СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___інформаційних технологій та електроніки_____

Кафедра _____інформаційних технологій та програмування_____

# Пояснювальна записка

до магістерської дипломної роботи

_____магістр_____
(освітньо-кваліфікаційний рівень)

на тему: ___Інформатизація процесу надання послуг з оренди помешкань з використанням AI чат-бота_____

Виконав: студент 2 курсу, групи_ІСТ-22зм_
126 «Інформаційні системи та технології»___
(шифр і назва спеціальності)
_____Бакланов Є.В._____
(прізвище та ініціали)
Керівник ___Лифар В.О._____
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2023 року

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___інформаційних технологій та електроніки_____
Кафедра _____інформаційних технологій та програмування_____
Освітньо-кваліфікаційний рівень __магістр_____
Спеціальність 126 «Інформаційні системи та технології»_____
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
_____ д.т.н., доц. Захожай О.І.
(підпис)
« ___ » _____2023 р.

# ЗАВДАННЯ

на магістерську дипломну роботу студенту
_____Бакланов Єгор Володимирович_____
(прізвище, ім'я, по батькові)

1.Тема роботи: Інформатизація процесу надання послуг з оренди помешкань з використанням AI._____
керівник роботи доцент, д.т.н. Лифар Володимир Олексійович ,
(вчене звання, науковий ступінь, прізвище, ім'я, по батькові)
затверджені наказом університету від «____»____ 2023року № _____
2. Строк подання студентом роботи: 06 грудня 2023 р._____
3. Вихідні дані до роботи: Матеріали науково-дослідної практики, науково-методична література; дані інтернет-мережі .
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ
4.2 Аналітичний огляд питання (огляд публічних джерел інформації)
4.3 Основна частина, в якої висвітлити методи, які будуть використовуватися для реалізації проекту.
4.4 Практична частина – огляд технологій, які використовуються під час реалізації проекту.
4.4 Висновки
4.5 Перелік використаних джерел

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____
_____

6. Консультанти розділів проєкту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання __20 жовтня 2023р._____

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Одержання завдання на виконання роботи | 20.10.2023 | |
| 2. | Укладання і погодження з керівником плану і етапів виконання роботи | 24.10.2023 | |
| 3. | Узагальнення даних літературних джерел | 28.10.2023 | |
| 4. | Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху виконання завдання | 01.11.2023 | |
| 5. | Аналіз технічних засобів та існуючих систем | 07.11.2023 | |
| 6. | Реалізація практичної частини завдання | 24.11.2023 | |
| 7. | Укладання, оформлення та погодження пояснювальної записки з керівником | 05.12.2023 | |
| 8. | Здача пояснювальної записки на кафедру | 06.12.2023 | |
| 9. | Підготовка доповіді та презентації | 09.12.2023 | |

Студент _____ Бакланов Є.В._
(підпис) (прізвище та ініціали)

Керівник роботи _____ _Лифар В.О.
(підпис) (прізвище та ініціали)

# РЕФЕРАТ

Об’єкт дослідження – процеси проєктування бази даних та розроблення програмного забезпечення надання послуг з оренди помешкань з використанням AI чат-бота.

Мета розробки – проєктування та розроблення інформаційної системи надання послуг з оренди помешкань з використанням AI чат-бота.

У ході виконання дипломної роботи були проаналізовані існуючі інформаційні системи в обраній сфері здачі в оренду власного житла та виявлені їх недоліки. Було проаналізовано засоби та технології для реалізації рішення, спроєктовано архітектуру системи, обрана оптимальна база даних та спроєктована її структура, натреновано AI, для розпізнавання введеного користувачем тексту. Сформовано та зафіксовано бізнес процеси роботи реалізованої системи.

Результатом роботи є розроблений чат-бот в месенджері Telegram з використанням AI та система для керування замовленнями на здачу квартири в оренду за допомогою мови програмування Node.js та AI двигунця Dialogflow. Система може бути практично використана в процесі роботи ріелторських агентств, яка здатна зробити процес здачі власного житла більш зручним та прозорим.

Ключові слова: Telegram bot, Telegram Bot API, NodeJS, Javascript, MongoDB, DIalogflow, AI chat bot.

**ЗМІСТ**

# ВСТУП

**Метою** магістерської роботи є проєктування та розроблення інформаційної системи надання послуг з оренди помешкань з використанням AI чат-бота.

**Об’єктом** магістерської роботи є процеси проєктування бази даних та розроблення програмного забезпечення надання послуг з оренди помешкань з використанням AI чат-бота.

**Предметом** магістерської роботи є методи та засоби для проєктування та розроблення інформаційної системи надання послуг з оренди помешкань з використанням AI чат-бота.

**Виконання завдання** магістерської роботи вимагає вирішення наступних задач:

- розглянути способи реалізації AI чат-ботів;
- розробити проєкт інформаційної системи;
- опрацювати проєкт інформаційної системи та бази даних;
- реалізувати комфортний та зрозумілий для користувача інтерфейс програмного забезпечення.

# РОЗДІЛ 1
# Аналітичний огляд літературних та інших джерел

## 1.1 Аналіз поточної ситуації надання послуг з оренди помешкань

Постійно зростаючий розвиток користувачів Мережі створив нову парадигму бізнесу та відкрив нову модель доходу для підприємства, яку називають електронним бізнесом чи електронними послугами. Однією з таких послуг є сервіс резервування готелів або оренди житла.

До розгортання електронних сервісів процес пошуку відповідного готелю або помешкання в потрібному місці був складним та витратним завданням, і він здійснювався головним чином за допомогою маклерів. Тож людям зручно та легко резервувати помешкання через Мережу за допомогою послуг резервування електронних готелів.

Існує безліч способів здійснення резервування, проте є два основні джерела, які широко розрізняються як:

- Інтернет-ресурси
- Офлайн-ресурси

Будь-яке резервування, що пройшло через онлайн-ресурс, називається онлайн-резервуванням. Прямий онлайн-ресурс означає, що вся взаємодія з бронюванням відбувалася між готелем та клієнтом, і жодний інший онлайн-агрегатор не брав участі. Непряме джерело резервування в Інтернеті - це резервування, при якому генерація бронювання відбувається через агрегатор або через якусь третю сторону платформи онлайн-бронювання, і за які готелям слід сплачувати значну суму комісії постачальнику послуг. Все більше набуває популярності кумунікація з клієнтами через чат-боти, які здатні полегшити та прискорити процеси бізнесу[8].

Загалом, існують основні джерела непрямих онлайн-бронювань:

- Інтернет-туристичні посередники
- Мета - пошукові системи

В наш час, основний сегмент готельного бізнесу залежить від онлайн посередників. Оскільки сьогодні майже кожному стало набагато простіше знайти відповідний готель, що відповідає їхнім фінансовим можливостям, і забронювати його всього лише за кілька клацань.

Сьогодні понад 70% бізнесу формується через цих онлайн посередників, тому це також стає обов'язковим для того, щоб самі переліки готелів на багатьох платформах онлайн-туристичних посередників отримували максимальну видимість і мали більше можливостей для конвертації. Зараз метапошукові системи - це база даних, яка отримує інформацію з усієї пошукової системи, і кожного разу, коли користувач вводить свій пошуковий запит, вона збирає всі свої дані та надає результат, який стосується пошуку, схожий на те, як це робиться в TripAdvisor та готельних оголошеннях Google.

Оскільки в кожному будинку або помешканні є обмежена кількість номерів, для особистого резервування стає дуже важко керувати всім процесом резервування з таким різноманітним набором джерел без помилок. Таким чином, програмне забезпечення резервування відігравало головну роль у безперебійній роботі системи резервування, проте наразі традиційні програмні засоби резервування не змогли вирішити необхідність усунення розриву з онлайн-каналами, а саме залучити плавну інтеграцію з усіма Інтернет-джерелами, щоб синхронізувати всю систему резервування, яка працює в синхронізації з фактичним статусом інвентаризації.

Таким чином, виникло сучасне хмарне програмне забезпечення резервування, яке постачається з інтегрованим менеджером каналів для миттєвого обслуговування. Це значною мірою автоматизує процес управління запасами та економить багато часу персоналу резервування та допомагає їм більше зосередитись на наданні обслуговування клієнтів та налагодженню роботи системи резервування готелів. Крім того, хмарне програмне забезпечення для резервування готелів або помешкань є дуже простим у

використанні, оскільки воно має зручний інтерфейс, який допомагає в швидкій доставці послуг.

Інтернет-платформа резервування надає різноманітні послуги, які корисні тим, хто часто подорожує у різні куточки світу. Використання технології мобільних агентів або чат-ботів для розробки системи онлайн-резервування задовольнило вимоги щодо ефективності, автоматизації та гнучкості розподілених додатків/систем. Також, безпека банківських даних клієнта в електронних транзакціях є важливою темою під час використання таких сервісів.

Основні вимоги безпеки для онлайн-платформи резервування визначені наступним чином:

- Конфіденційність – банківські реквізити клієнта повинні зберігатися в таємниці, коли він здійснює онлайн-платіж за рахунок бронювання. Конфіденційність банківської інформації клієнта забезпечується шифруванням їх, перш ніж фактично поширювати ці дані.
- Цілісність – реквізити електронних транзакцій, такі як сума трансакції, ім'я бенефіціара та номер рахунку, не повинні змінюватися. Квитанція про транзакцію із зазначенням підтвердження бронювання повинна бути доставлена замовнику без будь-яких змін.
- Аутентифікація гарантує, що люди, які використовують систему бронювання готелів, є уповноваженими користувачами цієї системи перед здійсненням транзакцій.
- Недопущення забезпечує, що ні замовник, ні постачальник не можуть відмовити у спілкуванні чи інших діях щодо інформації чи ресурсів у визначений час.
- Доступність забезпечує, що кінцева система та послуга будуть доступними для доступу постійно авторизованому користувачеві.

- Підзвітність – особистість усіх користувачів гарантується, а користувачі несуть відповідальність за їх дії.
- Захист від копіювання – ця функція забезпечує захист від несанкціонованого копіювання інтелектуальної інформації.

Функції безпеки онлайн систем бронювання готелів надаються на основі таких компонентів криптографії:

- Криптосистема відкритого ключа (PKC) може використовуватися для шифрування та дешифрування конфіденційної інформації, такої як номер кредитної картки / дебетової картки. І інфраструктура захищеного сокета (SSL), і інфраструктура відкритих ключів (PKI) базуються на PKC.
- Цифровий підпис використовується для забезпечення цілісності інформації, достовірності користувача та доступності інформації для аутентифікованого користувача.
- Автентифікація на основі пароля використовується для перевірки ідентичності користувачів. Це найпростіший і найдавніший метод аутентифікації суб'єктів.

### 1.2 Аналіз систем-аналогів

Під час пошуку житла, вибір ресурсів для порівняння повинен залежати від того, наскільки часто вони пропонують найвигідніші умови проживання. Для більшості запланованих подорожей оптимальною стратегією є порівняння цін з різних джерел, включаючи власний веб-сайт місця проживання, оскільки іноді там можуть бути пропозиції, недоступні на інших платформах або пропонують ті ж самі тарифи, за винятком прихованих комісій за бронювання.

Сайти готелів також надають пакетні пропозиції, що включають додаткові послуги та інше, які не доступні на зовнішніх ресурсах. Немає повного списку найкращих ресурсів для бронювання проживання, але ці 7 представляють

собою комбінацію великих популярних платформ разом з деякими новими аналогічними опціями для пошуку місць проживання.

### *1.2.1 Booking.com*

Booking.com повертає найоптимальніші результати пошуку на сьогоднішній день із цікавим поєднанням готелів, апартаментів та хостелів. Це перевага чи недолік, повністю залежить від особистих уподобань. Booking.com пропонує різноманіття для кожного користувача. Зручна пошукова система готелів відображає загальну суму передплатних витрат (крім податків), що, як і HotelsCombined, допомагає при порівнянні доступних пропозицій готелів; можливість бачити, що загальна вартість готелю на ранніх етапах допомагає швидко визначити, які готелі насправді вписуються у ваш бюджет[12]. (Рис.1.1).

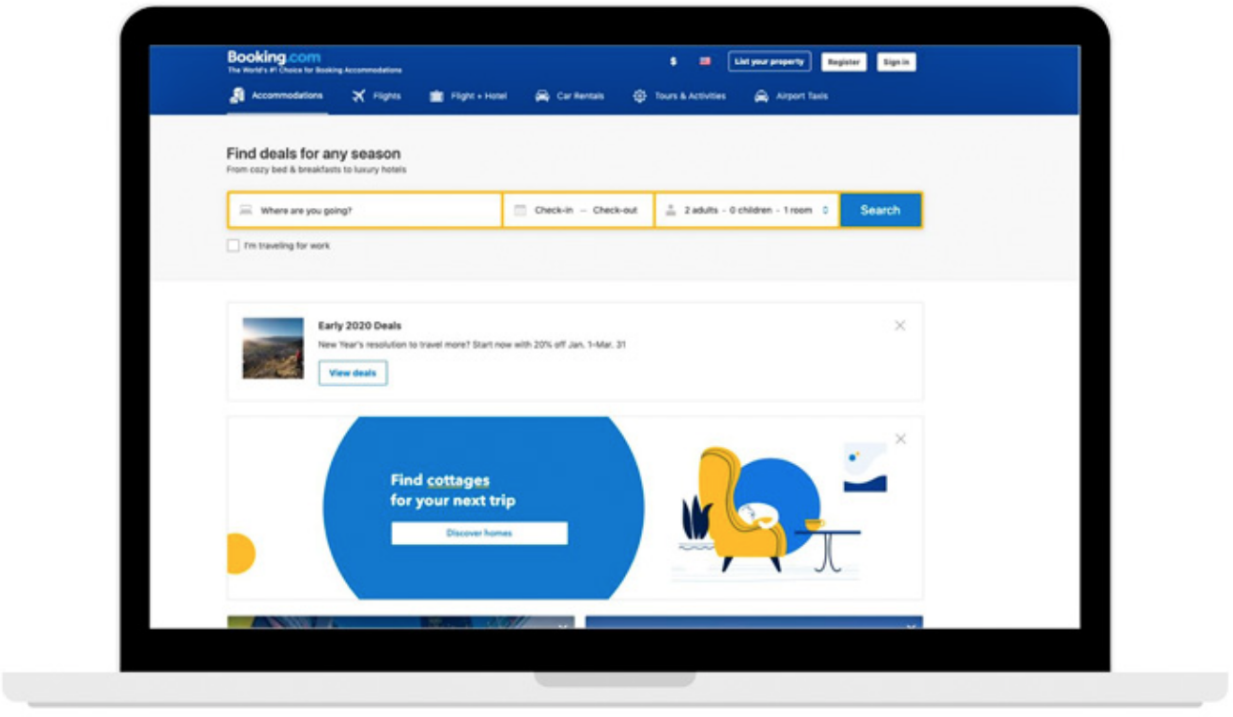


*Рис. 1.1 Головна сторінка Booking.com*

Найкраща особливість – це різноманітність типів житла та змішані результати пошуку. Booking.com - це хороший сайт пошуку готелів для широкого кола бюджетів.

### *1.2.2 Kayak*

Так само, як і у випадку пошуку авіабілетів, пошук житла пропонує простий у користуванні інтерфейс із багатьма доступними фільтрами. Він також відображає ціни інших сервісів резервування готелів, тому ви можете порівняти їх у одному місці. Проте перше значення ціни на готель часто вище, ніж більш вигідні пропозиції, що йдуть нижче в списку, якщо ви спеціально не фільтруєте за ціною[13].

Результати пошуку готелів на веб-сайті автоматично сортуються за певним коефіцієнтом, який є «Рекомендованим», і це стосується багатьох платформ для бронювання готелів. Зазвичай, вищі тарифи висвічуються на початку прокрутки результатів, і іноді ціна за бронювання значно вища, ніж у порівнянні з іншими варіантами готелів. Складно розібратися, наскільки це є корисним, особливо якщо головна мета пошукових систем готелів полягає в тому, щоб допомогти вам знайти найвигідніші ціни на проживання. (Рис.1.2). Мінімальний вигляд сайту є приємним, але має і зворотний бік:

- варіант сортування за ціною - важко знайти сіре спадне меню вгорі списку.
- найкраща особливість: простий у користуванні інтерфейс. Це також одна з найкращих програм для готелів, яке можна завантажити на свій телефон.

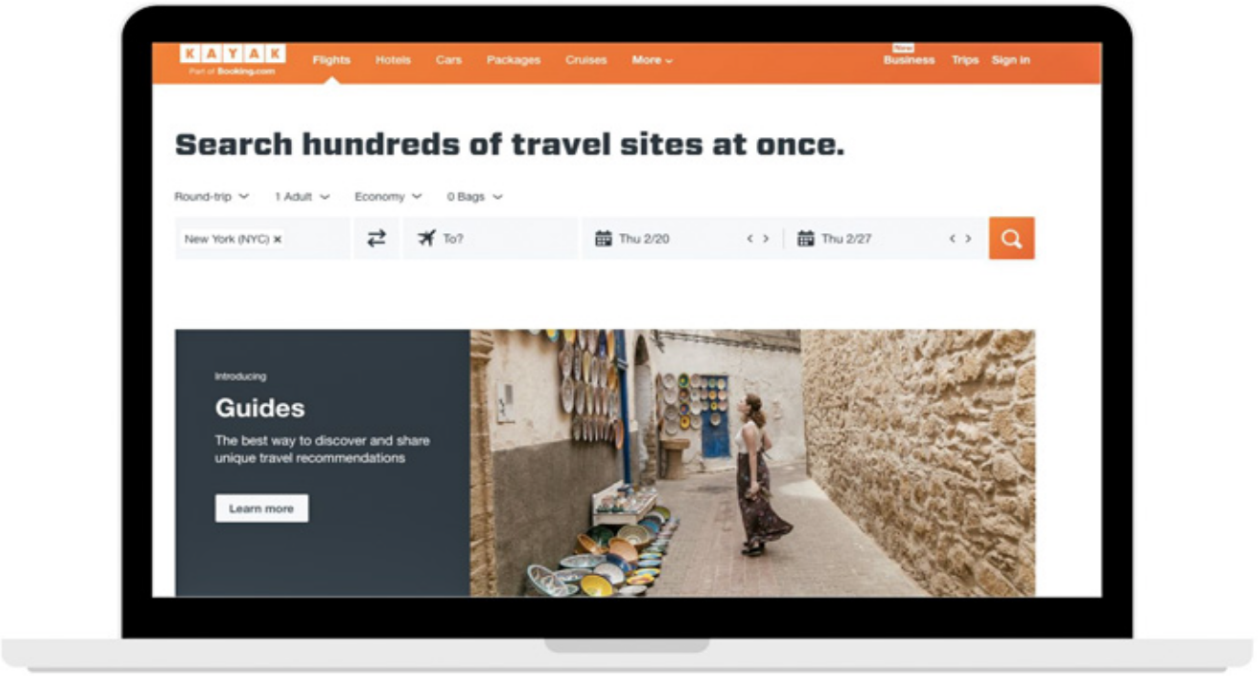


*Рис. 1.2 Головна сторінка Kayak*

### *1.2.3 Priceline*

У Priceline має один з найбільш привабливих макетів дизайну ключових пошукових систем готелів і, безумовно, є серед найпростіших веб-сайтів готелів для навігації. Жоден з цих порталів для пошуку готелів різко не відрізняється від інших за ставками чи порівнянням цін, тому зручність використання може пройти довгий шлях у поліпшенні досвіду пошуку найвигідніших цін на проживання.(Рис.1.3). Не дивно, що ціни Priceline нарівні з іншими туристичними агенціями в Інтернеті, і її результати в основному орієнтовані на міста та туристичні зони[14].

