

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПОШУКУ І
ОЦІНЮВАННЯ КОМПАНІЙ-ПІДРЯДНИКІВ
DEVELOPMENT OF A WEB APPLICATION FOR
SEARCHING AND RATING CONTRACTOR COMPANIES

Виконав: студент 4 курсу, групи КНТ-121сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ГАНСЬКИЙ Д.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ГАНСЬКОГО Данила Анатолійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку для пошуку і оцінювання
компаній-підрядників. Development of a Web Application for Searching and
Rating Contractor Companies

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “24” квітня 2024 року
№ 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно
розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та
експериментальне дослідження програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень,
кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т. В., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М. В., асистент		

7. Дата видачі завдання « 24 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент

_____ Данило ГАНСЬКИЙ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна ФЕДОРОНЧАК
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
91 с., 4 табл., 45 рис., 3 дод., 23 джерела.

JAVASCRIPT, REACT, GO, ELASTICSEARCH, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ПРИБЛИЗНИЙ (НЕЧІТКИЙ) ПОШУК, ПРОСТОРОВИЙ ПОШУК, САЙТ, ПІДРЯДНИК, ВІДГУК, РЕЙТИНГ.

Об'єкт дослідження – програмні засоби для пошуку компаній-підрядників.

Предмет дослідження – комп'ютерні програми для приблизного (нечіткого) пошуку оголошень підрядників за ключовими словами і місцезнаходженням.

Мета роботи – створення програмної системи для зручного і швидкого пошуку підрядників з урахуванням рейтингу підрядників і місцезнаходження користувача.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мови програмування JavaScript та Go (Golang), бази даних PostgreSQL та Elasticsearch, технологія контейнеризації Docker.

Результати. Створено програмний застосунок з інтуїтивним та простим для використання інтерфейсом для швидкого пошуку компаній-підрядників за ключовими словами та місцезнаходженням з урахуванням оцінок від користувачів.

Висновки. Розроблено програмну систему для швидкого і зручного пошуку та оцінювання компаній-підрядників, що дозволить користувачам знаходити кращих спеціалістів, а бізнесу – зручну платформу для поширення інформації про свої послуги.

Галузь використання – сфери послуг.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 91 pages, 4 tables, 45 figures, 3 appendixes, 23 sources.

JAVASCRIPT, REACT, GO, ELASTICSEARCH, DATABASE, WEB APPLICATION, APPROXIMATE (FUZZY) SEARCH, SPATIAL SEARCH, SITE, CONTRACTOR, REVIEW, RATING.

The object of the research is software tools for finding contractor companies.

The subject of the study is computer programs for approximate (fuzzy) search of contractor advertisements by keywords and location.

The purpose of the work is to create a software system for a convenient and quick search for contractors, taking into account the contractors' ratings and the user's location.

Materials, methods, and tools: object-oriented programming, JavaScript and Go (Golang) programming languages, PostgreSQL and Elasticsearch databases, Docker containerization technology.

Results. Created a software application with an intuitive and easy-to-use interface to quickly search for contractors by keywords and location, taking into account the contractors' ratings.

Conclusions. A software system has been developed for quick and convenient search and rating of contractor companies, which will allow users to find the best specialists, and businesses - a convenient platform for spreading information about their services.

The field of use is the service sector.

ЗМІСТ

С.

Перелік скорочень та умовних позначок.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Огляд основних термінів і понять предметної області.....	10
1.2 Огляд існуючих програмних продуктів для пошуку та оцінювання компаній-підрядників.....	11
1.3 Постановка завдання роботи.....	18
1.4 Висновки за розділом 1.....	18
2 Опис програми.....	19
2.1 Обґрунтування вибору мов програмування.....	19
2.2 Обґрунтування вибору супровідних технологій.....	23
2.3 Обґрунтування вибору середовища розробки.....	26
2.4 Загальний опис архітектури основних компонентів системи.....	27
2.5 Висновки за розділом 2.....	31
3 Розробка програми.....	32
3.1 Розробка серверних компонентів системи.....	32
3.2 Розробка та оформлення вебінтерфейсу користувача.....	42
3.3 Файлова структура проєкту.....	45
3.4 Висновки за розділом 3.....	48
4 Експлуатація, тестування та експериментальне дослідження програми.....	49
4.1 Опис застосування.....	49
4.2 Умови виконання програми.....	49
4.3 Інструкція по експлуатації програми.....	50
4.4 Тестування програмного забезпечення.....	61
4.5 Висновки за розділом 4.....	62
Висновки.....	63
Перелік джерел посилань.....	64
Додаток А Технічне завдання.....	66
Додаток Б Текст програми.....	70
Додаток В Слайди презентації.....	83

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ACID – Atomicity, Consistency, Isolation, Durability;

AJAX – Asynchronous JavaScript and XML;

AI – Artificial Intelligence;

API – Application Programming Interface;

DAL – Data Access Layer;

GC – Garbage Collector;

IDE – Integrated Development Environment;

JS – JavaScript;

JSON – JavaScript Object Notation;

ML – Machine Learning;

ORM – Object–relational mapping;

OS – Operating System;

REST – Representational State Transfer;

SEO – Search engine optimization;

SOAP – Simple Object Access Protocol;

SPA – Single Page Application;

SQL – Structured Query Language;

TS – TypeScript;

UUID – Universal Unique Identifier;

VM – Virtual Machine;

СКБД – Система керування базами даних.

ВСТУП

Наразі, найбільш зручним на швидким способом знайти місцевих контракторів, або людей, що надають різного роду послуги – це використовуючи мережу Інтернет та різноманітні платформи для пошуку. Спираючись на інформацію з відкритих джерел, нещодавнє дослідження показало, що у 2023 році 98% людей використовують мережу Інтернет та різні вебсервіси для пошуку місцевих компаній-контракторів, звертаючи увагу при цьому на відгуки інших користувачів. Це майже на 8% більше ніж у 2019 році за результатами того ж дослідження [1].

На сьогоднішній день існує чимало різноманітних застосунків та сервісів, що дозволяють шукати підрядників. Але часто вони, або не надають достатньої інформації про вид діяльності підрядника та результати виконаних ним робіт, або не забезпечують користувачів зручними інструментами для оцінки надавача послуг, або орієнтовані лише на специфічний регіон світу чи окрему країну.

Виходячи з цього, розробка програмного продукту для пошуку і оцінки компаній-підрядників є доволі актуальною, не дивлячись на велику кількість аналогів та конкурентів в цій сфері. Тож, основною метою цього проекту є створення інтуїтивного та ефективного інструменту, який, незалежно від місцезнаходження користувача та пристрою, який він використовує, дозволить швидко знайти надійного підрядника для виконання необхідної роботи клієнту в незалежності від роду роботи.

Розроблений програмний продукт має включати в себе швидку систему пошуку, яка дозволить здійснювати фільтрацію за ключовими словами та за різними критеріями, такими як: категорія, рейтинг і місцезнаходження. Також має бути реалізований функціонал для оцінювання та залишення відгуків про підрядників, що допоможе іншим користувачам зробити обдуманий та вдалий вибір.

Для бажаючих опублікувати інформацію про свою компанію та послуги, які вони надають, буде можливість зареєструвати свій бізнес профіль, що дозволить іншим користувачам знаходити інформацію про цих підрядників, бачити їх рейтинг, ставити оцінку та залишати відгук з фото виконаних робіт.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд основних термінів і понять предметної області

Основною сутністю обраної предметної області є компанія-підрядник, або просто підрядник. Підрядник – це сторона договору підряду, яка зобов'язується на свій ризик виконати певну роботу або послугу за завданням другої сторони (замовника) [2].

У світі розрізняють понад 600 різних видів послуг, які можна класифікувати наступним чином [3]:

- транспортне обслуговування з підрозділом на пасажирські та вантажні перевезення;
- мандрівки ділові та особисті;
- зв'язок;
- будівництво;
- страхування;
- фінансові;
- комп'ютерні, інформаційні послуги;
- роялті та ліцензування;
- інші бізнес послуги, такі як посередницькі, технічні, лізингові;
- персональні, культурні та рекреаційні послуги;
- послуги державних установ.

Як можемо бачити, в світі існує багато різноманітних видів послуг, які можуть надаватися, або окремими підрядниками, або цілими компаніями, і їх може бути безліч з різною якістю і професійністю надавання послуг.

Також при виборі підрядника важливим аспектом є відгуки від клієнтів, які раніше співпрацювали з цим підрядником, а також їх оцінка виконаній роботі. Якщо результат праці матеріальний, то можна ще продемонструвати результат роботи підрядника через фото. Ця інформація значно допоможе майбутнім клієнтам зробити правильний вибір на користь того чи іншого підрядника.

Кожна оцінка попередніх клієнтів впливає на загальний рейтинг підрядника, який буде публічно доступним для користувачів і буде впливати на позицію в списку результатів пошуку. Тому підрядник сам має бути зацікавлений, аби його загальний рейтинг був найбільш високим і приваблював нових клієнтів.

Тому, на сьогоднішній момент, стає важливим розробити систему, яка буде надавати спрощений механізм пошуку надійних і професійних підрядників.

1.2 Огляд існуючих програмних продуктів для пошуку та оцінювання компаній-підрядників

1.2.1 Yelp

Yelp – це вебсервіс для оцінювання та залишення відгуків про місцеві компанії-підрядників. Початково заснований у 2004 році, цей сервіс став одним з найпопулярніших джерел оглядів у США та більшості європейських країн. З його допомогою користувачі можуть знаходити, оцінювати та залишати відгуки про різноманітні заклади та підрядників (рис 1.1). Крім вебверсії, Yelp також пропонує мобільний застосунок, що дозволяє користувачам отримувати доступ до сервісу у будь-який час і з будь-якого місця. На сьогоднішній день він має понад 140 мільйонів відгуків і щомісячно відвідується близько 19 мільйонами користувачів [4].

Основною метою сервісу є надання користувачам можливості висловлювати свої враження про підприємства, а також знаходити відгуки інших користувачів. Кожен бізнес має свою сторінку на Yelp, де можна знайти інформацію про робочі години, адресу місцезнаходження, номери телефонів та інші ключові відомості, а також переглядати відгуки та оцінки.

Серед основних переваг сервісу можна виділити декілька:

- володіє широким спектром категорій, включаючи сервіси по дому, сервіси пов'язані з авто-обслуговуванням, заклади громадського харчування та інші послуги;
- має активну аудиторію, що постійно залишає відгуки, оцінює і ділиться враженнями про підрядників або заклади;
- дозволяє безкоштовно створити свій бізнес профіль;
- має сувору політику щодо фальшивих відгуків користувачів і вживає заходів, щоб переконатися, що на вебсайті розміщуються лише відгуки реальних людей.

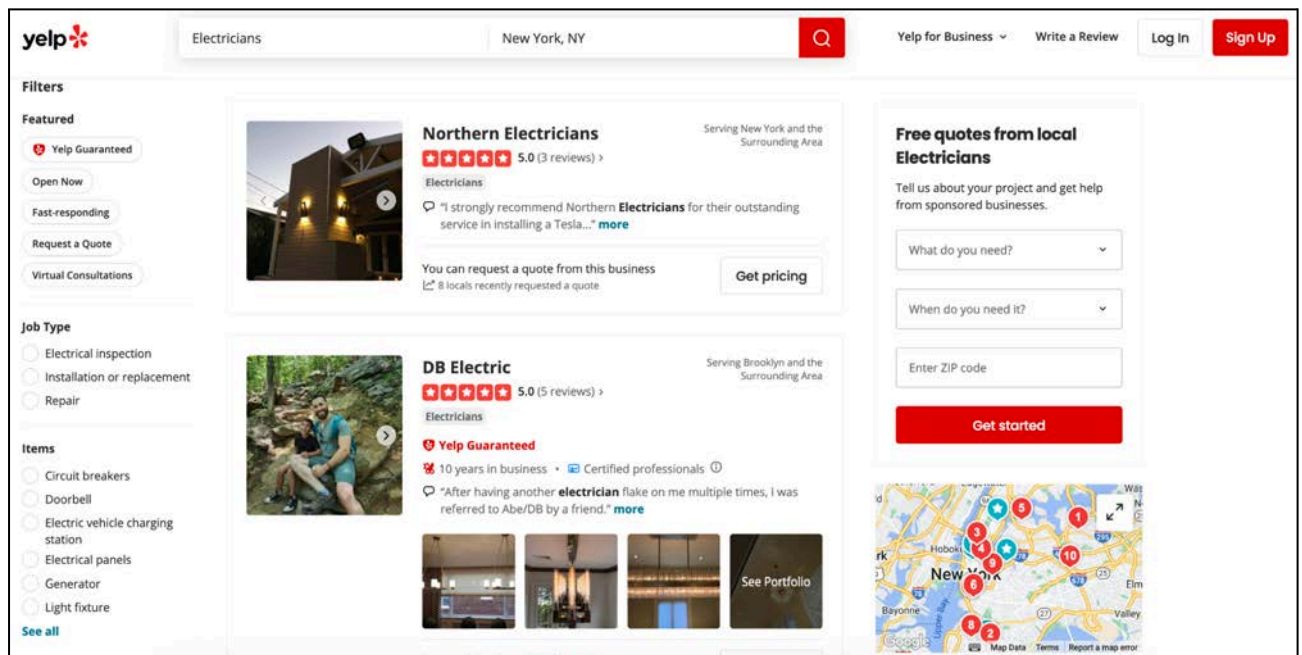


Рисунок 1.1 – Сторінка результатів пошуку у сервісі Yelp [5]

1.2.2 Google Reviews

Google Reviews – це інтегрований сервіс пошукової системи Google, яка дозволяє користувачам залишати відгуки та оцінки для різноманітних компаній, послуг та закладів, що відображаються у результатах пошуку (рис 1.2).

Оцінки та відгуки користувачів у Google Reviews мають значення як і для підприємств, так і для користувачів, оскільки вони впливають на репутацію компаній та служать як важливий фактор при виборі товарів чи послуг. Ця платформа стала важливим інструментом для спільноти та взаємодії, де думка кожного користувача має можливість вплинути на рішення інших у виборі надійного та якісного обслуговування.

Основними перевагами сервісу є:

- сервіс інтегрований у найбільш популярний пошуковий сервіс Google, а також Google Maps;
- доступний інтерфейс для перегляду і залишення відгуків;
- в пошуку Google одразу видно назву, робочі години і загальний рейтинг підприємця.

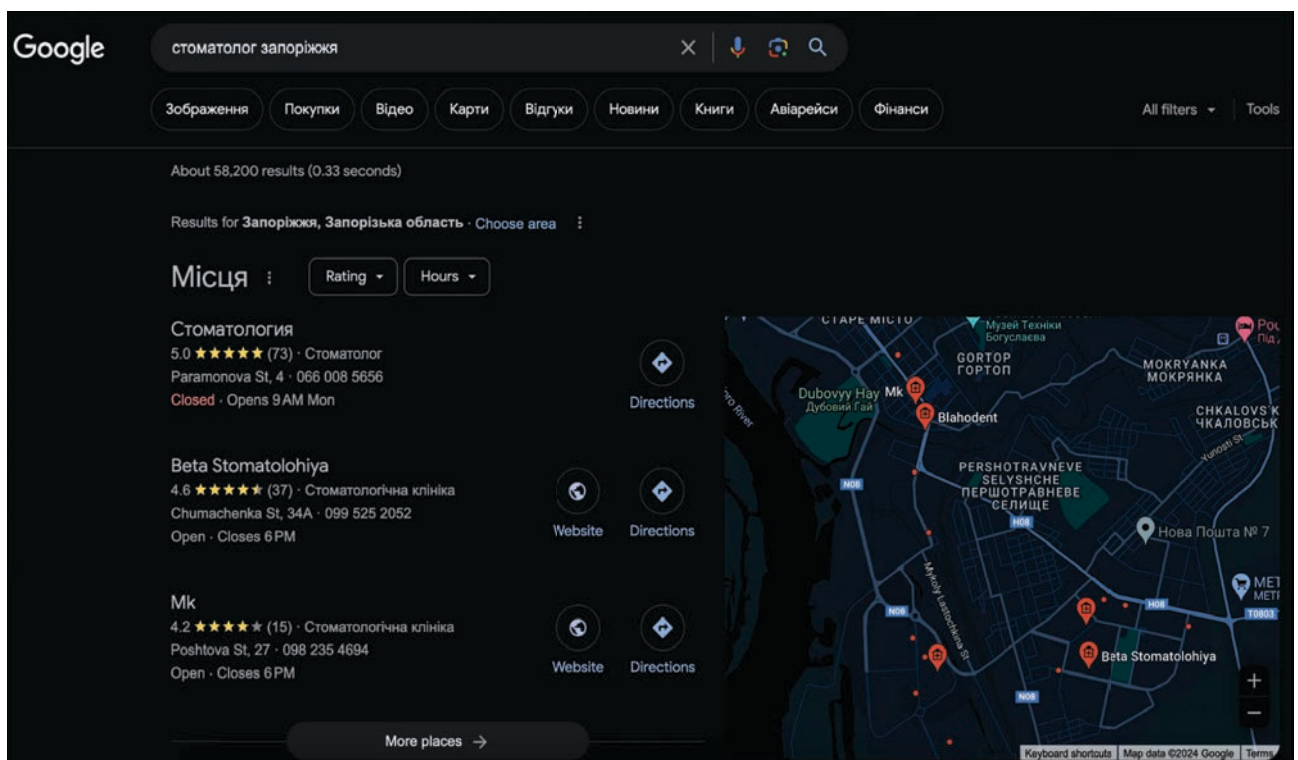


Рисунок 1.2 – Сторінка результатів пошуку у пошуку Google [6]

1.2.3 OLX

OLX.ua є найбільш популярною онлайн-торговельною платформою, що дозволяє користувачам в Україні безкоштовно розміщувати оголошення про продаж або обмін різноманітних товарів та послуг. Заснований у 2006 році, OLX.ua швидко став важливим ресурсом для тих, хто шукає можливості купівлі та продажу на другорядному ринку, або тих, хто шукає людей, які надають необхідні послуги. OLX також працює і в інших країнах світу, таких як Польща, Румунія, Болгарія [7].

За інформацією з відкритих джерел, на 2018 рік на платформі зареєстровано близько 1,5 млн продавців, розміщено понад 11 млн оголошень і кожен хвилину створюється близько 100 нових [8].

Серед особливостей сервісу можна виділити наступні:

- простота використання та доступність для широкого кола користувачів;
- можливість створити одне безкоштовне оголошення;
- висока популярність в Україні, що дозволяє швидко знайти клієнтів, покупців або підрядників;
- широкий спектр пошукових фільтрів для кожної категорії (рис 1.3);
- надає можливість взаємодіяти користувачу з власником оголошення, що дозволяє обговорювати деталі, уточнювати умови та узгоджувати угоди безпосередньо на платформі;
- дозволяє власнику оголошення продивлятися статистику за його оголошенням;
- існують функції для підвищення оголошення в загальному списку результатів.

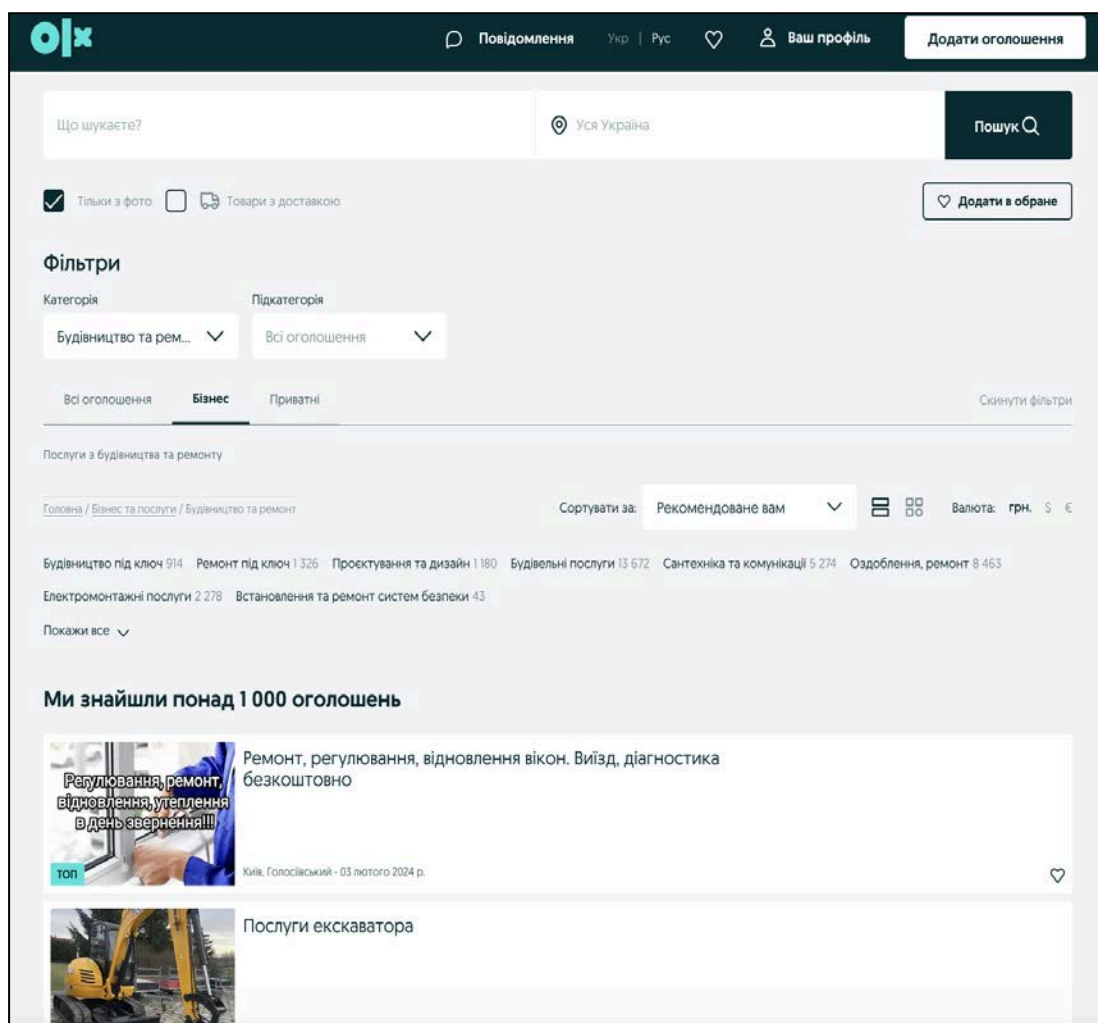


Рисунок 1.3 – Сторінка результатів пошуку у сервісі OLX [9]

1.2.4 Kabanchik.ua

Kabanchik.ua – це онлайн платформа, яка допомагає знаходити приватних фахівців для вирішення різноманітних побутових та бізнесових завдань. Цей сервіс об'єднує людей, які потребують допомоги з виконанням різних завдань, з досвідченими спеціалістами, які шукають можливості для підробітку або додаткового заробітку [10].

