Міністерство освіти і науки України НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Бакалаврська програма

Розробка iOS-додатку для пошуку та бронювання готельних номерів

Керівник курсової роботи

Горборуков В. В.

_____

(Підпис)

“___” _____ 2025 року

Виконав студент БП ІПЗ-3

Баранівський Я. Д.

“___” _____ 2025 року

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ

УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

доц., канд. наук

О.П. Жежерун

(підпис)

“____”_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Баранівському Ярославу Дмитровичу
факультету інформатики 3 курсу бакалаврської програми
ТЕМА: Розробка iOS-додатку для пошуку та
бронювання готельних номерів

Вихідні дані:

Зміст ТЧ до курсової роботи

Анотація

Вступ

1 Аналіз вимог та формулювання технічного завдання

2 Вибір інструментів для розробки

3 Проектування додатку

4 Технічна реалізація важливих компонентів

Висновки

Список використаних джерел

Дата видачі „___” _____ 2025 р. Керівник_____
(підпис)
Завдання отримав _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КУРСОВОЇ РОБОТИ
Тема: Розробка iOS-додатку для пошуку та бронювання готельних номерів

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	09.10.2024	
2.	Аналіз вимог та формулювання технічного завдання для додатку.	11.10.2024	
3.	Створення прототипу додатку.	20.11.2024	
4.	Впровадження автентифікації користувачів за допомогою Firebase.	28.11.2024	
5.	Розробка API з використанням фреймворка Django.	15.01.2025	
6.	Створення та налаштування бази даних користувачів у Firebase.	19.01.2025	

7.	Тестування інтеграції з базою даних та усунення виявлених недоліків.	30.01.2025	
8.	Реалізація основного функціоналу пов’язаної з готелями.	27.02.2025	
9.	Реалізація взаємодії користувача з додатком.	02.03.2025	
10.	Створення графічного інтерфейсу додатку.	15.03.2025	
11.	Розгортання серверної частини API на платформі Render.	20.03.2025	
12.	Тестування додатку та виявлення недоліків.	25.03.2025	
13.	Написання текстової частини в рамках курсової роботи.	28.03.2025	
14.	Аналіз отриманих результатів з керівником.	20.04.2025	
15.	Створення презентації та	29.04.2025	

	доповіді до захисту роботи.		
16.	Захист курсової роботи.	12.05.2025.	

Студент Баранівський Я.Д
Керівник Горборуков В. В.

“_____”_____

ЗМІСТ

АНОТАЦІЯ **7**
ВСТУП **7**
РОЗДІЛ 1. АНАЛІТИЧНА ЧАСТИНА **8**
1.1 Аналіз ринку туризму в Україні 8
1.2 Дослідження потреб іноземних туристів 9
1.3 Аналіз існуючих рішень 9
1.4 Визначення функціональних вимог до додатку 10
РОЗДІЛ 2. ІНСТРУМЕНТИ РОЗРОБКИ **12**
2.1 Обгрунтування вибору мобільної платформи iOS 12
2.2 Середовище розробки Xcode і мова програмування Swift 13
2.3 Використання Firebase для забезпечення автентифікації та клієнтської бази даних 14
2.4 Розробка серверної частини за допомогою Django 17
2.5 Платформа Render для розгортання серверної частини додатку 19
РОЗДІЛ 3. АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ДОДАТКУ **20**
3.1 Загальна архітектура додатку 20
3.2 Проектування бази даних 22
3.3 Проектування інтерфейсу користувача 24
3.4 Механізми взаємодії користувача з додатком. 26
РОЗДІЛ 4. ТЕХНІЧНА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ДОДАТКУ **34**
4.1 Робота з Firebase Database і Firebase Authentication 34
4.2 Робота з Django REST Framework 37
4.3 Зв’язок між Django REST Framework та Firebase Database 41
ВИСНОВОК **44**

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46

АНОТАЦІЯ

Ця робота присвячена iOS додатку, що дозволяє іноземним туристам ознайомлюватися з основною інформацією про готелі, переглядати доступні номери та здійснювати їх бронювання. Для цього дослідження були використані такі інструменти, як Xcode, Django REST Framework, Render та Google Firebase. Результатом даного дослідження став функціональний iOS застосунок із зручним інтерфейсом та привабливим дизайном.

ВСТУП

З кожним днем наш телефонний помічник набуває все більше і більше можливостей. Технології зростають щосекундно, а відповідно і потреби людей. Смартфони перетворилися на універсальних помічників, що роблять наше життя набагато комфортнішим. Немає такої людини, яка не має при собі свого маленького помічника. Люди потребують доступу до будь-якої інформації в лічені секунди. У сучасному світі люди прагнуть мати під рукою все необхідне, зокрема й можливість планувати правильно подорожі та відпустки.