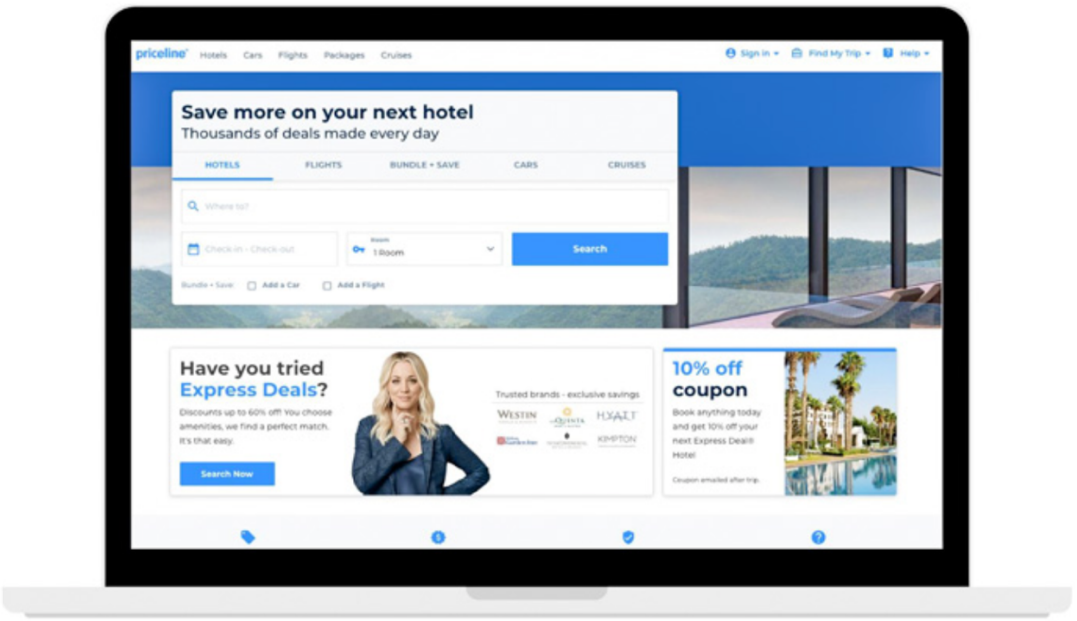


*Рис. 1.3 Головна сторінка Priceline*

Найкраща особливість: функції «Назвіть власну ціну» та «Експрес-пропозиція» є основними відмінностями від інших готельних сайтів зі знижками.

### *1.2.4 Hotels.com*

Hotels.com допомагає знаходити найкращі угоди на готелі за допомогою різноманітних фільтрів, які дозволяють звужати пошук. Його початкові результати, як правило, відображають збалансований вибір готелів різних класів, переважно в міських або їхніх околицях. Такий перелік добре розташованих варіантів є тим, що шукає більшість мандрівників у пошукових системах готелів: надійний і простий у вдосконаленні. З іншого боку, результати пошуку готелів також включають багато закладів, які «повністю заброньовані», що не дає клієнту жодних конкретних можливостей[15].

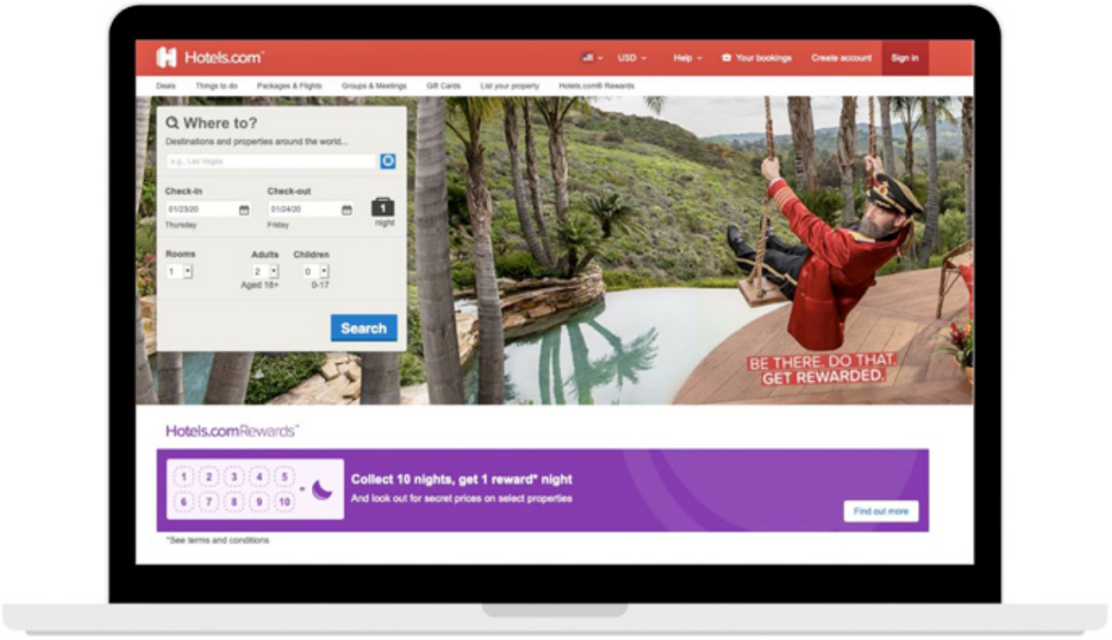


*Рис. 1.4 Головна сторінка Hotels.com*

Найкраща особливість: Hotels.com пропонує більше фільтрів пошуку готелів, ніж більшість мандрівників коли-небудь могли використовувати, але приємно мати ці варіанти.

### *1.2.5 HotelsCombined*

HotelsCombined - один із високоякісних порталів для бронювання готелів, представляє собою метапошуковий інструмент, який сканує різноманітні джерела для знаходження найвигідніших пропозицій на проживання, а також веб-сайти готелів. Результати пошуку включають декілька альтернатив з одного джерела, що дозволяє порівнювати різні категорії номерів. Можна переключатися між вартостями готелів, які враховують або не враховують податки (Рис.1.5).

HotelsCombined враховує значну кількість готелів біля аеропортів для отримання найбільш точних результатів, а параметр за замовчуванням визначає загальну вартість за увесь період подорожі, а не щоденний розрахунок тарифу. Жоден з цих аспектів не є проблемою сам по собі, проте це ускладнює процес

порівняння цін, оскільки більшість інших готельних сайтів відображають лише щоденні тарифи[16].

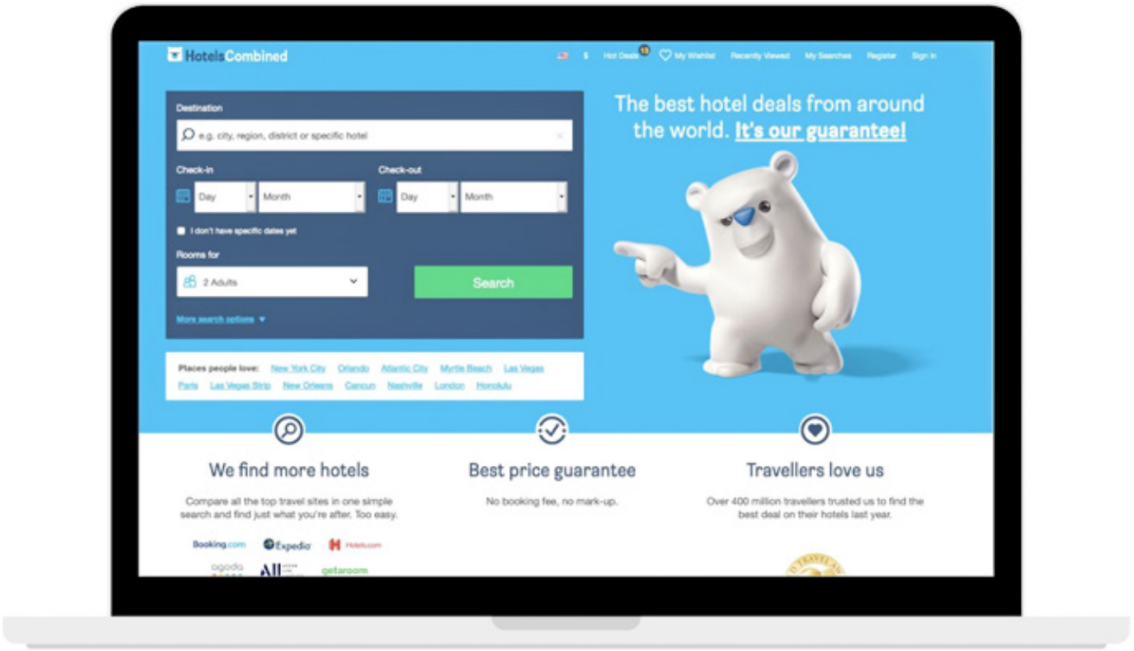


*Рис. 1.5 Головна сторінка HotelsCombined*

Найкраща особливість: абсолютний обсяг результатів робить це хорошим місцем для початку пошуку готелів, але обов'язково вивчайте ціни та варіанти, коли ви прибуваєте на фактичному сайті бронювання готелів, щоб переконатися, що вони відповідні.

### *1.2.6 Hotwire*

Один з високоякісних порталів для бронювання готелів з "гарячими" пропозиціями та "гарячими" цінами, які маскують назву готелю для отримання кращої ціни, Hotwire - це чудовий варіант, якщо особа не має нічого проти остаточного бронювання з обмеженими деталями. Hotwire також взаємодіє з відомими світовими готельними брендами, такими як Kimpton та Hyatt. Заощадження можуть варіюватися залежно від місця призначення, але "гарячі" ціни постійно залишаються надзвичайно конкурентоспроможними порівняно зі стандартними тарифами Hotwire, які не завжди є найкращими за ціною

порівняно з іншими готельними сайтами (Рис. 1.6). Основною характеристикою, якою користуються більшість клієнтів Hotwire, є їхнє круглодобове обслуговування, що, ймовірно, може бути досить корисною, навіть якщо вона завжди доступна[17].

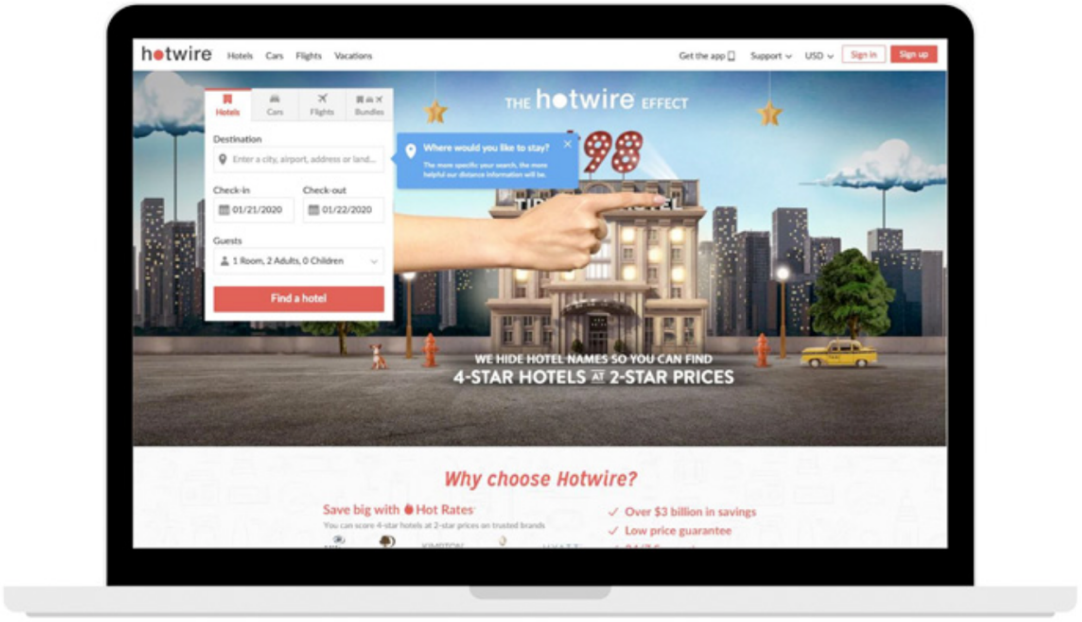


*Рис. 1.6 Головна сторінка Hotwire*

Найкраща особливість: гарячі ціни, які допоможуть вам розмістити в комфортабельному, якщо не чотиризірковому готелі, та за ціною двох зірок.

### *1.2.7 Google*

Пошукова система готелів Google працює, просто вводячи готелі прямо на Google.com. Ця функція вбудована в Карти Google, що відрізняє її від будь-якого з інших вище перерахованих сайтів для бронювання готелів. У Google місця розташування готелів відзначаються цінами на карті, а не іменем або будь-якою іншою ідентифікаційною характеристикою.

Таким чином, інструмент пошуку готелів Google є досить цінним і може служити метапошуком (Рис.1.7). При натисканні на ціну відображається назва готелю, функції та параметри бронювання готелю. Загалом, не дивно, що Google пропонує потужну, але не занадто вишукану пошукову систему для

мандрівників, які не бажають усіх складнощів дешевих готельних сайтів та інших комерційних пошукових платформ[18].

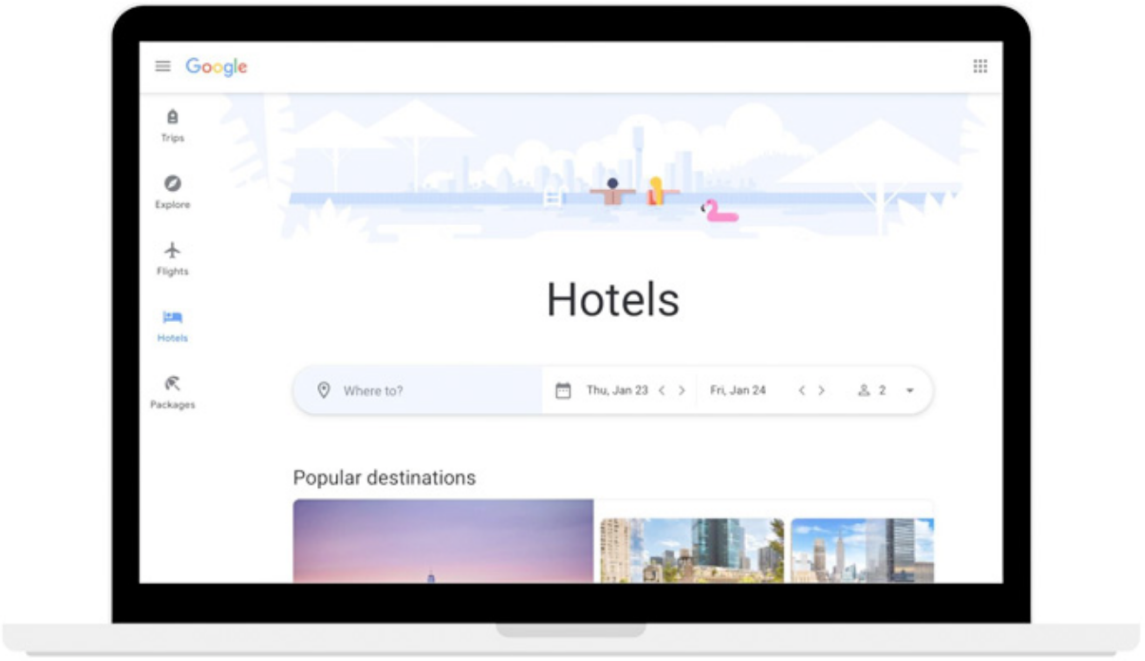


*Рис. 1.7 Головна сторінка Google*

Найкраща особливість: місце розташування, як правило, досить важливе при виборі готелю, а інтеграція Google Maps - із супутниковим і вуличним видом - дозволяє легко враховувати це при пошуку готелю.

# ВИСНОВОК ДО РОЗДІЛУ 1

Вивчено поточний стан наукових досліджень за допомогою літературних джерел. Також, проведено аналіз ряду аналогічних систем, зокрема веб-сайтів для бронювання і платформ, які надають суміжні послуги, а саме: Booking.com, Kayak, Priceline, Hotels.com, HotelsCombined, Hotwire, Google.

Для системного аналізу було визначено низку функцій, які повинні бути реалізовані у створюваній системі, а також функціонал, який може покращити існуючі можливості, а саме в існуючих системах є великий недолік, що мені як власнику житла, потрібно повністю контролювати процес здачі в оренду та комунікувати з орендарями, що не підходить для людей, котрі мають власне житло та хочуть його здавати в оренду при цьому з мінімальним контактом з орендарями, тому потрібно зробити таку систему, щоб можна було віддати своє житло, а компанія вже повністю займається пошуком орендарей та після здачі в оренду власник житла отримає гроші.

# РОЗДІЛ 2

## Системний аналіз об’єкта дослідження

### 2.1 Дерево цілей

Аналіз проблемного дерева виконує центральну роль у багатьох видах проєктного планування і є широко використовуваним. Аналіз проблемного дерева сприяє знаходженню рішень, відображаючи структуру причин і наслідків, пов'язаних із проблемою, але з більшою структурністю. Це призводить до ряду переваг:

- задачу можна розділити на визначені фрагменти. Це дозволяє чітше визначити пріоритетність чинників і сприяє фокусуванню на цілях;
- існує більше розуміння проблеми та її часто взаємопов'язаних і навіть суперечливих причин. Це часто є першим кроком у пошуку безпрограшних рішень;
- визначає складові питання та аргументи і може допомогти встановити, хто і що є акторами та процесами на кожному етапі;
- може допомогти встановити, чи потрібна додаткова інформація, докази чи ресурси для прийняття рішення;
- процес аналізу часто допомагає формувати спільне розуміння, мети та дії.

Мета дерева проблем - забезпечити широкий огляд проблеми, а також визначити конкретні причини та наслідки, що виникають. Використання негативної мови під час використання цього інструменту є корисним для виявлення конкретних проблем. Основна проблема буде розміщена в центрі дерева проблем. Щоб трансформувати проблемне дерево у дерево цілей, необхідно переформулювати кожен елемент у позитивне рішення. Завершальний етап включає визначення споріднених груп цілей, які можуть бути пов'язані з конкретним видом стратегії розвитку.

Мета полягає в тому, щоб визначити широкі категорії стратегій, доступних для досягнення цілей, і уточнити, який тип стратегії буде досягати кожної конкретної цілі. Під час аналізу предметної області було визначено ряд існуючих проблем, на основі чого було побудовано дерево цілей, включаючи наступні пункти (Рис 2.1):

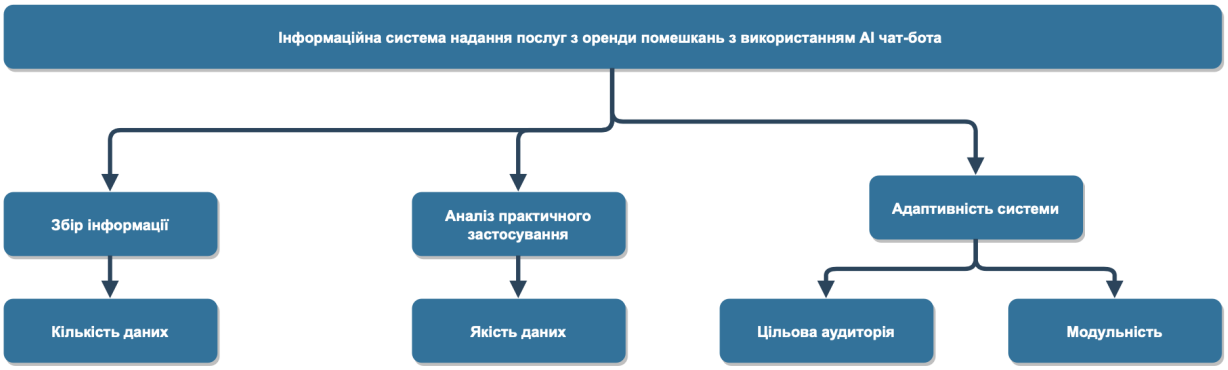


*Рис.2.1. Дерево цілей*

## 2.2 Конкретизація функціонування системи

Діаграма потоку даних (DFD) - це традиційне візуальне подання інформаційних потоків у системі. Чіткий DFD може графічно зобразити потрібну кількість системних вимог. Він показує, як дані входять та залишають систему, що змінює інформацію та де зберігаються дані.

