Для досягнення цієї мети, особливо в умовах частих оновлень та розширення функціоналу, впровадження автоматизованого end-to-end (E2E) тестування є необхідністю. E2E тести дозволяють перевіряти працездатність ключових користувацьких сценаріїв через взаємодію з інтерфейсом так, як це робив би реальний користувач. При виборі фреймворку для E2E тестування для даного проєкту розглядалися декілька популярних рішень [17].

Огляд та порівняння інструментів для E2E тестування:

Критерій	Playwright	Cypress	Selenium WebDriver
Підтримка браузерів	Chromium, Firefox, WebKit (Safari) "з коробки"	Переважно Chromium-based, експериментально Firefox/WebKit	Широка, але потребує окремих драйверів для кожного браузера
Швидкість виконання	Дуже висока, ефективна паралелізація тестів	Висока, але може поступатися Playwright в деяких сценаріях	Залежить від якості драйвера та архітектури тестів
Надійність ("Flakiness")	Низька, завдяки механізмам авто-очікування	Середня, іноді потребує більш явного налаштування очікувань	Може бути високою без ретельного налаштування очікувань

API та зручність розробки	Сучасний, інтуїтивно зрозумілий, відмінна підтримка TypeScript	Зручний, добре підходить для JavaScript/TypeScript розробників	Більш низькорівневий, може вимагати більше коду для тих самих дій
Ключові можливості	Auto-waits, перехоплення мережевих запитів, емуляція різних пристроїв, генерація коду, відеозапис виконання	Вбудований інтерактивний Test Runner, зручні інструменти для налагодження, хороший дашборд	Надзвичайна гнучкість, велика кількість мовних біндінгів, величезна спільнота
Налаштування та інтеграція	Відносно просте, хороша документація	Дуже просте, особливо для проєктів на JavaScript	Може бути більш складним, особливо для початківців

Таблиця 7.

Обґрунтування вибору Playwright:

Для проєкту розробки багаторівневої платформи пошуку роботи, де важливими є швидкість розробки, надійність тестів та підтримка сучасних веб-технологій, Playwright виділяється як оптимальний вибір. Його ключові переваги:

- **Кросбраузерність "з коробки":** Можливість тестувати на всіх основних рушіях (Chromium, Firefox, WebKit) з єдиного API без необхідності складних налаштувань є значним плюсом для забезпечення широкої сумісності.
- **Швидкість та надійність:** Архітектура Playwright та його механізми автоматичного очікування (auto-waits) сприяють швидшому та більш стабільному виконанню тестів, зменшуючи кількість хибних спрацьовувань ("flaky tests").
- **Відмінна підтримка TypeScript:** Оскільки значна частина проєкту (фронтенд на Next.js, бекенд на Nest.js) розробляється на TypeScript, нативна підтримка TypeScript у Playwright спрощує написання тестів та забезпечує кращу інтеграцію.
- **Потужні інструменти для розробника:** Можливості, такі як генератор коду (Code generación), інспектор, трасування та відеозапис виконання, значно полегшують створення та налагодження E2E тестів.
- **Активний розвиток та підтримка Microsoft:** Гарантує регулярні оновлення та впровадження нових можливостей.

Хоча Cypress пропонує чудовий "developer experience" та інтерактивний Test Runner, його обмеження в кросбраузерності (на момент аналізу) та деякі архітектурні особливості роблять Playwright більш універсальним рішенням. Selenium WebDriver, будучи ветераном у галузі, залишається потужним, але часто вимагає більше зусиль для налаштування та написання стабільних тестів порівняно з більш сучасними альтернативами.

Інтеграція з CI/CD: Передбачається тісна інтеграція обраного E2E фреймворку (Playwright) з системою неперервної інтеграції (CI), такою як **GitHub Actions**. Створення робочого процесу (наприклад, playwright.yml), який автоматично запускатиме набір E2E тестів при кожному пуші в репозиторій або при створенні пул-реквесту, є стандартною практикою для забезпечення якості коду.

Звітування: Важливою частиною E2E тестування є генерація детальних звітів. Playwright надає можливості для створення HTML-звітів, які включають інформацію про пройдені/невдалі тести, скріншоти, відео та трасування. Ці звіти можуть зберігатися як артефакти CI-пайплайну для подальшого аналізу [17].

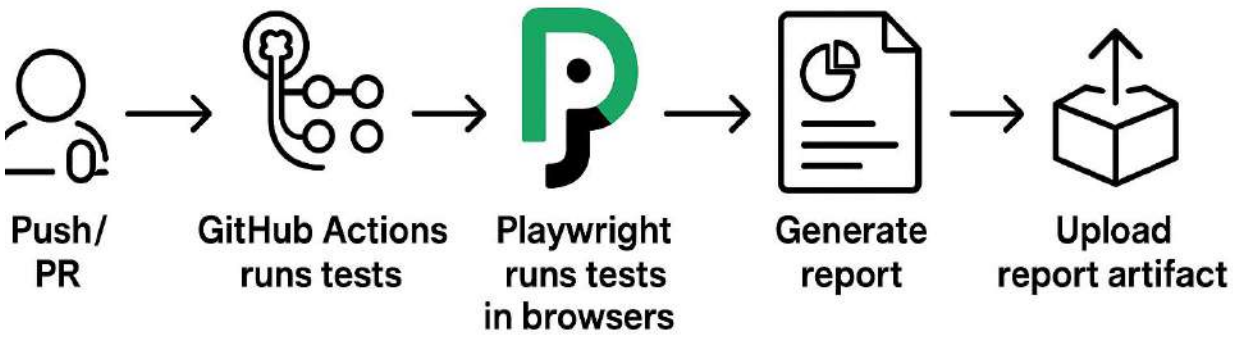

Рис. 5. Концептуальна схема інтеграції Playwright E2E тестів у CI/CD пайплайн на GitHub Actions

Таким чином, вибір Playwright як основного інструменту для автоматизованого E2E тестування дозволить забезпечити високий рівень якості фронтенд-застосунку, скоротити час на ручне тестування та підвищити загальну надійність платформи [17].

1.5. Моніторинг помилок

Для забезпечення стабільної та безперебійної роботи багатокомпонентної веб-платформи, а також для оперативного реагування на будь-які проблеми, що можуть виникати у користувачів або в інфраструктурі, необхідне впровадження ефективної системи моніторингу помилок та продуктивності. На ринку існує низка рішень, і вибір конкретного інструменту залежить від потреб проєкту, інтеграційних можливостей та функціоналу.

Огляд та порівняння інструментів моніторингу:

Критерій	Sentry	New Relic	Datadog	AWS CloudWatch (Logs + RUM + X-Ray)
Основний фокус	Відстеження помилок (error tracking), моніторинг продуктивності (APM)	Повноцінний APM, моніторинг інфраструктури, бізнес-аналітика	Повноцінний APM, моніторинг інфраструктури, логів, безпеки	Моніторинг ресурсів AWS, логів, трасування, RUM
Підтримка технологій	Дуже широка (JavaScript, Node.js, Python, Java, PHP, Ruby, .NET та ін.)	Широка, з акцентом на популярних стеках	Дуже широка, велика кількість інтеграцій	Переважно для сервісів AWS, можливість кастомних метрик

Відстеження помилок на клієнті	Відмінне, детальний контекст	Добре, але може бути менш детальним, ніж у Sentry	Добре	AWS RUM (Real User Monitoring)
Відстеження помилок на бекенді	Відмінне, інтеграція з багатьма фреймворкам и	Дуже потужне, глибока трасування	Дуже потужне, кореляція з логами та інфраструктуро ю	CloudWatch Logs, X-Ray для трасування
Моніторинг продуктивно сті	APM-функціонал (транзакції, спани)	Розширений APM, аналіз вузьких місць	Розширений APM, кореляція з інфраструктуро ю	CloudWatch Metrics, X-Ray
Ціноутворен ня	Модель Freemium, платні тарифи залежно від обсягу	Платна, залежить від кількості хостів, обсягу даних	Платна, модульна, залежить від обраних	Pay-as-you-go за використан ня сервісів AWS

	подій/користувачів, є self-hosted		продуктів та обсягу	
Зручність використання	Висока, інтуїтивний інтерфейс	Середня, може бути складною через велику кількість функцій	Середня/Висока, потужний, але вимагає налаштування	Інтегровано в екосистему AWS, але UI може бути складним

Таблиця 8.

Обґрунтування вибору Sentry:

Для даного проєкту, де розробка ведеться на стеку Next.js (фронтенд) та Nest.js (бекенд), **Sentry** є оптимальним вибором з наступних причин [18]:

- **Відмінна підтримка JavaScript/TypeScript екосистеми:** Sentry надає високоякісні SDK для Node.js та браузерного JavaScript, які легко інтегруються з Nest.js та Next.js. Це забезпечує глибокий збір контексту помилок, специфічного для цих фреймворків.
- **Фокус на відстеженні помилок з детальним контекстом:** Sentry відомий своїми потужними можливостями для збору детальної інформації про кожну помилку (стек викликів, "хлібні крихти", теги,

дані користувача), що значно прискорює процес діагностики та виправлення.

- **Зручний інтерфейс та ефективне групування помилок:** Інтерфейс Sentry інтуїтивно зрозумілий, а механізми групування схожих помилок допомагають уникнути інформаційного перевантаження та зосередитися на найважливіших проблемах.
- **Модель Freemium та можливість Self-hosting:** Наявність безкоштовного тарифу для хмарного SaaS-рішення дозволяє легко розпочати роботу та оцінити можливості сервісу. Можливість самостійного розгортання Sentry (self-hosted) надає гнучкість для компаній з особливими вимогами до безпеки або контролю даних.
- **Моніторинг продуктивності:** Окрім відстеження помилок, Sentry надає функціонал для моніторингу продуктивності (APM), що дозволяє аналізувати тривалість транзакцій та виявляти "вузькі місця" як на фронтенді, так і на бекенді.
- **Активна спільнота та хороша документація:** Забезпечують легкість інтеграції та вирішення можливих проблем.

Хоча комплексні APM-рішення, такі як New Relic або Datadog, пропонують ширший спектр інструментів для моніторингу інфраструктури, їхня вартість та складність можуть бути надлишковими для початкових етапів проєкту. AWS CloudWatch є потужним інструментом для моніторингу саме ресурсів AWS, але для детального відстеження помилок на рівні коду застосунку Sentry часто надає більш зручний та сфокусований інструментарій.

Інтеграція Sentry в архітектуру:

- SDK Sentry буде інтегровано на рівні ініціалізації фронтенд-застосунку (в _app.tsx або аналогічному файлі для Next.js) для перехоплення помилок JavaScript та проблем з рендерингом.
- На бекенді (Nest.js) Sentry SDK буде налаштовано як глобальний обробник виключень або інтегровано через спеціалізовані модулі для Nest.js, що дозволить перехоплювати помилки в контролерах, сервісах та інших частинах API.
- Обидві інтеграції надсилатимуть дані про помилки до єдиного проєкту Sentry (або окремих проектів для фронтенду та бекенду для кращого розділення).

Використання Sentry дозволить підтримувати високий рівень якості програмного продукту, оперативно реагувати на інциденти та постійно покращувати стабільність і продуктивність платформи.

1.6 Висновки до Розділу 1

При аналізі технологій для розробки **серверної частини (Бекенд API)** для багаторівневого веб-застосунку, було обрано платформу Node.js та прогресивний фреймворк Nest.js. Це рішення забезпечує модульну архітектуру, високу продуктивність для асинхронних операцій завдяки подієво-орієнтованій природі Node.js, а також надійність коду за рахунок використання TypeScript. Для взаємодії з базою даних PostgreSQL, розгорнутою на Amazon RDS, було обрано ORM TypeORM, що спрощує роботу з даними в TypeScript-середовищі та забезпечує чітке визначення сутностей. Інтерфейс API реалізовано з використанням GraphQL, що надає гнучкість запитів для клієнтських застосунків.

Для розробки **клієнтської частини (Фронтенд)** було обрано бібліотеку React у поєднанні з фреймворком Next.js. Цей вибір дозволяє створювати швидкі, SEO-оптимізовані та інтерактивні користувацькі інтерфейси, використовуючи переваги серверного рендерингу та статичної генерації. Для ефективного управління станом, пов'язаним із запитами до GraphQL API, та кешування даних на клієнті, було інтегровано Apollo Client.