Ми можемо помітити, що все більше і більше іноземців приїжджають в Україну за новими враженнями, адже наша країна славиться різноманітною культурою і гостинністю. У післявоєнний період одним із важливих завдань для підтримки економіки є розвиток туризму, адже він сприяє залученню іноземних інвестицій і популяризації української культури.

Іноземці потребують зручних та простих рішень для задоволення власних потреб. Одною з таких потреб є бронювання житла. Часто іноземці стикаються з нестачею достовірної інформації про українські готелі або ж мають труднощі при спробі забронювати готельний номер, а наявні на даний момент сервіси іноді містять застарілу або некоректну інформацію.

У зв’язку з цим було прийнято рішення створити такий додатку, який буде орієнтований саме на іноземних туристів. Основна мета - значно

спростити весь процес пошуку, забезпечити доступ до більш відкритої та актуальної інформації та забезпечити зручний процес бронювання. Для реалізації додатку були використані технології iOS, Firebase, Render та Django, адже саме використання цих інструментів забезпечить створити надійний та якісний сервіс.

Дослідження складається з 4 розділів для повного ознайомлення зі структурою додатку.

Перший розділ містить в собі аналітичну частину, де описано основну концепцію додатку.

У другому розділі описано інструменти додатку, які були використані для реалізації додатку.

У третьому розділі – архітектура та проектування додатку.

У четвертому розділі – технічна реалізація компонентів додатку.

РОЗДІЛ 1
АНАЛІТИЧНА ЧАСТИНА

1.1 Аналіз ринку туризму в Україні

Після декількох важких років, пов’язаних із пандемією та військовими діями, туристична галузь в Україні поступово відновлюється. Після завершення активної фази бойових дій, туристична галузь відновиться у новій, раніше небаченій формі. Відновлення потоку туристів зумовлює зростанню потреб у якісному сервісі, зокрема у сфері готельного обслуговування.

Україна має великий потенціал, адже ця країна унікальна своїм колоритом, багатою культурною спадщиною, історичними пам’ятками, мальовничими краєвидами, гостинністю мешканців. Усе це створює сприятливі передумови для активного залучення іноземних туристів.

За даними Державного агентства розвитку туризму України (ДАРТ), у 2024 році сума туристичного збору зросла майже на 23% у порівнянні з 2023 роком, досягнувши майже 273 млн грн [1]. Ця статистика вказує на зростаючий інтерес до України як до перспективного туристичного напрямку, що підкреслює актуальність потреби у пошуку сучасних рішень, які б забезпечили користувачам легкий доступ до інформації щодо проживання та бронювання.

Попри позитивні зрушення, на ринку все ще існують такі проблеми, як обмежена кількість сучасних сервісів для бронювання, відсутність актуальної інформації про готельні комплекси та складна система бронювання. Тому створення додатку є важливим кроком для покращення туристичного досвіду, особливо для іноземних мандрівників.

1.2 Дослідження потреб іноземних туристів

Іноземні мандрівники мають дещо інші очікування від туристичних сервісі, ніж внутрішні мандрівники нашої країни. Для туристів з-за кордону особливо важливим аспектом є наявність зрозумілої, структурованої та достовірної інформації про готельні комплекси. Враховуючи зазначене, можна сформулювати такі основні вимоги:

- Простота бронювання:
  Туристи очікують максимально спрощеної процедури бронювання, бажано - у декілька кліків, без зайвих контактів із адміністраторами або заповненням складних форм.
- Актуальність даних:
  Інформація про готелі та номери, зокрема ціни, типи номерів, тощо, повинна бути завжди актуальною.

- Інтеграція з навігаційними сервісами:
  Можливість перегляду розташування готелю на інтерактивній мапі є важливою функцією, яка сприяє кращій орієнтації на місцевості.

- Гнучкі фільтри пошуку:
  Користувацький досвід значно покращується завдяки можливості фільтрувати готелі та готельні номери за різними критеріями.

- Система відгуків:
  Наявність відгуків інших мандрівників дозволяє отримати повне уявлення про готелі та розуміння того, що насправді з себе представляє готельний комплекс.

- Доступ до інформації про готельні послуги:
  Туристи очікують чіткого переліку послуг, які готельний комплекс здатний надати.

Отже, розробка цього додатку вимагає врахування потреб міжнародних гостей та має бути орієнтована на зручність і надійність сервісу.

1.3 Аналіз існуючих рішень

У сучасному світі вже існує низка популярних мобільних сервісів для здійснення пошуку та бронювання житла. Серед таких: Booking.com, Airbnb та Hotels.com. Кожен з цих сервісів має значну аудиторію та ефективний

функціонал, однак, як і будь-який інший сервіс, він має свої переваги та недоліки.