Завдання DFD - показати масштаби та межі системи в цілому. Він може використовуватися як інструмент комунікації між системним аналітиком та будь-якою особою, яка відіграє певну роль у порядку, який є вихідною точкою для переробки системи. DFD також називається графіком потоку даних або бульбашковою діаграмою.

Наступні спостереження щодо DFD є важливими:

- усі імена повинні бути унікальними. Це полегшує посилання на елементи в DFD.

- DFD не є блок-схемою. Стрілки - блок-схема, яка представляє порядок подій; стрілки в DFD представляють поточні дані. DFD не передбачає жодного порядку подій.
- у діаграмах потоку використовується ромбоподібний ящик для відображення точок рішення з кількома існуючими шляхами, з яких взято єдине. Це передбачає впорядкування подій, що не має сенсу в DFD.

DFD може використовуватися для системи або програмного забезпечення на будь-якому рівні абстракції. У взаємодії, DFD можуть бути розподілені на рівні, які представляють зростаючий потік інформації та функціональні деталі. Рівні в DFD нумеруються 0, 1, 2 або вище.

0-рівень DFDM, також відомий як основна модель системи, або контекстна діаграма представляє всі вимоги програмного забезпечення як єдине середовище із вхідними та вихідними даними, позначеними вхідними та вихідними стрілками. Потім система розкладається і описується як DFD з декількома підрівнями. Частини системи, представлені кожним із цих рівнів, потім розкладаються та документуються як більш детальні DFD. Цей процес може бути повторений на стільки рівнів, скільки необхідно, доки флоу не буде добре зрозумілим. Важливо зберегти кількість входів і виходів між рівнями, це поняття називається DeMacro вирівнюванням.

DFD рівня 0, яка також називається контекстною діаграмою системи управління результатами. У DFD на 1 рівні контекстна діаграма розкладається на кілька процесів. На цьому рівні виділяються основні цілі системи та розбивається високорівневий процес DFD на 0 рівня на підпроцеси.

Дворівневий DFD проходить один процес глибше, в частини 1-рівня DFD. З його допомогою можна проєктувати або записувати конкретні / необхідні деталі щодо функціонування системи.

Було створено ряд діаграм, найперше була побудована загальна діаграма в якій описано принципи функціонування системи.

Основна взаємодія відбувається:

1. між Користувачем(власник житла) та Інформаційна система надання послуг з оренди помешкань з використанням AI чат-бота (Рис.2.2).

Потоки включають:

- Запуск чат-боту, натиском кнопки
- Вивід інформації про статус користувача (зареєстрований/не зареєстрований)
- Запит для отримання переліку послуг
- Надання переліку послуг
- Запуск процесу здачі житла в певний період часу
- Створення ордеру для пошуку зацікавленого орендаря
- Інформування про знайденого орендаря
- Підтвердження/скасування ордеру


*Рис.2.2. DF-діаграма взаємодії ІС з Користувачем(власником житла)*

2. між Користувачем(менеджер системи) та Інформаційна система надання послуг з оренди помешкань з використанням AI чат-бота (Рис.2.3).

Потоки включають:

- Натиснення кнопки "Apartment Owner Onboarding"

- Вивід реєстраційної форми для заповнення даних і кнопки "Submit"
- Заповнення форми і натискання кнопки "Submit"
- Повідомлення про успішну реєстрацію/відмову в реєстрації
- Натиснення кнопки "Apartment Owners"
- Вивід інформації про всіх власників житла
- Натиснення кнопки "Opened renting order list"
- Вивід інформації з заявками від власників житла
- Натиснення на певну заявку
- Вивід форми для підтвердження ордеру
- Заповнення форми і підтвердження замовлення

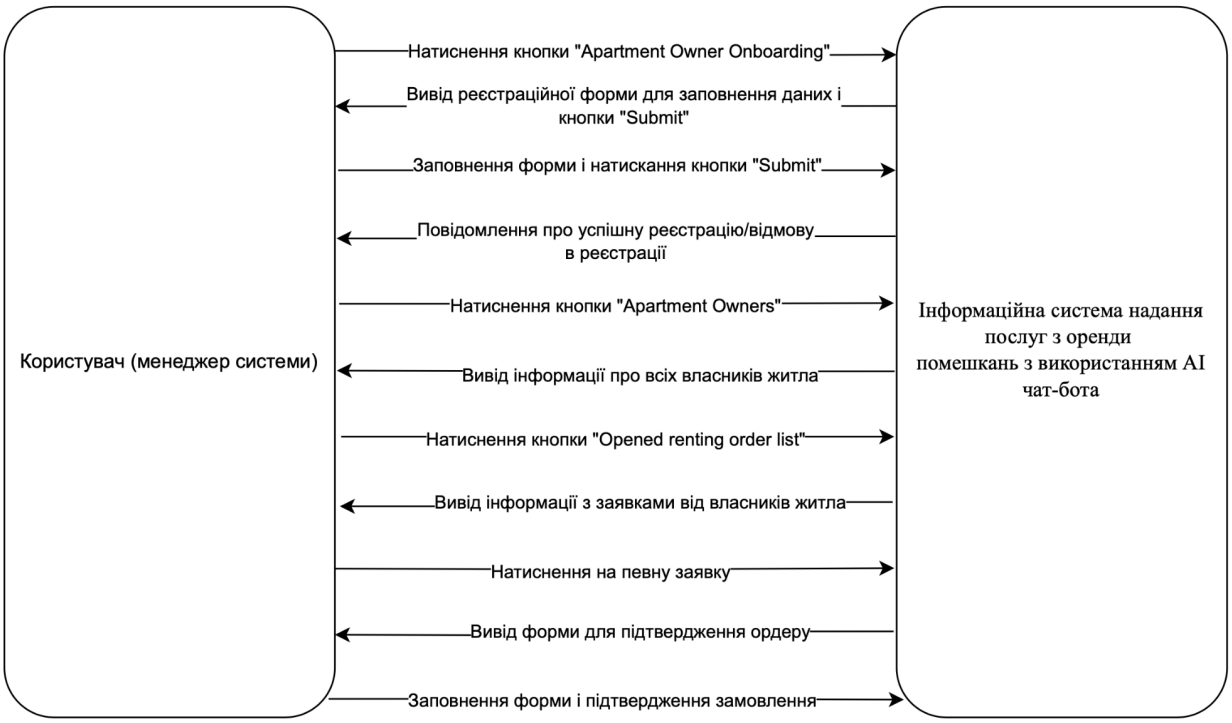


*Рис.2.3. DF-діаграма взаємодії ІС з Користувачем(менеджером системи)*

Була створена декомпозиція першого рівня(Рис. 2.4) для зв’язки ІС та Користувач(власник житла). Основними процесами є вивід інформації, аналіз

запиту, пошук користувача, валідація даних, створення нової події бронювання, БД. Наявні наступні потоки:

1. Користувач(власник житла) – Аналіз запиту
    - Запуск чат-боту, натиском кнопки
    - Запит для отримання переліку послуг
    - Запуск процесу здачі житла в певний період часу
2. Вивід інформації – Користувач(власник житла)
    - Вивід інформації про статус користувача (зареєстрований/не зареєстрований)
    - Надання переліку послуг
    - Створення ордеру для пошуку зацікавленого орендаря
    - Підтвердження/скасування ордеру
3. Аналіз запиту - Пошук користувача
    - Пошук користувача за telegram chatId
4. Пошук користувача – БД
    - Запит на отримання даних по id
    - Повідомлення про успішний/неуспішний пошук
5. Аналіз запиту - Валідація даних
    - Запит на перевірку коректності даних анкетної форми
6. Валідація даних - Створення нової події бронювання
    - Валідація успішна
7. Валідація даних - Вивід інформації
    - Валідація не успішна
8. Підтвердження замовлення - Вивід інформації
    - Оновлення статусу менеджером

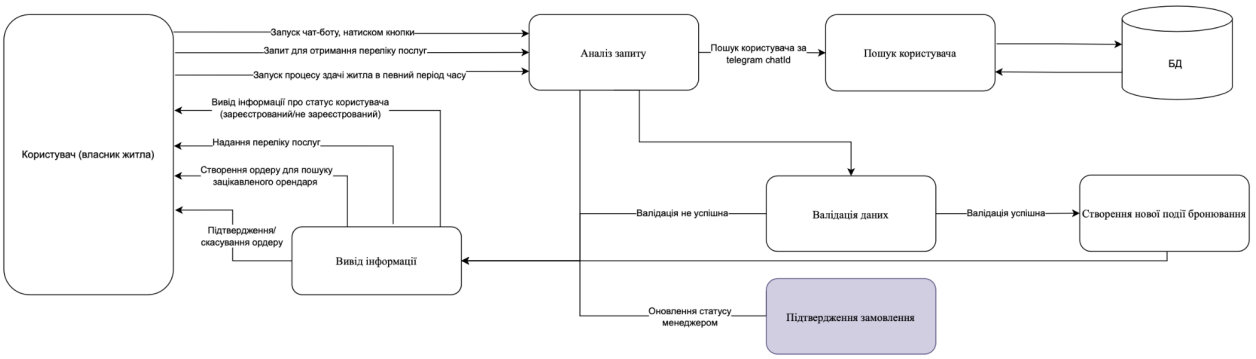


*Рис.2.4. DF-діаграма першого рівня декомпозиції ІС з Користувачем(власником житла)*

Також, була створена декомпозиція першого рівня(Рис. 2.5) для зв'язки ІС та Користувач(менеджер системи). Основними процесами є вивід інформації, аналіз запиту, пошук користувача, валідація даних, створення нової події бронювання, БД. Наявні наступні потоки:

1. Користувач(менеджер системи) – Аналіз запиту
    - Натиснення кнопки "Apartment Owner Onboarding"
    - Заповнення форми і натискання кнопки "Submit"
    - Натиснення кнопки "Apartment Owners"
    - Натиснення кнопки "Opened renting order list"
    - Натиснення на певну заявку
    - Заповнення форми і підтвердження замовлення
2. Вивід інформації – Користувач(менеджер системи)
    - Вивід форми для підтвердження ордеру
    - Вивід інформації з заявками від власників житла
    - Вивід інформації про всіх власників житла
    - Повідомлення про успішну реєстрацію/відмову в реєстрації
    - Вивід реєстраційної форми для заповнення даних і кнопки "Submit"
    - Інформування про знайденого орендаря

3. Аналіз запиту – Пошук користувача
    - Пошук користувача
4. Аналіз запиту – Пошук заявок оренди
    - Пошук заявок
5. Пошук користувача – БД
    - Запит на отримання даних
    - Повідомлення про успішний/неуспішний пошук
6. Пошук заявок оренди – БД
    - Запит на отримання даних
    - Повідомлення про успішний/неуспішний пошук
7. Аналіз запиту - Валідація даних
    - Запит на перевірку коректності даних анкетної форми
8. Валідація даних - Вивід інформації
    - Валідація не успішна
9. Підтвердження замовлення - Вивід інформації
    - Оновлення статусу менеджером (валідація успішна)

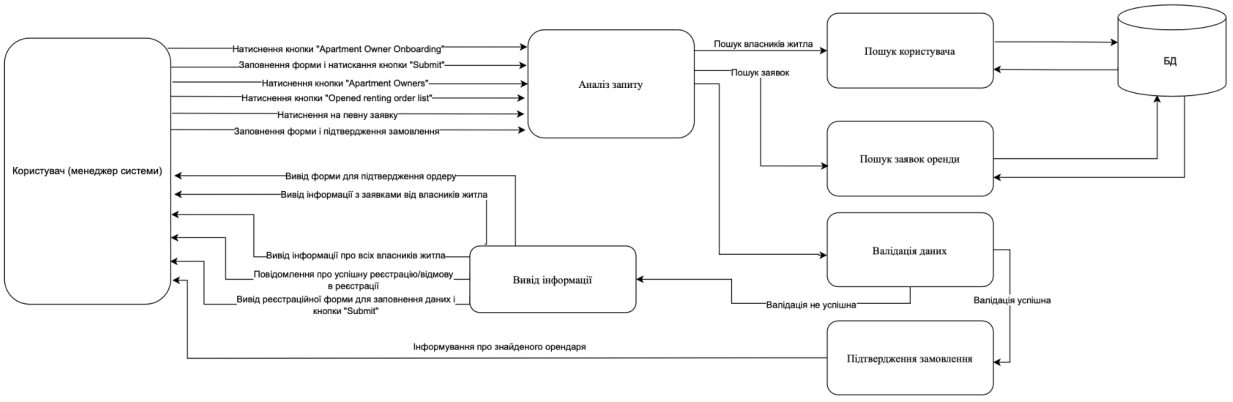


*Рис.2.5. DF-діаграма першого рівня декомпозиції ІС з Користувачем(менеджером системи)*

# ВИСНОВОК ДО РОЗДІЛУ 2

Проведено системний аналіз системи для здачі в оренду власного житла. Побудовано дерево цілей, створено DF-діаграми та проведено декомпозицію першого рівня, а також деталізовано певні процеси, а саме:

- процес взаємодії ІС з власником житла
- процес взаємодії ІС з менеджером системи

Визначено функції, які необхідно реалізувати в модулях, основними функціями є вивід інформації, аналіз запиту до системи, пошук користувача, валідація даних, створення нової події бронювання, комунікація з БД, аналізування отриманих повідомлень з чат-бота за допомогою AI[9].

# РОЗДІЛ 3
# Методи та засоби вирішення проблеми

## 3.1. Вибір та обґрунтування засобів розв’язання задачі

Різноманітні завдання програмування можна розв'язувати по-різному: може знадобитися створити функцію, створити окремий клас із методами та інше. Усі такі можливості об'єднуються в різноманітні підходи програмування, які також називаються парадигмами. Існує дві основні парадигми: імперативна та декларативна, а також їхні види.

Майже всі сучасні мови є мультипарадигмальними. Вони легко поєднують можливості імперативного та декларативного підходів.

Імперативна парадигма є однією з найстаріших парадигм програмування. Вона тісно пов’язана з архітектурою машини. Командна програма подібна до наказів, виражених наказовим способом у природних мовах. Це послідовність інструкцій, які процесор повинен виконувати крок за кроком. Основна увага цієї парадигми зосереджена на тому, як досягти мети. Парадигма складається з кількох інструкцій, і після виконання їх усіх результат зберігається або відображається.

Імперативне програмування поділяється на три великі категорії:

- парадигма процедурного програмування;
- об’єктно-орієнтоване програмування;
- підхід до паралельної обробки.

Парадигма процедурного програмування базується на концепції викликів процедур, у яких інструкції структуровані в процедури, також відомі як підпрограми або функції. Вони представляють собою перелік інструкцій, які повідомляють комп’ютеру, що робити крок за кроком. Іншими словами, комп’ютер приймає вхідні дані та обробляє їх послідовно, запам’ятовуючи кожну нову зміну.

Процедурне програмування підходить для програмування загального призначення для виконання типових завдань. Отже, це може бути невелика обчислювальна проблема, наприклад, обчислення факторіалу, або знаходження площі фігури, або відображення деякої інформації/фрази, як-от «Привіт, світ!». Крім того, код можна повторно використовувати в різних частинах програми без необхідності його копіювання. Мовами програмування, у яких реалізована парадигма процедурного програмування, є C, Java, C++, Kotlin, ColdFusion та Pascal.

Об’єктно-орієнтоване програмування або ООП – це парадигма, у якій програма написана як набір класів. Кожен клас має екземпляри, які називаються об’єктами. Клас — це спосіб загального опису сутності, визначення звичайного стану та поведінки, яка залежить від цього стану, а також звичайні правила взаємодії з цією сутністю. Формально клас розглядається як набір даних, таких як поля, атрибути, члени класу та функції, тобто методи для роботи з ними. ООП може вирішувати майже всі типові проблеми реального життя, де потрібно моделювати типові об’єкти та працювати з ними. Мовами програмування, які реалізували парадигму ООП, є Ruby, Kotlin, Java, C++, Python, Simula (перша мова ООП), Smalltalk, Visual Basic .NET та Objective-C.

Паралельна обробка сприяє зменшенню часу виконання інструкцій. Це здійснюється шляхом спільного використання або розпаралелювання інструкцій між кількома процесорами. Суть цього підходу можна виразити фразою: «розділяй і володарюй». Прикладами мов програмування, які підтримують паралельну обробку, є NESL (одна з найстаріших) та C / C++ (також підтримується завдяки деяким функціям бібліотеки).

Декларативне програмування - це парадигма програмування, в якій важливо визначити проблему та очікуваний результат її вирішення. Тобто на відміну від імперативної парадигми, де необхідно відповісти на запитання "Як це зробити?" потрібно поставити питання "Що потрібно зробити?" та "Який буде результат роботи?". Отже, замість того, щоб надавати покрокові інструкції,

ви повідомляєте системі, що вам потрібно, і дозволяєте їй спробувати знайти рішення. Декларативне програмування поділяється на типи парадигми логічного, функціонального та бази даних.

Логічне програмування — це парадигма програмування, яка значною мірою базується на формальній логіці. Будь-яка програма, написана мовою логічного програмування, — це набір речень у логічній формі, які виражають факти та правила щодо певної проблемної області. Отже, основні твердження логічного програмування такі:

- факти — це фундаментальні твердження про предметну область;
- правила дозволяють робити висновки щодо фактів у домені;
- запити – це запитання щодо цього домену.

Загалом тут стоїть завдання знайти відповідь на запит на основі фактів і правил. Основні сімейства мов логічного програмування включають Пролог (Prolog), програмування набору відповідей (Answer Set Programming - ASP) і журнал даних (Datalog).

Функціональне програмування – це методологія програмування, де процес обчислення інтерпретується як визначення значень функцій. Функція тут подібна до математичної, тобто це відображення незмінного масиву на новий масив з новими даними. Ця математична функція відрізняється від функції в імперативному програмуванні, де функція – це послідовність дій, що змінюють вихідні дані.

Ця методологія програмування ґрунтується на роботі з даними. Дані зберігаються в базі даних, і запити до цієї бази виконуються спеціальною мовою, такою як SQL. За допомогою цих мов можна отримувати доступ до даних для фільтрації, перетворень, обчислення статистики тощо. Програмні інструкції визначаються даними, а не жорстко закодованою серією кроків.

Програма бази даних є основою бізнес-інформаційної системи, що дозволяє створювати файли, вводити дані, оновлювати, виконувати запити та створювати звіти.

Отож, існують дві основні парадигми програмування: імперативна та декларативна. Імперативна парадигма зосереджена на досягненні результату за допомогою послідовних інструкцій, що обробляють дані послідовно. Декларативна парадигма зосереджена на визначенні задачі та намагається отримати очікуваний результат.

Більшість сучасних мов програмування можуть працювати з кількома парадигмами.

## 3.2. Засоби та технології реалізації

### *3.2.1. Мова програмування Python*

Python є однією з найпоширеніших мов програмування сьогодні і є легкою мовою для вивчення початківців через її читабельність. Це вільна мова програмування з відкритим кодом з широкими модулями підтримки та розвитком спільноти, легкою інтеграцією з веб-службами, зручними структурами даних та настільними додатками на основі GUI. Це популярна мова програмування для машинного навчання та програм глибокого навчання. Python використовується для розробки 2D-пакетів для зображень та 3D-анімації, таких як Blender, Inkscape та Autodesk. Він також використовувався для створення популярних відеоігор, включаючи Civilization IV, Vegas Trike та Toontown. Python використовується для таких наукових та обчислювальних програм, як FreeCAD та Abacus, а також на таких популярних веб-сайтах, як YouTube, Quora, Pinterest та Instagram.