17 вересня 2012 року була запущена перша версія проекту, який спочатку працював лише в Києві. Вже у 2015 році український торговий майданчик Prom.ua придбав сервіс Kabanchik.ua, що дозволило значно розширити його географію.

1 червня 2016 року було випущено мобільний додаток для фахівців на Android, а згодом і для замовлення послуг.

Також 28 лютого 2022 року, у зв'язку з воєнними діями в Україні, на сервісі Kabanchik.ua були створені волонтерські категорії. Цей розділ дозволяє українцям, які потребують допомоги, знаходити виконавців-волонтерів.

Серед переваг сервісу можна виділити наступні:

- фахівці можуть створювати свій профіль, вказувати свої навички та досвід, а також отримувати відгуки від клієнтів;
- користувачі мають можливість вибирати серед різних пропозицій фахівців, порівнюючи їх рейтинг, відгуки та ціни, що дозволяє знайти оптимального виконавця для свого завдання;
- платформа пропонує широкий спектр категорій послуг та фахівців.

На рис. 1.4 зображено головну сторінку Kabanchik.ua [11].

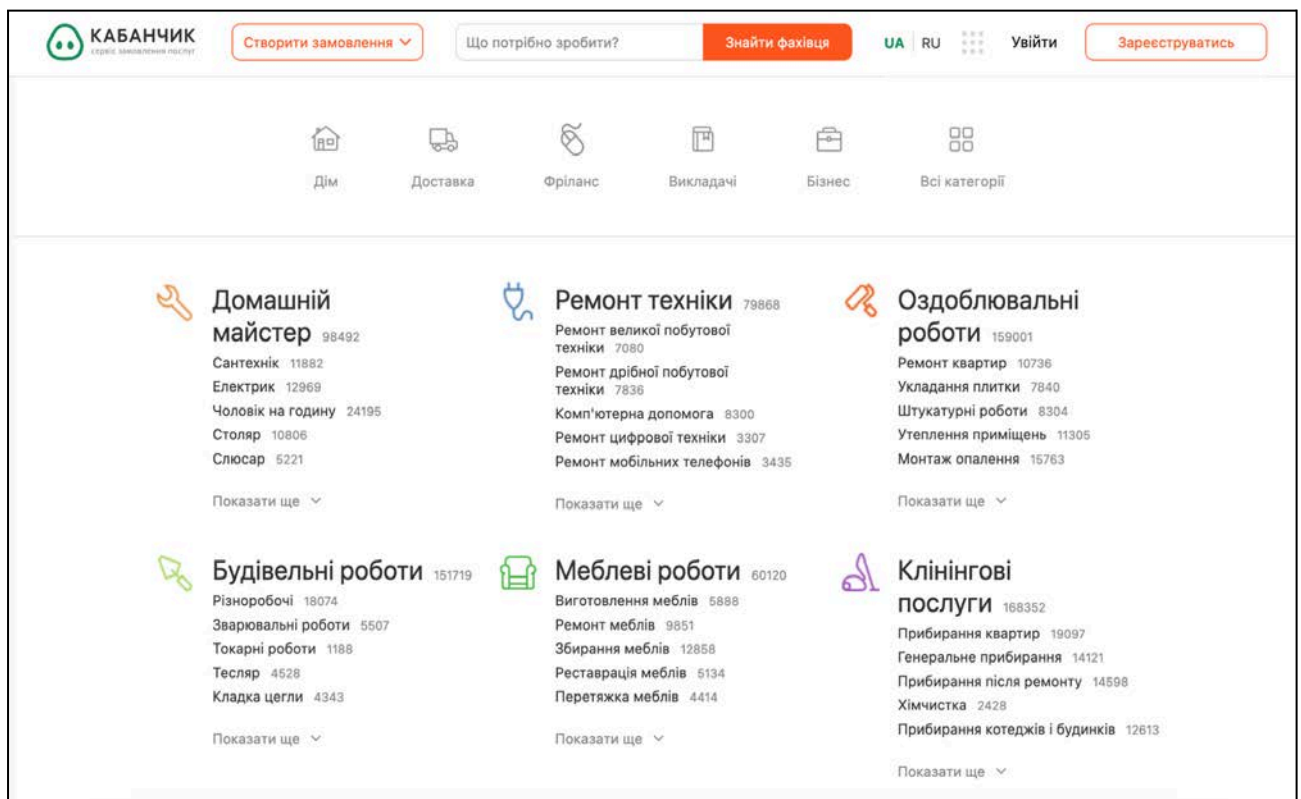


Рисунок 1.4 – Головна сторінка сервісу Kabanchik.ua [11]

1.2.5 Порівняльна таблиця існуючих програмних продуктів

Нижче представлена порівняльна таблиця наявних наразі програмних продуктів для пошуку і оцінювання контракторів (табл. 1.1).

Таблиця 1.1 – Порівняння програмного забезпечення для пошуку і оцінювання контракторів

Критерій порівняння	Існуючі програмні продукти			
	Yelp	Google Reviews	OLX	Kabanchik.ua
Доступний в скількох країнах	32	Усюди	15	1
Наявність відгуків	+	+	-	+
Наявність пошуку з фільтрами	+	-	+	+
Детальний опис компаній-контракторів	+	-	+	+
Є можливість безкоштовно створити бізнес профіль	+	+	+	+

Як можемо бачити з табл. 1.1, у кожного з перелічених сервісів є ряд переваг, а також недоліків. Серед спільного можна виділити, що кожен з цих сервісів дозволяє створити свій бізнес профіль безкоштовно, а також мають велику активну аудиторію користувачів.

1.3 Постановка завдання роботи

Виходячи з порівняння, наведеного в табл. 1.1, можемо виділити ряд основних функціональних вимог необхідних для розроблюваного програмного продукту, а саме:

- можливість безкоштовного створення одного бізнес профілю;
- показувати детальну інформацію про компанії у пошуку, включаючи: назву та опис компанії, фото виконаних робіт, місцезнаходження та контактні дані (номера телефону, електронна поштова адреса), оцінки та відгуки користувачів;
- надавати можливість гнучкого пошуку, у разі якщо користувач зробить помилку у запиті;
- надавати можливість користувачам залишати відгуки та ставити оцінки підрядникам.

1.4 Висновки за розділом 1

В першому розділі було проаналізовано предметну область, основні поняття та найбільш популярні програмні аналоги розроблюваного програмного продукту.

У ході проведеного аналізу було визначено сильні та слабкі сторони існуючих програмних аналогів, а також основні та необхідні для реалізації функціональні вимоги для розроблюваного програмного продукту.

2 ОПИС ПРОГРАМИ

2.1 Обґрунтування вибору мов програмування

Основними мовами програмування для розробки програмного продукту пошуку компаній-підрядників було обрано Go (Golang) для серверної частини застосунку та JavaScript (з TypeScript) для клієнтської частини застосунку. Для побудови користувацьких інтерфейсів обрано фреймворк React.

Додатково, для взаємодії з реляційною базою даних PostgreSQL, використовується мова структурних запитів SQL.

2.1.1 Go (Golang) як мова програмування серверної частини застосунку

Мова програмування Go, також відома як Golang, є кросплатформенною, високопродуктивною, компільованою мовою з вбудованими засобами для паралельних обчислень, розробленою в компанії Google. Основна мета створення Go - надати простий і зрозумілий інструмент для розробки швидких та надійних програм, особливо високонавантажених (highload) систем [12].

Однією з ключових особливостей мови Go є її простота та чистота синтаксису. Мова пропонує лише невеликий набір ключових слів і конструкцій, які базуються на елементах мови C, що спрощує процес написання коду і робить його більш зрозумілим для розробників. Крім того, мова має вбудовану систему управління та очищення пам'яті (garbage collector або GC), що полегшує роботу з динамічною пам'яттю та зменшує ймовірність витоків пам'яті.

Не менш важливою особливістю є швидкодія Go. Програмний код на мові Go компілюється безпосередньо в машинний код, що дозволяє програмам на Go працювати дуже швидко, навіть на великих обсягах даних. Крім того, Go підтримує паралельне виконання завдяки своїй вбудованій підтримці горутин,

що дозволяє розробникам легко створювати конкурентні програми та ефективно використовувати ресурси обчислювальної системи.

Завдяки своїм перевагам у швидкості, простоті та надійності, мова програмування Go стала популярним вибором для розробки серверної частини вебзастосунків, мікросервісів, СКБД, інструментів автоматизації, контейнеризації, оркестрації контейнерів, системного програмування та інших областей розробки програмного забезпечення. Вона продовжує набирати популярність серед розробників завдяки своїй простоті використання і потужності для вирішення широкого спектру завдань. Серед найвідоміших компаній, які активно використовують Go в своїх системах, можна виділити Uber, X (Twitter), Google, Microsoft, Meta, Dropbox, Cloudflare та інші [13].

2.1.2 JavaScript (з TypeScript) як мова програмування клієнтської частини застосунку

JavaScript є однією з найпопулярніших мов програмування для веброзробки, що використовується для створення динамічних та інтерактивних вебсайтів. JavaScript є мовою скриптів, яка виконується безпосередньо в браузері користувача і дозволяє додавати функціональність до вебсторінок. Станом на 2024 рік, 98.9% вебсайтів розроблено саме мовою JavaScript [14].

Однією з особливостей JavaScript є його динамічна та слабка типізація, що дає розробникам велику гнучкість у написанні коду. Проте, при необережному використанні мови, слабка типізація може також призводити до помилок під час розробки через неявність типів даних. Саме для цього була розроблена мова TypeScript.

TypeScript є суперсетом JavaScript, який надає статичну типізацію, що дозволяє визначати типи даних для змінних, параметрів функцій та інших елементів коду. Статична типізація дозволяє значно зменшити кількість помилок у коді під час розробки, зробити програмний код більш передбачуваним, покращити розробку в інтегрованих середовищах розробки,

полегшити рефакторинг великих кодових систем та підвищити стабільність програм.

Також JavaScript відомий великою кількістю фреймворків для розробки користувацького інтерфейсу. Серед найбільш відомих фреймворків можна виділити такі.

1. React. – розроблений компанією Meta (Facebook), є одним з найпопулярніших та найбільш використовуваних фреймворків. Фреймворк базується на компонентах, що дозволяє розробникам створювати складні інтерфейси з допомогою множини пере використання компонентів [15].

2. Angular – розроблений Google, є фреймворком для створення односторінкових застосунків (SPA) на основі TypeScript з використанням парадигми об'єктно-орієнтованого програмування. Angular має широкий набір функцій, таких як маршрутизація, управління станом застосунку та обробка подій [16].

3. Vue.js – це прогресивний JavaScript-фреймворк для створення інтерфейсів користувача. Фреймворк дозволяє розробникам поступово впроваджувати його у свої проекти, а також має простий і зрозумілий синтаксис [17].

4. Svelte – це новий фреймворк, який відрізняється від React, Angular та Vue, оскільки компілює код в чистий JavaScript під час збірки, що дозволяє забрати зайвий JS з браузера, що знижує розмір коду. Порівнюючи Svelte з альтернативними інструментами, цей фреймворк має один із найменших пакетів розміром лише 2 Кб [18].

Ці фреймворки дозволяють розробникам швидко та ефективно створювати потужні вебзастосунки, використовуючи сучасні підходи та інструменти.

2.1.3 Порівняння та обґрунтування вибору мов програмування

Нижче представлена порівняльна таблиця мов програмування для розробки клієнт-серверних програмних продуктів (табл. 2.1).

Таблиця 2.1 – Порівняння мов програмування для розробки серверної та клієнтської частин програмного забезпечення

Критерій порівняння	Мова програмування			
	Go (Golang)	Java	JavaScript (NodeJS)	Python
Типізація	Статична, строга	Статична, строга	Динамічна, нестрога	Динамічна, строга
Чи підтримує декілька парадигм програмування	+	+	+	+
Чи підтримує компіляцію	+	+/- (компілюється у байт-код)	-	-
Чи підтримує паралельне виконання коду	++	+	+/- (за допомогою декількох процесів)	+/- (за допомогою декількох процесів)
Сфери застосування	Серверні застосунки, highload системи	Серверні та мобільні застосунки	Веброзробка (клієнт і сервер)	AI, ML
Складність вивчення	Низька	Середня	Середня	Середня
Складність використання	Низька	Середня	Середня	Низька

Як видно з табл. 2.1, для реалізації клієнтської частини програмного застосунку є небагато опцій, а саме тільки JavaScript. Всі інші мови програмування більше підходять для реалізації серверної частини програмного продукту.

Для розробки користувацького інтерфейсу було обрано React фреймворк, так як React є найбільш популярним серед існуючих альтернатив, що дозволяє значно швидше знаходити рішення для проблем, які можуть виникати при розробці. Також, завдяки високій популярності, React має безліч сторонніх модулів та компонентів для розробки користувацького інтерфейсу, що значно пришвидшить конструювання інтерфейсу, бо непотрібно буде розробляти компоненти з нуля.

Для вибору серверних мов програмування можна орієнтуватися за такими критеріями: швидкодія програмного коду, написаного на обраній мові програмування, використання обчислювальних ресурсів, складність вивчення/використання мови для розробки програмних продуктів, тощо. Спираючись на дані з таблиці, наведеної вище, можна сказати, що Go, порівняно з іншими мовами, має більше переваг саме для написання серверної логіки програмних продуктів. Go розроблявся з акцентом на простоту вивчення та використання, на швидкодію програмного коду, з можливістю додатково запускати частини коду конкурентно або паралельно. Також, завдяки тому, що Go компілюється в машинний код, виконання написаного коду відбувається значно швидше.

2.2 Обґрунтування вибору супровідних технологій

2.2.1 PostgreSQL як реляційна система керування базами даних (СКБД)

PostgreSQL – це потужна та розширювана об'єктно-реляційна система управління базами даних (СКБД), яка пропонує широкий набір функцій для зберігання та обробки даних. PostgreSQL є однією з найпопулярніших

відкритих СКБД та широко використовується у веброзробці, програмуванні підприємств та наукових дослідженнях.

PostgreSQL розробляється незалежно від однієї компанії; розвиток програмного продукту можливий завдяки співпраці численних людей та компаній, які бажають використовувати цю систему керування базами даних і впроваджувати в неї останні досягнення [19].

PostgreSQL підтримує стандарт SQL та включає ряд розширених функцій, таких як: підтримка UUID типу даних, тригерів, транзакцій (з підтримкою ACID властивостей), географічних структур даних, та інші. База даних також підтримує ряд розширень, які дозволяють розширити її функціональність, наприклад: розширення для повнотекстового пошуку, геоданих, обробки XML та JSON, а також інші.

Однією з переваг PostgreSQL є його висока швидкість та надійність, що робить його ідеальним вибором для великих та важких завдань обробки даних. Крім того, вона є відкритою та безкоштовною для використання, що робить її доступною для широкого кола розробників та організацій.

2.2.2 Elasticsearch як пошукова система для повнотекстового пошуку

Elasticsearch – це потужний та розширюваний пошуковий двигун, який використовується для аналізу та пошуку великих обсягів даних у реальному часі. Elasticsearch базується на відкритому коді та розроблений компанією Elastic. Elasticsearch використовується для створення пошукових систем, аналізу журналів, моніторингу та аналітики даних, а також для реалізації повнотекстового пошуку у вебзастосунках [20].

Основою Elasticsearch є розподілена архітектура, яка дозволяє розподілити дані на кілька вузлів та обробляти їх паралельно. Це забезпечує високу швидкість та масштабованість пошуку, навіть при обробці великих обсягів даних. Крім того, Elasticsearch має розширений мовний API, що

дозволяє виконувати різноманітні запити та аналізувати дані з великою гнучкістю [21].

Однією з ключових особливостей Elasticsearch є його можливість виконувати повнотекстовий пошук з високою швидкістю та точністю. Він також підтримує ряд додаткових функцій, таких як фасетний пошук, агрегації, фільтрація, сортування та інші, що дозволяють розробникам створювати потужні та ефективні пошукові застосунки.

Крім того, для відображення даних у Elasticsearch існує ще один продукт від компанії Elastic – Kibana. Зазвичай вони використовуються разом, де Elasticsearch виступає як сховище журналів (логів), а Kibana – для їх фільтрації та візуалізації.

2.2.3 Docker як технологія контейнеризації програмних продуктів

Docker – це популярна платформа для контейнеризації застосунків, що дозволяє упаковувати та запускати застосунки разом з їхнім середовищем в контейнерах. Контейнери Docker відрізняються від традиційних віртуальних машин тим, що вони використовують віртуалізацію на рівні операційної системи (OS), що робить їх легкими та ефективними у використанні [22].

Основна мета Docker полягає в уніфікації процесу розгортання та управління застосунками. За допомогою Docker, розробники можуть створювати ізольовані контейнери, які містять усі необхідні залежності та середовище для роботи застосунка, і розгортати їх на будь-якому сервері або хості, що підтримує Docker. Це дозволяє запускати застосунки в будь-якому середовищі, будь то розробка на локальному комп'ютері, тестування на стенді чи розгортання в хмарному середовищі.

Різниця між контейнерами і віртуальними машинами полягає в тому, як вони віртуалізують апаратне забезпечення. Віртуальні машини (VMs) віртуалізують апаратне забезпечення, що дозволяє запускати кілька окремих операційних систем на одному фізичному сервері. Контейнери, з іншого боку,

віртуалізують операційну систему, дозволяючи запускати декілька ізольованих середовищ на одному операційному ядрі.

Серед сильних сторін контейнерів перед віртуальними машинами можна виділити декілька:

- порівняно з контейнерами, віртуальні машини вимагають значних обсягів ресурсів, таких як пам'ять та процесор, через необхідність віртуалізації апаратного забезпечення;
- запуск віртуальних машин зазвичай займає більше часу, ніж запуск контейнеру;
- більшість інструментів оркестрації застосунків працюють на основі контейнерів.

Для Docker також існує публічний репозиторій з великою кількістю різних образів Docker, відомий як DockerHub. Більшість популярних сервісів, інструментів та застосунків доступні у DockerHub, що дозволяє зручно налаштувати локальне середовище розробки, а також робити розгортання у виробниче середовище на сервері.

2.3 Обґрунтування вибору середовища розробки

Середовищем для розробки програмного продукту для пошуку компаній-підрядників було обрано – Goland, інтегроване середовище розробки (IDE) для мови програмування Go, розроблене компанією JetBrains. Це потужний інструмент, спеціально створений для розробки програм на мові Go, який надає розробникам широкі можливості та інструменти для зручної та продуктивної роботи, такі як: компіляція та запуск коду, відладчик коду, профілювання всього застосунку або окремих функцій, індексація проєкту та необхідних для проєкту бібліотек, що покращує автодоповнення коду, потужні інструменти для рефакторингу та кодогенерації та інші. Goland також підтримує розробку на JavaScript та TypeScript [23].

Серед альтернативних програмних продуктів для розробки на Go та

JavaScript, можна зазначити редактор коду Visual Studio Code. Цей редактор пропонує базовий набір функціоналу, необхідного для розробки, такий як автодоповнення коду та підсвітка синтаксису. Однак, порівняно з Goland, він має обмежений функціонал.

Goland також має ряд плагінів, які дозволяють розширити можливості середовища розробки. Один із основних плагінів, які широко використовуються, – це Git плагін, призначений для інтеграції з системою контролю версій Git. Також доступний плагін для інтеграції з Docker, який дозволяє створювати, запускати та управляти контейнерами, а також надає підтримку для написання нових Docker образів у Dockerfile. Додатково, цей плагін додає підтримку Docker Compose для створення та керування декількома контейнерами одночасно.

Однією з ключових особливостей Goland є його інтелектуальні функції, такі як автоматичне завершення коду, підказки по синтаксису, рефакторинг, форматування та перевірка коду, що сприяють швидкій та ефективній розробці. Останні версії Goland мають інтегровану систему автодоповнення на основі штучного інтелекту.