- Booking.com:
  Один із наймасштабніших сервісів для бронювання житла. Цей сервіс надає широкий вибір готелів та різних типів житла по всьому світу. Його сильними сторонами є велика база доступних варіантів, гнучка система фільтрів та детальна інформація про житло. Проте, користувачі відзначають такі недоліки, як велика кількість реклами та перевантажений інтерфейс. Також трапляються випадки, коли інформація про українські готелі виявляється застарілою або неповною.

- Airbnb:
  Основний напрям цього сервісу - оренда приватного житла від інших господарів. Airbnb пропонує інтуїтивно зрозумілий та легкий інтерфейс, що приваблює багатьох користувачів. Водночас, відсутність чіткої класифікації за готельними стандартами може створити незручності для мандрівників, які звикли до формату, притаманного традиційним готелям.

- Hotels.com
  Сервіс схожий на Booking.com, однак має свою особливість у вигляді програми лояльності, що дозволяє отримувати знижки на майбутні бронювання. До недоліків цього сервісу можна віднести застарілий інтерфейс як веб-сайту, так і мобільного додатку, що може знизити його привабливість для молодої аудиторії.

Загалом, хоч і вибір додатків для бронювання номерів досить великий, для українського туризму все ще існує дефіцит якісних рішень. Сучасний турист потребує простого і зручного мобільного інструмента, тому рішення має зосереджуватися, перш за все, на актуальність інформації і простому, зрозумілому інтерфейсі, в якому легко орієнтуватися без сторонньої допомоги.

1.4 Визначення функціональних вимог до додатку

Для успішної реалізації iOS додатку з функціоналом пошуку, перегляду готелів та бронювання номерів, необхідно чітко врахувати всі ключові вимоги до функціональної частини. Основна мета додатку – задовільнити всі необхідні потреби туристів, забезпечивши інтуїтивність інтерфейсу і надійність у використанні.

Ключові функції додатку включають:

- Реєстрація та автентифікація користувача:
  Для доступу до повного функціоналу додатку передбачається наявність авторизації через Google акаунт. Гарантує безпеку і персоналізований досвід для кожного користувача.

- Перегляд основної інформації про готелі
  Головний екран додатку повинен містити основний список доступних готелів із базовою інформацією – назвою, фотографією, рейтингом і місцезнаходженням.

- Розгорнута інформацію про готель:
  При переході до детального перегляду користувач має бачити повну інформацію про готель, зокрема список доступних номерів, рецензії, опис і перелік наданих послуг.

- Відгуки та оцінювання:
  Користувачі додатку повинні мати змогу оцінити готель та поділитися власними враженнями про готель, у якому вони прожили.

- Функціонал пошуку та фільтрація готелів:
  Можливість здійснювати пошук готелів за назвою або містом та фільтрації за містом, середньою ціною і рейтингом для зручного підбору потрібних варіантів.

- Пошук та фільтрація готельних номерів:
  Має бути реалізований пошук за типом або номером кімнати. Для зручності забезпечити фільтрацію за кількість проживаючих, ціною за одну ніч та доступними сервісами.

- Бронювання номеру:
  Користувач має швидко забронювати обраний номер прямо в додатку. Після успішної резервації, на електронну пошту користувача автоматично надсилається лист підтвердження.

- Список вподобаних готелів та номерів:
  Додаток має надавати можливість зберігати вподобані готелі та номери в окремі персональні списки для швидкого доступу до них у майбутньому.

- Інтеграція з мапою:
  Карта з можливістю перегляду готелів допоможе туристам краще орієнтуватися на місцевості.

Чітке визначення функціональних вимог є ключем успіху до успішної розробки мобільного додатку. Реалізація вищезазначених вимог забезпечить повний цикл взаємодії з сервісом – від пошуку до бронювання.

РОЗДІЛ 2
ІНСТРУМЕНТИ РОЗРОБКИ

2.1 Обгрунтування вибору мобільної платформи iOS

При розробці мобільного застосунку важливим кроком є вибір операційної системи. Від цього залежить подальше існування додатку у майбутньому, потенційне охоплення аудиторії та стабільність додатку. Для реалізації додатку з пошуку готелів та бронювання готельних номерів обрана операційна система iOS.

Згідно зі статистикою StarCounter, частка iOS на канадському ринку мобільних операційних систем становить 59,16% [2], а за даними Consumer Intelligence Research Partners (CIRP), у США частка користувачів iOS у 2024 році зросла до 39%, порівняно з 37% у 2022 році [3]. Ці дані свідчать про те, що iPhone залишається основним мобільним пристроєм для іноземних туристів, особливо з таких розвинених країн, як Канада та США. Тому орієнтація на користувачів iOS є вдалим стратегічним рішенням для залучення цільової аудиторії.