### *3.2.2. Мова програмування JavaScript*

JavaScript - це об'єктно-орієнтована мова програмування, яка зазвичай використовується для створення інтерактивних ефектів всередині веб-браузерів. Typescript - це JavaScript, який додає до мови додаткове статичне введення тексту. Разом з HTML та CSS, JavaScript є однією з трьох основних технологій всесвітньої павутини. Він також використовується в передній частині декількох популярних веб-сайтів, таких як Google, Wikipedia, YouTube, Facebook та

Amazon. Більше того, він використовується в таких популярних веб-структурах, як AngularJS, Node.js і React.JS.

### *3.2.3. Мова програмування Node.js*

Node.js – одна з найкращих технологій, яку може використовувати розробник. Крім простоти у роботі, він також дуже швидко працює, надсилаючи відповідь клієнту за лічені секунди [1]. Основні характеристики:

- Node.js написаний мовою JavaScript. На сьогоднішній день це найпопулярніша мова програмування у світі. Більшість програмістів вже добре знайомі з JavaScript, його роботою та іншими базовими та просунутими концепціями. Це робить Node.js простим для розуміння та вивчення. Крім того, JavaScript також використовується в стеку технологій для розробки клієнтського інтерфейсу, а додаючи використання Node.js розробники можуть створювати повноцінні робочі веб-проєкти, знаючи тільки JavaScript;
- важлива особливість Node.js – асинхронний характер. Термін асинхронний означає, що сервер, створений з використанням Node.js, не повинен чекати, поки дані повернуться, при виконанні різних внутрішніх запитів. При цьому він також має неблокуючий введення-виведення. Це означає, що кілька різних процесів можуть виконуватись паралельно, не блокуючи один одного. Обидві ці властивості роблять Node.js вкрай швидким і забезпечують кращий інтерфейс користувача;
- архітектура, керована подіями, тобто код перед виконанням чекає на якусь подію. У Node.js, при старті виконання будь-якої операції, можна одразу ж передати функцію, яка має бути виконана після закінчення цього завдання. Такі функції називаються функціями зворотного дзвінка, також відомі як обробники події. Функції зворотного дзвінка вимагають менше ресурсів на стороні сервера і займають менше пам'яті;
- усі запити однопотокові та збираються в циклі обробки подій (Event loop). Це означає, що всі програми виконуються в одному потоці, починаючи з

отримання запиту і закінчуючи виконанням необхідного завдання та надсиланням відповіді клієнту назад. Ця функція Node.js запобігає повторному завантаженню запитів і скорочує час їх обробки, що робить його більш економічним у використанні;

- сумісність з кількома платформами, Node.js можна використовувати на різних системах, від Windows до Mac OS, Linux і навіть на мобільних платформах. Це дозволяє створити самодостатнє середовище у будь-якій галузі розробки;
- швидка потокова передача даних Node.js використовує двигун виконання JavaScript V8. Цей движок також використовується у браузері Google Chrome. Завдяки цьому робота Node.js значно прискорюється, а отже забезпечується дуже швидка потокова передача даних для веб-додатку.

### *3.2.4. Двигунець AI Dialog Flow*

Dialogflow (раніше називався API.AI) — це платформа розробки чат-ботів, що належить Google. Він використовується для обробки природної мови за допомогою машинного навчання. Dialogflow — це продукт на основі SaaS. Він працює на інфраструктурі Google і може легко масштабуватися для мільйонів користувачів [2,11]. Основні переваги Dialogflow серед інших платформ наступні:

- Ефективність. Чат-боти на основі штучного інтелекту можуть виконувати рутинні та повторювані завдання, що може звільнити співробітників для зосередження на більш складних і стратегічних обов’язках[9].
- багатоканальна підтримка: двигунець підтримує інтеграцію в один клік для понад 20 платформ, включаючи Slack, Facebook Messenger, Twitter;
- можна використовувати безкоштовно;
- багатомовна підтримка, двигунець підтримує більше ніж 14+ мов у всьому світі;
- має краще машинне навчання порівняно з конкурентами навіть із меншою кількістю навчальних даних.

### *3.2.5. Двигунець AI ChatGPT*

ChatGPT — це чат-бот зі штучним інтелектом (AI), який використовує обробку природної мови для створення людського розмовного діалогу. Мовна модель може відповідати на запитання та створювати різний письмовий вміст, включаючи статті, публікації в соціальних мережах, есе, код і електронні листи. Тобто, це форма генеративного штучного інтелекту — інструмент, який дозволяє користувачам вводити підказки для отримання людських зображень, тексту чи відео, створених штучним інтелектом.

Двигунець схожий на автоматичні служби чату, які можна знайти на веб-сайтах служби підтримки клієнтів, оскільки люди можуть задавати йому запитання або запитувати роз’яснення щодо відповідей ChatGPT.

GPT розшифровується як «Generative Pre-trained Transformer», що означає, як ChatGPT обробляє запити та формулює відповіді. Двигунець навчається за допомогою навчання з підкріпленням за допомогою зворотного зв’язку людини та моделей винагороди, які класифікують найкращі відповіді. Цей відгук допомагає розширити ChatGPT за допомогою машинного навчання для покращення майбутніх відповідей.

На відміну від інших чат-ботів, ChatGPT може запам’ятовувати різноманітні запитання, щоб продовжити розмову більш плавно. Деякі переваги двигунця включають наступне:

- Кращий час відповіді. ChatGPT забезпечує миттєві відповіді, що скорочує час очікування для користувачів, які шукають допомоги.
- Підвищена доступність. Моделі AI доступні цілодобово, щоб забезпечити постійну підтримку та допомогу.
- Персоналізація. Чат-боти штучного інтелекту можуть адаптувати відповіді до вподобань і поведінки користувача на основі попередніх взаємодій.

- Розуміння природної мови. ChatGPT розуміє та генерує людський текст, тому він корисний для таких завдань, як створення вмісту, відповіді на запитання, участь у розмовах і надання пояснень.

# ВИСНОВОК ДО РОЗДІЛУ 3

Здійснено ознайомлення з основними технологіями, які можуть бути використані для впровадження інформаційної системи здачі в оренду власного житла. Створено фундамент для розробки програмних модулів. Для розробки обрано мову програмування Node.js разом із AI двигунцем Dialogflow, які дозволять зробити реалізацію програмного продукту максимально ефективно, тому що Node.js дуже гнучка мова програмування та дозволяє писати код в різних парадигмах, дуже добре тримає навантаження на велику кількість запитів, чого потребує розробка чат-ботів. Легко масштабується при зростанні навантаження. Dialogflow є один з найкращих двигунців AI, які дуже добре інтегруються з чат-ботами. Для тренування моделей розпізнавання тексту Dialogflow має найкращій інтерфейс та низку дуже зручних функцій, така як вбудована статистика комунікації з юзерами, також найбільший плюс, що це розробка корпорації Google, та вони підтримують і регулярно оновлюють систему, що дозволяє розраховувати на довготривалу та стабільну роботу розробляємої системи для здачі в оренду власного житла.

# РОЗДІЛ 4
# Практична реалізація

## 4.1. Функціональна структура системи

В процесі розробки системи для здачі в оренду власного житла через чатбот, було створено ряд пов’язаних між собою обробників інформації. Створені обробники зв’язані ланцюжком, тим самим реалізуючи процес комунікації з власником житла. Система передає отриманий текст з чатбота та передає цей текст в Dialogflow, який за допомогою аналізу тексту та спираючись на ті фрази, котрі його навчили, віддає відповідь, чого хоче користувач чат-бота. Після отримання інформації від Dialogflow вже в коді побудована стратегія обрання відповідного обробника, котрий може одразу дати відповідь користувачу, або запитати додаткові дані, якщо таких недостатньо, або дані введені некоректно, наприклад користувач ввів неіснуючу дату то обробник відправить користовачу уточнення, щоб він ввів коректне значення.

Обробники розроблялись з використанням модульного підходу, суть якого полягає в тому, що система будується окремими незалежними модулями, в нашому випадку це обробники, конектор до бази даних, конектор до Dialogflow. Загалом модульне програмування тісно пов'язане зі структурованим та об'єктно-орієнтованим програмуванням. Усі ці підходи спрямовані на досягнення спільної мети - побудова великих програмних систем та систем шляхом розкладання на менші компоненти. Навіть якщо історичне використання цих термінів було неоднозначним, зараз "модульне програмування" означає розбиття коду програми на високому рівні на окремі модулі. Структуроване програмування включає в себе використання коду низького рівня та структурованого управління потоком, а об'єктно-орієнтоване програмування базується на використанні об'єктів даних як основної структури даних. Такий підхід розробки функціональності програми дозволяє забезпечити значну гнучкість та легкість в розширенні, оновленні функціональностей, бо

потрібно буде реалізувати новий обробник та підключити його до бізнес логіки програми і не потрібно вносити правки в інші модулі, що пришвидшує розробку та подальшу підтримку кодової бази системи.

Серед наявних в даному проєкті модулів можна визначити:

- модуль підключення з базою даних
- модуль пошуку необхідної інформації
- модуль комунікації з Dialogflow
- модуль комунікації з Telegram Bot Api

В ході робіт було спроєктовано сховище даних, яке в подальшому використовувалося для інтеграції з розробленою інформаційною системою з метою автоматизації та полегшення функціонування програмного забезпечення.

Модуль пошуку необхідної інформації містить в собі всі операції з базою даних, запит потрібних даних, щоб система мала змогу коректно працювати та обирати потрібні їй дані із сховища, використовуючи не пряме підключення до бази даних, а через модуль, котрий в собі інкапсулює всі деталі та структуру запиту до бази даних. Важливими функціями цього модулю також є збереження та оновлення існуючих записів в базі даних, такі операції відбуваються в момент внесення нових даних до системи, наприклад створення нового замовлення на здачу в оренду, або додавання нового власника житла[5,7].

Модуль комунікації з Dialogflow містить в собі всі операції, котрі можливо виконати з цим сервісом, а саме відправити запит з текстом від користувача та отримати результат аналізу. Також модуль разбирає відповідь від сервісу, та форматує інформацію в об’єкт за даними у внутрішній формат, котрий зрозумілий для розробленої системи. Таким чином, розроблена система не має залежності від змін формата відповіді з Dialogflow, це все інкапсулюється в цьому модулі[2,6,10].

Модуль комунікації з Telegram Bot Api містить повну інтеграцію з сервісом Telegram для відправки повідомлень у відповідь користувачу[3].

## 4.2. Структура бази даних

База даних (БД) - це структуроване сховище даних, яке дозволяє ефективно зберігати, організовувати і керувати інформацією. Бази даних використовуються для зберігання інформації так, щоб було зручно виконувати операції додавання, оновлення, видалення і вибору даних. Бази даних використовуються в різних галузях, від бізнесу та науки до веб-розробки та мобільних додатків. Вони дозволяють ефективно зберігати та отримувати доступ до великої кількості даних, спрощуючи управління інформацією для користувачів і програм.

Є кілька основних типів баз даних, і кожен з них має свої особливості та застосування в залежності від конкретних вимог проєкту та задачі. Загалом можна виділити два найпопулярніші типи баз даних:

- Реляційні бази даних (RDBMS) - використовують табличну структуру для зберігання даних, де кожна таблиця має набір стовпців і рядків, таблиці зв’язуються між собою за допомогою первинних та вторинних ключів. У таблицях може бути багато стовпців, кожний з яких позначено описовою назвою (скажімо, віком, наприклад) та мають певний тип даних. Зазвичай реляційні бази даних використовують мову SQL для доступу до даних.

  Хоча реляційні бази даних є потужними та широко використовуваними, вони також мають свої мінуси, наприклад проблема гнучкості схеми, зміни в схемі бази даних можуть бути складними та важкими для впровадження, тому часто для малих проєктів або для динамічних проєктів, коли схема даних часто змінюється, це може стати достатньо великою проблемою, тому для нашого проєкту було обрано другий тип БД.
- Нереляційні бази даних (NoSQL) - використовують різноманітні моделі для зберігання та отримання даних, не обов'язково в табличній формі, наприклад є такі підтипи:

- Документ-орієнтовані (наприклад, MongoDB): Зберігають дані у вигляді документів, зазвичай у форматі JSON або BSON.
- Ключ-значення (наприклад, Redis): Зберігають пари ключ-значення.

Виходячи з аналізу наявних типів баз даних для проєкту було обрано нереляційну базу даних, а саме MongoDB, тому що це популярна нереляційна база даних, орієнтована на документи з дуже гнучкою схемою даних, що для наявної системи дуже потрібна гнучкість у зберіганні та обробці даних, щоб можна було просто та швидко зберігати дані [4]. Також змінювати схему БД в ході життєвого циклу продукту. Основні переваги MongoDB включають в себе:

- Гнучкість сховища - дозволяє зберігати дані в форматі BSON (бінарний JSON), що спрощує роботу з різноманітними типами даних та дуже просто змінювати схему даних
- Горизонтальне масштабування - легко розширюється на великі обсяги даних та високий трафік шляхом додавання нових серверів
- Індексація - підтримує різні види індексів для оптимізації швидкодії пошукових операцій
- Зручна мова запитів - використовує JavaScript-подібну мову запитів для зручного взаємодії з базою даних, що є дуже великим плюсом для проєкта на Node.js

Під час аналізу предметної області, визначено основні дані, які мають зберігатись системою. В подальшому всі ці дані були структуровані у вигляді відношень колекцій(таблиць) в базі даних MongoDB. Спроєктоване сховище (рис 4.1) включає в себе наступні колекції(таблиці):

- apartment_buildings
  - id
  - title

- users
    - id
    - firstName
    - lastName
    - email
    - chatId
    - apartmentBuildingId
    - apartmentSize
    - apartmentAddress
- orders
    - id
    - userId
    - customerId
    - orderStatus
    - serviceType
    - date
    - price
- customers
    - id
    - chatId
    - email
    - rating
    - firstName
    - lastName

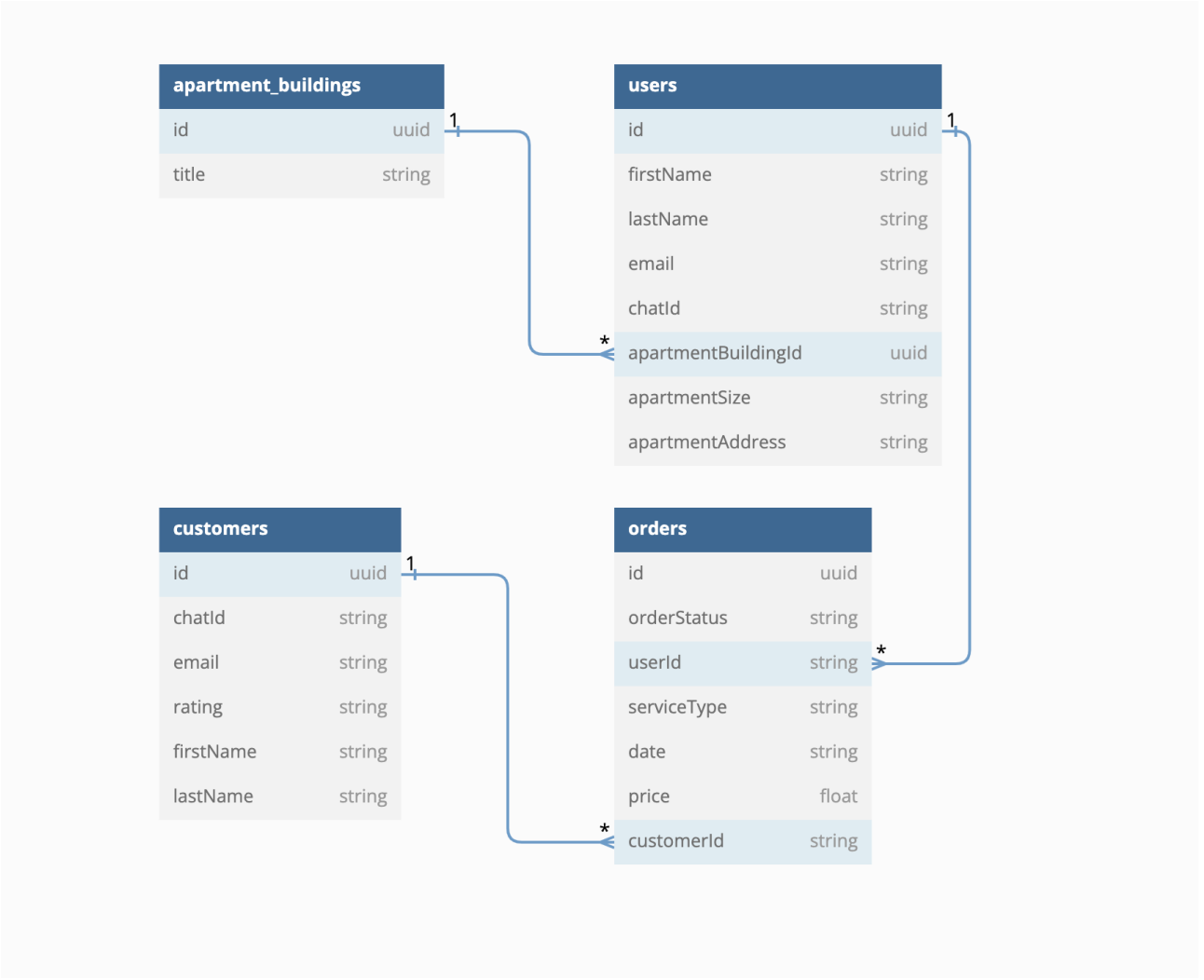


*Рис. 4.1. Схема бази даних*

## 4.3. Інтерфейс системи

Розроблена система має декілька інтерфейсів:

- бот в месенжері Telegram
- вебсайт адмін панель для керуванням системою

### 4.3.1 Інтерфейс бота в Telegram

Інтерфейс бота нам повністю забезпечує месенжер Telegram, котрий є на всі популярні платформи Android, IOS, MacOS, Windows.

При першому контакті з чат-ботом власник житла побачить пусту історію повідомлень та знизу кнопку «Запустити»(Рис. 4.2).