Goland також надає можливість роботи з різними типами проектів, включаючи вебзастосунки, мікросервіси, консольні програми та інші. Він має інтегровану систему тестування та запуску бенчмарків, яка допомагає розробникам створювати надійний і тестований код.

2.4 Загальний опис архітектури основних компонентів системи

В табл. 2.2 наведено основний перелік компонентів розроблюваної програмної системи, а також опис до кожного компонента. Кожен компонент системи тим чи іншим способом пов'язаний з іншими компонентами, але кожен компонент замість виклику необхідних процедур та функцій напряму з іншого компоненту, буде використовувати контракти, які чітко описують, яким чином можна взаємодіяти з компонентом іншим компонентам. Це дозволяє

мінімізувати кількість помилок, якщо необхідно розробити зміни в середині одного компонента і при цьому не пошкодити роботу іншого залежного компонента.

Таблиця 2.2 – Перелік і опис основних компонентів розроблюваної системи

Назва модулю	Опис та призначення
Search	Модуль пошуку компаній-підрядників за ключовими словами, місцезнаходженням, категорією і т.д. Дозволяє переглядати результати пошуку та інформацію про підрядників.
Reviews	Модуль, що дозволяє залишати відгуки підряднику та отримувати список вже зроблених відгуків. Також цей модуль займається підрахунком середнього рейтингу на основі оцінок в відгуках користувачів.
Identity	Модуль реєстрації, аутентифікації та авторизації користувачів в системі. Також модуль дозволяє управляти власним профілем.
Business	Модуль керування власним бізнес-профілем. Дозволяє зареєструвати свій бізнес-профіль на основі користувацького профілю та управляти ним.
Вебінтерфейс користувача	Модуль, що комбінує функціональність усіх вищезгаданих модулів та надає її у вигляді вебінтерфейсу.

Перші чотири модулі, наведені в табл. 2.2, знаходяться на стороні вебсерверу. Вони обробляють HTTP запити, взаємодіють з базами даних, та запускаються на окремій інфраструктурі, незалежній від клієнтської. Останній

модуль – це вебінтерфейс користувача. Це клієнтська частина застосунку, розроблена на JavaScript, яка буде виконуватись на стороні користувача, тобто в його браузері.

Для розробки серверної частини застосунку було вирішено використовувати архітектурний підхід REST. Серед переваг використання REST підходу можна виділити такі:

- API, що розроблені використовуючи REST (також називаються RESTful API), використовують стандартні HTTP методи (GET, POST, PUT, PATCH, DELETE), які вже знайомі розробникам;
- RESTful API властива масштабованість, завдяки якій можливо легко горизонтально масштабувати вебсервер, адже кожен запит, що надходить від клієнта, має містити усю необхідну для обробки інформацію;
- використання REST не прив'язане до конкретних мов програмування або фреймворків, що дозволяє впроваджувати REST підхід у будь-які вебзастосунки;
- RESTful API відомі використанням єдиного інтерфейсу, що спрощує комунікацію між клієнтом і сервером, а також впровадження нового функціоналу;
- в REST підході є розділення обов'язків клієнта та сервера, що дозволяє їм розвиватися незалежно, що також спрощує інтеграцію нового функціоналу;
- REST побудований на HTTP протоколі, що дозволяє використовувати вже існуючі методи і інструменти розробки для вебсерверів.

Наразі існують й інші архітектурні підходи для розробки вебзастосунків, такі як: SOAP, gRPC, WebSocket та GraphQL. Кожен з них має ряд своїх переваг та недоліків, але вони розроблені для окримих випадків, коли, наприклад, потрібна двостороння комунікація між клієнтом та сервером, або потрібна можливість фільтрування даних, які мають повертатися в відповідях від сервера. Тому перелічені варіанти архітектурних підходів не були використані у

розроблюваному застосунку, але вони можуть бути корисними в інших типах вебзастосунків.

Клієнтська частина вебзастосунку була розроблена як SPA. SPA – це такий вебзастосунок, що завантажується у браузері лише один раз і, при взаємодії користувача з компонентами інтерфейсу, не потребує оновлювати сторінку кожен раз, коли треба запросити або оновити дані на стороні серверу.

Створення вебзастосунку у форматі SPA істотно покращує користувацький досвід, оскільки такий формат дозволяє отримати більш плавний та безперервний користувацький досвід, без необхідності постійно перезавантажувати сторінку або чекати на її повне завантаження.

Також розробляти SPA застосунки значно простіше завдяки великій кількості існуючих інструментів розробки.

Проте, одним з ключових факторів у виборі SPA для розробки клієнтської частини була масштабованість. Розробка SPA дозволяє розділити процес написання серверної та клієнтської частин, що прискорює розробку та дозволяє відокремити логіку відображення даних для користувача від обробки даних на сервері.

Але як кожна технологія, мова програмування, або архітектурний підхід мають свої недоліки, є недоліки й у SPA. Одним з основних недоліків є обмежена сумісність з SEO. Проте зараз ця проблема вирішується простіше, а також сучасні пошукові системи вже вміють працювати з SPA.

Підсумовуючи, можна описати загальну архітектуру клієнта і сервера застосунку із супровідними технологіями у вигляді спрощеної діаграми, зображеної на рис. 2.1.

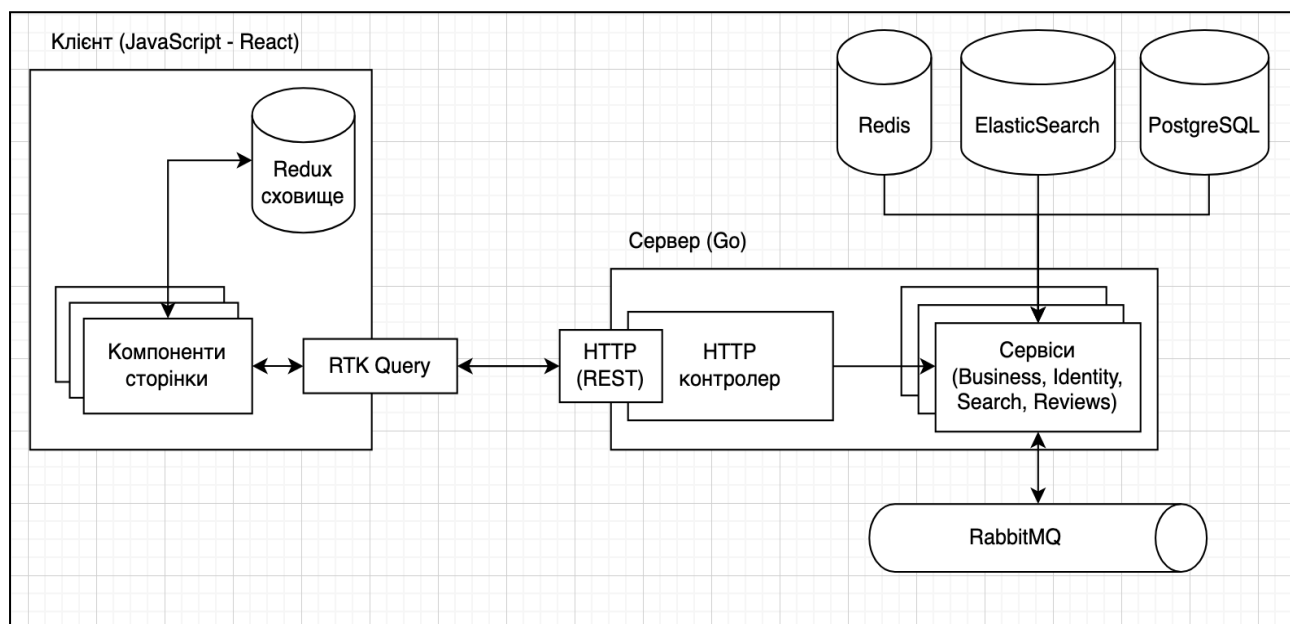


Рисунок 2.1 – Загальна архітектура клієнта і сервера

2.5 Висновки за розділом 2

У другому розділі було розглянуті можливі варіанти мов програмування для написання серверної та клієнтської частин програмного продукту. Було зроблено вибір на користь JavaScript (з TypeScript) для клієнтської частини застосунку, React фреймворк для побудови користувацького інтерфейсу та Go (Golang) для серверної частини застосунку.

Було обрано супроводжуючі технології, як Docker для контейнеризації програмного продукту, PostgreSQL як реляційну СКБД та Elasticsearch як пошукову систему для повнотекстового пошуку.

Також було зроблено опис архітектури програмного продукту, наведено список основних компонентів системи, їх опис та призначення.

3 РОЗРОБКА ПРОГРАМИ

Програмна система для пошуку компаній-підрядників складається з чотирьох основних серверних компонентів, які відповідають за різний набір функціоналу, і які композуються в єдину систему на стороні вебінтерфейсу користувача.

Структура і взаємодія основних компонентів програмної системи, баз даних та інших супровідних компонентів наведено на рис. 3.1.

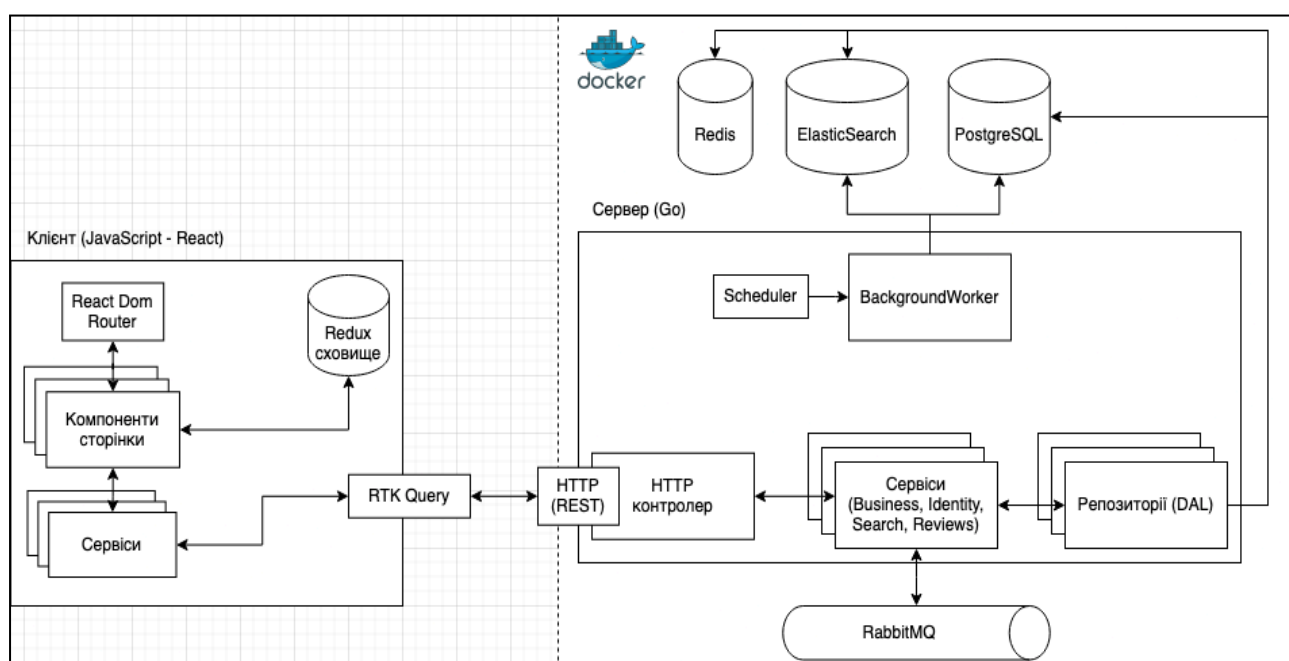


Рисунок 3.1 – Діаграма компонентів програмної системи

3.1 Розробка серверних компонентів системи

Усі серверні компоненти працюють незалежно один від одного, тому, в разі потреби, можна виділити один з компонентів і винести в окремий вебсервіс з окремою базою даних. Це може бути необхідно, якщо частина функціоналу користується попитом в користувачів і не вистачає обчислювальних ресурсів. Відокремлюючи цю частину функціоналу в окремий вебсервіс, з'являється можливість масштабувати цей сервіс незалежно від інших, що сприяє

покращенню швидкодії застосунку і дозволяє підтримувати більшу кількість активних користувачів.

Також, кожен сервіс на стороні сервера запускається у Docker контейнерах, що дозволяє зручно розгортати сервіси на будь-яких операційних системах з ріним налаштованим середовищем. Варто зазначити, що Docker розроблений і орієнтований під Linux ядро, тому на інших операційних системах знадобиться додаткова інсталяція Docker Desktop, який слугує легковісною віртуальною машиною з інсталюваним Linux ядром.

3.1.1 Опис класової ієрархії модулів

Кожен з модулів побудований за загальною схемою, використовуючи майже однаковий набір інтерфейсів і класів (рис. 3.2). Також, функції та класи, що використовуються у різних частинах застосунку, були винесені у окремий незалежний модуль – common. Модуль common переважно містить інфраструктурні речі для моніторингу (логування, метрики, і т.д.), для роботи з користувацьким сесіями (Cookies, SessionStore, і т.д.), тощо.

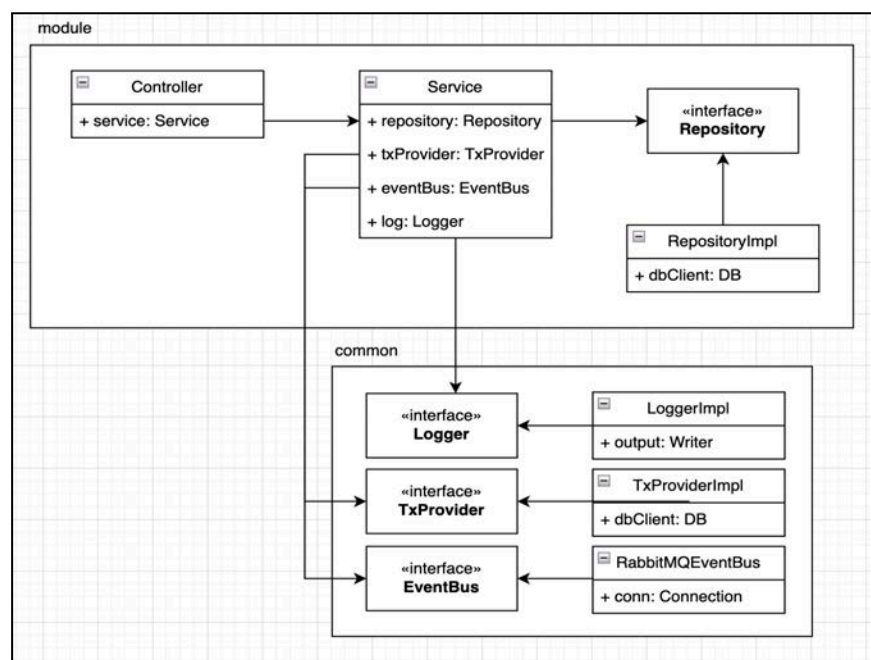


Рисунок 3.2 – Загальна діаграма класів модулю системи

Кожен модуль складається з класу Controller, який відповідальний за обробку HTTP запитів, парсинг даних, відправлених в запитих, і передачі на наступний рівень – рівень бізнес-логіки.

Додатково, у класі Controller може проводитися примітивна валідація вхідних даних, типів даних, тощо. Валідація правил предметної області відбувається на рівні класів типу Service, або на рівні самих сутностей предметної області.

В класах типу Controller прописані шляхи по яким вебсервер буде оброблювати запити, а також реалізація цих обробників. Обробники запитів зазвичай не мають в собі ніякої логіки пов'язаною з предметною областю, що автоматизується.

Наступним є рівень бізнес-логіки. Рівень бізнес-логіки представляється класом Service, який відповідальний за отримання даних з класу Controller, обробки отриманих даних згідно бізнес-логіки, зберігання оброблених даних в базу даних через клас Repository, і відправлення результатів роботи назад до класу Controller. Також частина правил предметної області може бути реалізована в класах сутностей.

У класі Service описується логіка самого застосунку. Там не має бути логіки обробки HTTP запитів, або логіки створення SQL запитів до баз даних. Рівень бізнес-логіки не знає нічого про набір технологій, що його оточують, а працює тільки сутностями, які представляють об'єкти з реального світу, їх властивості і поведінку.

Як вже було зазначено, клас Service не взаємодіє напряду зі сховищами даних. Взаємодія з базами даних відбувається через інтерфейс Repository. Клас Service не знає нічого про реалізацію інтерфейсу Repository, а тільки набір методів, що представляється для використання. На рис. 3.2 реалізація Repository інтерфейсу представлена класом RepositoryImpl, який зазвичай матиме доступ до класу для взаємодії з базою даних. Сам клас RepositoryImpl відповідальний за реалізацію інтерфейсу Repository, генерацію необхідних запитів, і отримання даних з БД у відповідні сутності, описані на рівні бізнес-логіки.

Також, клас `Service` може взаємодіяти із іншими інфраструктурними інтерфейсами, наприклад `TxProvider`, що дозволяє поєднувати виклик декількох методів в різних `Repository` в одну атомарну операцію. Тобто, якщо одна з операцій не відпрацює, не відпрацюють й інші, і система буде повернута до попереднього стану.

Тут варто зазначити, реалізація `TxProvider` залежить від використаної бази даних, бо не кожна база даних має підтримку транзакцій та реалізує властивості ACID (atomicity, consistency, isolation, durability), що є важливою властивістю реляційних баз даних, як, наприклад, PostgreSQL. NoSQL бази даних зазвичай або взагалі не реалізують властивості ACID та не мають підтримки транзакцій, або мають часткову підтримку, як, наприклад, DynamoDB.

Є інший інтерфейс – `EventBus`, який представляє publish-subscribe інтерфейс для роботи з системними подіями, що дозволяє відправляти події до брокеру, а також підписуватись на окремі типи подій і обробляти їх.

Іншим загальним інтерфейсом є – `Logger`, що відповідальний за логування системних повідомлень. Це важливий компонент будь-якої системи, який дозволяє розуміти поточний стан програмної системи і, у разі виявлення помилок в роботі, виправляти їх.

Додатково, модуль може містити фоновий процес – `BackgroundWorker`, що періодично, з використанням класу `Scheduler`, обробляє якісь дані, або з використанням `EventBus`, якщо треба обробляти системні події.

Наприклад, таким процесом, є обробка опублікованих відгуків під оголошеннями підрядників у модулі `Reviews`. Ця обробка включає оновлення рейтингу оголошень в базі даних `ElasticSearch` з метою автоматичного сортування результатів на сторінці пошуку за критерієм від найбільшого до найменшого рейтингу.

Для демонстрації загальної структури модуля, на рисунку 3.3 наведено приклад одного з модулів системи – модуль `Identity`, який відповідальний за управління акаунтами і профілями користувачів, а також за перевірку доступу.

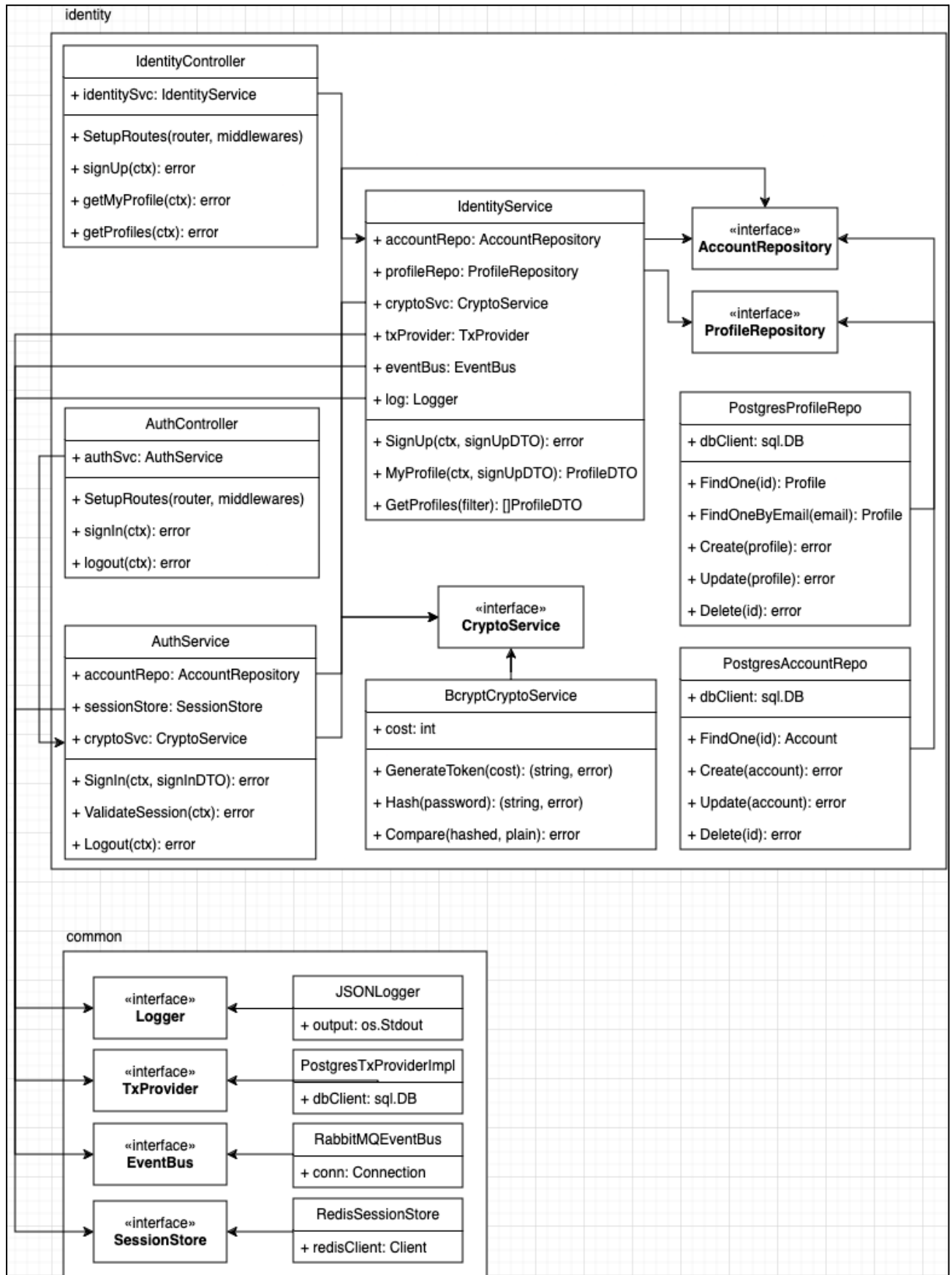


Рисунок 3.3 – Детальна діаграма класів модулю Identity

Модуль Identity містить по два класи типу Controller, Service та Repository, так як взаємодіє з двома сутностями - Account та Profile. Класи з префіксом Identity слугують в цілому для створення нового акаунту та профілю, тобто відповідальний за процес реєстрації користувача. Класи з префіксом Auth відповідальні за управління користувацькими сесіями.