Для **розгортання бекенд-застосунку (Nest.js)**, розгортнутого через Docker-контейнери, було обрано сервіс AWS Fargate, що працює в поєднанні з Amazon ECS. Це забезпечує масштабованість, керованість та усуває необхідність управління базовими серверами. Доступ до API здійснюється через Amazon API Gateway, що надає додатковий шар безпеки та управління.

Для **розгортання фронтенд-застосунку (Next.js)** було обрано платформу Vercel, яка тісно інтегрована з Next.js та забезпечує оптимальну продуктивність, глобальну CDN та автоматизовані процеси CI/CD.

Інфраструктура **зберігання та обробки даних** побудована на сервісах AWS: Amazon S3 використовується як Data Lake, а Amazon Athena – для виконання SQL-запитів до даних в S3, зокрема для підготовки даних для **модуля машинного навчання**. Для реалізації функцій ML було обрано AWS SageMaker Canvas, що дозволяє створювати та розгортати моделі за допомогою AutoML.

Обраний комплекс технологій, інструментів та хмарних сервісів (включаючи GitHub Actions для CI/CD, Playwright для E2E-тестування та Sentry для моніторингу помилок) дозволяє створювати надійний, масштабований та легкий у підтримці веб-застосунок. Вони сприяють

ефективній розробці, забезпечують високу продуктивність та допомагають мінімізувати типові проблеми, що виникають при створенні складних багаторівневих систем.

Розділ 2. Програмна реалізація

У даному розділі детально розглядаються етапи проєктування та програмної реалізації ключових компонентів багаторівневого веб-застосунку для платформи пошуку роботи. Описується технічне завдання, визначаються основні функціональні та нефункціональні вимоги, проєктується архітектура системи та структура бази даних. Також наводяться приклади реалізації фронтенд-частини, бекенд API та модуля машинного навчання, з акцентом на технологіях та підходах, обраних у Розділі 1. Цей розділ є основою для розуміння практичної імплементації запропонованого рішення.

2.1 Технічне завдання

Технічне завдання (ТЗ) є фундаментальним документом, що визначає мету, призначення, вимоги та основні характеристики розроблюваного програмного продукту. Воно слугує основою для проєктування, розробки, тестування та подальшого супроводу системи.

2.1.1 Функціональне призначення

Розроблюваний веб-застосунок призначений для створення інноваційної та інтелектуальної платформи для пошуку роботи, яка кардинально покращує взаємодію між шукачами роботи (надалі – кандидатами) та роботодавцями. Ключовою диференціюючою особливістю платформи є глибока інтеграція моделей машинного навчання для надання високоперсоналізованих рекомендацій вакансій кандидатам та найбільш релевантних кандидатів роботодавцям, а також для прогнозування успішності потенційних взаємодій, як-от прийняття запрошення на вакансію.

Основні функціональні призначення застосунку можна деталізувати наступним чином:

Для кандидатів (шукачів роботи):

- Комплексне управління профілем: Можливість реєстрації через email/пароль або соціальні мережі, створення та редагування розширеного профілю, що включає особисті дані, контактну інформацію, детальну освіту, сертифікати, вичерпний досвід роботи, професійні навички (з можливістю самооцінки), бажані типи робіт (наприклад, повна зайнятість, часткова), географічні вподобання, очікуваний рівень заробітної плати, володіння мовами (з акцентом на рівні німецької для ринку Німеччини), завантаження резюме та супровідних документів.
- Інтелектуальний пошук вакансій: Розширений пошук вакансій за множиною критеріїв: ключові слова, назва посади, спеціалізація, тип медичного закладу, регіон, місто, поштовий індекс, рівень заробітної плати, тип зайнятості, графік роботи, компанія-роботодавець. Можливість збереження пошукових запитів.
- Персоналізовані рекомендації вакансій: Динамічна стрічка вакансій, що формується індивідуально для кожного кандидата за допомогою ML-моделі. Рекомендації базуються на комплексному аналізі профілю кандидата, його попередньої активності на платформі (перегляди, відгуки, збережені вакансії), семантичній схожості з описом вакансій та прогнозованій ймовірності інтересу.

- Взаємодія з вакансіями: Можливість детального перегляду інформації про вакансію та компанію, подання відгуку на вакансію (з можливістю додати супровідний лист), збереження вакансій до списку "Обране" для подальшого розгляду.
- Управління запрошеннями від роботодавців: Отримання сповіщень про запрошення на вакансії від роботодавців, можливість переглянути деталі запрошення та прийняти або відхилити його.
- Безпечна комунікація: Вбудований месенджер (чат) для прямого та безпечного спілкування з представниками роботодавців щодо вакансій та співбесід.
- Профільна аналітика та зворотний зв'язок: Доступ до статистики власної активності: кількість переглядів профілю, кількість надісланих відгуків, статус відгуків, кількість отриманих запрошень. Можливість отримання зворотного зв'язку щодо повноти та привабливості профілю.

Для роботодавців (представників медичних закладів та компаній):

- Управління профілем компанії: Реєстрація компанії, створення та редагування детального профілю компанії, включаючи опис, місію, цінності, фото/відео матеріали, інформацію про умови праці, переваги для співробітників, розташування філій.
- Публікація та управління вакансіями: Створення нових вакансій з детальним описом посадових обов'язків, вимог до кандидатів (освіта, досвід, навички, сертифікати), умов праці, рівня заробітної плати (з можливістю вказати діапазон або приховати).

Можливість редагування, активації/деактивації та архівації вакансій.

- Розширений пошук кандидатів: Доступ до бази даних кандидатів з можливістю фільтрації за численними параметрами: навички, досвід, освіта, локація, тип бажаної роботи, рівень володіння мовами, готовність до переїзду тощо.
- Інтелектуальні рекомендації кандидатів: Отримання списку найбільш релевантних кандидатів для кожної відкритої вакансії, згенерованого ML-моделлю на основі зіставлення вимог вакансії та характеристик кандидатів.
- Система "Розумних запрошень" (Smart Invites): Використання прогнозів ML-моделі для оцінки ймовірності прийняття кандидатом запрошення на конкретну вакансію. Це дозволяє рекрутерам пріоритезувати кандидатів та оптимізувати процес залучення.
- Управління відгуками та кандидатами: Перегляд відгуків на вакансії, сортування кандидатів, зміна їхнього статусу в процесі найму (наприклад, "новий", "розглядається", "співбесіда", "пропозиція", "відхилено").
- Комунікація з кандидатами: Використання вбудованого чату для спілкування з кандидатами, призначення співбесід.

Для адміністраторів платформи:

- Управління користувачами та контентом: Модерація профілів кандидатів та компаній, управління вакансіями, блокування користувачів за порушення правил.

- Моніторинг системи: Відстеження працездатності всіх компонентів платформи, включаючи ML-моделі (точність, дрейф даних).
- Аналітика та звітність: Збір та аналіз статистики використання платформи (кількість реєстрацій, активних користувачів, опублікованих вакансій, успішних наймів), ефективності рекомендаційної системи.
- Конфігурація системи: Налаштування параметрів платформи, включаючи параметри ML-моделей (наприклад, пороги для рекомендацій).

Таким чином, функціональне призначення застосунку охоплює повний цикл взаємодії на ринку праці, збагачений інтелектуальними можливостями для підвищення ефективності та зручності для всіх учасників.

2.1.2 Архітектура та основні вимоги

Застосунок проєктується як сучасна багаторівнева система, що забезпечує чітке розділення відповідальності між компонентами, гнучкість та масштабованість. Основні архітектурні компоненти та їх взаємодія були описані в Розділі 1 і включають:

1. **Фронтенд (Клієнтська частина):** Розроблена на базі фреймворку **Next.js** (з використанням React та TypeScript), ця частина відповідає за весь користувацький інтерфейс та взаємодію з користувачем у браузері. Розгортання здійснюється на платформі **Vercel**, що забезпечує глобальну CDN, серверний рендеринг (SSR/ISR) та Edge Functions.

2. **Бекенд (Серверна частина API):** Є центральним вузлом, що реалізує бізнес-логіку системи. Побудований на фреймворку **Nest.js** (Node.js, TypeScript), він надає **GraphQL API** для комунікації з фронтендом та іншими потенційними клієнтами. Бекенд відповідає за обробку запитів, автентифікацію, авторизацію, валідацію даних та взаємодію з базою даних та модулем машинного навчання. Розгортається на хмарній інфраструктурі **AWS**, використовуючи такі сервіси, як **AWS Fargate** для контейнеризованих додатків або **AWS Lambda** для безсерверних функцій [1] [4].

3. **База даних:** Як основне сховище даних використовується реляційна СУБД **PostgreSQL**, розгорнута на **Amazon RDS**. Для підтримки векторних операцій, необхідних для семантичного пошуку та ML-рекомендацій [14].

4. **Модуль машинного навчання (ML Module):** Цей модуль є серцем інтелектуальних функцій платформи. Він охоплює весь життєвий цикл ML-моделей:

- **Збір та підготовка даних:** Дані збираються з **Amazon S3 Data Lake** та обробляються за допомогою запитів **Amazon Athena** та скриптів у **SageMaker Processing Jobs**.
- **Тренування та оцінка моделей:** Використовується **Amazon SageMaker Studio** з Jupyter Notebooks для розробки та експериментів, **AutoGluon** для автоматизованого вибору та тренування моделей, та **SageMaker Training Jobs** для масштабованого тренування.

- **Реєстрація та зберігання моделей:** Треновані моделі та їх артефакти зберігаються на **S3** та реєструються в **SageMaker Model Registry**.
- **Розгортання та інференс:** Схвалені моделі розгортаються як **AWS Lambda** функції, доступ до яких здійснюється через **Amazon API Gateway**.

5. **Система MLOps:** Забезпечує автоматизацію та управління життєвим циклом ML-моделей. Включає **GitHub Actions** для CI/CD коду та запуску пайплайнів, **SageMaker Pipelines** для оркестрації кроків тренування та валідації, **Amazon EventBridge** для реагування на події (наприклад, реєстрація нової моделі) та запуску відповідних процесів [14] [15].

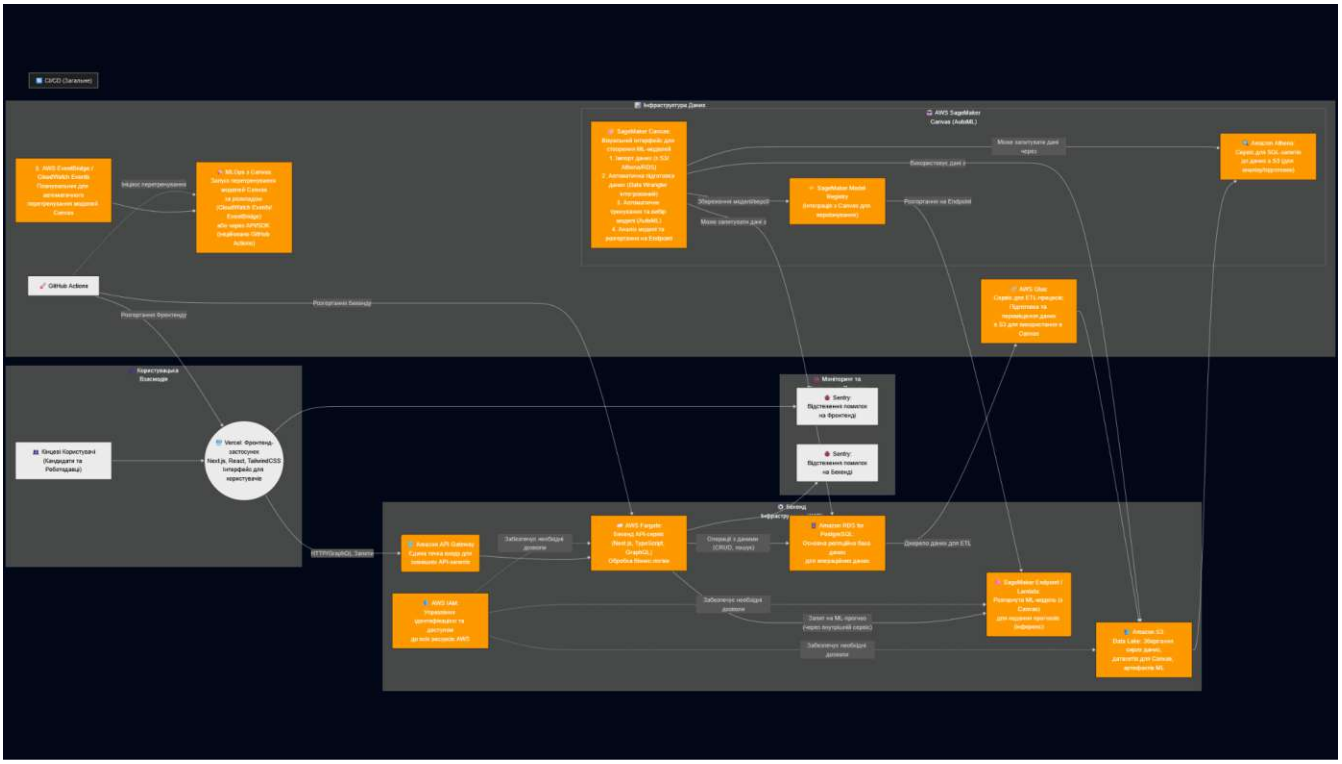

Рис. 6. Високорівнева архітектура веб-застосунку

2.1.2.1 Вимоги до бекенд-частини та API

Бекенд є ядром системи, що забезпечує стабільну та безпечну роботу. Основні вимоги:

- **Управління даними:** Розробка структурованої схеми PostgreSQL; використання ORM (TypeORM/Prisma) для взаємодії з БД; забезпечення нормалізації та цілісності даних.
- **GraphQL API:** Проєктування чіткої схеми (запити, мутації, підписки); ефективний резолвінг; підтримка пагінації, сортування, фільтрації; WebSocket для реального часу.
- **Автентифікація/Авторизація:** Інтеграція з Amazon Cognito; JWT-автентифікація; рольова модель доступу (Nest.js Guards).
- **Бізнес-логіка:** Обробка профілів, вакансій, відгуків, запрошень; реалізація пошуку; логіка сповіщень та чату.
- **Інтеграція з ML:** Інтерфейси для взаємодії з SageMaker Endpoint/Lambda для отримання прогнозів; використання прогнозів для покращення рекомендацій.
- **Масштабованість/Доступність:** Використання AWS Fargate/Lambda.
- **Безпека:** Валідація вхідних даних (class-validator); захист від веб-атак (OWASP); шифрування даних (HTTPS, AWS KMS).
- **Модульність/Тестованість:** Дотримання архітектури Nest.js; юніт- та інтеграційні тести (Jest).

- **Документація/Моніторинг:** Автогенерація GraphQL схеми; інтерактивні інструменти (GraphQL Playground); логування (CloudWatch Logs); моніторинг помилок (Sentry).

2.1.2.2 Вимоги до клієнтської частини (фронтенд)

Фронтенд має забезпечувати зручний та ефективний користувацький досвід.

- **UI/UX:** Інтуїтивний, сучасний, адаптивний дизайн (Tailwind CSS, Headless UI); легка навігація.
- **Функціональність:** Повний набір інструментів для кандидатів (профіль, пошук, відгуки, чат) та роботодавців (профіль компанії, вакансії, пошук кандидатів, запрошення).
- **Продуктивність:** Швидке завантаження (Next.js SSR/ISR, оптимізації); чуйність інтерфейсу.
- **Взаємодія з API:** Ефективне використання GraphQL (Apollo Client); управління клієнтським станом.
- **Безпека/Доступність/SEO:** Валідація на клієнті; захист від XSS; дотримання WCAG; SEO-дружні URL та мета-теги.
- **Аналітика:** Інтеграція з системами веб-аналітики.

2.1.2.3 Вимоги до модуля машинного навчання та MLOps

Інтелектуальні функції платформи залежать від ефективності ML-модуля.

- **Якість прогнозів:** Досягнення бізнес-значущих метрик для моделей бінарної класифікації (Accuracy, F1-score, Precision, Recall).

- **Інженерія ознак:** Відтворюваний процес створення ознак (SQL в Athena, SageMaker Processing Jobs).
- **Автоматизація MLOps:** CI/CD для ML (GitHub Actions, SageMaker Pipelines); автоматизоване перетренування моделей (SageMaker Canvas, EventBridge); версіонування (дані, код, моделі в SageMaker Model Registry).
- **Моніторинг моделей:** Відстеження операційних метрик (CloudWatch) та якості моделі (дрейф даних/концепції через SageMaker Model Monitor).
- **Масштабованість/Ефективність інференсу:** Автомасштабування SageMaker Endpoint/Lambda; оптимізація для зменшення "холодних стартів".
- **Етика та безпека даних в ML:** Дотримання GDPR; анонімізація/псевдонімізація; аналіз на упередженість.

Дотримання цих вимог забезпечує створення якісного, надійного та конкурентоспроможного продукту.

2.2 Структура бази даних

Ефективна та добре продумана структура бази даних є критично важливим елементом для будь-якого складного веб-застосунку, особливо для платформи пошуку роботи, яка оперує великими обсягами різноманітних даних. Для даного проєкту, обрано реляційну систему управління базами даних PostgreSQL з розширенням pgvector. PostgreSQL надає надійність, ACID-транзакції, широкі можливості для запитів та розширюваність, тоді як

pgvector дозволяє ефективно зберігати та шукати векторні представлення (ембединги), що є ключовим для реалізації інтелектуальних функцій пошуку та рекомендацій.

Нижче наведено розширений опис основних сутностей (таблиць) та їхніх взаємозв'язків, що формують ядро бази даних платформи.

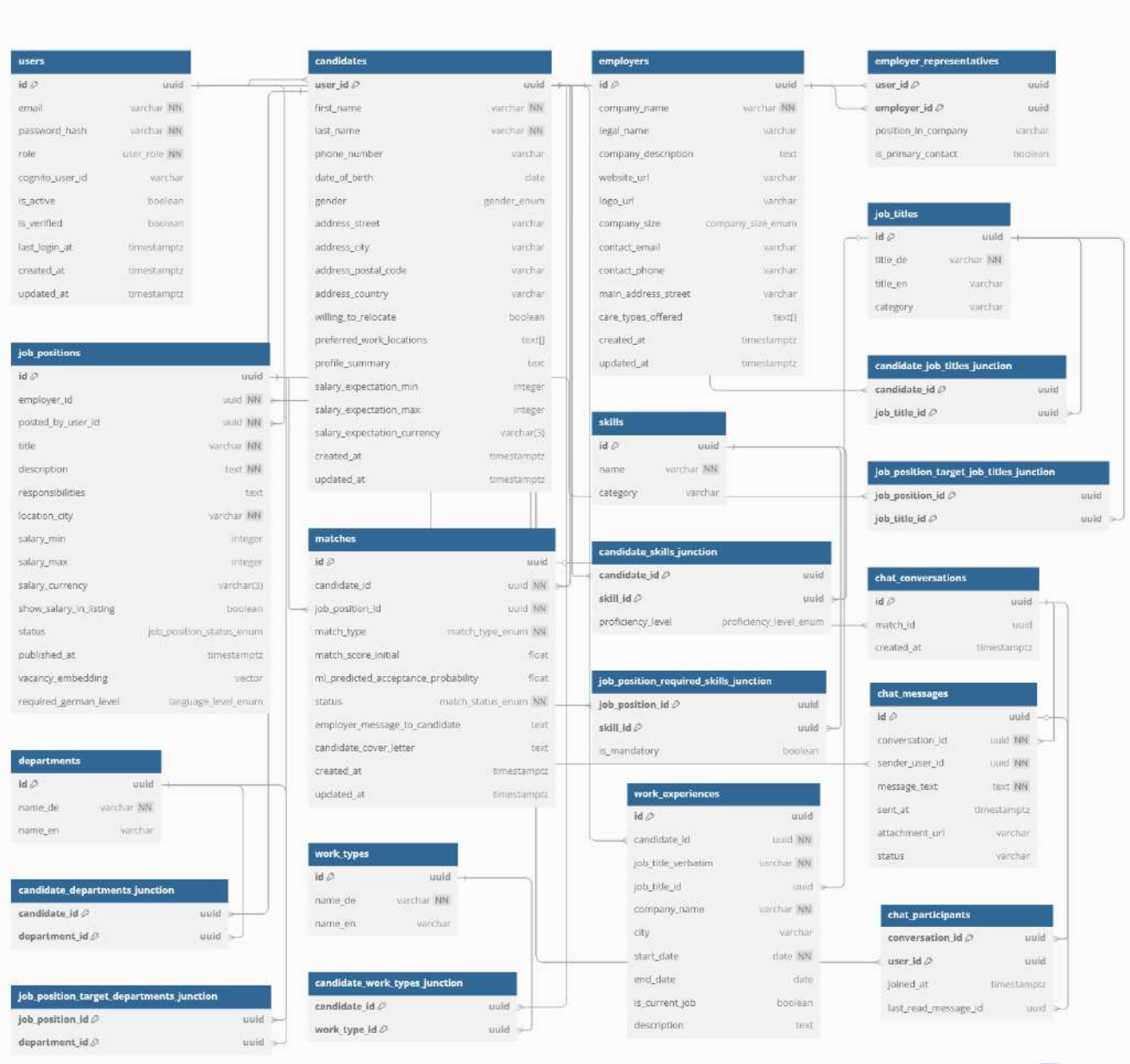

Рис. 7. Деталізована концептуальна ERD діаграма

2.2.1 Основні сутності та їх атрибути

Після аналізу вимог, розроблено архітектуру бази даних з урахуванням вимог до масштабованості, продуктивності, та безпеки даних.

- **users (Користувачі)**: Зберігає фундаментальну інформацію про всіх зареєстрованих користувачів системи.
- **candidates (Кандидати)**: Розширена інформація, специфічна для профілів шукачів роботи.
- **employers (Роботодавці/Компанії)**: Інформація про компанії, що розміщують вакансії.4. **employer_representatives (Представники роботодавців)**: Зв'язує користувачів з компаніями, дозволяючи одному користувачу представляти компанію.
- **job_positions (Вакансії)**: Детальна інформація про відкриті вакансії.
- **matches (Матчі/Взаємодії)**: Ключова таблиця, що фіксує взаємодії між кандидатами та вакансіями, включаючи згенеровані системою "матчі" та надіслані запрошення.

Довідникові таблиці (Look-up tables):

- **skills (Навички)**: id (UUID, PK), name (VARCHAR, UNIQUE, NOT NULL), category (VARCHAR, NULL).
- **job_titles (Посади)**: id (UUID, PK), title_de (VARCHAR, UNIQUE, NOT NULL), title_en (VARCHAR, NULL), category (VARCHAR, NULL).

- **departments (Департаменти/Відділи)**: id (UUID, PK), name_de (VARCHAR, UNIQUE, NOT NULL), name_en (VARCHAR, NULL).
- **industries (Галузі)**: id (UUID, PK), name_de (VARCHAR, UNIQUE, NOT NULL), name_en (VARCHAR, NULL).
- **work_types (Типи робіт)**: id (UUID, PK), name_de (VARCHAR, UNIQUE, NOT NULL), name_en (VARCHAR, NULL).

Таблиці зв'язків "багато-до-багатьох":

- **candidate_skills_junction**: candidate_id (FK), skill_id (FK), proficiency_level (ENUM('beginner', 'intermediate', 'advanced', 'expert'), NULL). (PK: candidate_id, skill_id).
- **candidate_job_titles_junction**: candidate_id (FK), job_title_id (FK). (PK: candidate_id, job_title_id).
- **candidate_departments_junction**: candidate_id (FK), department_id (FK). (PK: candidate_id, department_id).
- **candidate_work_types_junction**: candidate_id (FK), work_type_id (FK). (PK: candidate_id, work_type_id).
- **job_position_required_skills_junction**: job_position_id (FK), skill_id (FK), is_mandatory (BOOLEAN, DEFAULT TRUE). (PK: job_position_id, skill_id).
- **job_position_target_job_titles_junction**: job_position_id (FK), job_title_id (FK). (PK: job_position_id, job_title_id).
- **job_position_target_departments_junction**: job_position_id (FK), department_id (FK). (PK: job_position_id, department_id).
- **work_experiences (Досвід роботи кандидатів)**:
- **education_experiences (Освіта кандидатів)**:

- **chat_conversations (Розмови в чаті)**:
- **chat_participants (Учасники розмов)**:
- **chat_messages (Повідомлення в чаті)**:.