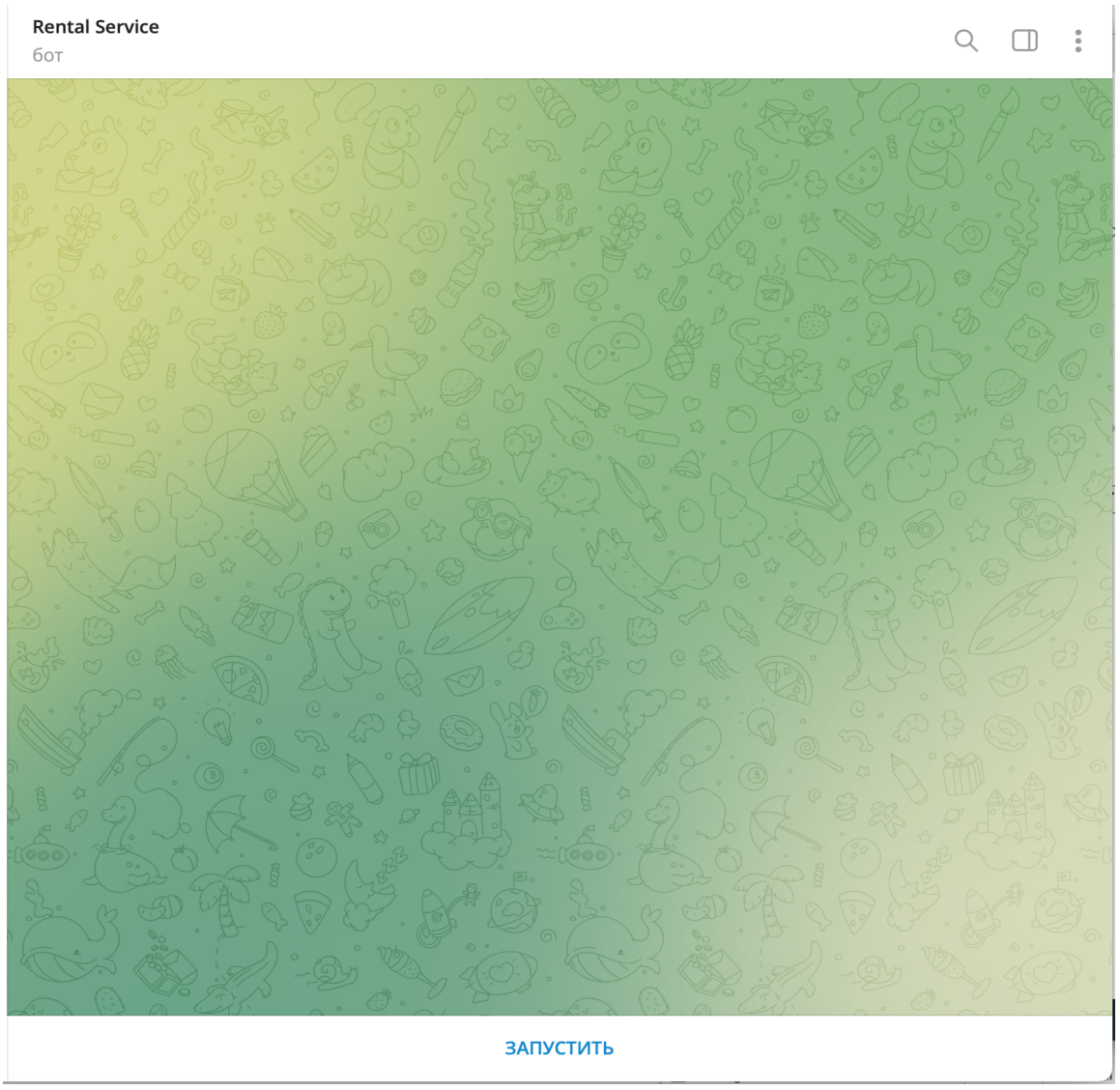


*Рис. 4.2. Старт діалога з ботом*

Після натискання на кнопку кнопку «Запустити» буде розпочато діалог з чат-ботом, автоматично в чат буде відправлено команду /start. Якщо юзер раніше не зареєстрований менеджером в системі як власник житла, то юзер отримає відповідне повідомлення, що йому потрібно звернутись до менеджера передавши йому свій chat-id, щоб юзера зареєстрували в системі (Рис. 4.3)

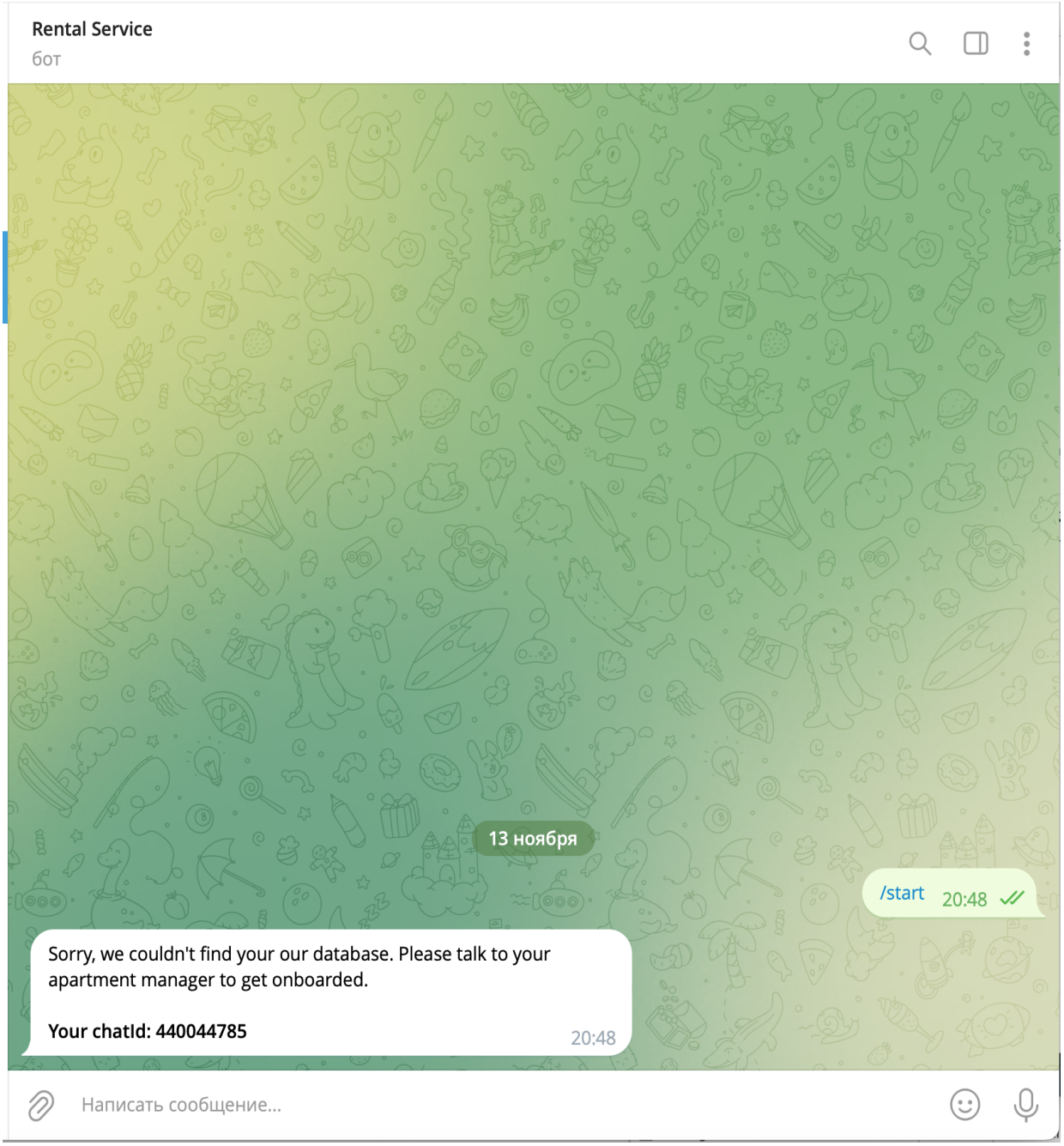


*Рис. 4.3. Юзер не зареєстрований в системі*

Після того, як менеджер зареєструє юзера в системі як власника житла, юзер отримає про це повідомлення (Рис. 4.4)

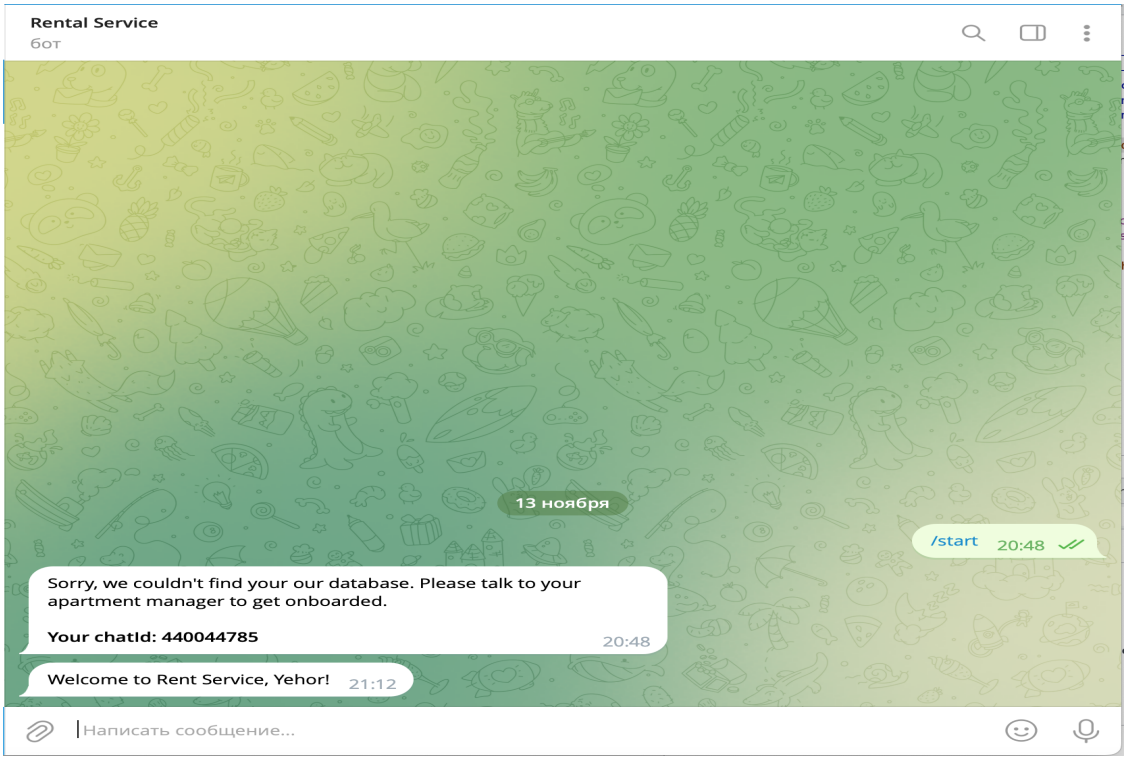


*Рис. 4.4. Юзер отримав повідомлення про реєстрацію*

Після цього юзер може повноцінно користуватись ботом, наприклад привітатись з ботом, та отримати який сервіс бот може забезпечити. Бот за допомогою Dialogflow зрозуміє, що з ним привітались, та відправить привітальне повідомлення. Так само й з наступними фразами відправленими боту (Рис. 4.5)

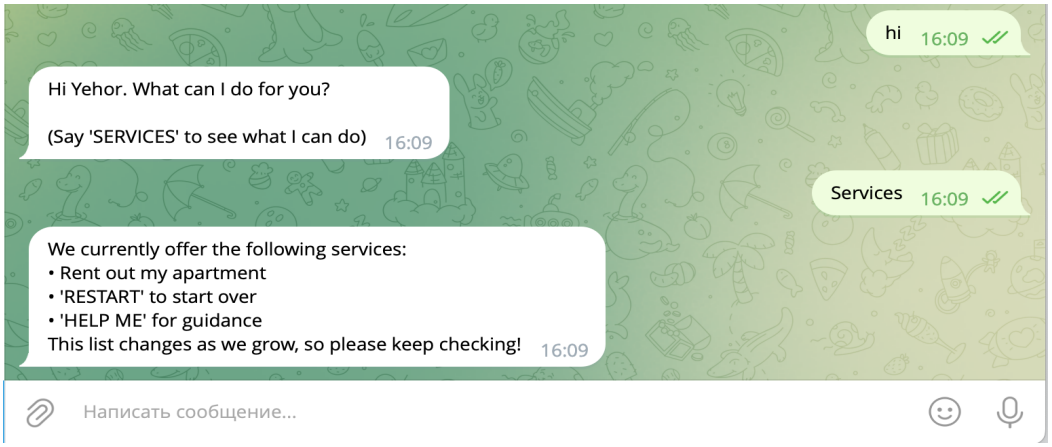


*Рис. 4.5. Запитали в бота сервіси*

Після цього, наприклад ми хочемо створити заявку на здачу власного житла. Для цього нам потрібно написати боту в довільній формі, що ми хочемо здати в аренду житло бот за допомогою Dialogflow розпізнає чого хоче юзер, та запустить опитування всієї потрібної інформації для створення заявки. Відповідати можна теж в довільній формі, бот розпізнає потрібні дані(Рис. 4.6)


*Рис. 4.6. Створення заявки на здачу в оренду житла*

Після створення заявки її отримає менеджер системи та коли знайде потенційного орендаря, в бота прийде повідомлення (Рис. 4.7)

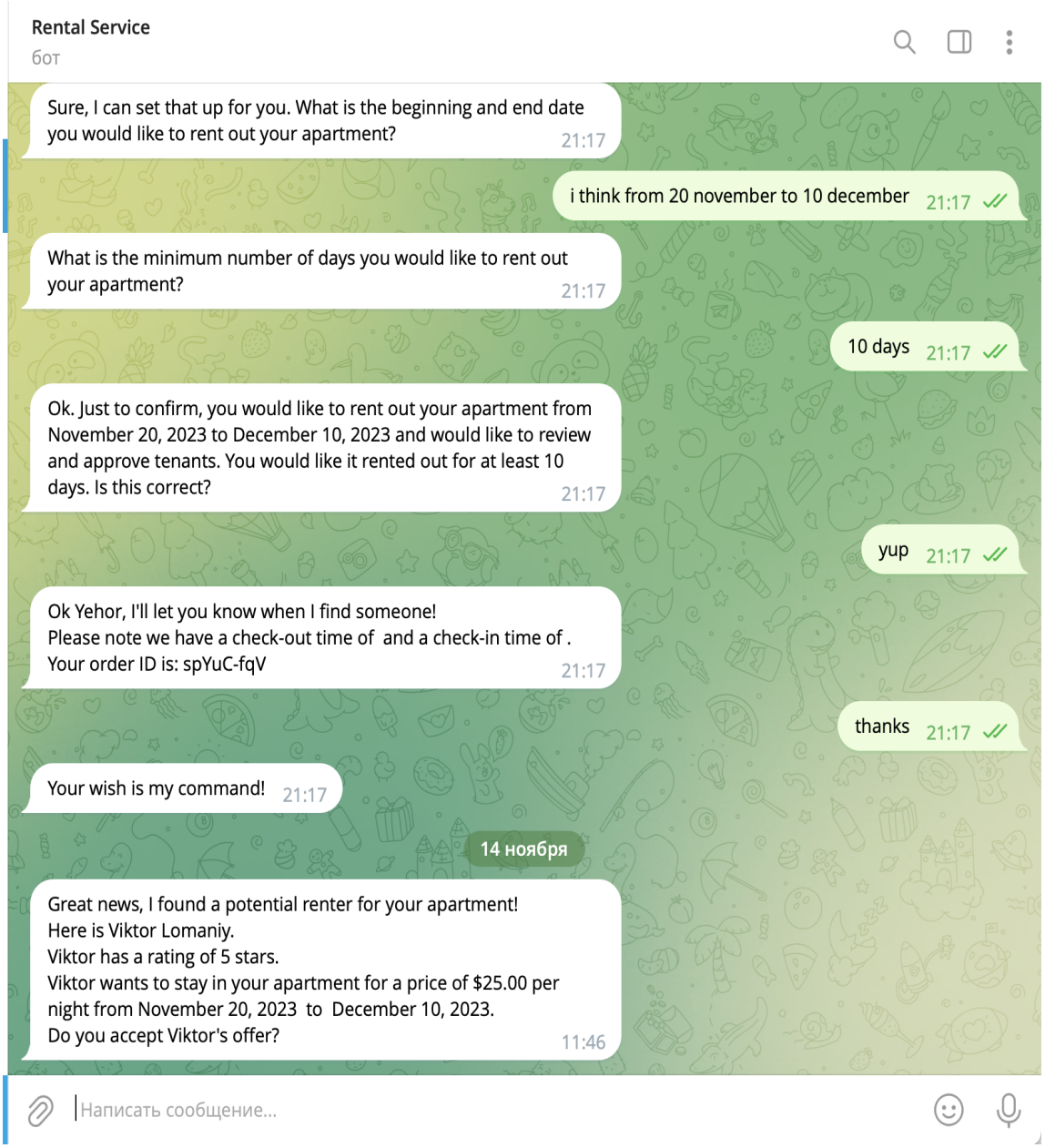


*Рис. 4.7. Повідомлення з потенційним орендарем*

Власник житла може підтвердити цього орендаря, якщо йому все підходить, або відмовится від цієї пропозиції, та менеджер продовжить пошук нового орендаря(Рис. 4.8)

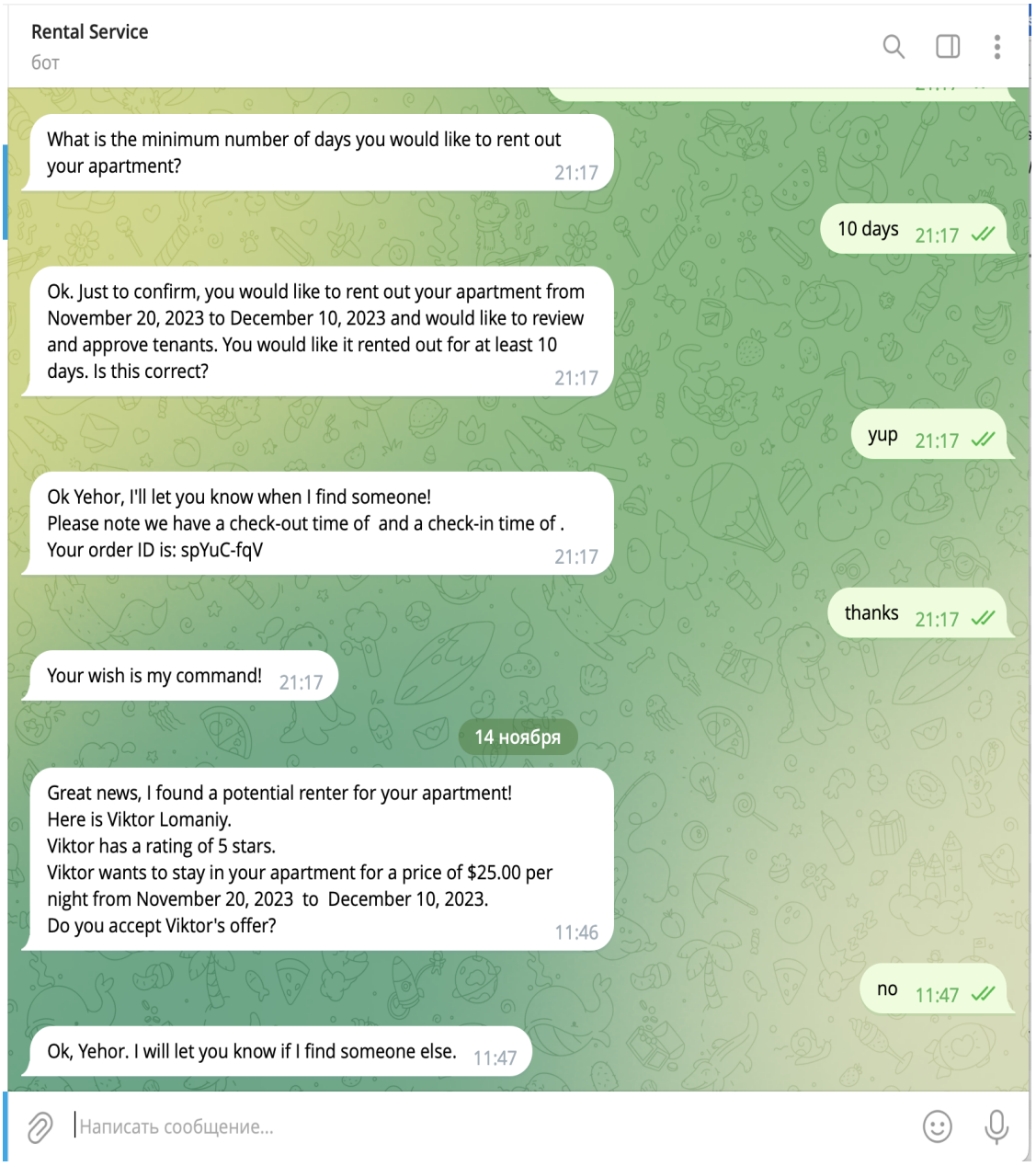


*Рис. 4.8. Власник житла відмовив в оренді*

Коли менеджер знайде нового орендаря, то в чат прийде також сповіщення, але на цей раз власник житла вже вирішив, що йому все підходить, та підтвердив пропозицію. В такому випадку бот закриває заявку як виконану, та менеджер починає працювати з орендарем(Рис. 4.9)