Додатково у модулі Identity є ще два супровідних класи – RedisSessionStore, що реалізує інтерфейс SessionStore, та BcryptCryptoService, що реалізує інтерфейс CryptoService. Як можна побачити по назві, SessionStore відповідальний за управління та збереження користувацьких сесій, тоді як BcryptCryptoService – дозволяє генерувати токен сесії, хешувати паролі користувачів та перевіряти введений пароль користувачем на формі входу в систему зі збереженим захешованим паролем.

Важливим моментом в реалізації AuthService є те, що паролі користувачів не зберігаються у сирому вигляді в базі даних, бо це відкриває значно більшу прощину для кібератак на розроблювану систему. Тому в будь-якій програмній системі паролі хешуються тим чи іншим механізмом, бібліотекою, тощо. Як можна помітити з назви класу, у розробленій системі використовується бібліотека bcrypt, що є майже стандартним рішенням при розробці подібних функцій.

Також у класі AuthService є метод ValidateSession, який використовується у багатьох запитах на вебсервер для перевірки доступу до ресурсу у користувача. Викликається цей метод не напряму у кожному методі, який потребує перевірки доступу, а у HTTP middleware. Класи типу Middleware дозволяють попередньо, не допускаючи запит до обробників, зробити необхідні дії з вхідним запитом, або обробити запит вже після виклику обробника. Зазвичай Middleware класи використовують для виділення якоїсь загальної логіки, як от перевірка токена авторизації.

3.1.2 Реалізація процесу авторизації користувача

Кожен запит на сервер, що потребує доступу до користувацьких даних, має бути супроводжений токеном авторизації. Токен надається користувачу після успішного входу в систему через електронну пошту та пароль.

Токен авторизації генерується у класі `EncryptCryptoService` та за допомогою криптографічної бібліотеки `crypto`, що належить до стандартного набору бібліотек `Go`. Токен являє собою звичану строку, яка додатково є ще ідентифікатором користувацької сесії.

Потім, коли токен авторизації передається на сторону класу `Controller`, тобто на HTTP рівень, токен передається браузеру користувача у формі `cookies` і тому автоматично надсилається в усіх запитах на сервер застосунку, що спрощує роботу з токеном авторизації на стороні вебінтерфейсу.

Крім того, для запобігання витоку токена авторизації до злоумисників, сервер використовує токен із параметром `HttpOnly`, що блокує доступ до нього з будь-якого JavaScript коду, що використовується в застосунку або в бібліотеці, а також параметр `SameSite`, який контролює відправку `cookies` на зовнішні сервери.

Отримавши токен авторизації через модуль `Identity`, користувач отримує доступ до інших модулів і їх функціоналу, такого як:

- реєстрація власного бізнес-профілю та управління ним через модуль `Business`;
- створення відгуків та надання оцінок роботі підрядників через модуль `Reviews`;
- пошук підрядників через модуль `Search`.

На рис. 3.4 наведено повну схему переміщення даних у програмній системі від моменту реєстрації та авторизації користувача до створення відгуку під оголошенням підрядника.

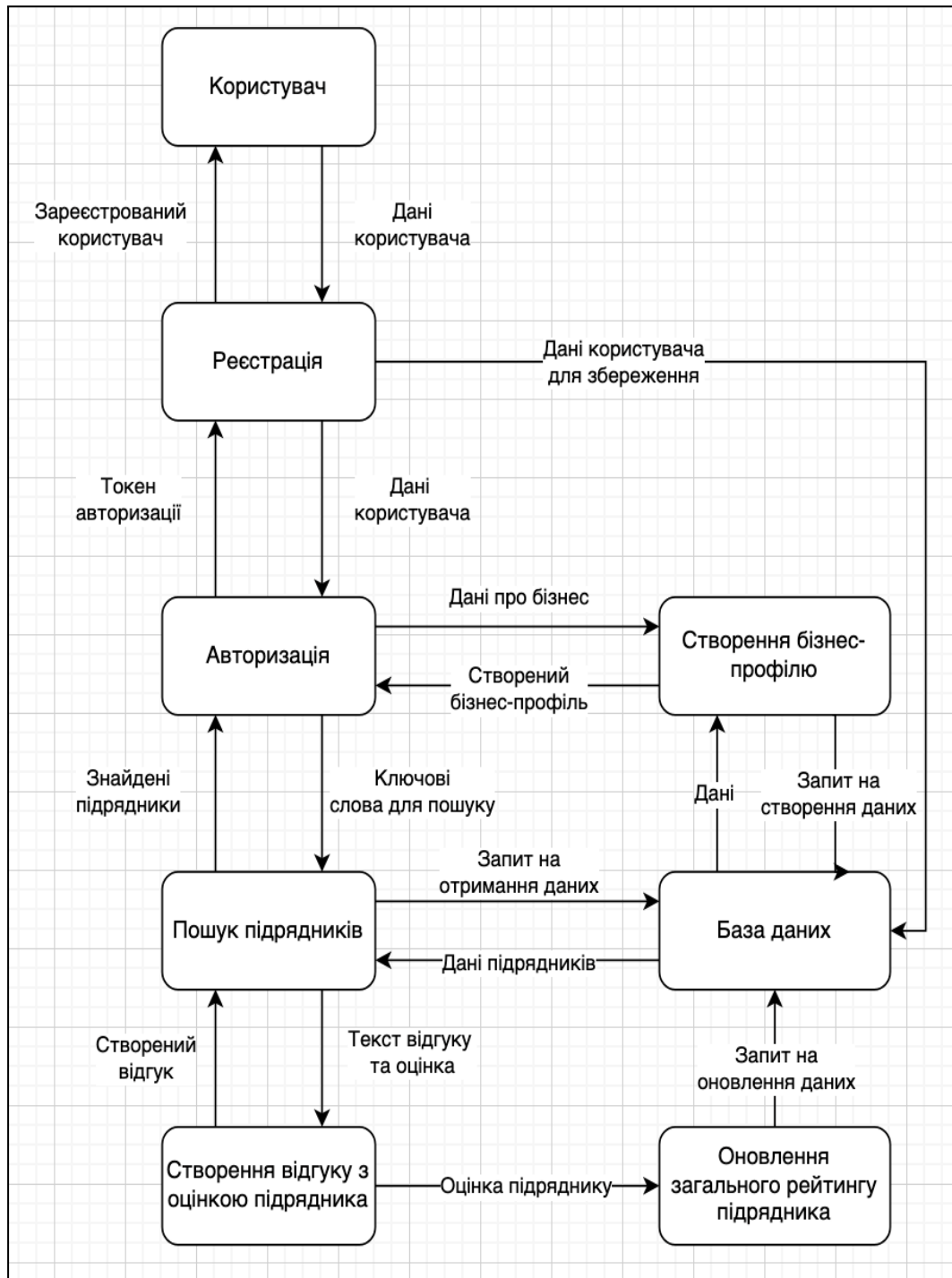


Рисунок 3.4 – Діаграма руху даних між процесами і сховищами

3.1.3 Опис взаємодії з базами даних

Повертаючись до рис. 3.1, що зображує загальну архітектуру застосунку видно, що більшість процесів та модулів в програмній системі працюють з

базами даних. Загалом, в програмній системі використовується три різні бази даних, які використовуються для різних потреб.

Elasticsearch використовується як база даних для повнотекстового пошуку, в якій знаходяться майже ідентичні дані підрядників, що й в PostgreSQL, які вводив користувач при створенні бізнес-профілю.

Використання Elasticsearch дозволяє оптимізувати збереження даних, що дозволяє значно пришвидшити пошук по великих масивах текстових даних, навіть якщо в запиті присутня одруківка, або запит сформульовано по-іншому, ніж це зроблено в збережених даних.

Взаємодія з Elasticsearch відбувається через класи типу Repository з використанням REST API Elasticsearch, що дозволяє передавати параметри пошуку та сортування підрядників. Також важливо відзначити, що дані в Elasticsearch можуть з'являтися з невеликою затримкою через те, що інформація про створення бізнес-профілю з модуля Business спочатку надсилається у чергу RabbitMQ до модуля Search, де дані підрядника додаються до бази даних.

Так як при розробці застосунку основним акцентом було створення системи, яка могла б легко розширюватися в майбутньому, було прийняте рішення розділити процес створення бізнес-профілю на два окремі процеси: створення сутності для пошуку та створення сутності для підтримки зв'язків між таблицями, що робить модуль Search та Business розділеними і покращує загальну масштабованість системи.

Для збереження даних підрядників, метаданих, даних користувачів, відгуків та оцінок використовується PostgreSQL.

Взаємодія з PostgreSQL відбувається також через класи типу Repository, проте використовується мова SQL та ORM інструмент go-jet, що спрощує створення та виконання різних видів запитів таких, як SELECT, INSERT, UPDATE, DELETE з змінним набором параметрів та фільтрів.

Загальна структура даних та зв'язків між сутностями у базі даних PostgreSQL представлена на рис. 3.5.

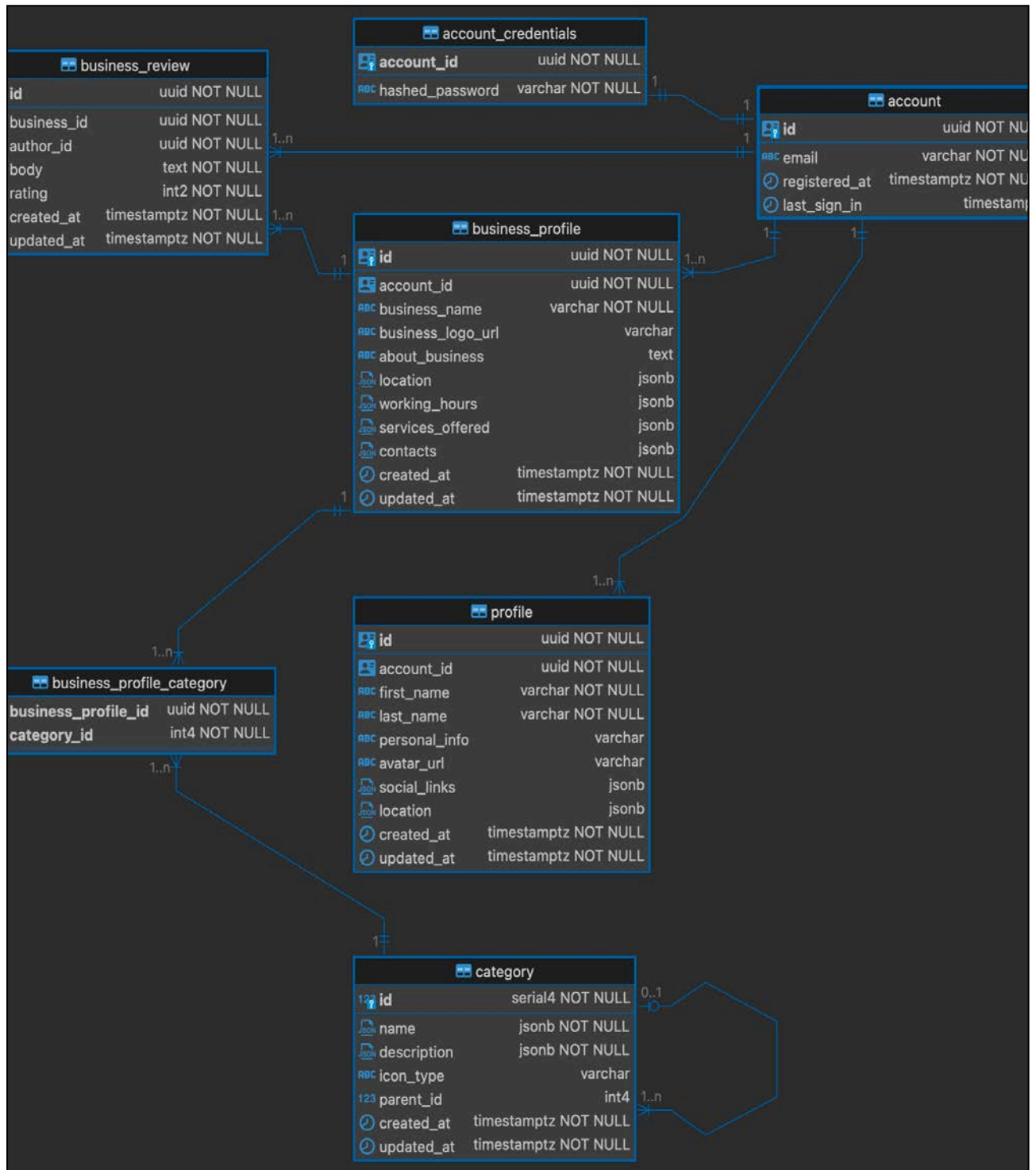


Рисунок 3.5 – Діаграма атрибутів сутностей і зв'язків між ними

Схема бази даних складається з семи таблиць, які пов'язані зв'язками один-до-одного та один-до-багатьох.

В табл. 3.1 наведено перелік і опис основних таблиць і даних в них, що зберігаються в PostgreSQL.

Таблиця 3.1 – Опис основних таблиць в PostgreSQL

Назва таблиці	Опис та призначення
Account	Містить дані про акаунти користувачів.
Account Credentials	Зберігає захешовані паролі користувачів.
Profile	Зберігає додаткові дані про користувача, такі як: ім'я та прізвище, посилання на соц. мережі, тощо.
Business Profile	Містить дані про бізнес, такі як: назва бізнесу, опис, місцезнаходження, сервіси, що надає, робочі години, контакти, тощо.
Business Review	Містить дані відгуку: оцінку та текст відгуку.
Business Profile Category	Містить зв'язки між таблицями Business Profile та Category.
Category	Зберігає інформацію про існуючі категорії в системі, їх назву, опис та іконку.

Для збереження та швидкого отримання і валідації токенів авторизації використовується key-value база даних Redis. Redis працює як in-memory база даних, але також зберігає дані на диск, на відміну від Memcached, який може слугувати тільки як кеш-сховище.

3.2 Розробка та оформлення вебінтерфейсу користувача

Для розробки візуальних компонентів користувацького інтерфейсу була використана бібліотека компонентів (UI Kit) Mantine для React. В бібліотеці є безліч різних компонентів та прикладів їх використання, які добре вписуються в тематику програмного продукту.

Також для перекладу тексту на інтерфейсі було використано бібліотеку `i18next`. Бібліотека дозволяє описати переклади заголовків та інших текстових компонентів через JSON файли та використати їх в компонентах інтерфейсу (рис. 3.6).

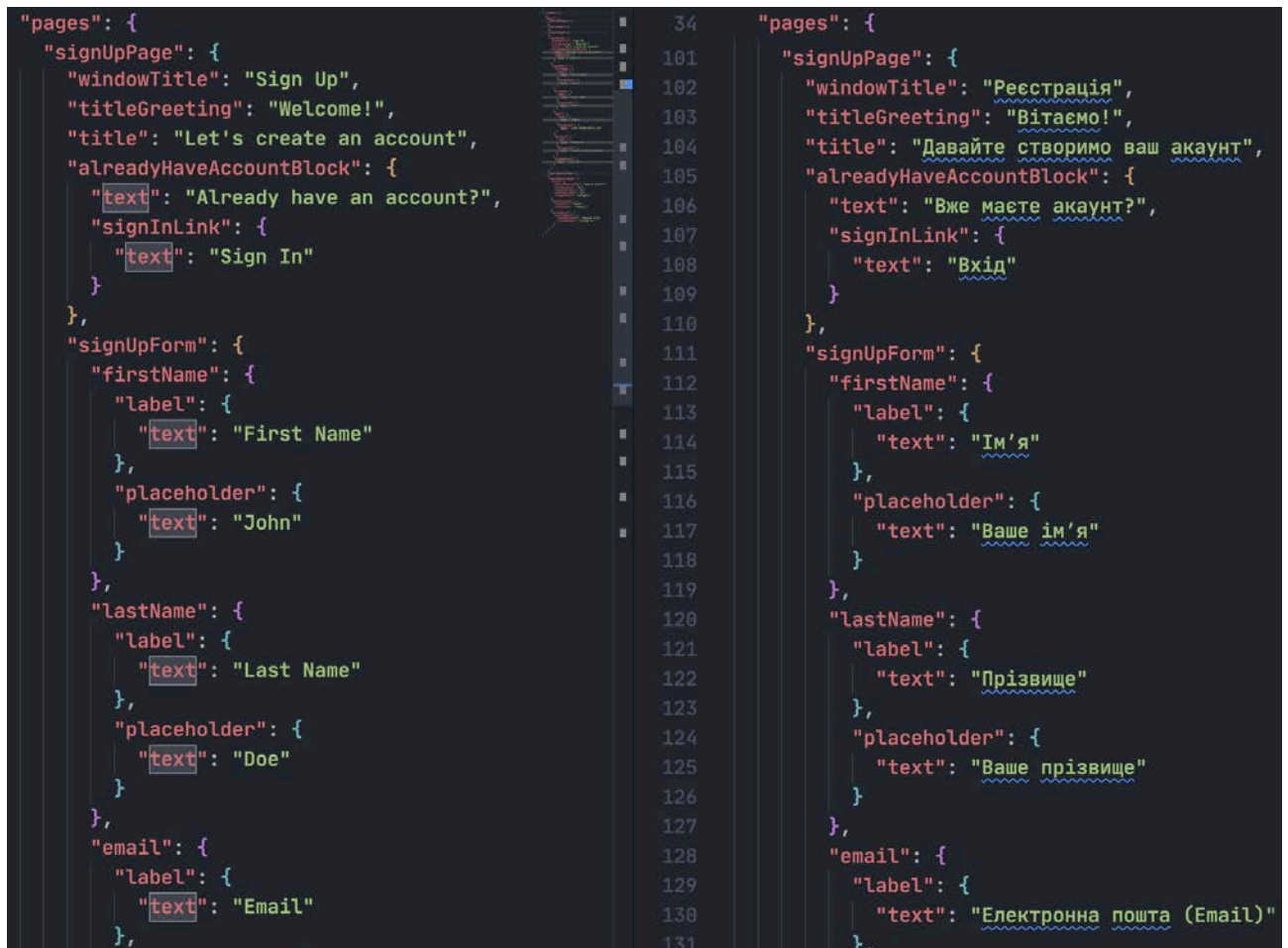


Рисунок 3.6 – Приклад файлів перекладу

Для створення AJAX запитів, використовується бібліотека `RTK Query`, що є частиною `Redux Toolkit` для управління станом застосунку. Завдяки `RTK Query` є можливість кешувати відповіді з сервера, що зменшує навантаження на вебсервер та робить взаємодію з застосунком більш плавною.

І останнім важливим компонентом клієнтської частини застосунку є `React Dom Router`, який дозволяє перенаправляти користувача на необхідні сторінки, без повного перезавантаження сторінки в браузері, що теж допомагає зробити взаємодію з інтерфейсом більш плавним. На рис. 3.7 зображено

механізм реєстрації компонентів, які будуть відображатися при переході по вказаним шляхам.