2.3 Реалізація клієнтської частини: Next.js, Apollo GraphQL

2.3.1 Створення, конфігурація та структура проєкту Next.js

Ініціалізація проєкту була здійснена за допомогою офіційного інструменту create-next-app, що дозволило швидко отримати налаштований скелет застосунку з усіма необхідними залежностями та конфігураціями. Команда для створення проєкту виглядала наступним чином [4]:
pnpm create next-app@latest pflegia-job-platform --typescript --eslint --tailwind --src-dir --app --import-alias "@/"*

Такий вибір параметрів забезпечив:

- **TypeScript:** Використання статичної типізації для підвищення надійності коду, покращення автодоповнення в IDE та полегшення рефакторингу, що є критично важливим для великих проєктів.
- **ESLint:** Інтеграція лінтера для автоматичного аналізу коду на відповідність стандартам кодування та виявлення потенційних помилок.
- **Tailwind CSS:** Використання утилітарного CSS-фреймворку для швидкого та гнучкого створення користувацьких інтерфейсів без написання великої кількості кастомного CSS.

- **Директорія src/:** Організація вихідного коду в окремій директорії src/ для кращої структуризації.
- **App Router:** Використання нового App Router, представленого в Next.js 13, що надає більш гнучкі та потужні можливості для маршрутизації, створення макетів (layouts) та рендерингу компонентів на сервері (Server Components).
- **Аліас для імпортів (@/*):** Налаштування абсолютних шляхів для імпортів, що починаються з @/, для спрощення навігації по проєкту та уникнення складних відносних шляхів (../../..).

Після створення проєкту, його **структура** в директорії src/ була організована наступним чином для забезпечення модульності та легкості підтримки:

- app/: Ключова директорія для App Router. Тут визначаються маршрути, сторінки, макети та серверні компоненти.
  - layout.tsx: Головний кореневий макет, що застосовується до всього застосунку. Він включає загальну структуру HTML (теги <html>, <body>), підключення глобальних стилів, провайдерів (наприклад, ApolloProvider, контекст теми).
  - page.tsx: Головна сторінка застосунку.
  - Групи маршрутів (Route Groups), наприклад, (auth)/login/page.tsx для сторінок автентифікації, (dashboard)/candidate/profile/page.tsx для сторінок особистого кабінету кандидата. Групи дозволяють організовувати файли без впливу на URL-шлях.
  - Динамічні маршрути, наприклад, jobs/[id]/page.tsx для сторінок окремих вакансій.

- loading.tsx та error.tsx: Спеціальні файли для відображення стану завантаження та обробки помилок на рівні маршрутів.
- api/: Опціональна директорія для створення Next.js API Routes, якщо потрібні прості бекенд-ендпоінти безпосередньо на фронтенді (наприклад, для проксі-запитів або обробки веб-хуків).

● components/: Містить перевикористовувані UI-компоненти, розбиті за функціональністю або типом (наприклад, ui/ для базових елементів як кнопки, інпути; features/ для більш складних компонентів, специфічних для певних функцій платформи).

● lib/: Допоміжні утиліти, константи, кастомні хуки React, функції для роботи з датами, валідації тощо.

● graphql/: Директорія для зберігання GraphQL-запитів (queries), мутацій (mutations) та підписок (subscriptions), визначених за допомогою gql тегу з @apollo/client. Це дозволяє централізовано управляти всіма GraphQL операціями.

● store/: (Опціонально) Якщо для управління певними аспектами глобального стану UI використовується бібліотека типу Zustand або Jotai, тут можуть знаходитися відповідні сховища (stores).

● styles/: Глобальні CSS-файли (наприклад, globals.css для Tailwind CSS директив та базових стилів).

● types/: Визначення глобальних TypeScript типів та інтерфейсів, що використовуються в усьому застосунку.

Конфігураційні файли, такі як tailwind.config.ts, postcss.config.js, next.config.mjs, tsconfig.json, були налаштовані відповідно до потреб проєкту для забезпечення коректної роботи Tailwind CSS, транспіляції TypeScript та інших аспектів збірки Next.js [4] [13].

2.3.2 Організація взаємодії з GraphQL API за допомогою Apollo Client

Для ефективної та типізованої взаємодії фронтенд-застосунку на Next.js з бекенд API, що надає GraphQL інтерфейс, було обрано бібліотеку **Apollo Client**. Вона є повнофункціональним клієнтом GraphQL, що надає потужні можливості для виконання запитів, управління кешем, обробки станів завантаження та помилок, а також підтримки оптимістичних оновлень та підписок [8] [9].

Конфігурація екземпляра Apollo Client:

Було створено централізований модуль для ініціалізації та конфігурації Apollo Client (наприклад, src/lib/apolloClient.ts).

```tsx
// src/app/ApolloWrapper.tsx (приклад для App Router)
"use client";
import { ApolloLink, HttpLink } from "@apollo/client";
import {
  ApolloNextAppProvider,
  NextSSRApolloClient,
  NextSSRInMemoryCache,
  SSRMultipartLink,
} from "@apollo/experimental-nextjs-app-support/ssr";

function makeClient() {
  const httpLink = new HttpLink({
      uri: process.env.NEXT_PUBLIC_GRAPHQL_ENDPOINT,
      fetchOptions: { cache: "no-store" },
  });
  // Логіка для authLink (як у попередньому прикладі)
  // const authLink = ...
```

Рис. 8. Конфігурація Apollo Client

Це дозволяє налаштувати URI GraphQL-ендпоінта. Розроблено стратегію кешування та механізми для автоматичного додавання JWT-токенів автентифікації до кожного запиту.

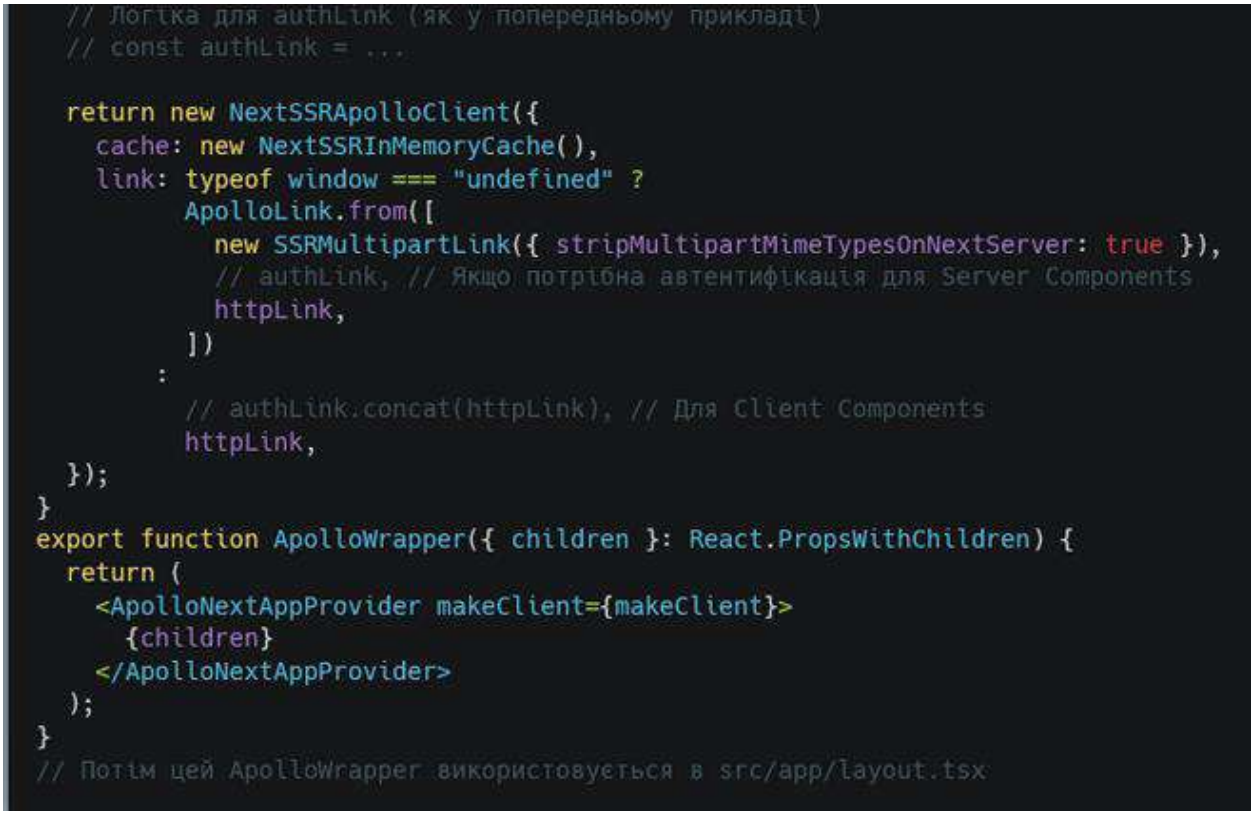

Рис. 9. Налаштування стратегії кешування та механізмів для автоматичного додавання JWT-токенів.

Важливою особливістю є налаштування authLink для автоматичного включення JWT-токенів. Токен зазвичай зберігається в localStorage після успішного входу користувача і витягується перед кожним запитом. Для Next.js App Router та Server Components механізм роботи з автентифікацією та передачею токенів може бути складнішим і вимагати використання Server Actions або передачі токенів через контекст на сервері.

Надання Apollo Client застосунку:

Для того, щоб компоненти мали доступ до Apollo Client, застосунок (або його частина) обгортається в ApolloProvider. У Next.js App Router це зазвичай

робиться в кореневому layout.tsx або в специфічних макетах для клієнтських компонентів.

Виконання GraphQL-операцій у компонентах:

У React-компонентах для виконання запитів (queries), змін (mutations) та підписок (subscriptions) було використано відповідні хуки, що надаються Apollo Client [10]:

useQuery: Для отримання даних. Він автоматично обробляє стани завантаження (loading), помилок (error) та отриманих даних (data). Також надає функції для повторного виконання запиту (refetch) та пагінації.

Приклад отримання профілю поточного користувача:

```
// src/graphql/queries/currentUser.ts
import { gql } from '@apollo/client';
export const GET_CURRENT_USER = gql`
  query GetCurrentUser {
    me {
      id
      email
      role
      candidateProfile { # Припускаючи, що є зв'язок
        firstName
        lastName
      }
    }
  }
`;
```

Рис. 10. Запит GET_CURRENT_USER

```tsx
// src/components/UserProfile.tsx
"use client"; // Якщо це клієнтський компонент
import { useQuery } from '@apollo/client'; // або з @apollo/experimental-nextjs-app-support/ssr для Server Components
import { GET_CURRENT_USER } from '@/graphql/queries/currentUser';

function UserProfile() {
  // Для Server Components виклик може бути через getClient().query(...)
  const { loading, error, data } = useQuery(GET_CURRENT_USER);

  if (loading) return <p>Завантаження профілю...</p>;
  if (error) return <p>Помилка завантаження: {error.message}</p>;
  if (!data?.me) return <p>Ви не авторизовані.</p>;

  return (
    <div>
      <h1>Вітаємо, {data.me.candidateProfile?.firstName || data.me.email}!</h1>
      <p>Ваша роль: {data.me.role}</p>
    </div>
  );
}
```

Рис. 11. Використання GET_CURRENT_USER для відображення профілю користувача

- **useMutation:** Для виконання операцій, що змінюють дані на сервері (створення, оновлення, видалення). Повертає функцію для виклику мутації та об'єкт зі станом операції. Дозволяє оновлювати кеш Apollo після успішного виконання мутації.
- **useSubscription:** Для отримання даних у реальному часі через WebSocket (наприклад, для нових повідомлень у чаті або миттєвих сповіщень).

2. Використання Apollo Client значно спростило управління станом, пов'язаним із серверними даними, забезпечило декларативний підхід до отримання даних та покращило загальну структуру коду фронтенд-застосунку.

2.5 Реалізація серверної частини: Бекенд API на Nest.js та GraphQL

Серверна частина платформи була розроблена як центральний компонент, що відповідає за бізнес-логіку, управління даними та надання API. Для цього було обрано фреймворк **Nest.js** (Node.js, TypeScript) з **GraphQL** (через **Apollo Server**) для API. Розгортання здійснюється на **AWS Fargate**.