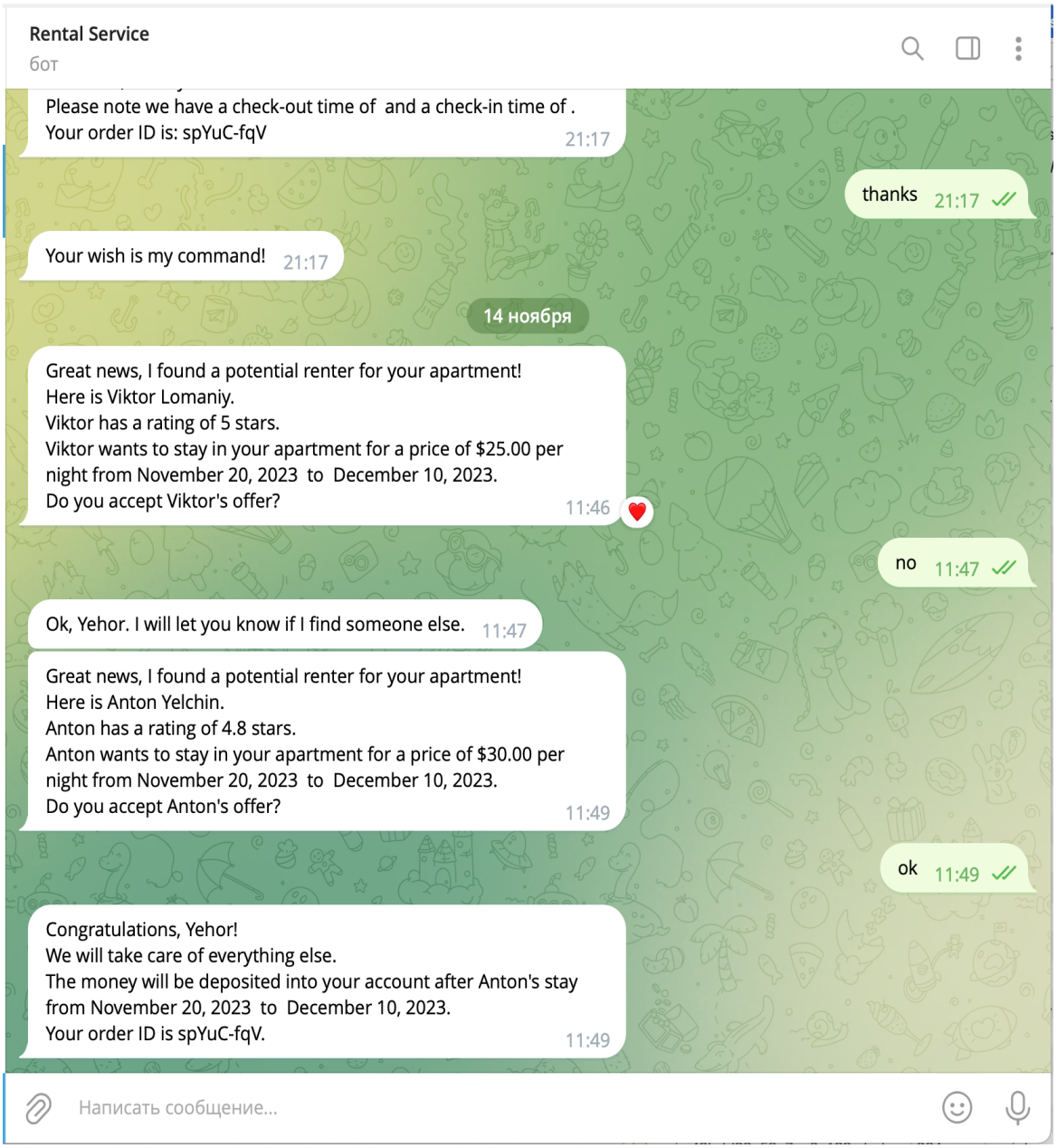


*Рис. 4.9. Власник житла підтвердив пропозицію*

### 4.3.2 Інтерфейс адмін панелі

Головна сторінка графічного інтерфейсу адмін панелі для керування системою має навігацію до доступного функціоналу адмін панелі (Рис. 4.10):

- Кнопка «Apartment Owner Onboarding»
- Кнопка «Apartment Owners» переглянути всіх власників житла в системі
- Кнопка «Opened renting orders list» переглянути всі відкриті замовлення на здачу своєю квартири в оренду


*Рис. 4.10. Головна сторінка*

При натискані на кнопку «Apartment Owner Onboarding» переводить на сторінку додавання нового власника житла, котрий зможе користуватись чат-ботом. На сторінці є форма для заповнювання інформації про власника житла з не активною кнопкою «Submit»(Рис. 4.11), яка містить в собі:

- Поле для вводу Ім’я - валідація, тільки літери, дефіс, апостроф
- Поле для вводу Призвище - валідація, тільки літери, дефіс, апостроф
- Поле для вводу Telegram chatId - валідація, тільки цифри
- Поле для вводу Адреси поштової скриньки - валідація, на дійсність емейл адреси
- Поле для вводу Адреса житла - валідація, дозволені всі символи
- Поле для вводу Обрати розмір житла - вибирається з обмеженого списку, а саме: Studio, 1 Bedroom, 2 Bedroom

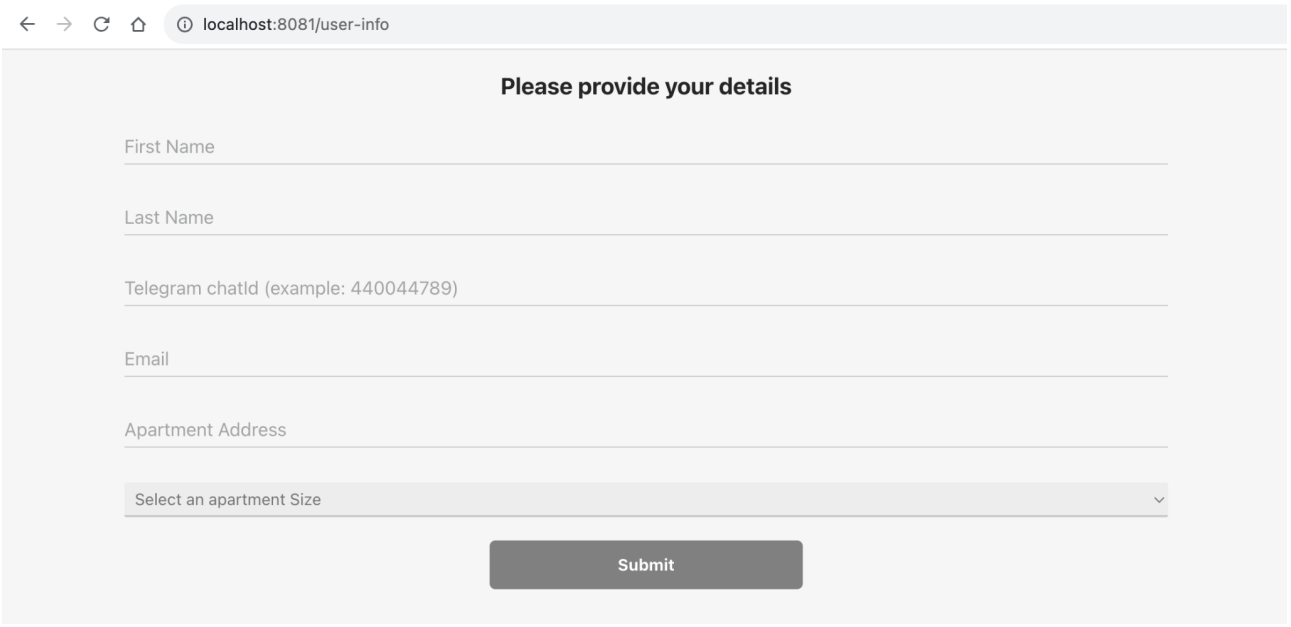


*Рис. 4.11. Сторінка Послуги та ціни*

Після заповнення всіх полей кнопка «Submit» розблокується(Рис. 4.12), після натискання на яку новий власник збережеться в базу даних та зможе користуватись чат-ботом та отримає про це повідомлення в чат-бот. Також ми отримаємо повідомлення на інтерфейсі про успішне збереження (Рис. 4.13)

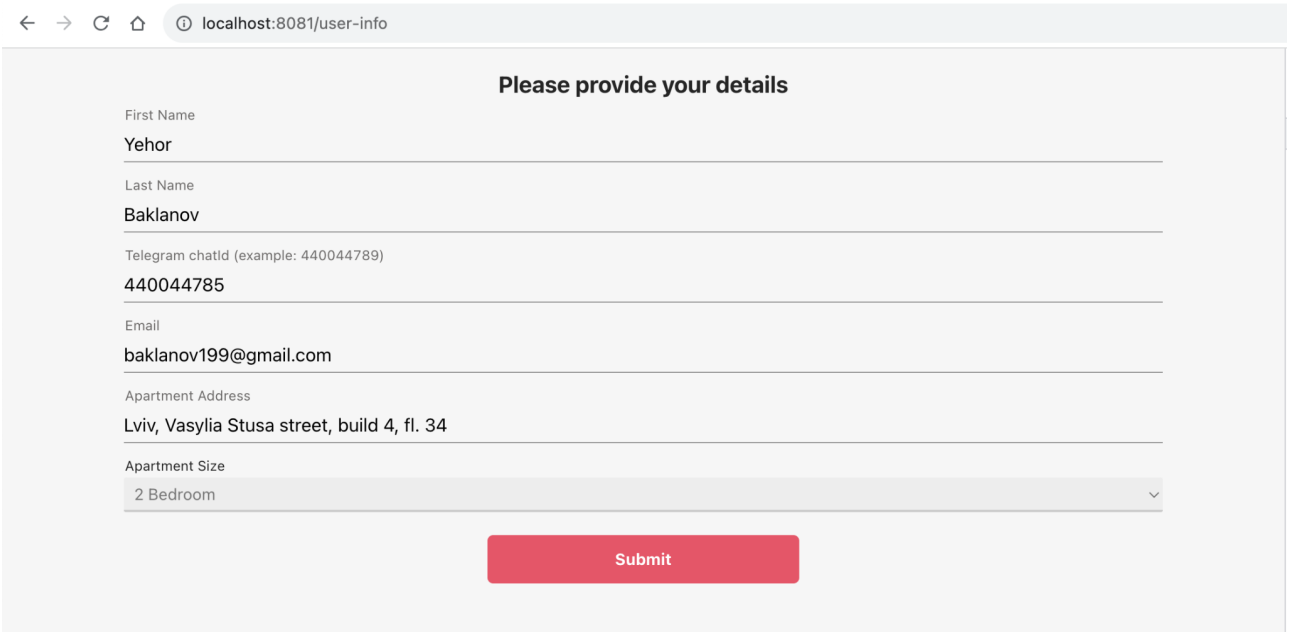


*Рис. 4.12. Сторінка з заповленними данними власника житла*

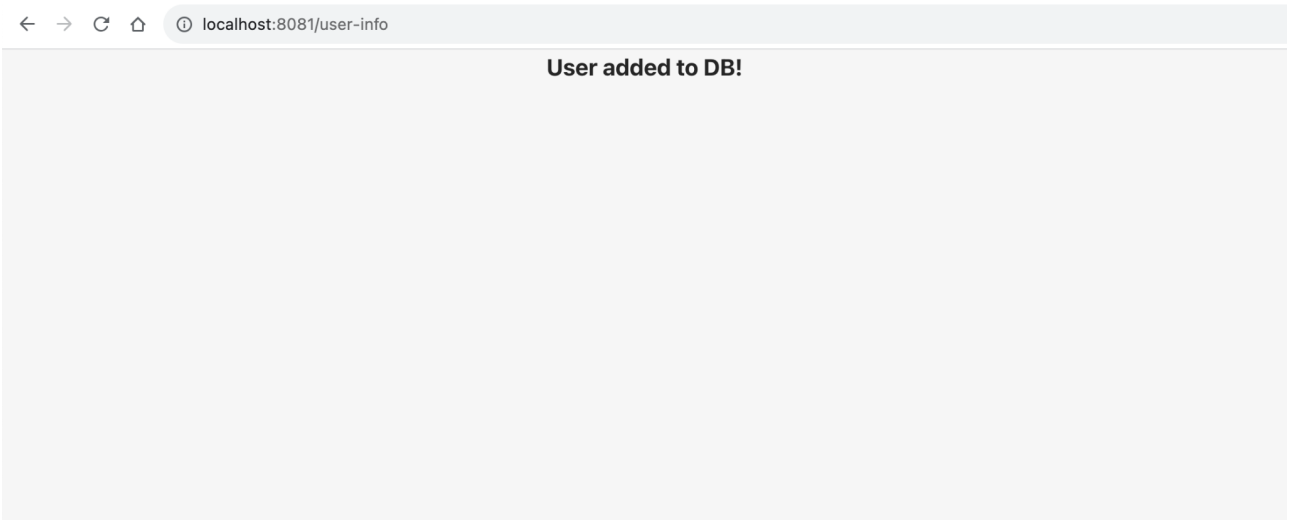


*Рис. 4.13. Сторінка власника житла успішно додано*

При натискані на кнопку «Apartment Owners» на головній сторінці (Рис. 4.10) відкривається сторінка на можна переглянути всіх зареєстрованих власників житла в системі(Рис. 4.14):

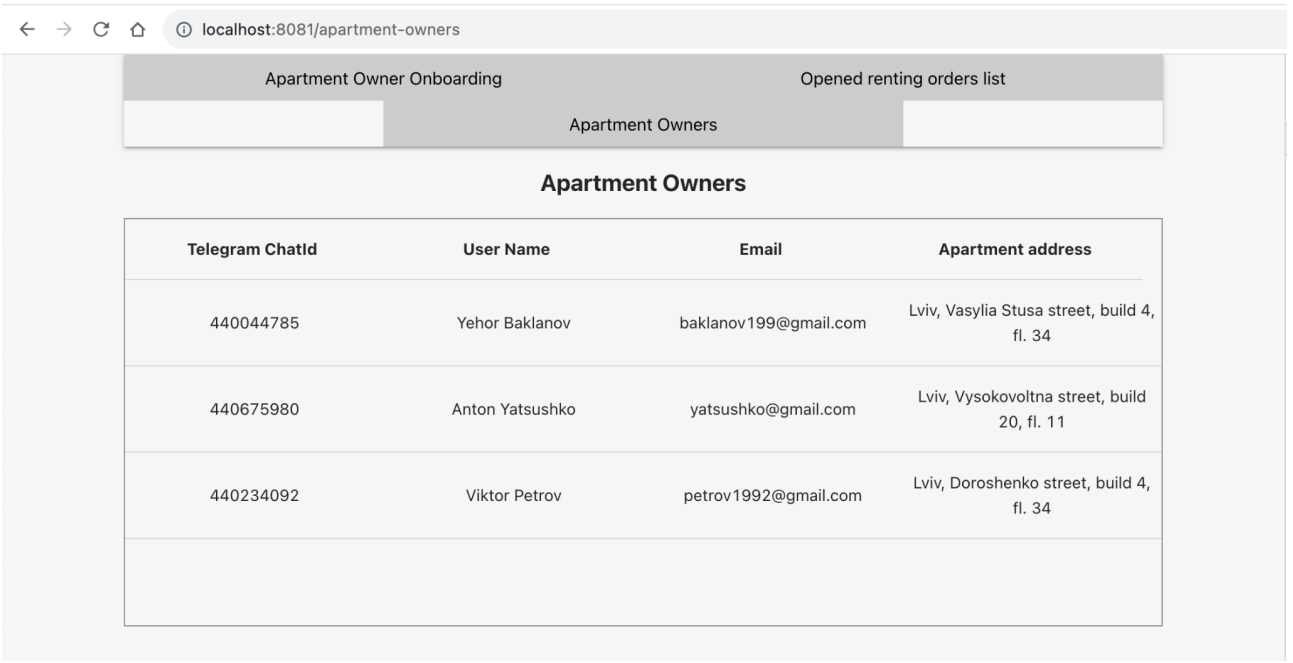


*Рис. 4.14. Сторінка всіх зареєстрованих власників житла*

При натискані на кнопку «Opened renting orders list» на головній сторінці(Рис. 4.10) відкривається сторінка на котрій можна переглянути ві відкриті для заявки на здачу свого житла (Рис. 4.15):

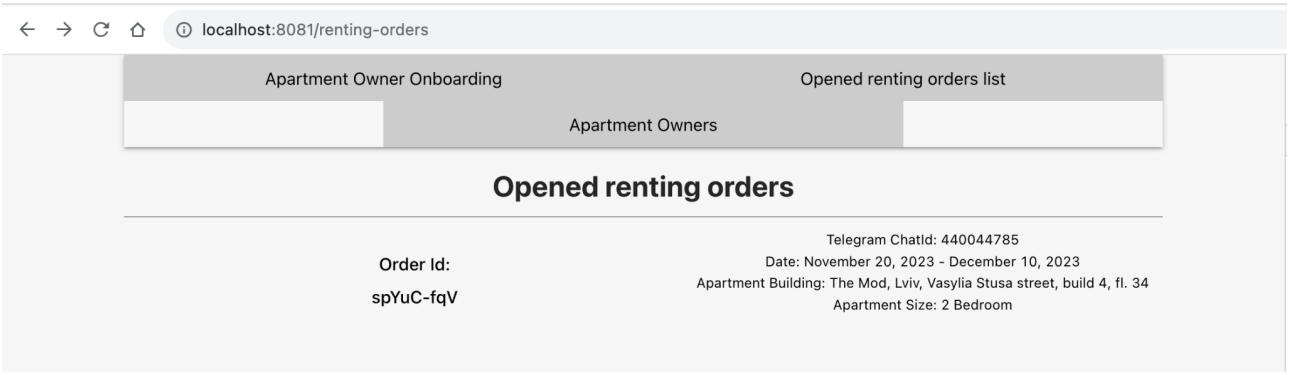


*Рис. 4.15. Сторінка з заявками на здачу свого житла*

При натискані на потрібну нам заявку, з'являється сторінка з деталями заявки. На цій сторінці ми бачимо детальну інформацію про заявку на здачу в оренду житла та є форма для заповнення інформації про потенційного орендаря житла з не активною кнопкою «Submit»(Рис. 4.16), яка містить в собі:

- Поле для вводу Номера телефону - валідація, номера телефонів тільки з + в початку та довжина не більше 14 цифр
- Поле для вводу Ім'я - валідація, тільки літери, дефіс, апостроф
- Поле для вводу Призвище - валідація, тільки літери, дефіс, апостроф
- Поле для вводу Адреси поштової скриньки - валідація, на дійсність емейл адреси
- Поле для вводу Рейтингу орендаря - валідація, тільки цифри
- Поле для вводу Ціна за ніч - валідація, тільки цифри