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    errorElement: <NotFoundPage />,
    children: [
      {
        path: "",
        element: <SearchPage />,
      },
      {
        path: "search",
        element: <SearchResultsPage />,
      },
      {
        path: "business/:resultId",
        element: <SearchResultPage />,
      },
      {
        path: "signin",
        element: <SignInPage />,
      },
      {
        path: "signup",
        element: <SignUpPage />,
      },
      {
        path: "create-business",
        element: <CreateBusinessProfile />,
      },
      {
        path: "help",
        element: <HelpPage />,
      },
    ],
  },
],
```

Рисунок 3.7 – Приклад реєстрації шляхів на компоненти сторінок

3.3 Файлова структура проєкту

Загальна файлова структура проєкту представлена на рис. 3.8. Найбільш важливими директоріями в проєкті є `app`, `build`, `web` та `database`.

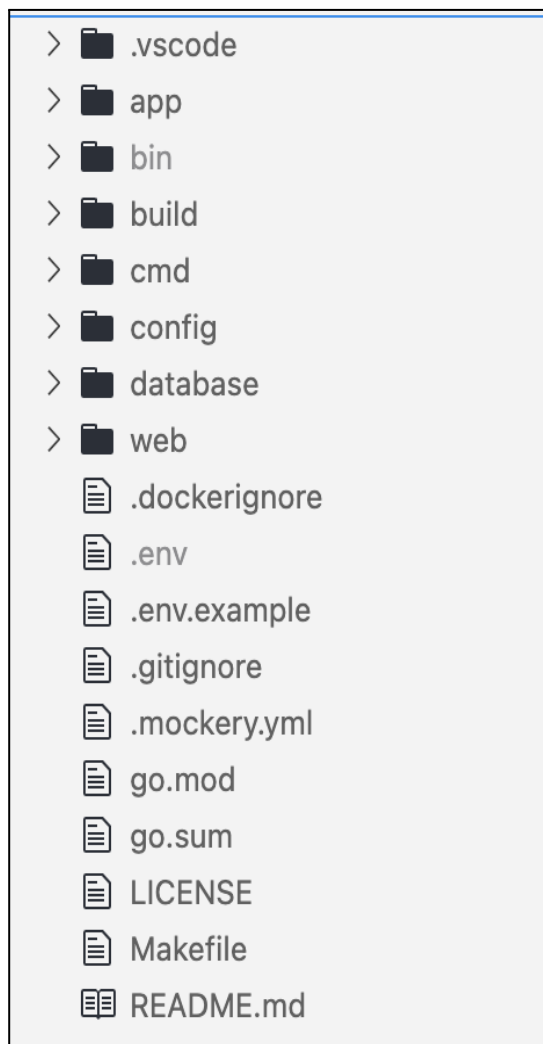


Рисунок 3.8 – Загальна файлова структура проєкту

Директорія `app` містить реалізацію вебсерверу, бізнес-логіки та взаємодію з базами даних. `build` директорія містить файли конфігурації для розгортання застосунку. `Web` директорія слугує як окремим підпроєктом для клієнтської частини застосунку. `Database` директорія – це директорія, що містить файли міграцій бази даних, тобто створення і модифікації схеми даних в БД,

файли ініціалізації основних даних в застосунку, наприклад, список високорівневих і дочірніх категорій.

Нижче, на рис. 3.9, представлена файлова структура директорії app, тобто серверної частини застосунку.

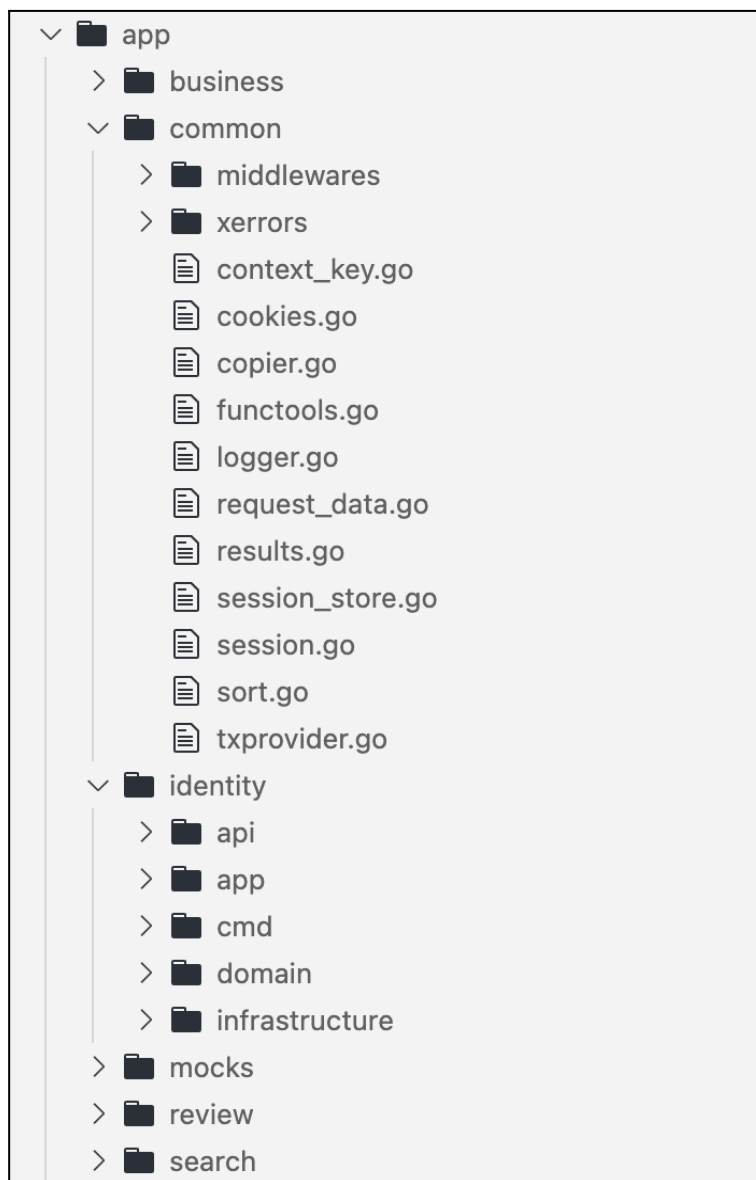


Рисунок 3.9 – Файлова структура серверної частини на прикладі модуля Identity та Common

На рисунку можемо бачити, що директорія app складається з декількох інших директорій, кожна з яких відповідає або за основний модуль застосунку, або за допоміжний модуль, що використовується основними.

Кожен з основних модулів складається з інших директорій, як:

- `api`, що містить логіку обробки HTTP запитів, тобто класи типу `Controller`;
- `app`, який містить класи типу `Service`, що містять основну логіку застосунку;
- `cmd` – для створення об'єктів класів `Controller`, `Service`, `Repository` та інших, що використовуються у модулі;
- `domain` містить класи сутностей;
- `infrastructure` слугує місцем, де зберігаються файли реалізацій інтерфейсів для взаємодії з інфраструктурою застосунку, як от: бази даних, брокери системних повідомлень, логування, моніторинг, тощо.

Як вже було зазначено, клієнтська частина застосунку знаходиться в директорії `web`, файлова структура якої зображена на рис. 3.10.

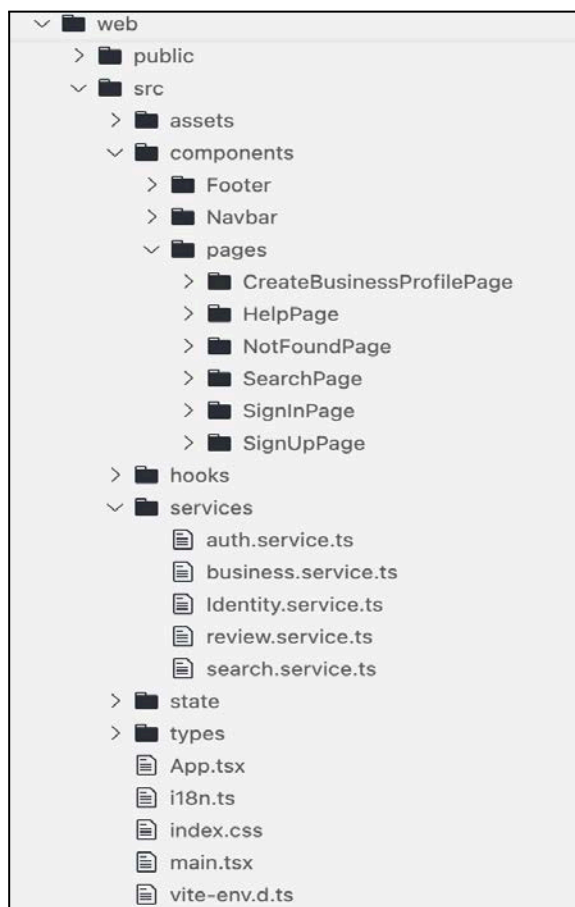


Рисунок 3.10 – Файлова структура клієнтської частини

Основною директорією є `src`, що містить усі файли клієнтської частини. В ній є директорія `components`, в якій зберігаються файли усіх основних компонентів, тобто сторінок застосунку, а також допоміжні компоненти, що використовуються на цих сторінках.

Директорія `services` слугує для зберігання функцій, що дозволяють взаємодіяти з вебсервером з компонентів.

`Types` директорія містить класи та інтерфейси, які використовуються в `services` для передачі даних з вебсервера до компонентів і навпаки.

3.4 Висновки за розділом 3

Наведено детальний опис основних компонентів розроблюваної програмної системи, їх взаємодію, взаємодію клієнтської частини застосунку із серверною та основними рішеннями щодо безпеки даних користувачів. Наведено діаграми, що показують структуру компонентів в програмній системі, рух даних між процесами та зв'язки між головними сутностями в системі.

Описано основні рішення щодо дизайну вебінтерфейсу користувача, використаних бібліотек та організації файлів і директорій у проєкті.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування

Розроблений програмний продукт призначений для зручного та ефективного пошуку компаній-підрядників для виконання тих чи інших завдань клієнта.

У програмному продукті реалізовано наступні можливості:

- пошук компаній-підрядників за ключовими словами, місцезнаходженням та категорією;
- залишення та перегляд відгуків на сторінці підрядника;
- можливість оцінити підрядника та переглянути оцінки інших користувачів і переглянути середню оцінку по підряднику;
- реєстрація акаунту та створення власного профілю;
- реєстрація власного бізнес-профілю.

4.2 Умови виконання програми

Для коректної роботи програмного продукту на стороні клієнту повинні виконуватися наступні умови до апаратної та програмної частини:

- підтримувані операційні системи включають Windows 7 та вище, macOS 10.10 та вище, Linux (різні дистрибутиви);
- браузер з підтримкою HTML версії 5 або пізніше, CSS3, ECMAScript 6 або пізніше;
- клавіатура і миша або аналогічний пристрій введення.

Для коректної роботи програмного продукту на стороні сервера повинні виконуватися наступні умови до апаратної та програмної частини:

- чотирьохядерний процесор з тактовою частотою 2,5 ГГц або вище;
- 4 ГБ ОЗП або більше;
- 25 ГБ або вище вільного місця на диску;

- підтримка Gigabit Ethernet для швидкої передачі даних між сервером і клієнтами;
- підтримувані операційні системи включають Linux (найкраще підходить для використання Docker), такі як Ubuntu, CentOS, Debian тощо;
- потрібно мати встановлену останню версію Docker для контейнеризації та управління програмними контейнерами;
- для належного моніторингу та управління сервером рекомендується мати доступ до монітора або інтерфейсу для віддаленого керування.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Перед тим як користувачі зможуть взаємодіяти з вебінтерфейсом програмного застосунку, необхідно запустити сервер для надання статичних даних, таких як файли HTML, CSS, JS, які використовуються для відображення інтерфейсу. Також потрібно запустити сервер для обробки HTTP запитів, до якого звертатиметься клієнтська частина застосунку, а також інші необхідні сервіси. Для розгортання всіх компонентів програмної системи використовується Docker. Результат розгортання зображено на рис. 4.1.