2.5.1 Архітектура та структура проєкту Nest.js

Проєкт структуровано за модульним принципом Nest.js. Вхідна точка main.ts ініціалізує застосунок, а app.module.ts об'єднує функціональні модулі (наприклад, AuthModule, UsersModule, JobPositionsModule, MlModule).

Приклад функціонального модуля (JobPositionsModule):

```
// src/job-positions/job-positions.module.ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { JobPositionsService } from './job-positions.service';
import { JobPositionsResolver } from './job-positions.resolver';
import { JobPosition } from './entities/job-position.entity';

@Module({
  imports: [TypeOrmModule.forFeature([JobPosition])],
  providers: [JobPositionsResolver, JobPositionsService],
})
export class JobPositionsModule {}
```

Рис. 12. Приклад функціонального модуля (JobPositionsModule)

Сервіси (@Injectable()): Інкапсулюють бізнес-логіку.

```
// src/job-positions/job-positions.service.ts
@Injectable()
export class JobPositionsService {
  constructor(
    @InjectRepository(JobPosition)
    private readonly jobRepo: Repository<JobPosition>,
  ) {}
  async findOneById(id: string): Promise<JobPosition | null> {
    return this.jobRepo.findOneBy({ id });
  }
}
```

Рис. 13. Сервіс JobPositionService

GraphQL Резолвери (@Resolver()): Обробляють GraphQL-операції.

```
// src/job-positions/job-positions.resolver.ts
@Resolver(() => JobPosition)
export class JobPositionsResolver {
  constructor(private readonly service: JobPositionsService) {}

  @Query(() => JobPosition, { name: 'jobPosition', nullable: true })
  async findOne(@Args('id', { type: () => ID }) id: string) {
    return this.service.findOneById(id);
  }
}
```

Рис. 14. Graphql Resolver

DTO (Data Transfer Objects) з валідацією (class-validator):

```
// src/job-positions/dto/create-job-position.input.ts
@InputType()
export class CreateJobPositionInput {
  @Field() @IsNotEmpty() @MinLength(5)
  title: string;

  @Field() @IsNotEmpty() @IsUUID()
  employerId: string;
  // ... інші поля
}
```

Рис. 15. DTO CreateJobPositionInput

Глобальний ValidationPipe використовується в main.ts для автоматичної валідації.

2.5.2 Налаштування GraphQL з Apollo Server

Інтеграція GraphQL здійснюється через @nestjs/graphql та @nestjs/apollo.

1. Встановлення залежностей: pnpm add @nestjs/graphql @nestjs/apollo graphql apollo-server-express.

Конфігурація GraphQLModule в app.module.ts:

```
// Фрагмент з src/app.module.ts
GraphQLModule.forRoot<ApolloDriverConfig>({
  driver: ApolloDriver,
  autoSchemaFile: 'schema.gql', // Автогенерація схеми
  sortSchema: true,
  playground: process.env.NODE_ENV !== 'production', // Playground для розробки
  context: ({ req }) => ({ req }), // Передача запиту в контекст
  subscriptions: { 'graphql-ws': true }, // Підтримка WebSocket
}),
```

Рис. 16. Код генерації схеми, інтерактивний Playground та підтримку підписок

Ця конфігурація забезпечує генерацію схеми, інтерактивний Playground та підтримку підписок.

2.5.3 Реалізація автентифікації та авторизації

Безпека API реалізована за допомогою JWT та інтеграції з **Amazon Cognito**.

1. **Автентифікація через Cognito:** Клієнт отримує JWT від Cognito та надсилає його в заголовку Authorization: Bearer <token>.

Валідація JWT на Бекенді: Було створено JwtStrategy (використовуючи passport-jwt), що валідує токен за допомогою JWKS URI від

Cognito. JwtAuthGuard захищає GraphQL-операції.

```
// src/auth/guards/jwt-auth.guard.ts
@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {
  getRequest(context: ExecutionContext) {
    const ctx = GqlExecutionContext.create(context);
    return ctx.getContext().req;
  }
}
```

Рис. 17. Застосування: @UseGuards(JwtAuthGuard) до резолверів/мутацій.

Рольова авторизація: Реалізовано кастомний RolesGuard для перевірки ролей користувача з JWT.

2.5.4 Взаємодія з базою даних (TypeORM/Prisma)

Для взаємодії з базою даних PostgreSQL було використано ORM []. Якщо це **TypeORM**:

- Сутності (Entities) визначаються класами з декораторами (@Entity(), @Column(), @PrimaryGeneratedColumn(), @ManyToOne(), @ManyToMany() тощо).
- Репозиторії (Repositories) надають методи для CRUD-операцій та побудови складних запитів. Сервіси інжектують відповідні репозиторії.

Якщо це **Prisma**:

- Схема даних визначається у файлі schema.prisma.

- PrismaClient генерується на основі схеми та надає типізований API для взаємодії з базою даних. Сервіси інжектують PrismaService (кастомний сервіс, що інкапсулює PrismaClient).

2.5.5 Реалізація сервісу для взаємодії з SageMaker Endpoint

Як було показано у наданому тобою прикладі коду, я створив сервіс (наприклад, MlService або AppService), який інкапсулює логіку виклику SageMaker Endpoint.

Інжектування SageMakerRuntimeClient:
Клієнт AWS SDK для SageMaker Runtime інжектується в сервіс. Я налаштував його як провайдер у відповідному модулі, можливо, з використанням ConfigService для отримання регіону та облікових даних (хоча при розгортанні на AWS IAM ролі для Fargate/Lambda є кращим підходом для автентифікації) [15].

```
    // src/ml/ml.module.ts
import { Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { SageMakerRuntimeClient } from '@aws-sdk/client-sagemaker-runtime';
import { MlService } from './ml.service';

@Module({
  imports: [ConfigModule], // Для доступу до конфігурацій
  providers: [
    MlService,
    {
      provide: SageMakerRuntimeClient,
      useFactory: (configService: ConfigService) => {
        return new SageMakerRuntimeClient({
          region: configService.get<string>('AWS_REGION'), // Змінна середовища
          // Credentials тут не потрібні при використанні IAM ролей на Fargate/Lambda
        });
      },
      inject: [ConfigService],
    },
  ],
  exports: [MlService],
})
export class MlModule {}
```

Рис. 18. Підключення клієнта Sagemaker.

- **Метод predict:**

  Метод, аналогічний твоєму прикладу, приймає масив ознак, валідує їх, готує дані у форматі CSV, викликає InvokeEndpointCommand та трансформує відповідь.

  EndpointName, NumberOfFeatures, MaxPayloadSize отримуються з ConfigService для гнучкості конфігурації.

2.6 Реалізація модуля машинного навчання (AWS SageMaker Canvas)

Інтелектуальне ядро платформи, відповідальне за надання персоналізованих рекомендацій та прогнозів (наприклад, ймовірності прийняття запрошення кандидатом), було реалізовано з використанням сервісів Amazon Web Services, зокрема **AWS SageMaker Canvas**. Цей інструмент дозволяє створювати моделі машинного навчання за допомогою візуального інтерфейсу, що значно прискорює процес прототипування та розгортання [14].

2.6.1 Підготовка та завантаження даних для SageMaker Canvas

Основою для будь-якої ML-моделі є якісні дані. У рамках проєкту, дані для тренування моделей SageMaker Canvas готувалися наступним чином:

1. **Джерела даних:** Основними джерелами даних слугували операційна база даних **Amazon RDS for PostgreSQL** та логи активності користувачів, що накопичувалися в **Amazon S3 Data Lake**.
2. **ETL-процеси з AWS Glue:** Для вилучення, трансформації та завантаження даних з RDS та інших джерел до S3 Data Lake у форматі, придатному для аналізу, було налаштовано завдання **AWS Glue**.
3. **Використання Amazon Athena:** Для додаткового аналізу та вибірки даних з S3 Data Lake було використано **Amazon Athena**, що дозволяло виконувати SQL-запити та створювати специфічні вітрини даних.

4. **Імпорт даних в SageMaker Canvas:** Підготовлені датасети (у форматі CSV або Parquet) з S3 імпортувалися безпосередньо в інтерфейс SageMaker Canvas.

2.6.2 Побудова та тренування моделі в SageMaker Canvas

SageMaker Canvas надає інтуїтивно зрозумілий візуальний інтерфейс для побудови моделей машинного навчання.

1. **Вибір типу моделі:** Для задачі прогнозування прийняття запрошення було обрано тип моделі "Прогноз (2 категорії)" (бінарна класифікація), з цільовою колонкою match_connection_status_accepted.
2. **Автоматична підготовка даних в Canvas:** Після завантаження датасету, Canvas проводить його аналіз та базову підготовку.
3. **Тренування моделі (AutoML):** SageMaker Canvas автоматично проводить процес тренування, тестуючи різні алгоритми та оптимізуючи гіперпараметри для вибору найкращої моделі. Наприклад, для моделі Model_Invite_Acceptance_Predict (версія 56), як показано на скріншотах, було обрано "Standard build" для отримання кращої точності.

2.6.3 Аналіз моделі в SageMaker Canvas

Після завершення тренування, SageMaker Canvas надає детальний інтерфейс для аналізу продуктивності та характеристик побудованої моделі.

- **Огляд продуктивності (Overview Tab):**

На вкладці "Overview", як видно на **Рис. 19**, представлена загальна інформація про модель. Ключовими показниками є **Accuracy**

(Точність) 83.762% та **F1-score 0.846**. Пояснення під метрикою Accuracy вказує, що "модель правильно прогнозує значення Match_connection_status_accepted у 83.762% випадків". Це дає загальне уявлення про якість моделі. Також на цій вкладці доступний аналіз впливу ознак (Column impact).

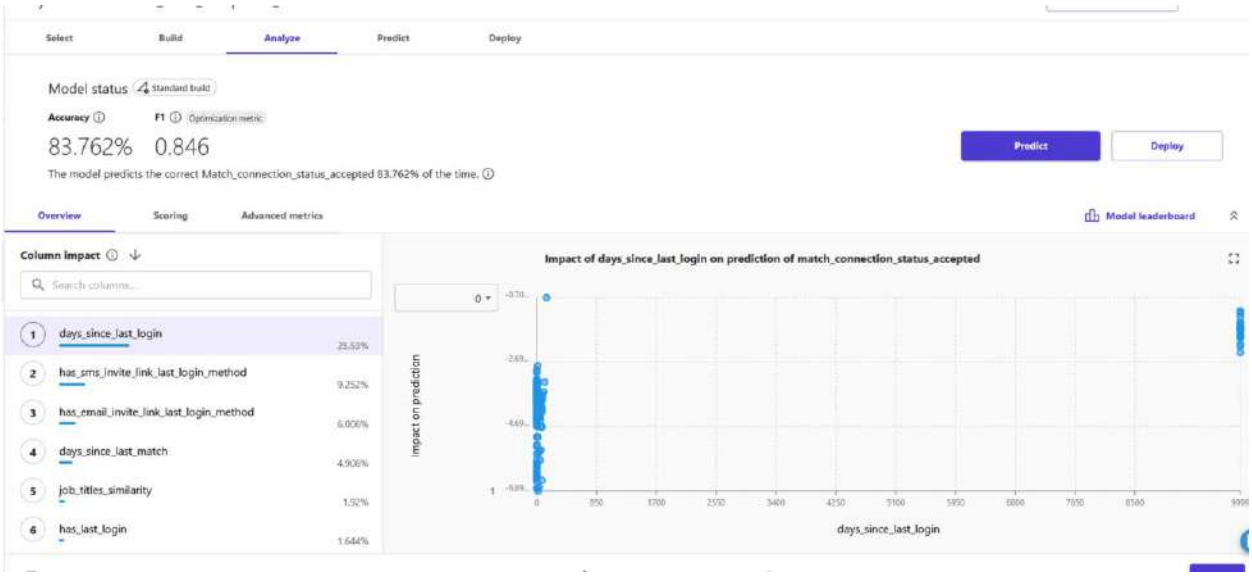

Рис. 19. – Вкладка "Overview" аналізу моделі в SageMaker Canvas

Найбільш впливовими ознаками для даної моделі виявилися:

- days_since_last_login (кількість днів з моменту останнього входу кандидата) – вплив 25.53%.
- has_sms_invite_link_last_login_method (чи був останній вхід через SMS-запрошення) – вплив 9.252%.
- has_email_invite_link_last_login_method (чи був останній вхід через email-запрошення) – вплив 6.006%.