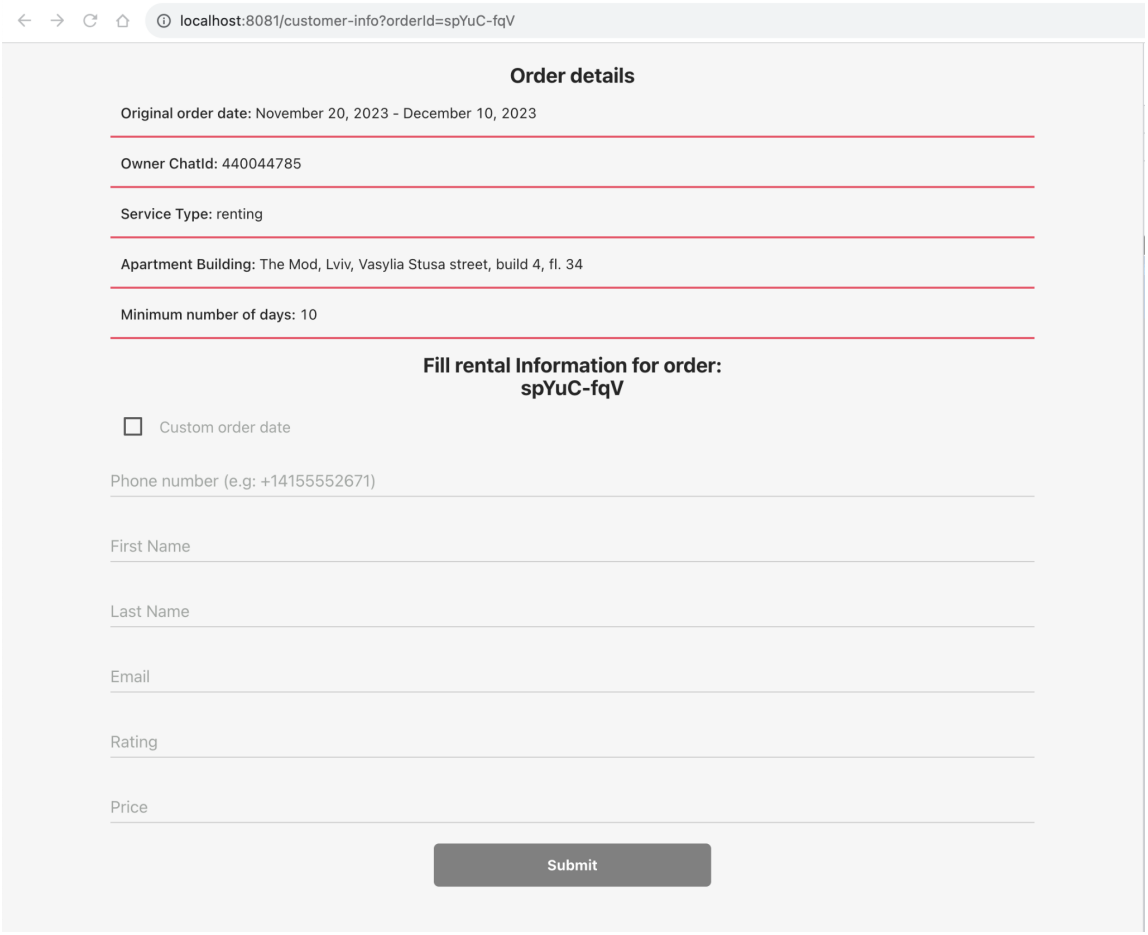


*Рис. 4.16. Сторінка деталі заявки на здачу в оренду*

Після заповнення всіх полей кнопка «Submit» розблокується(Рис. 4.17), після натискання на яку до бази даних збережеться інформація про орендара житла, та власнику житла за цією заявкою буде відправлене повідомлення з пропозицією про здачу житла цьому орендару. Після збереження до бази даних та відправки повідомлення власнику житла, ми отримуємо повідомлення на інтерфейсі про успіх(Рис. 4.18).

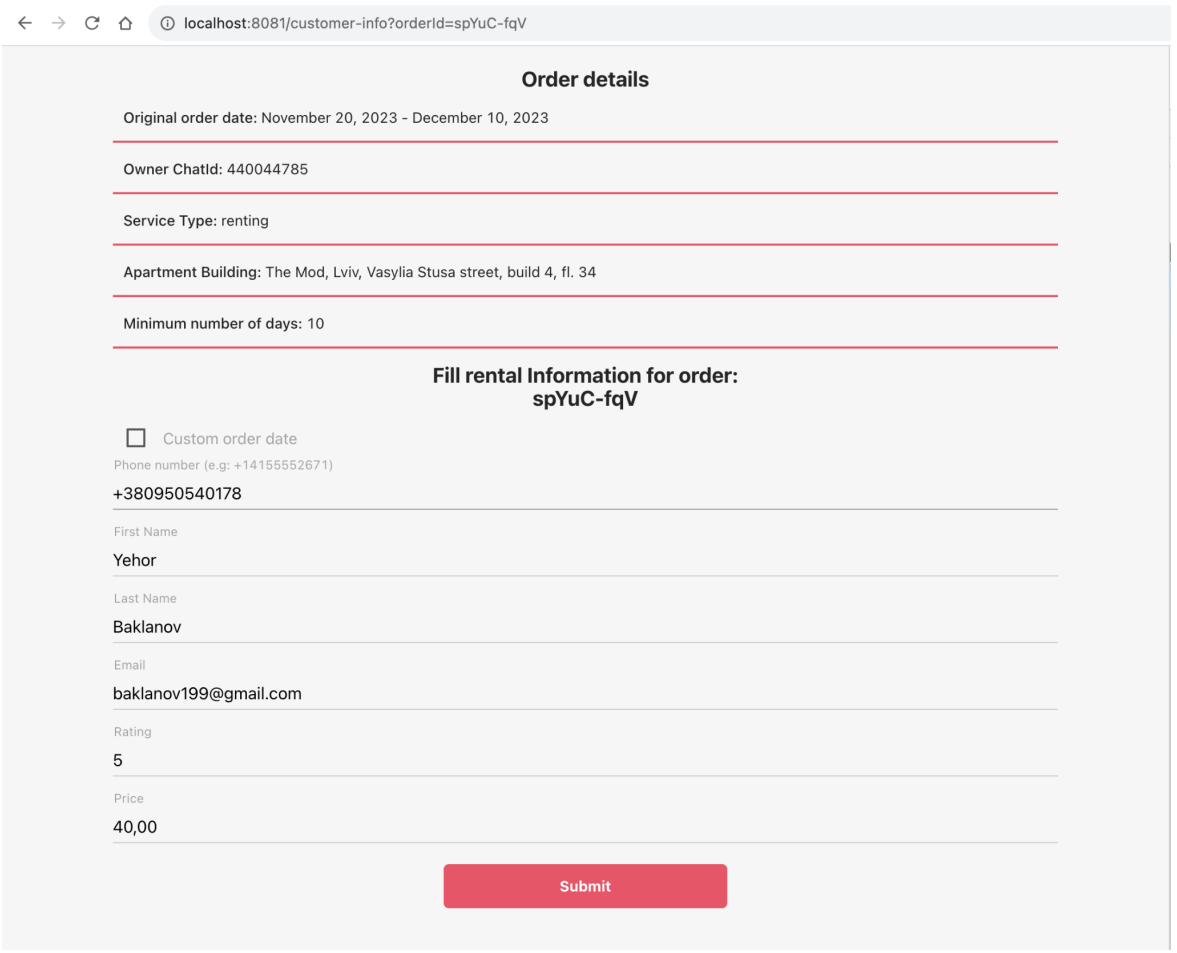


*Рис. 4.17. Сторінка з заповненими даними орендаря*

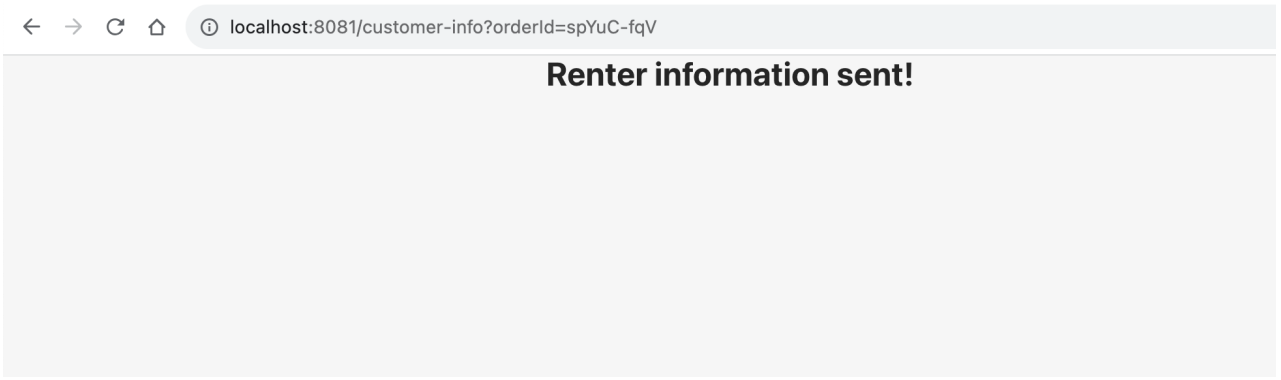


*Рис. 4.18. Сторінка про успішну відправку інформації про орендаря*

## 4.4. Інструкція користувачеві

Загалом в системі існує два типи користувачів: власник житла, менеджер системи.

Власник житла має доступ тільки до чат-бота, щоб мати можливість користуватись чат-ботом власник житла повинен звернутись до менеджера, котрий його зареєструє в системі, та тільки після реєстрації власник житла зможе на повну користуватись можливостями чат-бота. Через чат-бот він може сформувати заявку на здачу в оренду власного житла в проміжок необхідних дат, та після того як менеджер знайде потенційного орендаря, власник житла може дозволити здати в оренду знайденому орендарю, або відхилити пропозицію, в такому разі менеджер продовжить пошуки орендаря.

Менеджер системи має доступ до веб-сайту адмін панелі, на котрому може керувати системою, а саме:

- додавати нових власників житла
- переглядати всіх наявних власників житла в системі
- переглядати всі відкриті заявки на здачу в оренду житла від її власників.
- відправляти пропозиції власнику житла за заявкою на здачу нерухомості в оренду певному орендарю
- додавання нового орендаря в систему, виконується в процесі формування пропозиції за заявкою

# ВИСНОВОК ДО РОЗДІЛУ 4

Розроблено систему для здачі в оренду власного житла за допомогою AI чат-бота та адмін панель для керування системою. Реалізовано весь потрібний функціонал згідно вимог розділу 2.2.

За допомогою реалізованої системи, власник житла може через чат-бот здати своє житло на певний період та не комунікувати з орендарями, а цим всім займається менеджер. Чат-бот за допомогою Dialogflow розрізняє повідомлення власника житла, визначає чого хоче власником житла та відносно потреби будує комунікацію з ним.

Система вирішє проблему власників житла, котрі хочуть здавати в оренду житло при цьому без контактування з орендарями, а мати можливість віддати квартиру посереднику, котрий самостійно займається пошуком орендарей, комунікує з ними та повністю контролює процес здачі в оренду житла після чого власник житла отримує гроші.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Node.js Developers Docs [Електронний ресурс]: Режим доступу: https://nodejs.org/dist/latest-v20.x/docs/api/
2. Google Dialogflow Docs [Електронний ресурс]: Режим доступу: https://cloud.google.com/dialogflow/es/docs/reference
3. Telegram Bot Api Docs [Електронний ресурс]: Режим доступу: https://core.telegram.org/bots/api
4. Node.js MongoDB Docs [Електронний ресурс]: Режим доступу: https://www.mongodb.com/docs/drivers/node/current/
5. Node.js Express Framework Docs [Електронний ресурс]: Режим доступу: https://expressjs.com/
6. Dialogflow guides [Електронний ресурс]: Режим доступу: https://habr.com/ru/articles/502688/
7. Основи розробки чат-ботів на базі Node.js + Telegram [Електронний ресурс]: Режим доступу: https://habr.com/ru/articles/740796/
8. Використання технологій чат-ботів в умовах цифрової трансформації бізнесу [Електронний ресурс]: Режим доступу: http://confmanagement.kpi.ua/proc/article/view/230945
9. Rule-based chatbots vs Artificial intelligence chatbots [Електронний ресурс]: Режим доступу: https://mindtitan.com/resources/guides/chatbot/types-of-chatbots/
10. Best Practices for building Dialogflow Chatbot [Електронний ресурс]: Режим доступу: https://medium.com/axlewebtech/best-practices-for-building-dialogflow-chatbot-50b305bdd5d
11. Dialogflow vs Competitors [Електронний ресурс]: Режим доступу: https://research.aimultiple.com/dialogflow/

12. Офіційний веб-сайт Booking [Електронний ресурс]: Режим доступу: https://www.booking.com/
13. Офіційний веб-сайт Kayak [Електронний ресурс]: Режим доступу: https://www.kayak.com/
14. Офіційний веб-сайт Priceline [Електронний ресурс]: Режим доступу: https://www.priceline.com/
15. Офіційний веб-сайт Hotels.com [Електронний ресурс]: Режим доступу: https://www.hotels.com/
16. Офіційний веб-сайт HotelsCombined [Електронний ресурс]: Режим доступу: https://www.hotelscombined.com/
17. Офіційний веб-сайт Hotwire [Електронний ресурс]: Режим доступу: https://www.hotwire.com/
18. Офіційний веб-сайт Google Travel [Електронний ресурс]: Режим доступу: https://www.google.com/travel

## Додаток А

## Файл package.json

```
{
 "name": "rental-service",
 "version": "1.0.0",
 "author": {
  "name": "Yehor Baklanov",
  "email": "baklanov199@gmail.com"
 },
 "main": "server.js",
 "scripts": {
  "start": "node index.js",
  "start:dev": "nodemon index.js"
 },
 "license": "ISC",
 "dependencies": {
  "actions-on-google": "^2.12.0",
  "axios": "1.6.1",
  "body-parser": "^1.19.0",
  "cors": "^2.8.5",
  "dialogflow": "^1.2.0",
  "dialogflow-fulfillment": "^0.6.1",
  "dotenv": "^7.0.0",
  "express": "4.17.0",
  "lodash": "^4.17.15",
  "moment": "^2.24.0",
  "moment-timezone": "^0.5.31",
  "mongodb": "4.17.1",
  "pb-util": "^0.1.3",
```

```
    "shortid": "^2.2.15",
    "string-template": "^1.0.0",
    "uuid": "^8.2.0",
    "winston": "^3.2.1"
  },
  "devDependencies": {
    "@types/axios": "^0.14.0",
    "claudia": "^5.12.0",
    "claudia-local-api": "^2.0.0",
    "eslint": "^5.16.0",
    "eslint-config-airbnb": "^17.1.0",
    "eslint-config-airbnb-base": "^13.1.0",
    "eslint-plugin-import": "^2.17.3",
    "nodemon": "^1.19.1",
    "prettier": "3.0.3"
  }
}
```

**Файл server.js**

```
const express = require('express');
const cors = require('cors');
const { authorization } = require('./middlewares/authorization');
const mongodb = require('./db/mongodb-connector');
const Controllers = require('./controllers');
const logger = require('./helpers/logger');
const Config = require('./config');
const app = express();
async function startServer() {
  app.use(cors());
  app.use(express.json());
```

```
  app.use(express.urlencoded({ extended: true }));
  await mongodb.initConnectionToDB();
  app.post('/telegram-webhook', Controllers.telegramBotControllerHandler);
  app.post('/fulfillment', authorization, Controllers.setFollowupEventHandler);
  app.get('/get-all-users', Controllers.getAllUsersHandler);
  app.get('/orders-info', Controllers.getOrdersInfoHandler);
  app.get('/customer-info/:phoneNumber', Controllers.getCustomerInfoHandler);
  app.get('/apartment-buildings-list', Controllers.getApartmentBuildingsListHandler);
  app.post('/order-info/:orderId', Controllers.getOrderInfoHandler);
  app.post('/user-info', Controllers.createNewuserHandler);
  app.post('/customer-info/:orderId', Controllers.sendCustomerInfoHandler);
  process.on('unhandledRejection', (reason, promise) => {
    logger.error('[Critical] Unhandled Rejection at Promise', { reason, promise });
    process.exit(1);
  });
  process.on('uncaughtException', (err) => {
    logger.error('[Critical] Uncaught Exception thrown', err);
    process.exit(1);
  });
     app.listen(Config.PORT, () => logger.info(`Server started at ${Config.PORT}
port`));
}
startServer();
module.exports = app;
```

**Файл mongodb.connector.js**

```
const { MongoClient } = require('mongodb');
const logger = require('../helpers/logger');
const CONFIG = require('../config');
class MongoDBConnector {
  constructor() {
    this._mongoClient = new MongoClient(CONFIG.MONGO_DB_URL);
  }
  async initConnectionToDB() {
    try {
      await this._mongoClient.connect();
      /**
       * Проксуємо виклики функцій цього класу
       * щоб централізувати обробку та логування помилок
       * які виникли при проботі з БД
       */
      this.db = new Proxy(this._mongoClient.db(), {
        get: (service, property) => {
          if (typeof service[property] === 'function') {
            return new Proxy(service[property], {
              apply(target, _this, args) {
                try {
                  return Reflect.apply(target, service, args);
                } catch (error) {
                  logger.error(`[${MongoDBConnector.name}] Throw error when try to use
${property}`, error);
                  throw error;
                }
              },
```

```
          });
        }
        return service[property];
      },
    });
    logger.info(`Successfully connected to MongoDB. Database name: ${this.db.databaseName}`);
  } catch (e) {
    logger.error('Catch error when try to connect to MongoDB!', e);
    throw e;
  }
 }
}
module.exports = new MongoDBConnector();
```