```

$ docker-compose -f ./build/local/docker-compose.yml up -d
[+] Running 9/9
✓ Volume "moja-posluga-elasticsearch"    Created    0.0s
✓ Volume "moja-posluga-postgres"         Created    0.0s
✓ Volume "moja-posluga-redis"            Created    0.0s
✓ Volume "moja-posluga-rabbitmq"         Created    0.0s
✓ Container moja-posluga-postgres        Started    0.1s
✓ Container moja-posluga-redis           Started    0.1s
✓ Container moja-posluga-rabbitmq        Started    0.1s
✓ Container moja-posluga-elasticsearch   Started    0.1s
✓ Container moja-posluga-kibana          Started    0.0s

```

Рисунок 4.1 – Результат розгортання компонентів системи через Docker

Після успішного розгортання усіх компонентів системи, взаємодія користувачів з програмним продуктом має відбуватися через браузер, встановлений на пристрої користувача. Після введення необхідної адреси в рядку для пошуку браузера, відкриється головна сторінка застосунку, де можна почати пошук підрядників.

Головна сторінка програмного сервісу складається лише з заголовку вебзастосунку, де знаходяться основні посилання та кнопки для реєстрації і входу в систему, та простої форми пошуку, що дозволить швидко адаптуватись новим користувачам (рис. 4.2).

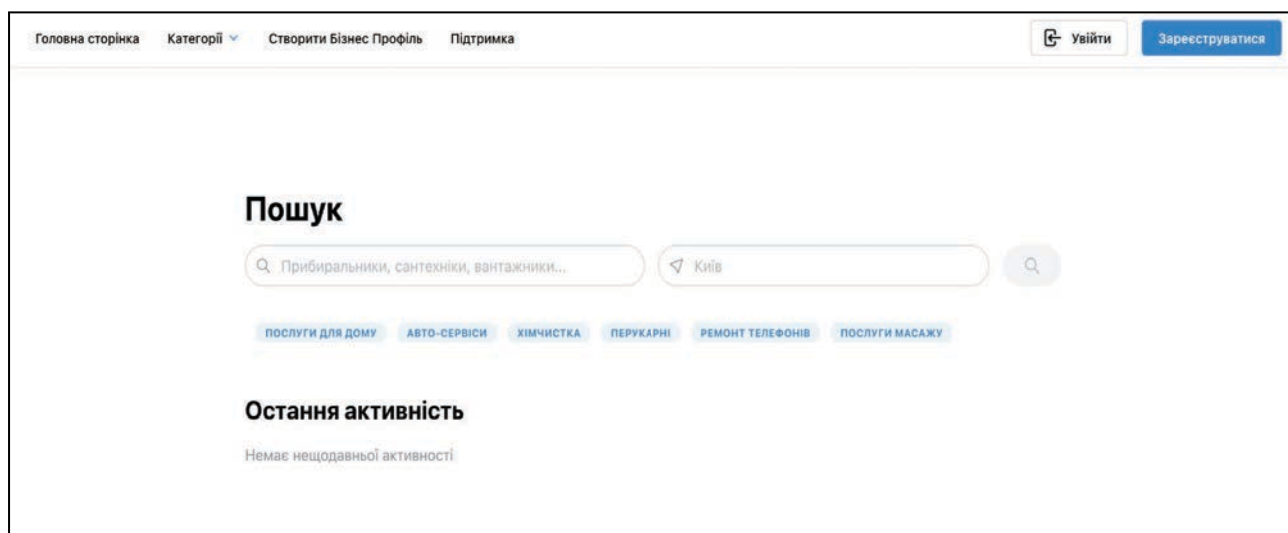


Рисунок 4.2 – Головна сторінка вебзастосунку

Форма пошуку складається з поля вводу запиту та ключових слів, які мають міститись в назві або описі оголошення підрядника, поля вводу назви населеного пункту, де користувач буде шукати оголошення, і списком доступних категорій для пошуку.

На початку, коли користувач ще не обрав категорію для пошуку, показуються лише високорівневі категорії, як під собою мають декілька дочірніх категорій. При пошуку по окремій категорії, в списку категорій для пошуку будуть лише дочірні категорії, що користувач обрав для пошуку.

Щоб залишати відгуки в під оголошеннями компаній-підприємців, треба зареєструватись і увійти в систему. В заголовку вебзастосунку можна знайти кнопку для реєстрації нових користувачів та кнопку авторизації для користувачів, що вже мають зареєстрований профіль.

Після натискання кнопки “Зареєструватися”, користувача перенаправляє на відповідну сторінку з формою реєстрації (рис. 4.3).

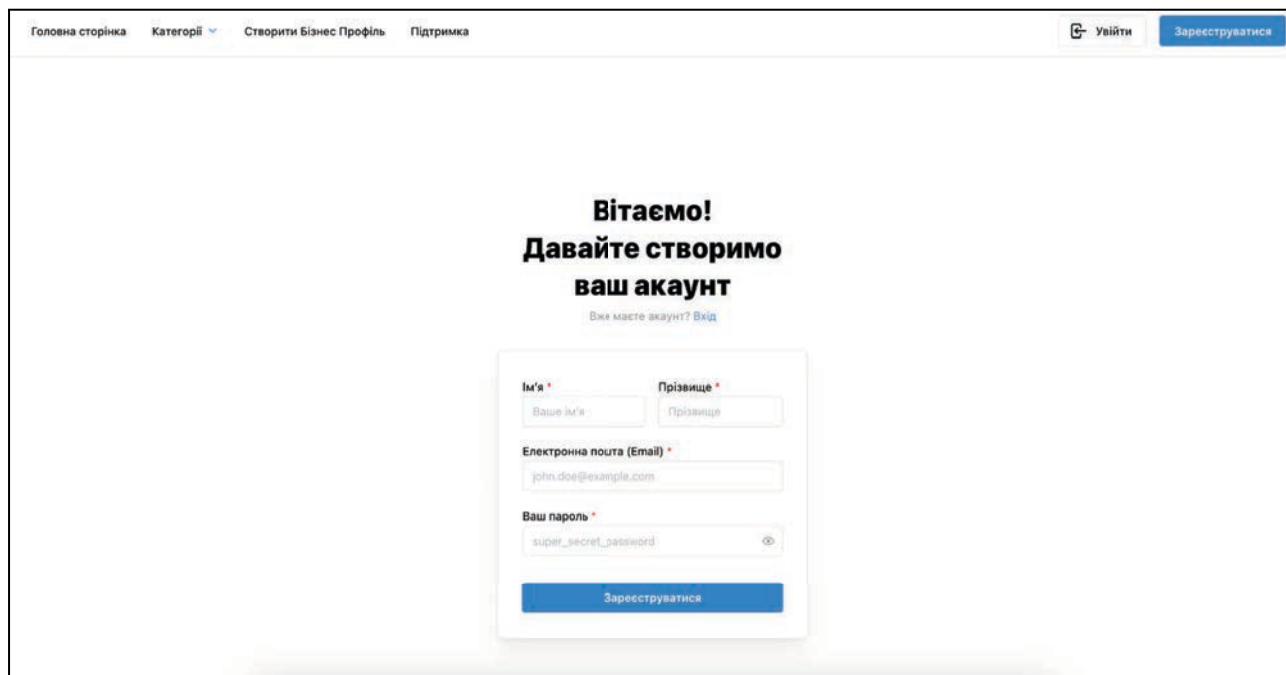


Рисунок 4.3 – Вигляд сторінки створення акаунту та профілю

Форма реєстрації складається з полів вводу імені та прізвища користувача, адреси електронної пошти та паролю, який буде використовуватись при вході в систему. Також є посилання для переходу на форму входу, якщо користувач помилився сторінкою, і вже має акаунт.

Після успішної реєстрації, користувача буде перенаправлено на сторінку входу (рис. 4.4), де потрібно ввести адресу електронної пошти і пароль, що були вказані попередньо на сторінці реєстрації.

The screenshot shows the login interface of a web application. At the top, there is a navigation bar with links: 'Головна сторінка', 'Категорії', 'Створити Бізнес Профіль', and 'Підтримка'. On the right side of the navigation bar are two buttons: 'Увійти' (Login) and 'Зареєструватися' (Register). The main content area features a large heading 'Увійти в Моя Послуга' (Login to My Service) with a sub-link 'Ще не створили акаунт? Зареєструватися' (Don't have an account? Register). Below this is a login form with two input fields: 'Електронна пошта (Email)' (Email) and 'Пароль' (Password). The password field has a toggle icon for visibility. Below the fields are two checkboxes: 'Запам'ятати мене' (Remember me) and 'Забули пароль?' (Forgot password?). A blue 'Увійти' (Login) button is at the bottom of the form.

Рисунок 4.4 – Вигляд сторінки авторизації

Після успішної авторизації в системі, користувача буде перенаправлено на головну сторінку, де вже можна починати пошук оголошень підрядників.

Один з способів пошуку оголошень в застосунку – це пошук по категоріям. В програмному застосунку є два рівні категорій – високорівневі та дочірні. Різниця полягає в тому, що високорівневі категорії не належать до жодної з категорій, а дочірні – навпаки, мають належати до однієї з високорівневих категорій, для більш детального класифікування оголошень.

Щоб побачити список усіх високорівневих категорій, що доступні в програмному застосунку, у заголовку знаходиться посилання, що при натисканні показує список категорій з їх коротким описом і невеликою іконкою, для візуалізації і інтуїтивного розуміння теми оголошень в категорії (рис. 4.5).

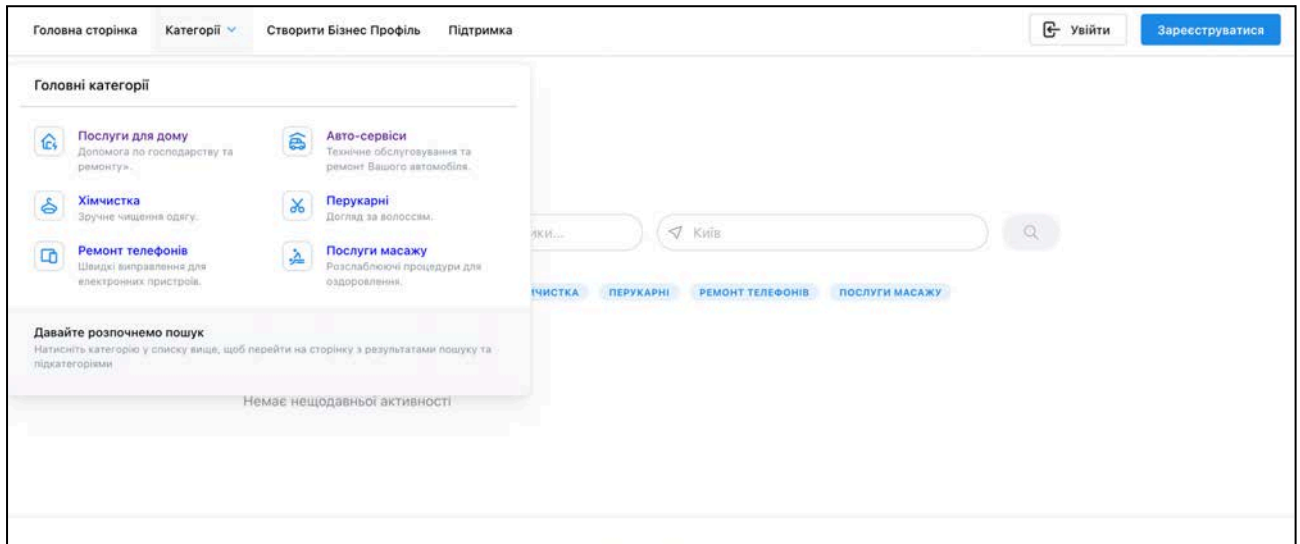


Рисунок 4.5 – Список основних категорій

Після натискання користувачем на одну з категорій, завантажиться сторінка з результатами пошуку в обраній категорії (рис. 4.6).

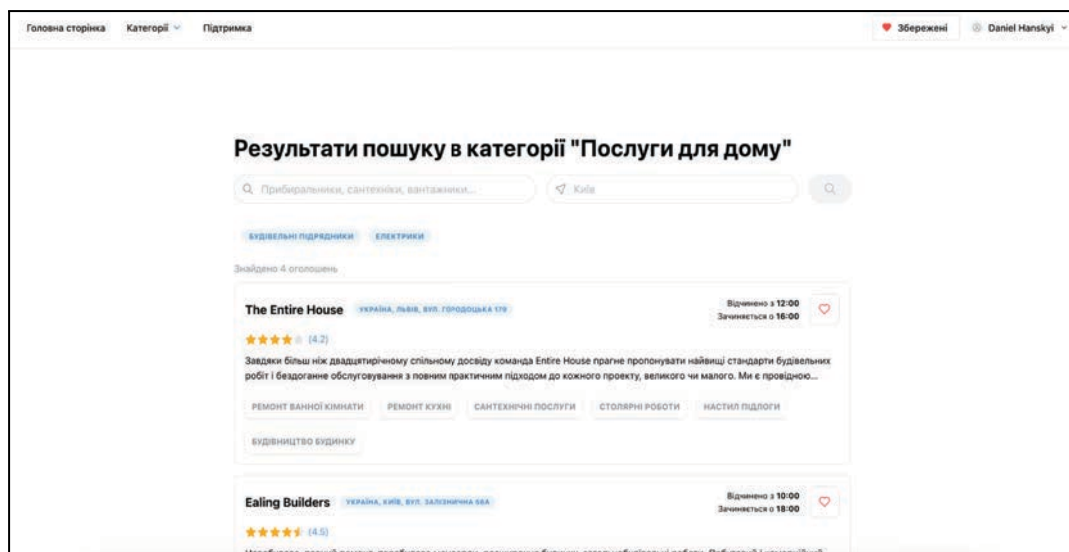


Рисунок 4.6 – Список результатів пошуку по категорії

На сторінці результатів пошуку по категорії можемо бачити ту саму форму, що була присутня на головній сторінці, а також з оновленим списком категорій, що є дочірніми до тієї, що була обрана користувачем попередньо.

Нижче виводяться самі оголошення підрядників, що були розміщені в категорії, обраній для пошуку.

В кожному оголошенні присутня назва компанії-підрядника, короткий опис діяльності, послуги, що надає підрядник, адреса, робочі години в день перегляду оголошення, а також рейтинг компанії, розрахований на основі опублікованих відгуків користувачами під оголошенням підрядника. Для більш детального ознайомлення, користувач може натиснути на оголошення в списку, що його зацікавило, після чого буде завантажена окрема сторінка з додатковою інформацією про підрядника, послуги, що надаються, та списком відгуків від інших користувачів.

Як було вже зазначено, пошук по категорії є лише одним з способів пошуку оголошень підрядників. Іншим способом можна виділити – пошук за ключовими словами і, додатково, за місцезнаходженням. Запит може бути уточнюючим, по попередньо обраній категорії, або зроблений з головної сторінки застосунку, без фільтру по категорії (рис 4.7).



Рисунок 4.7 – Результати пошуку тільки по ключовим словам і місцезнаходженню

Після натискання на будь-яке з оголошень в списку результатів, відкриється сторінка з додатковою інформацією по компанії-підряднику (рис 4.8).

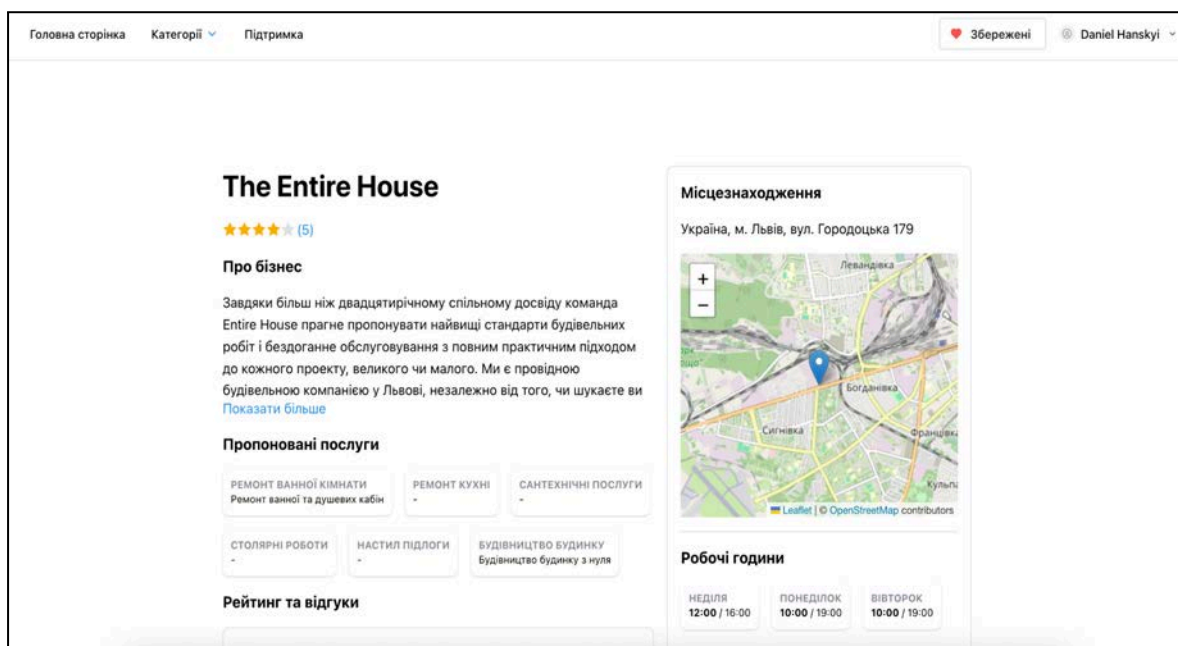


Рисунок 4.8 – Вигляд сторінки з деталями по компанії-підряднику

На рис. 4.8 можна побачити, що уся сторінка складається з двох частин. Перша – це основна інформація про компанію-підрядника, список послуг з коротким описом, що надаються, рейтинг підрядника та список відгуків користувачів. Друга – додаткова інформація про підрядника, як: місцезнаходження, відображене на мапі, робочі години у кожен день тижня (день, у який користувач переглядатиме оголошення, буде виділено додатковою рамкою, як показано на рис. 4.9), та контактні дані підрядника.

На рис. 4.10 відображено форму публікації відгуку. Вона складається з невеликого поля вводу тексту відгуку і компоненту для вводу рейтингу у вигляді п'яти зірочок, при натисканні на які можна обрати рейтинг від одного до п'яти.

Робочі години

НЕДІЛЯ

12:00 / 16:00

ПОНЕДІЛОК

10:00 / 19:00

ВІВТОРОК

10:00 / 19:00

СЕРЕДА

10:00 / 19:00

ЧЕРТВЕР

10:00 / 19:00

П'ЯТНИЦЯ

10:00 / 19:00

СУБОТА

12:00 / 16:00

Контакти

+3800500673828

+3800960633828

Напишіть нам

Веб-сторінка

Рисунок 4.9 – Вигляд блоку сторінки з робочими годинами компанії-підрядника та контактними даними

Рейтинг та відгуки

Відгуків ще немає

Залишити відгук

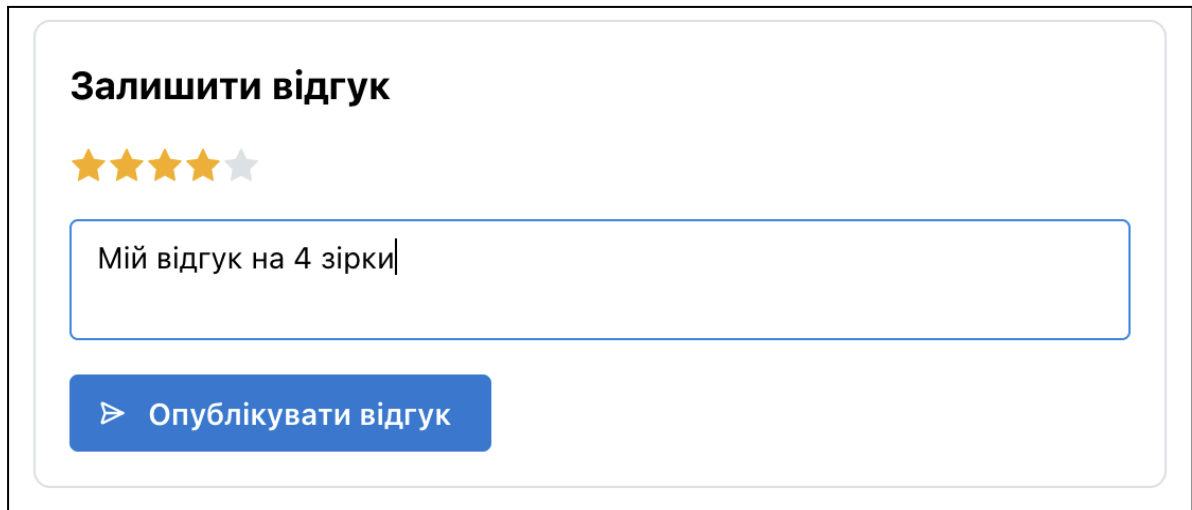
★ ★ ★ ★ ★

Напишіть свої думки тут...

➤ Опублікувати відгук

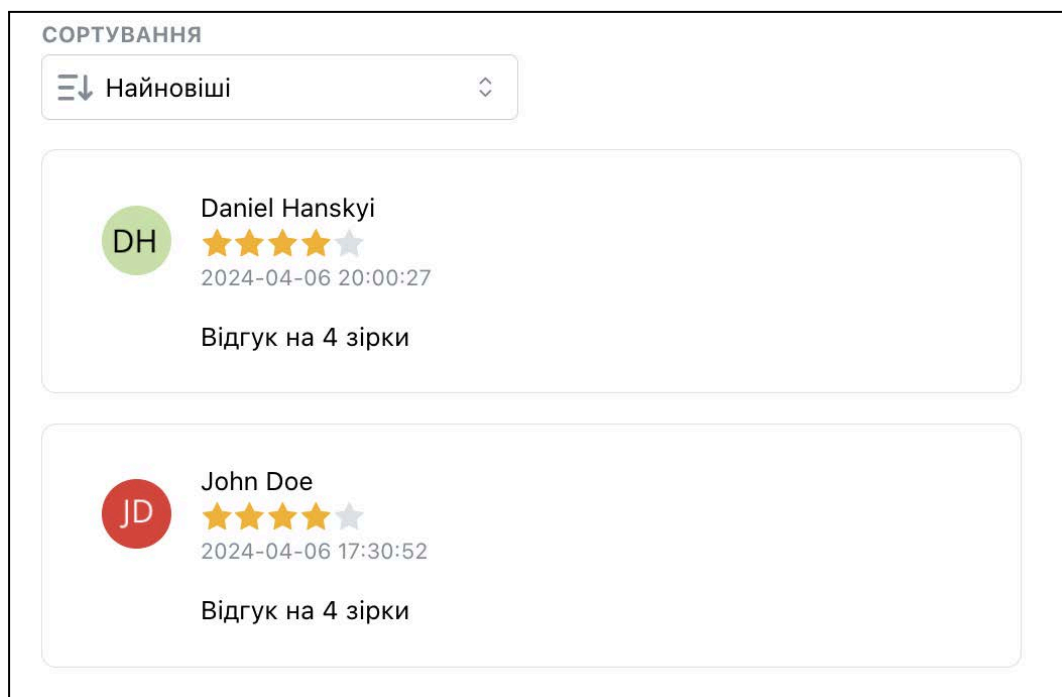
Рисунок 4.10 – Форма опублікування відгуку під оголошенням

Коли поле вводу відгуку заповнене та рейтинг виставлений, можна побачити, що кнопка публікації відгуку стала активною і користувач може її натискати (рис 4.11). Після натискання, відгук користувача буде додано в загальний список (рис 4.12).



The image shows a form titled "Залишити відгук" (Leave a review). It features a 5-star rating system with 4 stars filled and 1 star empty. Below the stars is a text input field containing the text "Мій відгук на 4 зірки" (My review on 4 stars). At the bottom of the form is a blue button with a right-pointing arrow and the text "Опублікувати відгук" (Publish review).

Рисунок 4.11 – Заповнена форма опублікування відгуку



The image shows a list of reviews under the heading "СОРТУВАННЯ" (Sorting). A dropdown menu is set to "Найновіші" (Newest). There are two review entries:

- Review 1:** User "Daniel Hanksyi" (profile picture with initials "DH"), 4 stars, timestamp "2024-04-06 20:00:27", and text "Відгук на 4 зірки" (Review on 4 stars).
- Review 2:** User "John Doe" (profile picture with initials "JD"), 4 stars, timestamp "2024-04-06 17:30:52", and text "Відгук на 4 зірки" (Review on 4 stars).

Рисунок 4.12 – Список відгуків

Додатково, відгуки у списку можна сортувати за часом публікації та за рейтингом, як це показано на рис. 4.13.

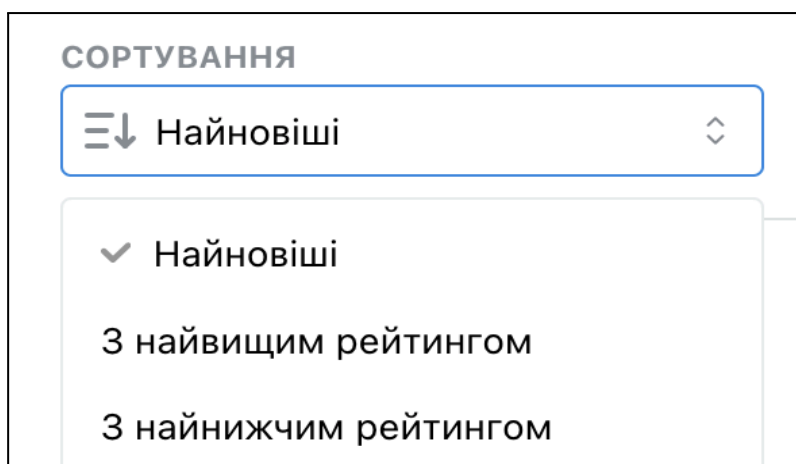


Рисунок 4.13 – Варіанти сортування відгуків

Також, якщо оголошення отримує більше одного відгуку, користувачі можуть переглянути загальну кількість опублікованих відгуків, середній рейтинг і графічне зображення кількості оцінок (рис. 4.14).

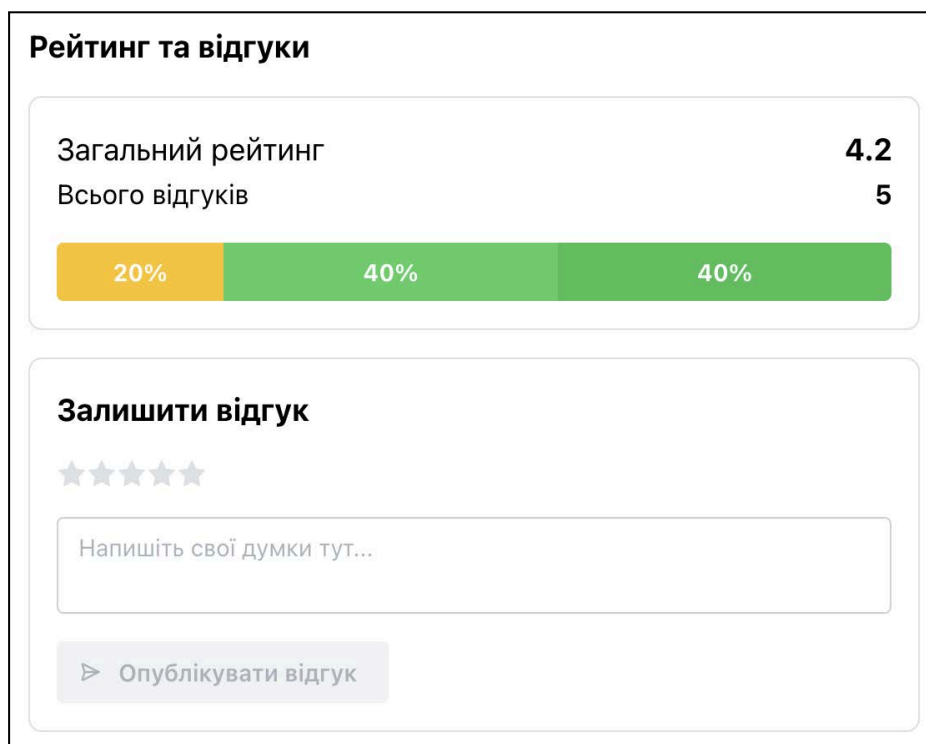


Рисунок 4.14 – Графічне зображення кількості оцінок

Важливим є також, що користувач, аби залишити відгук, має бути авторизований в системі. Для неавторизованих користувачів буде відображатися повідомлення показане на рис. 4.15.

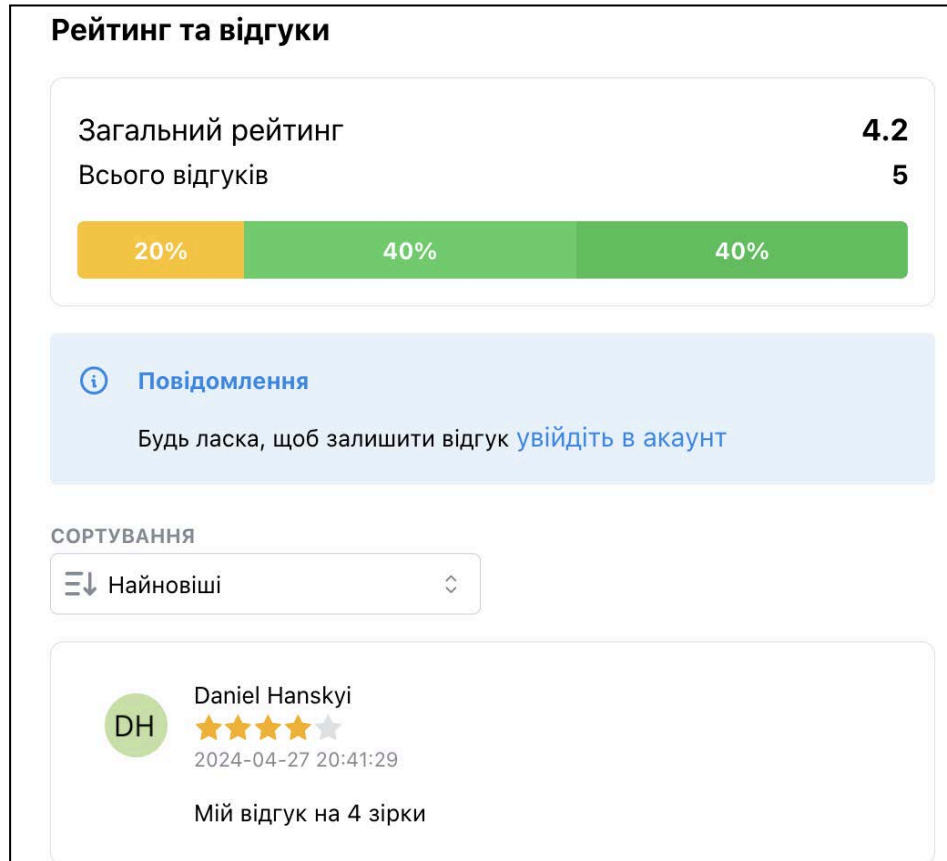


Рисунок 4.15 – Повідомлення, що користувач має бути авторизованим

4.3.2 Вхідні й вихідні дані

Вхідними даними є: пошукові запити користувача, інформація про користувача, введена при реєстрації акаунту та заповнені профілю, текст відгуку та оцінка підряднику, інформація про підрядника, заповнена при створенні бізнес-профілю.

Вихідними даними є: результати пошуку за запитом користувача, список відгуків по підряднику та середня оцінка по підряднику.

4.3.3 Повідомлення

Серед основних типів повідомлень системи до користувача можна виділити наступні:

- повідомлення про некоректність введених даних, як от: повідомлення при реєстрації, що електронна адреса вже використовується в системі, невірний логін або пароль і т.д.;
- повідомлення про успішне створення або оновлення того чи іншого об'єкту, як от: після створення профілю користувача чи бізнес-профілю, залишення відгуку на сторінці підрядника, оновлення профілю та відгуку користувача і т.д.

4.4 Тестування програмного забезпечення

На етапі тестування програмного продукту було проведено декілька різних видів тестування, а саме:

- модульне тестування – тестування окремих класів та функцій для перевірки коректності поведінки при різних вхідних даних;
- інтеграційне тестування – тестування декількох взаємопов'язаних компонентів системи з реальними даними та реальними базами даних та іншими технологіями;
- end-to-end тестування – перевірка роботи всього програмного продукту зі сторони користувача.

Використовуючи ці методи тестування було проведено перевірку процесу реєстрації та авторизації в системі, процесів створення та пошуку підрядників, процесу залишення відгуків та оцінок на сторінках підрядників.

У разі вводу користувачем некоректних даних, серверна частина застосунку повертає відповідний HTTP статус код – 400, із додатковими деталями помилки. Прикладом такої помилки є введення електронної пошти при реєстрації, що не відповідає загальноприйнятому формату.

Помилки, що повертаються з серверу, коректно обробляються вебінтерфейсом і виводяться користувачеві у належних компонентах інтерфейсу.

Під час тестування не було виявлено системних помилок, можливості несанкціонованого доступу до користувацьких даних, проблем пов'язаних з швидкістю роботи. Виходячи з цього можна підсумувати, що програмний продукт працює коректно, відповідає вказаним вимогам та готовий до використання.

4.5 Висновки за розділом 4

У цьому розділі розглянуто призначення та функції розробленого програмного забезпечення. Визначено технічні та програмні вимоги, необхідні для його встановлення та підтримки як на комп'ютері користувача, так і на сервері. Зроблено загальний огляд взаємодії з системою користувачем. Наведено типи і опис вхідних та вихідних даних, а також типи повідомлень для користувача. Проведено тестування програмного продукту на декількох рівнях, яке показало, що система готова для використання користувачами.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено предметну область та програмні засоби для пошуку оголошень компаній-підрядників.

Досліджено і проаналізовано найбільш відомі програмні рішення для пошуку підрядників, а також їх необхідність у сучасному світі. На основі проведеного аналізу, було визначено основні функціональні вимоги до розроблюваного програмного продукту.

Проведено дослідження існуючих мов програмування та супроводжуваних технологій для розробки програмного застосунку для пошуку підрядників, як от: СКБД, база даних для повнотекстового пошуку, інструменти контейнеризації, тощо.

Для серверної частини програми вибрано мову програмування Go (Golang), а для побудови користувацького інтерфейсу використано JavaScript (TypeScript) з фреймворком React.

В якості супроводжуваних технологій було обрано: Elasticsearch як пошуковий двигун, PostgreSQL як реляційну базу даних. Для контейнеризації програмного продукту використовується Docker.

Було створено програмний продукт, що дозволяє користувачам швидко та ефективно знаходити підрядників для виконання потрібних завдань, незалежно від їх типу, за допомогою ключових слів, місцезнаходження користувача та рейтингу. Крім того, в програмному продукті реалізована можливість перегляду та залишення відгуків на оголошеннях підрядників.

Під час перевірки не виявлено жодних системних помилок, можливих порушень безпеки щодо доступу до користувацьких даних або проблем з швидкістю обробки запитів та загальною продуктивністю додатку. З цього можна зробити висновок, що програмний продукт працює коректно, відповідає вимогам і готовий до використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Paget S. Local Consumer Review Survey 2024: Trends, Behaviors, and Platforms Explored [Electronic resource]. – Access mode: <https://www.brightlocal.com/research/local-consumer-review-survey>.
2. Підрядник [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Підрядник>.
3. Любіцева О. О. Ринок туристичних послуг (геопросторові аспекти) [Текст] / О. О. Любіцева. – К.: Альтерпрес, 2002. – 436 с.
4. Окраку Е., WHAT IS YELP: How to Use It & Guidelines [Electronic resource]. – Access mode: <https://businessyield.com/bs-business/what-iss-yelp>.
5. Yelp [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Yelp>.
6. Add, edit, or delete Google Maps reviews & ratings [Electronic resource]. – Access mode: <https://support.google.com/maps/answer/6230175>.
7. OLX, Про нас [Електрон. ресурс]. – Режим доступу: <https://blog.olx.ua/pro-nas-uk>.
8. Локальний е-commerce: чим українці торгують у мережі [Електрон. ресурс]. – Режим доступу: <https://mind.ua/publications/20180858-lokalnij-e-commerce-chim-ukrayinci-torguyut-u-merezhi>.
9. OLX, [Електрон. ресурс]. – Режим доступу: <https://olx.ua>.
10. Kabanchik.ua, Про проєкт [Електрон. ресурс]. – Режим доступу: <https://kabanchik.ua/ua/about>.
11. Kabanchik.ua, [Електрон. ресурс]. – Режим доступу: <https://kabanchik.ua>.
12. Go [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/Go_\(programming_language\)](https://en.wikipedia.org/wiki/Go_(programming_language)).
13. Go Case Studies [Electronic resource]. – Access mode: <https://go.dev/solutions/case-studies>.

14. JavaScript [Electronic resource]. – Access mode:
<https://en.wikipedia.org/wiki/JavaScript>.
15. React [Electronic resource]. – Access mode:
[https://en.wikipedia.org/wiki/React_\(software\)](https://en.wikipedia.org/wiki/React_(software)).
16. Angular [Electronic resource]. – Access mode:
[https://en.wikipedia.org/wiki/Angular_\(web_framework\)](https://en.wikipedia.org/wiki/Angular_(web_framework)).
17. VueJS [Electronic resource]. – Access mode:
<https://en.wikipedia.org/wiki/Vue.js>.
18. Svelte [Electronic resource]. – Access mode:
<https://en.wikipedia.org/wiki/Svelte>.
19. PostgreSQL [Electronic resource]. – Access mode:
<https://en.wikipedia.org/wiki/PostgreSQL>.
20. Elasticsearch [Electronic resource]. – Access mode:
<https://uk.wikipedia.org/wiki/Elasticsearch>.
21. The heart of the free and open Elastic Stack: Elasticsearch [Electronic resource]. – Access mode: <https://www.elastic.co/products/elasticsearch>.
22. Docker [Electronic resource]. – Access mode:
[https://en.wikipedia.org/wiki/Docker_\(software\)](https://en.wikipedia.org/wiki/Docker_(software)).
23. Golang [Electronic resource]. – Access mode:
<https://www.jetbrains.com/go>.

ДОДАТОК А
Технічне завдання

Вступ

Програмний застосунок для пошуку компаній-підрядників призначений для пошуку, перегляду і оцінки оголошень підрядників.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка вебзастосунку для пошуку і оцінки компаній-підрядників», затверджене наказом Національного університету «Запорізька політехніка» № 168 від 24 квітня 2024 р.

A.2 Призначення розробки

Програмний застосунок для пошуку компаній-підрядників спрямований на полегшення процесу пошуку підрядників шляхом надання мінімалістичного та інтуїтивно зрозумілого інтерфейсу, при цьому зберігаючи повний функціонал.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Розроблений програмний продукт має реалізовувати наступні функції:

- пошук оголошень підрядників для авторизованих та неавторизованих користувачів;
- можливість пошуку оголошень через ключові слова, навіть якщо користувач зробив помилку в запиті;
- можливість пошуку оголошень додатково за місцезнаходженням;
- можливість пошуку оголошень за категорією або підкатегорією;

- відображення списку усіх доступних категорій та підкатегорій для пошуку;
- показувати список усіх знайдених оголошень підрядників за вказаними пошуковими критеріями з сортуванням по рейтингу оголошень;
- оголошення в списку результатів пошуку має включати назву, короткий опис, загальний рейтинг та послуги, що надаються;
- на сторінці оголошення має відображатися розгорнута інформація про діяльність підрядника, його рейтинг та відгуки від користувачів;
- створення відгуків під оголошенням підрядників має бути доступним тільки для авторизованих користувачів;
- форма створення відгуку має включати текст відгуку та оцінку;
- після публікації відгуку, користувач має відразу бачити свій відгук у загальному списку відгуків під оголошенням.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс повинен бути легким у використанні та інтуїтивно зрозумілим для користувачів будь-якого рівня технічної освіти. Мінімізація кількості кроків для досягнення мети допоможе забезпечити позитивний досвід користувача.

Усі елементи інтерфейсу повинні мати однаковий стиль та розміщення, щоб забезпечити консистентність та зручність для користувача.

A.3.3 Вимоги до надійності

Застосунок повинен бути стійким до відмов, тобто повинен продовжувати працювати при виникненні критичних помилок або непередбачених умов.

Вебзастосунок повинен мати вбудовані заходи безпеки, які захищають його від різних видів атак, таких як SQL-ін'єкція, перехоплення сесій, кросс-сайтові атаки (XSS) та інші.

Всі дані, що обробляються в застосунку, повинні бути захищені від випадкових чи навмисних змін або втрати.

Зберігання паролів та токенів сесій користувачів має відповідати всім сучасним стандартам у веброзробці.

Потрібно мати можливість легко масштабувати вебзастосунок, щоб сервер міг витримувати зростання обсягу даних і навантаження.

A.3.4 Умови експлуатації

Для користування розробленим веб-застосунком користувачам необхідно мати пристрій з оновленою версією одного з популярних веббраузерів і стабільне з'єднання з мережею Інтернет. Щоб залишати відгуки, потрібно створити акаунт.

A.3.5 Вимоги до складу та параметрів технічний засобів

Для коректної роботи програмного продукту на стороні клієнту повинні виконуватися наступні умови до апаратної та програмної частини:

- підтримувані операційні системи включають Windows 7 та вище, macOS 10.10 та вище, Linux (різні дистрибутиви);
- браузер з підтримкою HTML версії 5 або пізніше, CSS3, ECMAScript 6 або пізніше;
- клавіатура і миша або аналогічний пристрій введення.

Для коректної роботи програмного продукту на стороні сервера повинні виконуватися наступні умови до апаратної та програмної частини:

- чотирьохядерний процесор з тактовою частотою 2,5 ГГц або вище;
- 4 ГБ ОЗП або більше;
- 25 ГБ або вище вільного місця на диску;
- підтримка Gigabit Ethernet для швидкої передачі даних між сервером і клієнтами;
- підтримувані операційні системи включають Linux (найкраще підходить для використання Docker), такі як Ubuntu, CentOS, Debian тощо;
- потрібно мати встановлену останню версію Docker для контейнеризації та управління програмними контейнерами;
- для належного моніторингу та управління сервером рекомендується мати доступ до монітора або інтерфейсу для віддаленого керування.

A.3.6 Вимоги до маркування і пакування

Програмне забезпечення не потребує маркування та упакування.

A.4 Вхідні та вихідні дані до роботи

Вхідними даними є: пошукові запити користувача, інформація про користувача, введена при реєстрації акаунту та заповнені профілю, текст відгуку та оцінка підряднику, інформація про підрядника, заповнена при створенні бізнес-профілю.

Вихідними даними є: результати пошуку за запитом користувача, список відгуків по підряднику та середня оцінка по підряднику.

ДОДАТОК Б
Текст програми