  Цей аналіз допомагає зрозуміти, які фактори є найважливішими

для прогнозування. Графік "Impact on prediction" візуалізує, як зміна значення ознаки days_since_last_login впливає на прогноз.

- **Оцінка (Scoring Tab):**

Вкладка "Scoring", зображена на **Рис. 20**, надає візуалізацію "Predicted vs. Actual" (Прогнозовані та Фактичні значення) у вигляді діаграми потоків (Sankey diagram). Для загальної кількості у 10000 прогнозів, діаграма показує, як розподілилися правильні та неправильні класифікації для обох класів (0 та 1). Наприклад, для класу 0 (неприйняття запрошення) модель правильно спрогнозувала 5018 випадків, а для класу 1 (прийняття запрошення) – 4882 випадки. Також надається інформація про те, що "якщо модель прогнозує 0, вона є правильною у 81.075% випадків" та "для значень, що є 0 у датасеті, модель спрогнозувала 90.155% з них як 0".

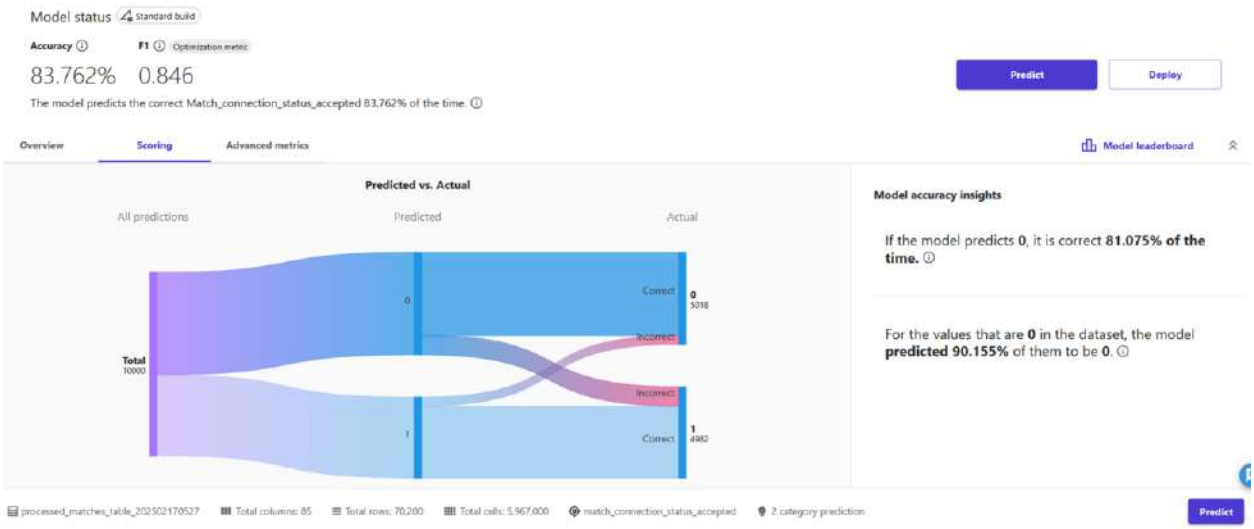

Рис.20. – Вкладка "Scoring" аналізу моделі в SageMaker Canvas

- **Розширені метрики (Advanced metrics Tab):**

На вкладці "Advanced metrics", як показано на **Рис. 21**, представлені детальні метрики класифікації для кожного класу та загалом. Для

позитивного класу (припустимо, це клас "1" - прийняття запрошення) бачимо:

- **F1-score:** 85.375% (що є цільовою метрикою оптимізації для цієї моделі, як позначено "Optimization metric").
- **Accuracy:** 84.5%
- **Precision:** 81.075% (точність позитивних прогнозів).
- **Recall:** 90.155% (повнота, або наскільки добре модель знаходить усі фактичні позитивні випадки).
- **AUC-ROC:** 0.919 (дуже хороший показник здатності моделі розрізняти класи).

  Таблиця "Metrics table" надає числові значення цих метрик.

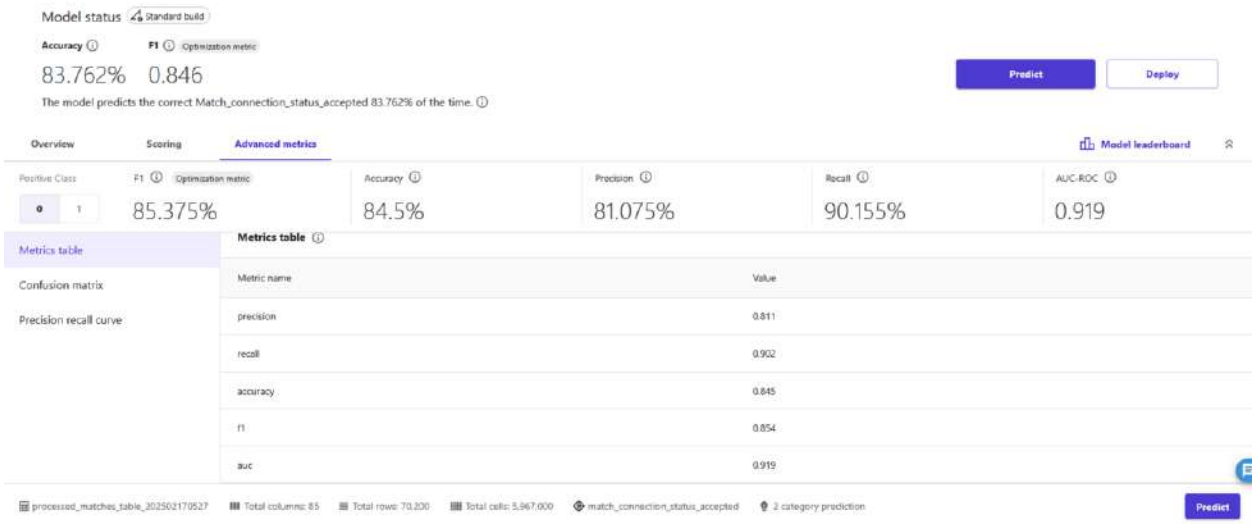

Рис. 21. – "Advanced metrics" аналізу моделі в SageMaker Canvas з детальними метриками

- **Лідерборд моделей (Model leaderboard):**

SageMaker Canvas під час "Standard build" автоматично тренує та оцінює декілька варіантів моделей. **Рисунок 14** демонструє "Model

leaderboard", де можна порівняти продуктивність різних моделей за ключовими метриками (F1, Accuracy, AUC, Precision, Recall, Log Loss, Inference latency). Наприклад, модель FULL-t123... (Default model) має F1 84.640%, тоді як інші моделі, такі як FULL-n423... або FULL-h823..., показують схожі, але трохи відмінні результати. Це дозволяє обрати модель, яка найкраще відповідає бізнес-вимогам (наприклад, баланс між точністю та швидкістю інференсу).

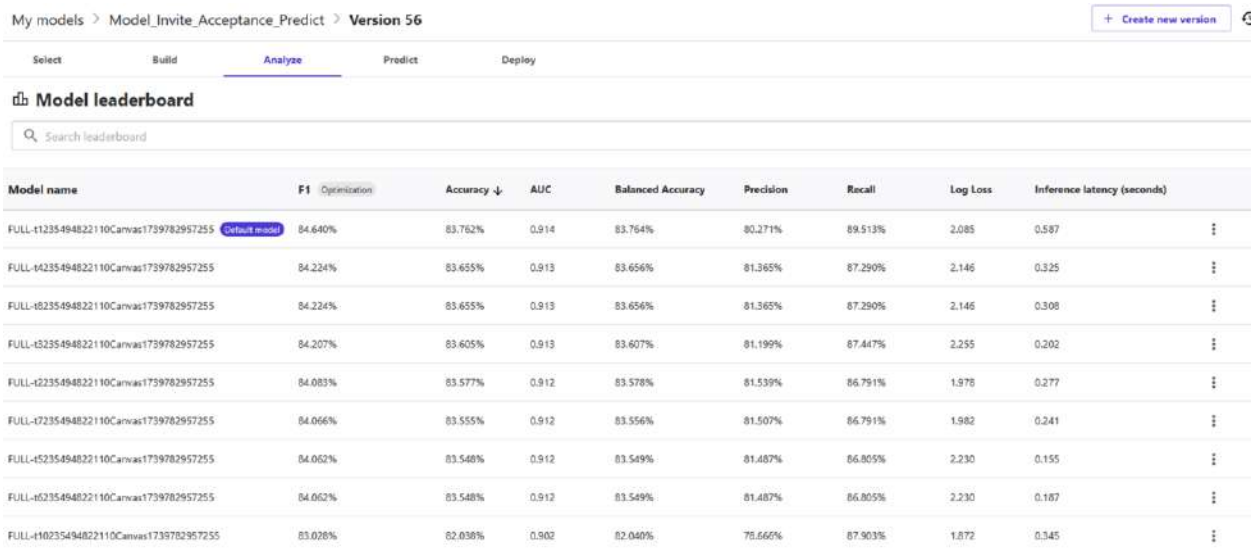

Рис. 22. – Скріншот "Model leaderboard" в SageMaker Canvas, що показує порівняння різних тренованих моделей

Глибокий аналіз, що надається SageMaker Canvas, дозволяє не тільки оцінити якість побудованої моделі, але й отримати цінні інсайти для подальшого вдосконалення як самої моделі, так і бізнес-процесів платформи.

2.6.4 Розгортання моделі та інтеграція з бекендом

Після успішного тренування та аналізу, модель була розгорнута для використання в реальному часі.

1. **Розгортання на SageMaker Endpoint:** З інтерфейсу SageMaker Canvas найкращу версію моделі (наприклад, "Default model" з лідерборду) було розгорнуто на керований **SageMaker Endpoint**. Цей процес автоматично створює необхідну інфраструктуру та налаштовує автомасштабування.
2. **Версіонування моделей:** Всі значущі ітерації моделей, створених у Canvas, версіонуються та можуть бути зареєстровані в **SageMaker Model Registry** для централізованого управління.
3. **Взаємодія з бекенд-сервісом:** Бекенд-сервіс на Nest.js (детально описаний у 2.5.5) використовує AWS SDK (SageMakerRuntimeClient) для надсилання запитів до цього SageMaker Endpoint. Він передає масив ознак для прогнозування та отримує у відповідь ймовірності прийняття запрошення.

Цей підхід з використанням SageMaker Canvas дозволив швидко реалізувати та інтегрувати функціональну ML-модель в платформу.

2.7 Висновки до Розділу 2

У другому розділі було детально розглянуто процес проєктування та програмної реалізації ключових компонентів багаторівневого веб-застосунку для платформи пошуку роботи.

Було сформульовано **технічне завдання**, що визначило функціональне призначення системи та основні вимоги до її архітектури, клієнтську та серверну частини, а також до інтегрованого модуля машинного навчання та пов'язаних з ним MLOps процесів.

Розроблено та описано **структуру бази даних** на основі PostgreSQL з урахуванням специфіки предметної області, вимог до масштабованості та продуктивності. Було детально розглянуто основні сутності, їх атрибути та зв'язки, а також обґрунтовано стратегії оптимізації, індексації та забезпечення цілісності даних.

Описано процес **налаштування середовища розробки**, включаючи управління версіями Node.js (nvm), використання пакетного менеджера pnpm, налаштування Docker Desktop [3] для локальної розробки та конфігурацію середовища для роботи з AWS SageMaker.

Представлено реалізацію **клієнтської частини (фронтенду)** на базі фреймворку Next.js, включаючи структуру проєкту, налаштування взаємодії з GraphQL API за допомогою Apollo Client.

Детально розглянуто реалізацію **серверної частини (бекенд API)** на фреймворку Nest.js. Описано архітектуру проєкту, ключові елементи Nest.js (модулі, сервіси, резолвери, DTO, guards), налаштування GraphQL з Apollo Server, реалізацію автентифікації та авторизації з Amazon Cognito, взаємодію з базою даних через ORM, а також створення спеціалізованого сервісу для взаємодії з SageMaker Endpoint.

Було детально описано процес реалізації **модуля машинного навчання** з використанням AWS SageMaker Canvas. Це включало етапи підготовки та завантаження даних (з використанням AWS Glue, S3, Athena), побудови, тренування та глибокого аналізу моделі бінарної класифікації в інтерфейсі Canvas, а також її розгортання на SageMaker Endpoint для

інтеграції з бекенд-системою. Аналіз результатів моделі, представлених на скріншотах з SageMaker Canvas, підтвердив її прогностичну здатність.