**Файл bot.js**

```
const moment = require('moment');
const { get } = require('lodash');
const texts = require('../const/text');
const logger = require('../helpers/logger');
const sendMessage = require('../helpers/send-message');
const getIntents = require('../services/dialogflow/get-intents');
const { getIntentInfoObject } = require('../helpers/form-object');
const {
 getUser, getServiceProviderByPhoneNumber, updateUser, getOrderById,
} = require('../helpers/mongodb.service');
const {
 show,
 greeting,
 complaint,
```

```
  editOrder,
  cancelOrder,
  completeOrder,
  cleaningService,
  decisionCustomer,
} = require('./skills');
module.exports = async (message) => {
  const { phoneNumber, mediaUrl } = message;
  let { userText } = message;
  let user;
  let contextName;
  let serviceProvider;
  let fallbacks;
  let access = true;
  let clearContext = false;
  try {
    user = await getUser(phoneNumber);
  } catch (err) {
    logger.error('DATABASE CLIENT ERROR', err);
  }
  try {
    serviceProvider = await getServiceProviderByPhoneNumber(phoneNumber);
  } catch (err) {
    logger.error('DATABASE CLIENT ERROR', err);
  }
  if (!user && !serviceProvider.length) {
    try {
        return sendMessage({ text: `${texts.USER_NOT_FOUND}\n\n*Your chatId:
${phoneNumber}*` }, phoneNumber);
```

```
    } catch (err) {
      logger.error("CAN'T SEND ANSWER TO USER", err);
    }
  }
  if (!user && serviceProvider.length) {
    access = false;
    user = {
      sessionId: serviceProvider[0].sessionId,
    };
  }
  if (access && !userText && mediaUrl && user.isWaitingForGroceryList) {
    userText = 'no text list';
  }
  if (access && user.ordersWaitingForDecision.length) {
    contextName = 'user-customer-decision';
    clearContext = true;
  } else if (access && user.providersWaitingForDecision
    && user.providersWaitingForDecision.length) {
    contextName = 'user-provider-decision';
    clearContext = true;
  }
  if (userText.toLowerCase() === 'restart') {
    clearContext = true;
  }
  let intent = await getIntents(userText, user.sessionId, contextName, clearContext);
    if (user && get(intent, 'queryResult.intent.displayName') === 'Default Fallback
Intent') {
    const lastMessageTime = moment(user.lastMessageTime);
    userText = lastMessageTime.diff(Date.now(), 'minutes') < -30 ? 'hi' : userText;
```

```
    intent = await getIntents(userText, user.sessionId, contextName, clearContext);
  }
  const neededIntentInfo = getIntentInfoObject(intent);
  let { botAnswer, serviceName } = neededIntentInfo;
  const { intentName } = neededIntentInfo;
  if (intentName === 'Default Fallback Intent') {
    if (user.fallbacks === 3) {
      botAnswer = `${botAnswer} Say 'HELP ME' if you need some guidance.`;
      fallbacks = 1;
    } else {
      fallbacks = user.fallbacks + 1 || 2;
    }
  }
  if (access && user) {
      await updateUser(user.phoneNumber, { lastMessageTime: Date.now(), fallbacks:
fallbacks || 2 });
    if (!user.providersWaitingForDecision) {
      await updateUser(user.phoneNumber, { providersWaitingForDecision: [] });
    }
  }
  if (access) {
                if (user.ordersWaitingForDecision.length && !(intentName ===
'decision-service-customer - yes' || intentName === 'decision-service-customer - no'))
{
      botAnswer = 'Please accept or reject renter offer first.';
      serviceName = '';
                        } else if (user.providersWaitingForDecision &&
user.providersWaitingForDecision.length && !(intentName ===
'decision-service-provider - yes' || intentName === 'decision-service-provider - no')) {
```

```
    const order = await getOrderById(user.providersWaitingForDecision[0]);
    botAnswer = `Please accept or reject ${order.serviceType} offer.`;
    serviceName = '';
  }
}
if (access) {
  switch (serviceName) {
    case 'complete':
      return completeOrder(
        {
          ...neededIntentInfo, phoneNumber, serviceProvider, sessionId: user.sessionId,
        },
      );
    case 'decision':
      return decisionCustomer({ ...neededIntentInfo, phoneNumber, user });
    case 'greeting':
      return greeting(phoneNumber, user.firstName, botAnswer);
    case 'complaint':
      return complaint({
        ...neededIntentInfo, phoneNumber, user, userText, botAnswer,
      });
    case 'edit':
      return editOrder({
        ...neededIntentInfo,
        phoneNumber,
        sessionId: user.sessionId,
        user,
      });
    case 'show':
```

```
      return show(user, phoneNumber, botAnswer);
    case 'cancel':
      return cancelOrder({ ...neededIntentInfo, phoneNumber, user });
    case 'cleaning':
      return cleaningService({ ...neededIntentInfo, phoneNumber, user });
    default:
      try {
        return sendMessage({ text: botAnswer }, phoneNumber);
      } catch (err) {
        logger.error("CAN'T SEND ANSWER TO USER", err);
      }
   }
  }
};
```

**Файл renting-service.js**

```
const logger = require('../../helpers/logger');
const sendMessage = require('../../helpers/send-message');
const createRentingOrder = require('../../helpers/create-orders/create-renting-order');
const deleteUnnecessaryContexts =
require('../../services/dialogflow/delete-unnecessary-contexts');
const {
  rentingValidationDate,
  rentingValidationTime,
  rentingValidationPrice,
  rentingValidationReview,
  rentingMinDays,
} = require('../../helpers/validation');
module.exports = async (intentInfo) => {
  const {
    contexts, serviceName, intentName, user, phoneNumber, botAnswer,
  } = intentInfo;
  let formatedText;
  const { sessionId } = user;
```

```
  const [serviceContext] = contexts.filter(context =>
context.name.split('/').reverse()[0] === serviceName);
  await deleteUnnecessaryContexts(sessionId, contexts, 'renting');
  console.log('RENTInt SErVICE: ', intentName);
  if (serviceContext) {
    switch (intentName) {
      case 'renting-service-date':
        formatedText = await rentingValidationDate(serviceContext, user, botAnswer);
        break;
      case 'renting-service-time':
        formatedText = await rentingValidationTime(serviceContext, user, botAnswer);
        break;
      case 'renting-service-price':
        await rentingValidationPrice(serviceContext, user);
        break;
      case 'renting-service-review - no':
        await rentingValidationReview(user, 'no');
        break;
      case 'renting-service-review - yes':
        await rentingValidationReview(user, 'yes');
        break;
      case 'renting-service-confirmation - yes':
        return createRentingOrder(serviceContext, user, botAnswer);
      case 'renting-service-min-days':
        formatedText = await rentingMinDays(serviceContext, user, botAnswer);
        break;
      default:
    }
  }
  try {
    return sendMessage({ text: formatedText || botAnswer }, phoneNumber);
  } catch (err) {
    logger.error("CAN'T SEND ANSWER TO USER ", err);
  }
};
```

**Файл renting-validation-date.js**

```
const moment = require('moment');
```

```
const { struct } = require('pb-util');
const format = require('string-template');
const { FORMAT } = require('../../../const');
const updateContext = require('../../../services/dialogflow/update-context');
module.exports = async (context, user, botAnswer) => {
  const {
    'start-date': rawStartDate, 'end-date': rawEndDate,
  } = struct.decode(context.parameters);
  let { reviewed } = struct.decode(context.parameters);
  const endDate = moment(rawEndDate).format(FORMAT.date);
  const startDate = moment(rawStartDate).format(FORMAT.date);

  if (!reviewed) {
    reviewed = user.isReviewed;
  }
  let reviewedText;
  if (reviewed === 'yes') {
    reviewedText = 'would like to review and approve tenants';
  } else {
    reviewedText = 'would not like to review and approve tenants';
  }
  const formatedText = format(botAnswer, {
    startDate,
    endDate,
    isReviewed: reviewedText,
    price: user.apartmentRentingPricePreference,
  });
  Object.assign(context.parameters.fields, struct.encode({
    'start-date': startDate,
    'end-date': endDate,
    reviewed,
  }).fields);
  await updateContext(context, user.sessionId);
  return formatedText;
};
```

**Файл renting-validation-min-days.js**

```
const { struct } = require('pb-util');
```

```
const moment = require('moment');
const format = require('string-template');
const updateContext = require('../../../services/dialogflow/update-context');
const { FORMAT } = require('../../../const');
module.exports = async (context, user, botAnswer) => {
  const {
    'start-date': rawStartDate, 'end-date': rawEndDate,
  } = struct.decode(context.parameters);
  let { reviewed, 'min-days-rent': minRentDays } = struct.decode(context.parameters);
  minRentDays = Math.abs(minRentDays).toFixed();
  const endDate = moment(new Date(rawEndDate)).format(FORMAT.date);
  const startDate = moment(new Date(rawStartDate)).format(FORMAT.date);
  if (!reviewed) {
    reviewed = user.isReviewed;
  }
  let reviewedText;
  if (reviewed === 'yes') {
    reviewedText = 'would like to review and approve tenants';
  } else {
    reviewedText = 'would not like to review and approve tenants';
  }
  const formatedText = format(botAnswer, {
    startDate,
    endDate,
    isReviewed: reviewedText,
    minRentDays,
  });
  Object.assign(context.parameters.fields, struct.encode({
    'start-date': startDate,
    'end-date': endDate,
    reviewed,
    'min-days-rent': minRentDays,
  }).fields);
  await updateContext(context, user.sessionId);
  return formatedText;
};
```

**Файл create-renting-order.js**

```
const { get } = require('lodash');
const shortid = require('shortid');
const { struct } = require('pb-util');
const format = require('string-template');
const logger = require('../logger');
const sendMessage = require('../send-message');
const { formOrderObject } = require('../form-object');
const { sendMail } = require('../../services/send-grid');
const { createNewOrder, getApartmentBuildingInfo } = require('../mongodb.service');
const updateContext = require('../../services/dialogflow/update-context');
module.exports = async (context, user, botAnswer) => {
  const {
    reviewed, 'start-date': startDate, 'end-date': endDate, 'min-days-rent':
minDaysForRent,
  } = struct.decode(context.parameters);
  const [apartmentBuildingInfo] = await
getApartmentBuildingInfo(user.apartmentBuilding);
  const { checkIn, checkOut } = apartmentBuildingInfo;
  let orderId = get(context, 'parameters.fields.orderId.stringValue');
  let requestType;
  if (!orderId) {
    orderId = shortid.generate();
    requestType = 'Order';
  } else {
    requestType = 'Change';
  }
  const formatedText = format(botAnswer, {
    orderId,
    checkIn,
    checkOut,
    firstName: user.firstName,
  });
  try {
    await sendMessage({ text: formatedText || botAnswer }, user.userId);
  } catch (err) {
    logger.error("CAN'T SEND ANSWER TO USER ", err);
```

```
  }
  const date = `${startDate} - ${endDate}`;
  const orderObject = formOrderObject('renting', {
    user,
    date,
    checkIn,
    orderId,
    checkOut,
    isReviewed: reviewed,
    apartmentSize: user.apartmentSize,
    minDaysForRent,
  });
  await createNewOrder(orderObject.orderId, orderObject);
  try {
    await sendMail(requestType, user, orderObject);
  } catch (err) {
    logger.error("CAN'T SEND MAIL TO USER", err);
  }
  try {
    await sendMail(requestType, user, orderObject, 'manager');
  } catch (err) {
    logger.error("CAN'T SEND MAIL TO MANAGER ", err);
  }
  context.parameters.fields = {};
  await updateContext(context, user.sessionId);
};
```

**Файл middlewares/authorization.js**

```
const config = require('../config');

const authorization = async (req, res, next) => {
  const authorizationHeader = req.headers.token;

  if (authorizationHeader && authorizationHeader === config.AUTH_TOKEN) {
    return next();
  }
  return res.status(401).send('Invalid token or token doesn\'t exist');
};
```

```
module.exports = { authorization };
```

**Файл controllers/orders/send-customer-info.js**

```
const uuid = require('uuid');
const logger = require('../../helpers/logger');
const sendMessage = require('../../helpers/send-message');
const {
  getUser,
  getOrder,
  updateUser,
  updateUserOrder,
  createOrUpdateNewCustomer,
} = require('../../helpers/mongodb.service');

module.exports = async (req, res) => {
  const { orderId } = req.params;
  const {
    email,
    price,
    rating,
    lastName,
    orderInfo,
    firstName,
    profileLink,
    phoneNumber,
    customDate,
  } = req.body;
  const dateCurrentOrder = customDate || orderInfo.date;
  let { customerId } = req.body;
  let isNew = false;
  if (!customerId) {
    isNew = true;
    customerId = uuid.v4();
  }
  await createOrUpdateNewCustomer(customerId, isNew, {
    email,
```

```
    price,
    rating,
    lastName,
    firstName,
    profileLink,
    phoneNumber,
  });
  const [order] = await getOrder(orderId, 'OPEN');
  let { customers } = order;
  let text;

  if (!customers || !customers.length) {
    customers = [];
    if (orderInfo.serviceType === 'renting') {
      if (orderInfo.isReviewed === 'no') {
        text = `Congratulation! We found a renter for your apartment. We will take care
of everything. The money will be deposited into your account after ${firstName}'s
stay from ${dateCurrentOrder}`;
        await updateUserOrder(orderId, {
          orderStatus: 'APPROVED',
          customers: [{
            customerId, firstName, lastName, rating, price, customDate,
          }],
          date: dateCurrentOrder,
          price: req.body.price,
        });
      } else {
        text = `Great news, I found a potential renter for your apartment!\nHere is
${firstName} ${lastName}.\n${firstName} has a rating of ${rating}
stars.\n${firstName} wants to stay in your apartment for a price of $${price} per
night from ${dateCurrentOrder.split('-')[0]} to ${dateCurrentOrder.split('-')[1]}.\nDo
you accept ${firstName}'s offer?`;
      }
    }
  }
  const customerWasAddedBefore = !!customers.find(customer =>
customer.customerId === customerId);
```

```
  if (!customerWasAddedBefore) {
    await updateUserOrder(orderId,
      {
        customers: [...customers, {
          customerId, firstName, lastName, rating, price, customDate,
        }],
      });
  }

  const user = await getUser(orderInfo.userId);

  const { ordersWaitingForDecision } = user;

  if (ordersWaitingForDecision.length) {
    const orderWasAddedBefore = !!ordersWaitingForDecision.includes(orderId);
    if (!orderWasAddedBefore && orderInfo.isReviewed !== 'no' &&
orderInfo.serviceType !== 'parking') {
      try {
        await updateUser(user.userId,
          { ordersWaitingForDecision: [...ordersWaitingForDecision, orderId] });
      } catch (error) {
        logger.info('', error); // to do
      }
    }
  } else {
    try {
      if (text) {
        await sendMessage({ text }, user.userId);
      }

      if (orderInfo.serviceType === 'parking') {
        const messageToRenter = `Congratulations, parking space
${order.additionalParams.parkingSpaceNumber} is yours from
${dateCurrentOrder}.`;
        await sendMessage({ text: messageToRenter }, phoneNumber);
      }
```

```
      if (orderInfo.isReviewed !== 'no' && orderInfo.serviceType !== 'parking') {
        await updateUser(orderInfo.userId,
          { ordersWaitingForDecision: [orderId] });
      }
    } catch (error) {
      logger.info('', error);
    }
  }

  res.end('Information sended');
};
```

**Файл mongodb.service.js**

```
const mongodb = require('../db/mongodb-connector');
const TableNames = {
  USERS: 'users',
  ORDERS: 'orders',
  CUSTOMERS: 'customers',
};

/**
 * Users
 */
const createNewUser = async (userId, data) =>
mongodb.db.collection(TableNames.USERS).insertOne({
  ...data,
  userId: String(userId),
});

const updateUser = async (userId, data) =>
mongodb.db.collection(TableNames.USERS).updateOne(
  { userId: String(userId) },
  {
    $set: data,
  },
);
```

```
const getUser = async userId =>
mongodb.db.collection(TableNames.USERS).findOne({ userId: String(userId) });

const getAllUsers = async () =>
mongodb.db.collection(TableNames.USERS).find().toArray();

const getAllUsersByBuilding = async building =>
mongodb.db.collection(TableNames.USERS).find({
  apartmentBuilding: building,
}).toArray();

/**
 * Orders
 */

const createNewOrder = async (orderId, orderObject) => {
  return mongodb.db.collection(TableNames.ORDERS).insertOne(
    {
      orderId: String(orderId),
      ...orderObject,
      notificationSended: false,
      orderReminderSent: false,
      orderReminderCount: 0,
    },
  );
};

const getOrder = async (orderId, orderStatus) =>
mongodb.db.collection(TableNames.ORDERS).find(
  {
    orderId: String(orderId),
    orderStatus,
  },
).toArray();

const getOrderById = async orderId =>
mongodb.db.collection(TableNames.ORDERS).findOne(orderId);
```

```
const getOrders = async (serviceType, orderStatus = 'OPEN') =>
mongodb.db.collection(TableNames.ORDERS).find(
  {
    serviceType,
    orderStatus,
  },
).toArray();

const getOrdersForNotification = async () =>
mongodb.db.collection(TableNames.ORDERS).find({
  orderStatus: 'APPROVED',
  orderReminderSent: false,
}).toArray();

const getAllOrders = async orderStatus =>
mongodb.db.collection(TableNames.ORDERS).find({
  orderStatus,
}).toArray();

const getOpenedUserOrders = async (userId, serviceType, orderStatus = 'OPEN') =>
mongodb.db.collection(TableNames.ORDERS).find(
  {
    userId: String(userId),
    serviceType,
    orderStatus,
  },
).toArray();
const getAllOpenedUserOrders = async userId =>
mongodb.db.collection(TableNames.ORDERS).find(
  {
    userId: String(userId),
    orderStatus: 'OPEN',
  },
).toArray();
const getAllApprovedOrders = async userId =>
mongodb.db.collection(TableNames.ORDERS).find({
```

```
  userId: String(userId),
  orderStatus: 'APPROVED',
}).toArray();
const getApprovedAndRejectedOrders = () =>
mongodb.db.collection(TableNames.ORDERS).find(
  {
    orderStatus: {
      $nin: [
        'OPEN',
        'DELIVERED',
        'CANCELED',
        'NOT_PAID',
        'COMPLETED',
      ],
    },
    notificationSended: false,
  },
).toArray();
const getApprovedAndOpenedOrders = userId =>
mongodb.db.collection(TableNames.ORDERS).find(
  {
    userId: String(userId),
    orderStatus: {
      $nin: [
        'REJECTED',
        'DELIVERED',
        'COMPLETED',
        'NOT_PAID',
        'CANCELED',
        'CounterOffer',
      ],
    },
  },
).toArray();
const updateUserOrder = async (orderId, data) =>
mongodb.db.collection(TableNames.ORDERS).updateOne(
  { orderId },
```

```
  {
    $set: data,
  },
);
/**
 * Customers
 */
const getAllCustomers = async () =>
mongodb.db.collection(TableNames.CUSTOMERS).find().toArray();
const createOrUpdateNewCustomer = async (customerId, isNew, data) => {
  if (isNew) {
    return mongodb.db.collection(TableNames.CUSTOMERS).insertOne({
customerId, ...data });
  }
  return mongodb.db.collection(TableNames.CUSTOMERS).updateOne({ customerId
}, {
    $set: data,
  });
};
const getCustomer = async customerId =>
mongodb.db.collection(TableNames.CUSTOMERS).findOne({ customerId });
const getCustomerInfo = phoneNumber =>
mongodb.db.collection(TableNames.CUSTOMERS).find(
  {
    phoneNumber,
  },
).toArray();
/** Appartment Buildings */
const getApartmentBuildingsList = async () => {
  return
mongodb.db.collection(TableNames.APARTMENT_BUILDINGS).find().toArray();
};
const getApartmentBuildingInfo = async (apartmentBuilding) => {
  return mongodb.db.collection(TableNames.APARTMENT_BUILDINGS).find({
    apartmentBuilding,
  }).toArray();
};
```

```
const createApartmentBuilding = async (apartmentBuilding) => {
  return
mongodb.db.collection(TableNames.APARTMENT_BUILDINGS).insertOne({
    apartmentBuilding,
  });
};
const updateApartmentBuildings = async (buildingId, data) => {
  return
mongodb.db.collection(TableNames.APARTMENT_BUILDINGS).updateOne(
    {
      buildingId,
    },
    {
      $set: data,
    },
  );
};
module.exports = {
  getUser,
  updateUser,
  getAllUsers,
  createNewUser,
  getAllUsersByBuilding,
  getCustomer,
  getAllCustomers,
  getCustomerInfo,
  createOrUpdateNewCustomer,
  getOrder,
  getOrders,
  getAllOrders,
  getOrderById,
  createNewOrder,
  updateUserOrder,
  getOpenedUserOrders,
  getAllApprovedOrders,
  getAllOpenedUserOrders,
  getOrdersForNotification,
```

```
  getApprovedAndOpenedOrders,
  getApprovedAndRejectedOrders,
  updateApartmentBuildings,
  getApartmentBuildingInfo,
  getApartmentBuildingsList,
  createApartmentBuilding,
};
```

**Файл logger.js**

```
const Winston = require('winston');
const config = require('../config');

const LoggerLevel = {
  error: 'error',
  warn: 'warn',
  info: 'info',
  verbose: 'verbose',
  debug: 'debug',
  silly: 'silly',
};
const defaultLevel = LoggerLevel[config.LOG_LEVELS] || LoggerLevel.info;
const logger = Winston.createLogger({
  level: defaultLevel,
  transports: [new Winston.transports.Console({
    format: Winston.format.combine(
      Winston.format.timestamp(),
      Winston.format.colorize(),
      Winston.format.simple(),
    ),
  })],
});
Winston.addColors({
  error: 'red',
  warn: 'yellow',
  info: 'cyan',
  debug: 'green',
});
module.exports = logger;
```