```

package v1 http

type SearchController struct {
    searchSvc *app.SearchResultService
    categorySvc *app.CategoryService
}

func NewSearchController(
    searchSvc *app.SearchResultService,
    categorySvc *app.CategoryService,
) *SearchController {
    return &SearchController{
        searchSvc: searchSvc,
        categorySvc: categorySvc,
    }
}

func (c *SearchController) SetupRoutes(router *echo.Echo) {
    group := router.Group("/search/v1")

    group.GET("", c.search)
    group.GET("/:resultId", c.getSearchResult)

    group.GET("/categories", c.getCategories)
    group.GET("/categories/:categoryId", c.getCategory)
}

func (c *SearchController) search(ctx echo.Context) error {
    var req searchRequest

    if err := ctx.Bind(&req); err != nil {
        return err
    }

    reqCtx := ctx.Request().Context()
    params := app.SearchResultFilterDTO(req)

    searchResults, totalCount, err := c.searchSvc.Search(reqCtx, params)
    if err != nil {
        return err
    }

    var searchResultsResp []searchResultResponse
    if err = common.Copy(searchResults, &searchResultsResp); err != nil {
        return err
    }

    resp := common.Results[searchResultResponse]{
        TotalCount: totalCount,
        Items: searchResultsResp,
    }
    return ctx.JSON(http.StatusOK, resp)
}

func (c *SearchController) getSearchResult(ctx echo.Context) error {
    var req getSearchResultRequest

    if err := ctx.Bind(&req); err != nil {
        return err
    }

    reqCtx := ctx.Request().Context()

```

```

        searchResult, err := c.searchSvc.FindOne(reqCtx, req.ResultID)
        if err != nil {
            return err
        }

        var resp searchResultResponse
        if err = common.Copy(searchResult, &resp); err != nil {
            return err
        }
        return ctx.JSON(http.StatusOK, resp)
    }

func (c *SearchController) getCategories(ctx echo.Context) error {
    var req getCategoryRequest

    if err := ctx.Bind(&req); err != nil {
        return err
    }

    reqCtx := ctx.Request().Context()
    dto := app.CategoryFilterDTO(req)

    categories, totalCount, err := c.categorySvc.FindAll(reqCtx, dto)
    if err != nil {
        return err
    }

    var categoriesResp []categoryResponse
    if err = common.Copy(categories, &categoriesResp); err != nil {
        return err
    }

    resp := common.Results[categoryResponse]{
        Items:    categoriesResp,
        TotalCount: totalCount,
    }
    return ctx.JSON(http.StatusOK, resp)
}

func (c *SearchController) getCategory(ctx echo.Context) error {
    var req getCategoryRequest

    if err := ctx.Bind(&req); err != nil {
        return err
    }

    category, err := c.categorySvc.FindOne(ctx.Request().Context(), req.ID)
    if err != nil {
        return err
    }

    var categoryResp categoryResponse
    if err = common.Copy(category, &categoryResp); err != nil {
        return err
    }
    return ctx.JSON(http.StatusOK, categoryResp)
}

package app

type SearchResultService struct {
    searchResultRepo domain.SearchResultRepository
    categoryRepo     domain.CategoryRepository
}

```

```

}

func NewSearchService(
    searchResultRepo domain.SearchResultRepository,
    categoryRepo domain.CategoryRepository,
) *SearchResultService {
    return &SearchResultService{
        searchResultRepo: searchResultRepo,
        categoryRepo: categoryRepo,
    }
}

func (svc *SearchResultService) Search(
    ctx context.Context,
    dto SearchResultFilterDTO,
) ([]SearchResultDTO, int64, error) {
    filter := MapSearchResultFilterFromDTO(dto)

    childCategories, _, err := svc.categoryRepo.FindAll(ctx, domain.CategoryFilter{
        ParentIDs: filter.CategoryIDs,
    })
    if err != nil {
        return nil, 0, err
    }

    categoryIDs := lo.Map(childCategories,
        func(category domain.Category, _ int) int64 {
            return category.ID
        })

    categoryIDs = append(categoryIDs, filter.CategoryIDs...)
    filter.CategoryIDs = lo.Uniq(categoryIDs)

    results, totalCount, err := svc.searchResultRepo.FindAll(ctx, filter)
    if err != nil {
        return nil, 0, err
    }

    var resultsDTOs []SearchResultDTO
    if err = common.Copy(results, &resultsDTOs); err != nil {
        return nil, 0, err
    }
    return resultsDTOs, totalCount, nil
}

func (svc *SearchResultService) FindOne(
    ctx context.Context,
    resultID string,
) (SearchResultDTO, error) {
    resultUUID, err := uuid.Parse(resultID)
    if err != nil {
        return SearchResultDTO{}, err
    }

    result, err := svc.searchResultRepo.FindOne(ctx, resultUUID)
    if err != nil {
        return SearchResultDTO{}, err
    }

    var resultDTO SearchResultDTO
    if err = common.Copy(result, &resultDTO); err != nil {
        return SearchResultDTO{}, err
    }
}

```

```

        return resultDTO, nil
    }

func (svc *SearchResultService) Save(ctx context.Context, dto SaveSearchResultDTO) error {
    resultUUID, err := uuid.Parse(dto.ID)
    if err != nil {
        return err
    }

    result, err := svc.searchResultRepo.FindOne(ctx, resultUUID)
    if err != nil && !errors.Is(err, xerrors.ErrNotFound) {
        return err
    }

    result.ID = dto.ID
    result.BusinessName = dto.BusinessName
    result.BusinessLogoURL = dto.BusinessLogoURL
    result.CategoryIDs = dto.Categories
    result.AboutBusiness = dto.AboutBusiness

    result.Location = (*domain.Location)(dto.Location)
    result.Contacts = (*domain.Contacts)(dto.Contacts)

    workingHours := make(domain.WorkingHours)
    if err = common.Copy(dto.WorkingHours, &workingHours); err != nil {
        return err
    }

    var servicesOffered []domain.Service
    if err = common.Copy(dto.ServicesOffered, &servicesOffered); err != nil {
        return err
    }

    result.WorkingHours = workingHours
    result.ServicesOffered = servicesOffered

    result.CreatedAt = dto.CreatedAt
    result.UpdatedAt = dto.UpdatedAt
    return svc.searchResultRepo.Save(ctx, result)
}

func (svc *SearchResultService) SaveRating(ctx context.Context, dto SaveRatingDTO) error {
    limit := 1

    filter := domain.SearchResultFilter{
        Keywords: []string{dto.BusinessID},
        Limit:    &limit,
    }

    results, totalCount, err := svc.searchResultRepo.FindAll(ctx, &filter)
    if err != nil {
        return err
    }
    if totalCount == 0 {
        return xerrors.ErrNotFound
    }

    result := results[0]
    result.Rating = &dto.Rating
    return svc.searchResultRepo.Save(ctx, result)
}

func (svc *SearchResultService) Delete(ctx context.Context, resultID string) error {

```

```

        return svc.searchResultRepo.Delete(ctx, resultID)
    }

package infrastructure

type SearchResultRepository struct {
    client *elasticsearch.TypedClient
    indexName string
}

func NewSearchResultRepository(
    client *elasticsearch.TypedClient,
    indexName string,
) *SearchResultRepository {
    return &SearchResultRepository{
        client: client,
        indexName: indexName,
    }
}

func (c *SearchResultRepository) FindAll(
    ctx context.Context,
    filter *domain.SearchResultFilter,
) ([]domain.SearchResult, int64, error) {
    query := types.Query{
        Bool: &types.BoolQuery{},
    }

    if len(filter.Keywords) > 0 {
        query.Bool.Must = append(query.Bool.Must, types.Query{
            MultiMatch: &types.MultiMatchQuery{
                Query: strings.Join(filter.Keywords, " "),
                Fuzziness: "AUTO",
            },
        })
    }

    if len(filter.CategoryIDs) > 0 {
        fieldValues := lo.Map(filter.CategoryIDs,
            func(id int64, _ int) types.FieldValue {
                return id
            })

        query.Bool.Must = append(query.Bool.Must, types.Query{
            Terms: &types.TermsQuery{
                TermsQuery: map[string]types.TermsQueryField{
                    "categoryIds": fieldValues,
                },
            },
        })
    }

    if filter.Location != nil {
        query.Bool.Must = append(query.Bool.Must, types.Query{
            MultiMatch: &types.MultiMatchQuery{
                Fields: []string{
                    "location.country",
                    "location.city",
                    "location.street",
                    "location.building",
                    "location.unit",
                    "location.postalCode5",
                },
            },
        })
    }
}

```

```

        Query: *filter.Location,
        Fuzziness: "AUTO",
    },
})
}

resp, err := c.client.Search().
    Index(c.indexName).
    Request(&search.Request{
        Query: &query,
        Size: filter.Limit,
        From: filter.Offset,
    }).
    Do(ctx)
if err != nil {
    return nil, 0, err
}

results := make([]domain.SearchResult, 0, len(resp.Hits.Hits))

for _, hit := range resp.Hits.Hits {
    var resultDTO searchResultDTO
    if err = json.Unmarshal(hit.Source_, &resultDTO); err != nil {
        return nil, 0, err
    }

    var result domain.SearchResult
    if err = common.Copy(resultDTO, &result); err != nil {
        return nil, 0, err
    }

    results = append(results, result)
}
return results, resp.Hits.Total.Value, nil
}

func (c *SearchResultRepository) FindOne(
    ctx context.Context,
    resultID uuid.UUID,
) (domain.SearchResult, error) {
    resp, err := c.client.Search().
        Index(c.indexName).
        Request(&search.Request{
            Query: &types.Query{
                Match: map[string]types.MatchQuery{
                    "id": {
                        Query: resultID.String(),
                    },
                },
            },
        }).
        Do(ctx)
    if err != nil {
        return domain.SearchResult{}, err
    }
    if len(resp.Hits.Hits) == 0 {
        return domain.SearchResult{}, xerrors.ErrNotFound
    }

    var resultDTO searchResultDTO
    if err = json.Unmarshal(resp.Hits.Hits[0].Source_, &resultDTO); err != nil {
        return domain.SearchResult{}, err
    }
}

```