Розділ 3. Автоматизоване розгортання, CI/CD

Автоматизація процесів неперервної інтеграції (CI) та неперервного розгортання (CD) є критично важливою для сучасної розробки програмного забезпечення. Вона дозволяє командам зосередитися на написанні якісного коду та швидкому впровадженні нових функцій, мінімізуючи ручну працю, пов'язану з оновленням серверів, тестуванням та моніторингом. У даному проєкті було реалізовано комплексний CI/CD пайплайн з використанням **GitHub, GitHub Actions, Docker, Amazon Web Services (AWS)** [14] [16] для бекенду та інфраструктури машинного навчання, **Vercel** для фронтенду, а також інструментів для тестування та моніторингу, таких як **Playwright** [17] та **Sentry** [18].

Загальна схема CI/CD передбачає, що розробка ведеться в Git-репозиторії на GitHub. При кожному пуші змін до основних гілок або при створенні pull-request автоматично запускаються робочі процеси **GitHub Actions**. Ці процеси відповідають за збірку коду, статичний аналіз, запуск тестів та розгортання оновлених компонентів системи на відповідні платформи.

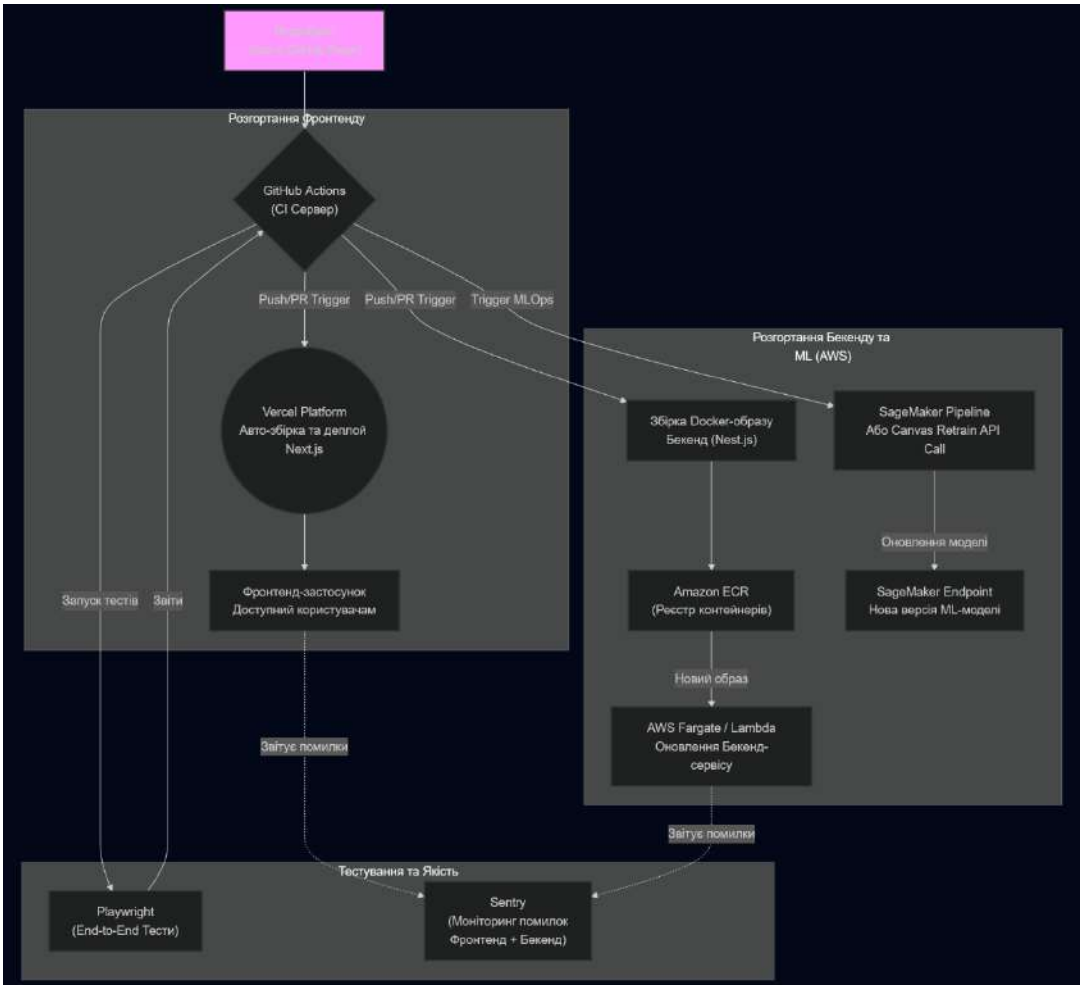

Рис. 23. Загальна схема CI/CD пайплайну: GitHub -> GitHub Actions (збірка, тести) -> Vercel (для фронтенду), AWS (Fargate/Lambda для бекенду, SageMaker для ML

3.1 CI/CD інтеграція для бекенд API (Nest.js) та ML-модуля (SageMaker)

Для серверної частини, розробленої на Nest.js, та модуля машинного навчання на AWS SageMaker, процеси CI/CD були побудовані з використанням Docker та GitHub Actions для розгортання на AWS Fargate (або Lambda) та управління MLOps пайплайнами.

3.1.1 Використання Docker для контейнеризації бекенду

Для забезпечення консистентності середовища розробки, тестування та продакшену, бекенд-застосунок на Nest.js було запаковано в Docker-контейнер. Було створено Dockerfile, що описує кроки для побудови образу, включаючи встановлення залежностей (за допомогою pnpm), компіляцію TypeScript коду та підготовку застосунку до запуску. Використання багатоетапної збірки (multistage build) дозволило мінімізувати розмір фінального образу.

3.1.2 Розгортання бекенду на AWS Fargate за допомогою GitHub Actions

Робочі процеси GitHub Actions були налаштовані для автоматичного розгортання Docker-образів бекенду на AWS Fargate.

Тригери: Запуск при пуші в гілку master або main, або при створенні релізу.

Кроки workflow:

1. Checkout коду.
2. Автентифікація в AWS (зазвичай через OIDC з IAM роллю).
3. Побудова Docker-образу за допомогою Dockerfile.
4. Завантаження образу в Amazon ECR (Elastic Container Registry).
5. Оновлення сервісу AWS Fargate (або ECS Task Definition та Service) для використання нового образу.

На **Рис. 17** показано приклад успішного виконання робочих процесів GitHub Actions, що відповідають за розгортання на AWS.

Видно різні запуски, такі як feat: sentry або fix: updatedAt, які були розгорнуті на хмарну платформу.

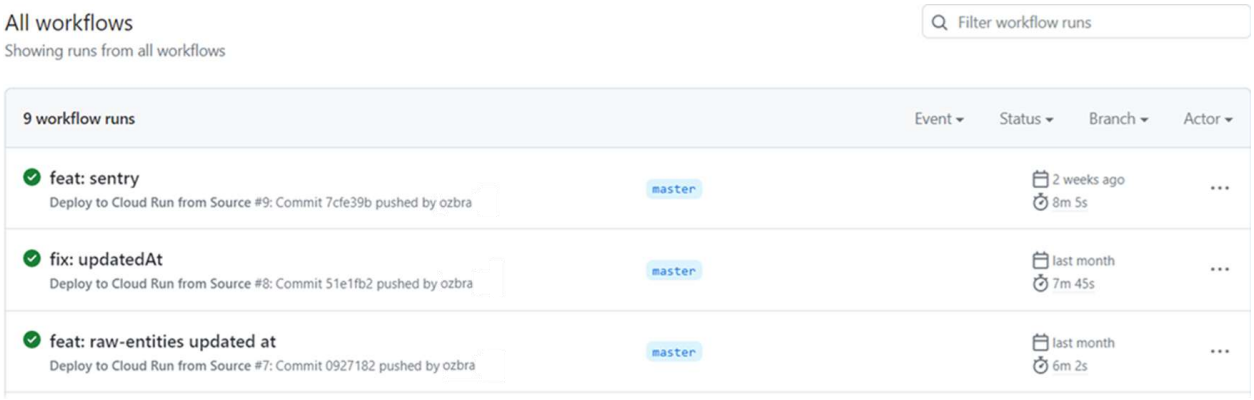

Рис. 17. – Списком виконаних workflow runs з GitHub Actions

3.1.3 CI/CD для MLOps з AWS SageMaker Canvas та SageMaker Pipelines

Для модуля машинного навчання, що використовує SageMaker Canvas, процеси MLOps також автоматизуються через GitHub Actions.

Тригери: Зміни в скриптах підготовки даних (Glue ETL), конфігураціях моделей, або за розкладом (через cron в GitHub Actions).

Кроки workflow:

1. Checkout коду та конфігурацій.
2. Автентифікація в AWS.
3. Запуск ETL-завдань AWS Glue для оновлення датасетів в S3/Athena.
4. Ініціація процесу перетренування або оновлення моделі в SageMaker Canvas через AWS SDK/CLI.

5. Або запуск SageMaker Pipeline (якщо використовується більш кастомний MLOps), який оркеструє кроки підготовки даних, тренування, валідації та реєстрації моделі в SageMaker Model Registry.
6. Після схвалення нової версії моделі, може запускатися окремий workflow для оновлення SageMaker Endpoint або Lambda-функції інференсу.

3.2 CI/CD інтеграція для клієнтської частини (Next.js на Vercel)

Для фронтенд-застосунку, розробленого на Next.js, було використано нативну інтеграцію платформи Vercel з GitHub.

3.2.1 Автоматичне розгортання на Vercel

Vercel надає безшовну інтеграцію з GitHub репозиторіями.

- **Підключення репозиторію:** Репозиторій проєкту на GitHub було підключено до Vercel.
- **Автоматичні деплої:** При кожному пуші в основну гілку (наприклад, master або main) Vercel автоматично запускає процес збірки та розгортання застосунку. Також Vercel створює прев'ю-деплої для кожного pull-request, що дозволяє тестувати зміни в ізольованому середовищі перед їх злиттям.

На **Рис. 18** показано інтерфейс Vercel з інформацією про успішне розгортання клієнтського застосунку. Видно статус "Ready",

середовище "Production", тривалість розгортання, активні домени (наприклад, data-shape-app.vercel.app).

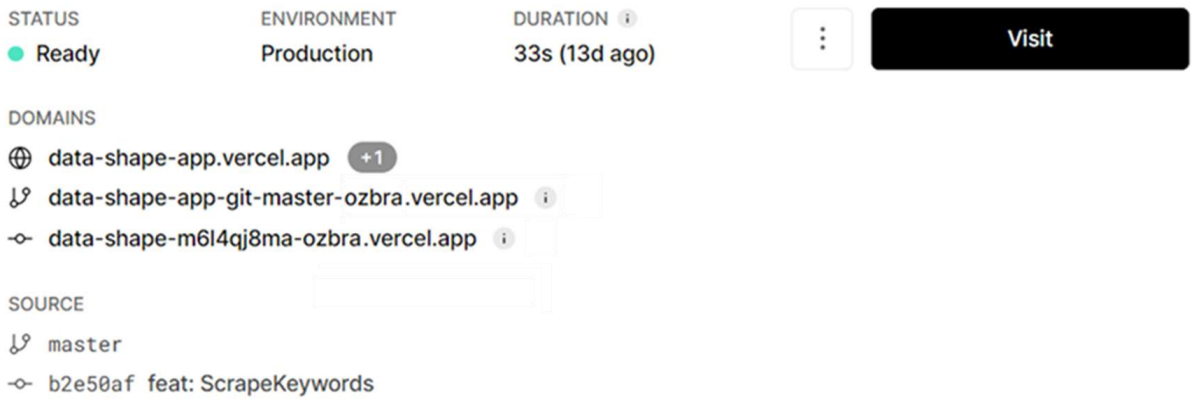

Рис. 18. – Скріншот з інтерфейсу Vercel, що показує деталі розгортання

3.3 Забезпечення якості: Автоматичне тестування та моніторинг помилок

Для підтримки високої якості програмного продукту було впроваджено автоматичне тестування та систему моніторингу помилок.