```

var result domain.SearchResult
if err = common.Copy(resultDTO, &result); err != nil {
    return domain.SearchResult{}, err
}
return result, nil
}

func (s *SearchResultRepository) Save(ctx context.Context, result domain.SearchResult) error {
    var resultDTO searchResultDTO

    if err := common.Copy(result, &resultDTO); err != nil {
        return err
    }

    _, err := s.client.Index(s.indexName).
        Id(result.ID).
        Request(resultDTO).
        Do(ctx)
    return err
}

func (s *SearchResultRepository) Delete(ctx context.Context, resultID string) error {
    _, err := s.client.Delete(s.indexName, resultID).Do(ctx)
    return err
}

package jobs

type SearchResultsUpdaterJob struct {
    subscriber    message.Subscriber
    searchResultSvc *app.SearchResultService
    log           common.Logger
}

func NewSearchResultsUpdaterJob(
    subscriber message.Subscriber,
    searchResultSvc *app.SearchResultService,
    log common.Logger,
) *SearchResultsUpdaterJob {
    return &SearchResultsUpdaterJob{
        subscriber:    subscriber,
        searchResultSvc: searchResultSvc,
        log:           log.WithGroup("SearchResultsUpdaterJob"),
    }
}

func (job *SearchResultsUpdaterJob) SetupJob(ctx context.Context) error {
    messages, err := job.subscriber.Subscribe(ctx, "business_profile_saved")
    if err != nil {
        return err
    }

    job.log.Info("started")
    defer job.log.Info("stopped")

    return job.processMessages(messages)
}

func (job *SearchResultsUpdaterJob) processMessages(messages <-chan *message.Message) error {
    for msg := range messages {
        if err := job.processMessage(msg); err != nil {
            job.log.Error("processMessage",

```

```

        "err", err.Error(), "msg", msg)
    }
}
return nil
}

func (job *SearchResultsUpdaterJob) processMessage(msg *message.Message) error {
    var businessUpdated businessUpdatedEvent
    if err := json.Unmarshal(msg.Payload, &businessUpdated); err != nil {
        return err
    }

    var dto app.SaveSearchResultDTO
    if err := common.Copy(businessUpdated, &dto); err != nil {
        return err
    }

    err := job.searchResultSvc.Save(msg.Context(), dto)
    if err != nil {
        return err
    }

    msg.Ack()
    return nil
}

export interface SearchFormProps {
    query?: string;
    location?: string;
}

export function SearchForm(props: SearchFormProps) {
    const navigate = useNavigate();

    const [query, setQuery] = useState<string>(props.query ?? "");
    const [location, setLocation] = useState<string>(props.location ?? "");

    const getSearchPageRoute = (query: string, location: string): string => {
        let route = "/search?";

        if (query) {
            route += "q=" + query;
        }
        if (location) {
            route += "&loc=" + location;
        }

        return route;
    };

    const onSubmit = (
        event: React.SyntheticEvent<HTMLFormElement | HTMLButtonElement>
    ) => {
        event.preventDefault();
        navigate(getSearchPageRoute(query, location));
    };

    return (
        <Box>
            <form onSubmit={onSubmit}>
                <Grid>
                    <Grid.Col span={6}>
                        <TextInput

```

```

placeholder="Cleaners, plumbers, movers..."
size="md"
radius="xl"
leftSection={
  <IconSearch
    style={{ width: rem(18), height: rem(18) }}
    stroke={1.5}
  />
}
rightSection={
  <CloseButton
    aria-label="Clear input"
    onClick={() => setQuery("")}
    style={{ display: query ? "initial" : "none" }}
  />
}
onChange={(event) => setQuery(event.target.value)}
value={query}
/>
</Grid.Col>

<Grid.Col span={5}>
  <TextInput
    placeholder="Kyiv"
    size="md"
    radius="xl"
    leftSection={
      <IconLocation
        style={{ width: rem(18), height: rem(18) }}
        stroke={1.5}
      />
    }
    rightSection={
      <CloseButton
        aria-label="Clear input"
        onClick={() => setLocation("")}
        style={{ display: location ? "initial" : "none" }}
      />
    }
    onChange={(event) => setLocation(event.target.value)}
    value={location}
  />
</Grid.Col>

<Grid.Col span={1}>
  <Button
    type="submit"
    variant="light"
    radius="xl"
    size="md"
    disabled={!query && !location}
    onClick={onSubmit}
  >
  <IconSearch
    style={{ width: rem(20), height: rem(20) }}
    stroke={1.5}
  />
</Button>
</Grid.Col>
</Grid>
</form>
</Box>
);

```

```

}

export interface CategoriesFilterProps {
  categoryId?: number;
}

export function CategoriesFilter(props: CategoriesFilterProps) {
  const { data: categories } = useGetCategoriesQuery({
    parentId: props.categoryId,
    onlyRoot: !props.categoryId,
  });

  return (
    <Box>
      {categories?.items.map((category) => (
        <Link key={category.id} to={` /search?category=${category.id} `}>
          <Badge variant="light" mt="sm" ml="sm">
            {category.name}
          </Badge>
        </Link>
      ))}
    </Box>
  );
}

export function SearchResultsPage() {
  const navigate = useNavigate();

  const [searchParams] = useSearchParams();
  const params = Object.fromEntries(searchParams.entries());

  useEffect(() => {
    if (!params.q || !params.loc || !params.categoryId) {
      navigate("/");
    }
  }, []);

  const getTitle = (q: string, loc: string, category: string): string => {
    let title = `Search results`;

    if (q) {
      title += ` for "${q}"`;
    }
    if (loc) {
      title += ` at "${loc}"`;
    }
    if (category) {
      title += ` in "${category}" category`;
    }

    return title;
  };

  const { data: category } = useGetCategoryByIdQuery(
    params.category as unknown as number,
    {
      skip: !params.category,
    }
  );

  const title = getTitle(params.q, params.loc, category?.name as string);
  useDocumentTitle(title);

```

```

const { data } = useGetSearchResultsQuery({
  q: params.q,
  loc: params.loc,
  categoryId: params.category,
});

return (
  <Box>
    <Title mb="md">{title}</Title>

    <SearchForm query={params.q} location={params.loc} />
    <br />

    <CategoriesFilter categoryId={params.category as unknown as number} />
    <br />

    <Text c="dimmed" size="sm">
      Found {data?.totalCount} businesses {" "}
    </Text>

    <Stack mt="xs">
      {data?.items.map((business) => (
        <SearchResultCard key={business.id} searchResult={business} />
      ))}
    </Stack>
  </Box>
);
}

```

ДОДАТОК В
Слайди презентації

Національний університет “Запорізька політехніка”
Кафедра програмних засобів

Розробка вебзастосунку для пошуку і оцінювання компаній-підрядників

Виконав
Студент гр. КНТ-121сп

Данило ГАНСЬКИЙ

Керівник
Доцент, к.т.н.

Тетяна ФЕДОРОНЧАК

1

Рисунок В.1 – Слайд №1

Мета, об’єкт та предмет дослідження

Мета роботи Створення програмної системи для пошуку підрядників.

Об’єкт дослідження Програмні засоби для пошуку компаній-підрядників.

Предмет дослідження Розробка програмної системи для повнотекстового пошуку оголошень підрядників.

2

Рисунок В.2 – Слайд №2

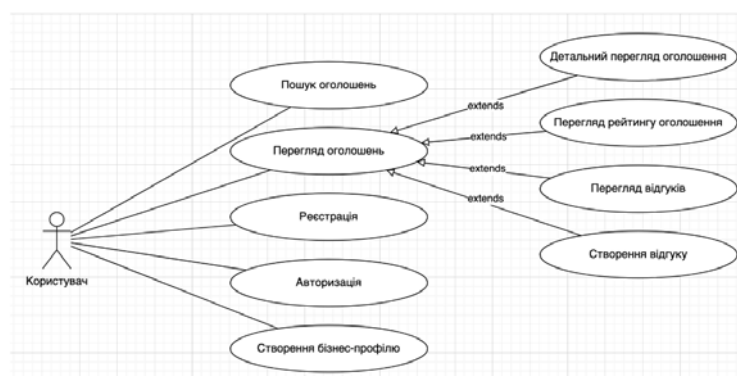
Постановка завдання

1. Розробити гнучкий пошук за ключовими словами і місцезнаходженням;
2. Показувати детальну інформацію про компанії у результатах пошуку;
3. Реалізувати можливість створення відгуків.

3

Рисунок В.3 – Слайд №3

Функціональні вимоги у вигляді use-case діаграми



4

Рисунок В.4 – Слайд №4

Огляд існуючих аналогів

Критерій порівняння	Yelp	Google Reviews	OLX	Kabanchik.ua
Наявність відгуків	+	+	-	+
Наявність пошуку з фільтрами	+	-	+	+
Детальний опис компаній	+	-	+	+
Доступний в скількох країнах	32	Усюди	15	1

5

Рисунок В.5 – Слайд №5

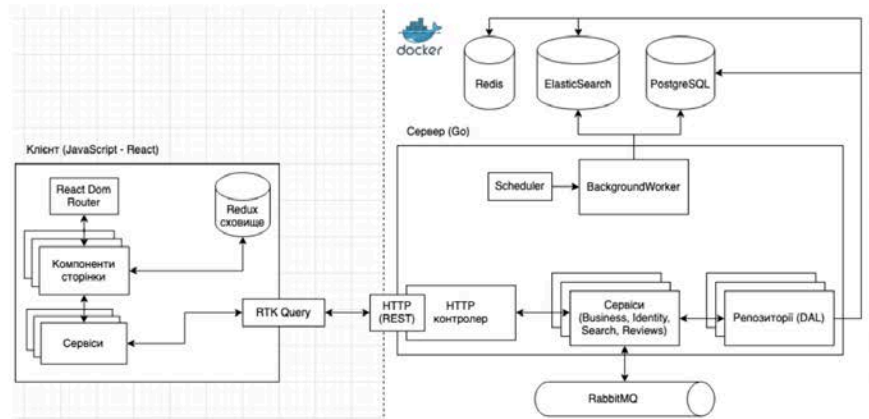
Обґрунтування вибору мов програмування

Критерій порівняння	Go (Golang)	Java	JavaScript	Python
Швидкодія	+	+	+	-
Паралельне виконання коду	++	+	+/- (за допомогою процесів)	+/- (за допомогою процесів)
Сфери застосування	Веб розробка	Веб розробка	Веб розробка	Веб розробка, AI
Кросплатформеність	+	+	+	+

6

Рисунок В.6 – Слайд №6

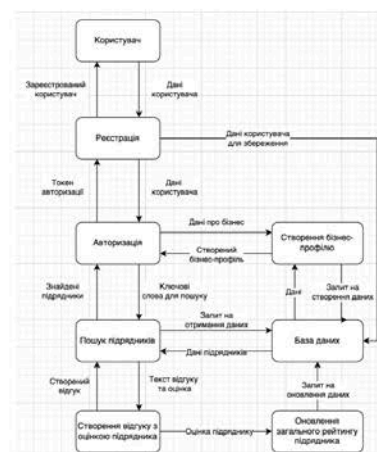
Архітектура системи



7

Рисунок В.7 – Слайд №7

Взаємодія користувача з системою



8

Рисунок В.8 – Слайд №8

Схема бази даних



Рисунок В.9 – Слайд №9

Графічний інтерфейс застосунку

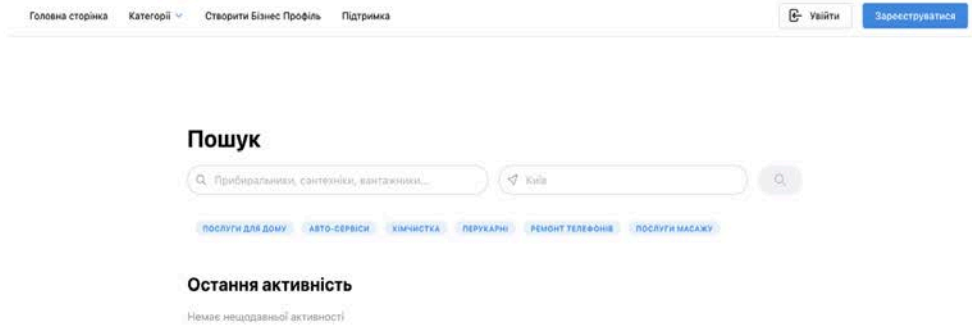


Рисунок В.10 – Слайд №10

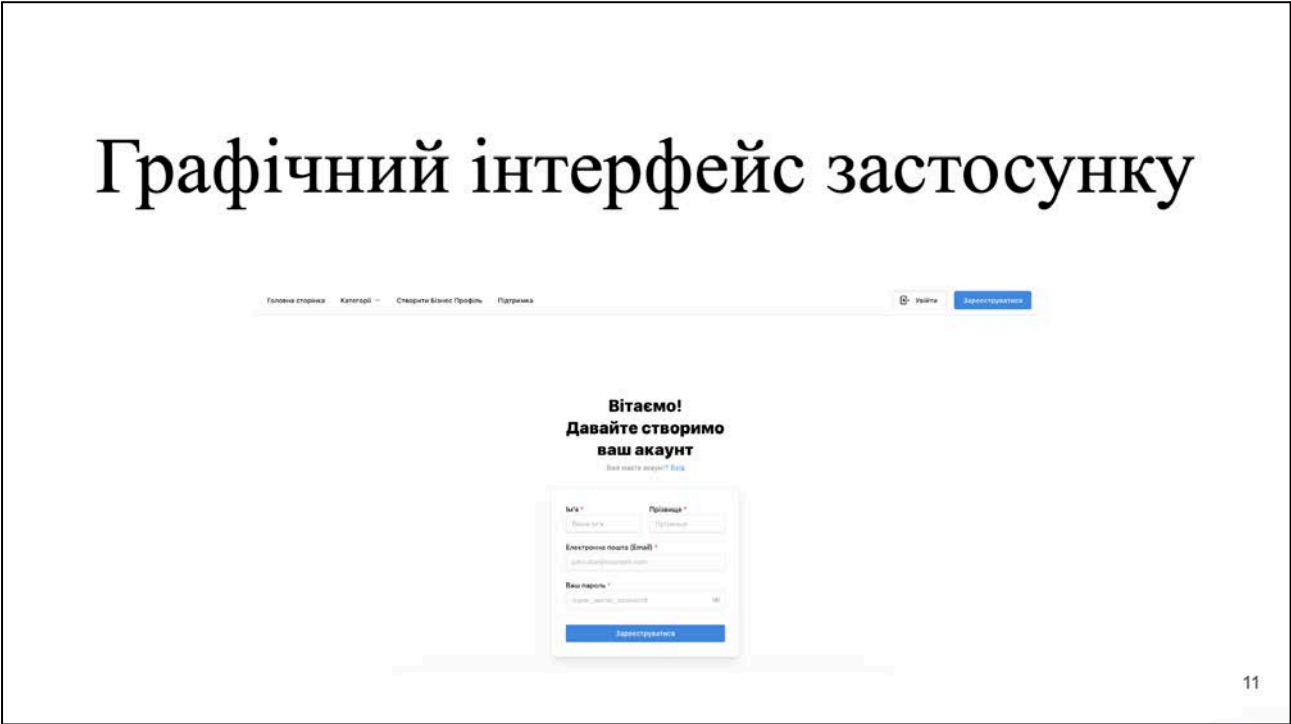


Рисунок В.11 – Слайд №11

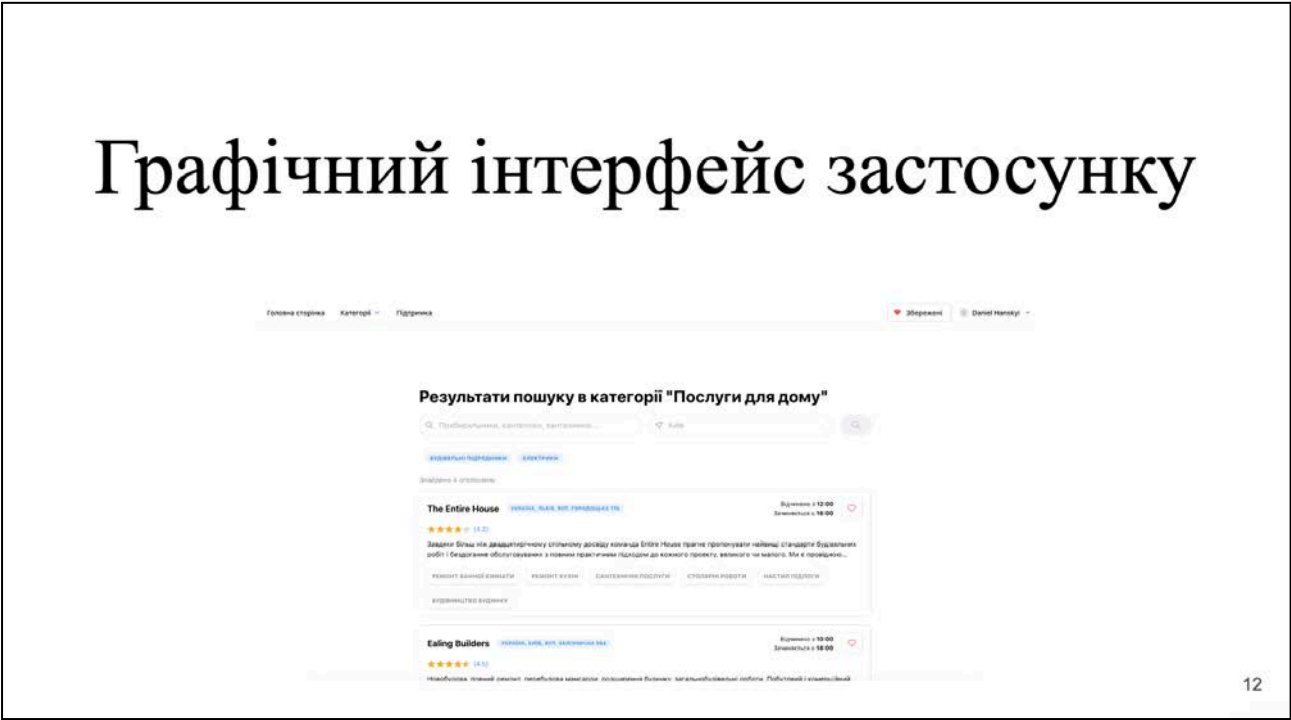
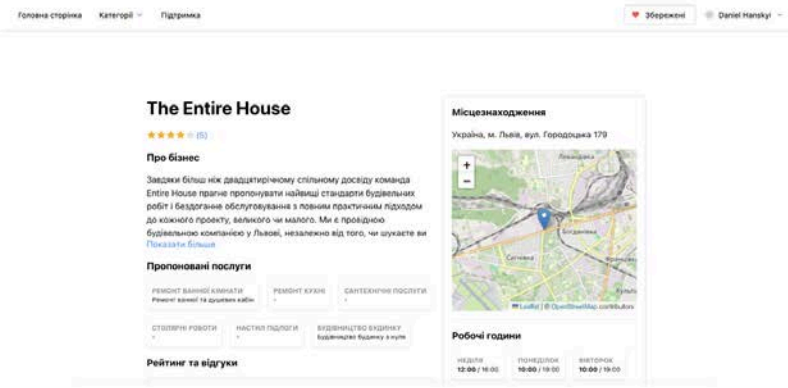


Рисунок В.12 – Слайд №12

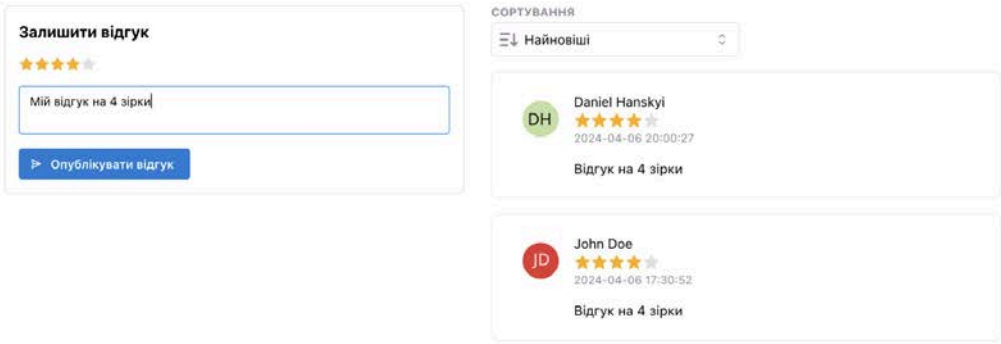
Графічний інтерфейс застосунку



13

Рисунок В.13 – Слайд №13

Графічний інтерфейс застосунку



14

Рисунок В.14 – Слайд №14

Висновки

1. Проаналізовано предметну область з пошуку підрядників;
2. Проаналізовано існуючі аналоги, їх переваги і недоліки;
3. Визначено вимоги до розроблюваної системи;
4. Розроблено програмну систему для пошуку та оцінювання компаній-підрядників;
5. Проведено тестування застосунку, під час якого не було виявлено жодних системних помилок, можливих порушень безпеки щодо доступу до користувацьких даних або проблем з швидкістю обробки запитів та загальною продуктивністю додатку.

Рисунок В.15 – Слайд №15