3.3.1 Автоматичне end-to-end тестування з Playwright

Для перевірки коректності роботи ключових сценаріїв взаємодії користувача з фронтенд-застосунком було налаштовано автоматичне end-to-end тестування за допомогою фреймворку **Playwright** [17].

- **Інтеграція з CI/CD:** Тести Playwright інтегровані в робочий процес GitHub Actions (визначений у файлі, наприклад, playwright.yml), який запускається автоматично при кожному pull-request або пуші в основні гілки.

- **Звітування:** Після виконання тестів генерується детальний звіт, який зберігається як артефакт робочого процесу.

На **Рис. 19** показано приклад результатів запуску Playwright тестів у GitHub Actions. Видно, що робочий процес playwright.yml (спрацював на подію pull_request) успішно завершив крок build за 3 хвилини 7 секунд. У секції "Artifacts" доступний для завантаження звіт playwright-report розміром 412 KB. Ці звіти містять інформацію про пройдені/невдалі тести, скріншоти та відеозаписи виконання, що значно допомагає в діагностиці проблем.

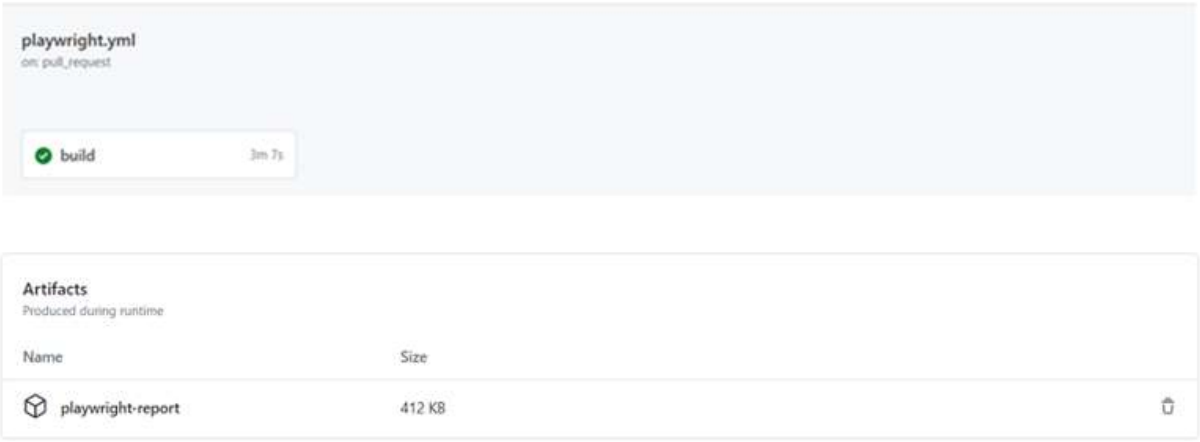

Рис. 19. – Результати запуску Playwright тестів у GitHub Actions

3.3.2 Відстежування помилок за допомогою Sentry

Для проактивного моніторингу та аналізу помилок, що виникають у роботі як фронтенд, так і бекенд частин застосунку, було інтегровано сервіс **Sentry**.

- **Інтеграція:** Sentry SDK було додано до Next.js (фронтенд) та Nest.js (бекенд) проєктів.

- **Збір помилок:** Sentry автоматично перехоплює необроблені виключення, помилки JavaScript, HTTP-помилки та інші аномалії.
- **Аналіз та сповіщення:** Sentry надає детальний дашборд для аналізу помилок, включаючи стек викликів, контекст виникнення, інформацію про браузер/сервер користувача та частоту повторень. Налаштовано сповіщення для команди розробників про критичні помилки.

На **Рис. 20.** продемонстровано інтерфейс Sentry зі списком зафіксованих проблем (Issues) для проєктів data-shape-api та data-shape-app. Видно приклади помилок, таких як QueryFailedError (пов'язана з PostgreSQL) або AxiosError (мережева помилка), кількість їх виникнень та час останньої появи. Це дозволяє оперативно реагувати на проблеми та покращувати стабільність системи.

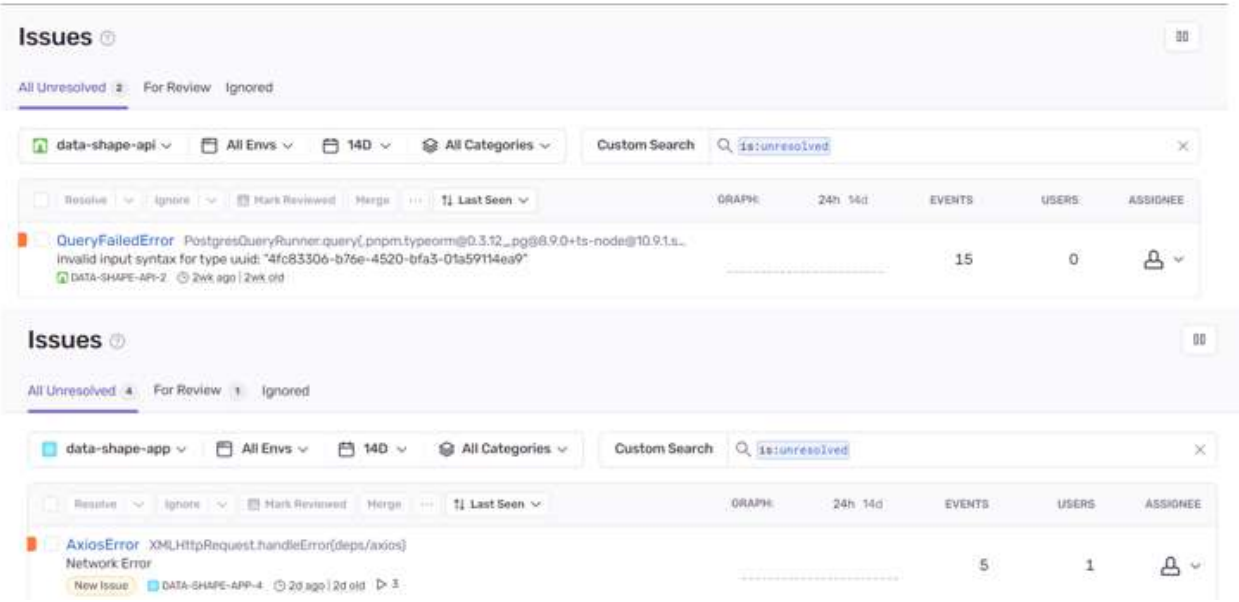

Рис. 20 – Інтерфейс Sentry, що показує список відстежуваних помилок

3.4 Висновки до розділу 3

У третьому розділі було детально розглянуто процеси автоматизованого розгортання, неперервної інтеграції та забезпечення якості для розробленого багаторівневого веб-застосунку. Було описано використання GitHub Actions для автоматизації збірки та розгортання бекенд-компонентів на AWS Fargate та управління MLOps-пайплайнами для SageMaker. Продемонстровано інтеграцію з Vercel для автоматичного розгортання фронтенд-застосунку на Next.js.

Особливу увагу було приділено аспектам забезпечення якості: впроваджено автоматичне end-to-end тестування за допомогою Playwright, інтегроване в CI/CD потік, та систему моніторингу й відстежування помилок Sentry для фронтенду та бекенду. Ці заходи дозволяють підтримувати високий рівень стабільності та надійності платформи, оперативно реагувати на проблеми та забезпечувати якісний користувацький досвід. Налаштований CI/CD

пайплайн з автоматизованим тестуванням та моніторингом є важливим кроком до побудови ефективної та сучасної системи розробки.

Висновки

У рамках даної магістерської роботи було здійснено комплексне дослідження, проєктування та теоретичне обґрунтування архітектури багаторівневого веб-застосунку для інтелектуальної платформи пошуку роботи. Вибір технологічного стеку, що базується на Next.js для розробки клієнтського інтерфейсу, Nest.js з GraphQL для реалізації серверної частини API, та PostgreSQL з розширенням pgvector для операційного зберігання даних, у поєднанні з екосистемою хмарних сервісів Amazon Web Services (AWS), забезпечує міцний фундамент для створення високопродуктивної, відмовостійкої та масштабованої системи. Застосування зазначених технологій дозволяє оптимізувати цикл розробки, забезпечити архітектурну гнучкість та мінімізувати ризики, пов'язані з розробкою складних розподілених систем.

У ході дослідження було детально проаналізовано та визначено ключові архітектурні патерни та етапи розробки компонентів застосунку. Акцент було зроблено на забезпеченні ефективної та безпечної взаємодії між клієнтською частиною, сервером додатків та модулем машинного навчання. Розглянуто стратегії управління великими обсягами даних, зокрема використання Amazon S3 для реалізації концепції Data Lake та інтеграцію Amazon Redshift як аналітичного сховища даних (Data Warehouse), що є критично важливим для агрегації, аналізу та підготовки даних для моделей

машинного навчання. Особливу увагу приділено інтеграції модуля машинного навчання на платформі AWS SageMaker, включаючи етапи підготовки даних за допомогою AWS Data Wrangler, розробку та тренування моделей бінарної класифікації (з потенційним використанням інструментів AutoML, таких як AutoGluon), та їх операціоналізацію через SageMaker Endpoints або безсерверні функції AWS Lambda.

Важливою складовою роботи стало проєктування та опис процесів автоматизації розгортання та MLOps (Machine Learning Operations). Використання інструментів CI/CD, таких як GitHub Actions та AWS CodePipeline, у поєднанні з технологією контейнеризації Docker та підходами Infrastructure as Code Terraform, дозволяє стандартизувати та автоматизувати процеси збірки, тестування та розгортання. Для управління життєвим циклом ML-моделей запропоновано використання SageMaker Pipelines та SageMaker Model Registry. Інтеграція систем комплексного моніторингу (Amazon CloudWatch, Sentry) є невід'ємною частиною забезпечення стабільності та надійності системи в експлуатації.

Аспекти безпеки були розглянуті через призму використання сервісів Amazon Cognito для управління ідентифікацією та доступом користувачів, AWS IAM для гранулярного контролю доступу до ресурсів AWS, а також AWS KMS та AWS Macie для забезпечення конфіденційності та цілісності даних.

Таким чином, проведена робота демонструє комплексний підхід до створення сучасного багаторівневого веб-застосунку. На основі системного аналізу предметної області, обґрунтованого вибору передових технологій та хмарних сервісів AWS, а також глибокого опрацювання інженерних аспектів

розробки, MLOps та безпеки, було сформовано концептуальну модель та описано методологію реалізації надійного, високопродуктивного та масштабованого програмного рішення. Запропоновані архітектурні та технологічні рішення створюють передумови для успішної практичної імплементації та подальшого розвитку платформи пошуку роботи з розширеними інтелектуальними можливостями.

Список використаної літератури та електронних ресурсів

1. Node.js Documentation. [Електронний ресурс]. Available: https://nodejs.org/en/docs/

2. TypeScript Documentation. [Електронний ресурс]. Available: https://www.typescriptlang.org/docs/

3. Docker Documentation. [Електронний ресурс]. Available: https://docs.docker.com/

4. Next.js Documentation. [Електронний ресурс]. Available: https://nextjs.org/docs
5. Angular Documentation: [Електронний ресурс]. Avaialble: https://angular.dev/
6. Vue documentation. [Електронний ресурс]. Available: https://vuejs.org/
7. React Documentation. [Електронний ресурс]. Available: https://react.dev/

8. Tailwind CSS Documentation. [Електронний ресурс]. Available: https://tailwindcss.com/docs

9. Apollo Client Documentation. [Електронний ресурс]. Available: https://www.apollographql.com/docs/react/

10. GraphQL Foundation. Introduction to GraphQL. [Електронний ресурс]. Available: https://graphql.org/learn/

11. NestJS Documentation. [Електронний ресурс]. Available: https://docs.nestjs.com/

12. PostgreSQL Documentation. [Електронний ресурс]. Available: https://www.postgresql.org/docs/

13. Amazon Web Services (AWS) Documentation. [Електронний ресурс]. Available: https://docs.aws.amazon.com/

14. Amazon SageMaker Developer Guide. [Електронний ресурс]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html

15. AWS Lambda Developer Guide. [Електронний ресурс]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

16. GitHub Actions Documentation. [Електронний ресурс]. Available: https://docs.github.com/en/actions

17. Playwright Documentation. [Електронний ресурс]. Available: https://playwright.dev/docs/intro

18. Sentry documantation. [Елктроний ресурс]. Available: https://sentry.io/welcome/