Крім того, дослідження Horace Dediu демонструє, що користувачі iOS витрачають у середньому $10,40, тоді як користувачі операційної системи Android витрачають $1,40. Тобто середній рівень витрат у користувачів iOS на додатки та пов’язані з ними сервіси у 7,4 рази вищий, ніж у користувачів Android [4]. Цей фактор є вагомою перевагою, оскільки з’являються певні перспективи для реалізації додаткових платних функцій у майбутньому, які зможуть знайти своїх клієнтів.

З технічної точки зору, розробка додатків для iOS є технічно простішою. На відміну від операційної системи Android, де існує широкий спектр пристроїв з різними характеристиками, iOS працює в рамках однієї екосистеми Apple, що спрощує процес тестування та підтримки додатку.

Серцем розробки додатку є мова програмування Swift. Ця мова програмування відзначається високою продуктивність, безпечним синтаксисом і сучасним підходом до розробки. Swift надає можливість створювати додатки з легким для читання кодом та використовувати різні потужні інструменти для якісної розробки.

Отже, вибір операційної системи iOS є цілком обґрунтованим. iOS забезпечує високу якість користувацького досвіду, платоспроможну аудиторію, простоту розробки й обслуговування, що є важливим аспектом для підтримки і розвитку додатку у майбутньому.

2.2 Середовище розробки Xcode і мова програмування Swift

У процесі реалізації даного додатку було прийнято рішення використовувати середовище розробки Xcode у поєднанні з мовою програмування Swift, що є офіційним стандартом для розробки iOS додатків.

Xcode – це основний інструмент для створення додатків під iOS, macOS, watchOS та tvOS. Це середовище надає розробникам повний набір функціоналу, який охоплює всі етапи створення додатку – від написання і формування коду до його тестування. Має багато вбудованих інструментів для комфортної розробки додатку, а саме кодовий редактор, інструменти для діагностики роботи коду, емулятор пристроїв тощо.

Наприклад, кодовий редактор підтримує підсвічування синтаксису та миттєву перевірку помилок, що дозволяє швидко їх виправляти. Емулятор пристроїв дозволяє протестувати роботу додатку без необхідності використовувати реальні пристрої.

Слід підкреслити зручну інтеграцію з Git, яка реалізована у вигляді вбудованої підтримки системи контролю версій. Це значно покращує командну розробку, даючи можливість працювати з репозиторіями, створювати гілки та працювати з пул-реквестами безпосередньо з Xcode, без необхідності використання додаткових застосунків. Xcode має потужну вбудовану систему відлагодження, що забезпечує ефективне виявлення помилок і неточностей у коді під час виконання додатку.

Також варто відзначити високий рівень підтримки цього середовища з боку компанії Apple, яка регулярно випускає оновлення. Це забезпечує стабільність, безпеку та комфорт під час розробки додатків.

Якщо говорити про мову програмуванняя, Swift є сучасним рішенням, створеним спеціально для розробки додатків для Apple. Вона поєднує в собі простоту, високу продуктивність безпечність та ефективність розробки.

Swift вирізняється своїм лаконічним та інтуїтивно зрозумілим синтаксисом, що спрощує написання коду. Swift дозволяє писати менше

рядків коду, необхідного для реалізації певного функціоналу, що дозволяє значно пришвидшити розробку.

Мова постійно розвивається і підтримується компанією Apple, тому розробники регулярно отримують оновлення з новими можливостями та виправленням помилок.

Swift спеціально був створений із урахуванням високих вимог до безпеки, тому ця мова стійка до різноманітних помилок, має спеціальні механізми для безпечного управління пам'яттю та дозволяє виявляти помилки вже на етапі компіляції.

Таким чином, поєднання середовища розробки Xcode з мовою програмування Swift гарантує забезпечення усіх необхідних умов для розробки сучасного, функціонального та стабільного застосунку.

2.3 Використання Firebase для забезпечення автентифікації та клієнтської бази даних

У межах цього дослідження для забезпечення роботи додатку було використано таку хмарну платформу як Firebase. Firebase – це середовище від компанії Google, яке пропонує розробникам широкий спектр інструментів, необхідних для створення сучасних додатків різного типу. Список його можливостей безмежний, починаючи від базового зберігання даних і закінчуючи створенням хмарних застосунків без потреби у власних серверах. У нашому додатку дане середовище використовується для автентифікації користувачів і керування базою даних, що містить інформацію про їхні облікові записи.

Firebase Authentication дозволяє швидко інтегрувати автентифікацію без потреби в складній логіці. Цей компонент дозволяє розробникам інтегрувати механізм автентифікації в додаток і реалізувати вхід за допомогою електронної пошти та пароля, або через облікові записи в соціальних мережах.

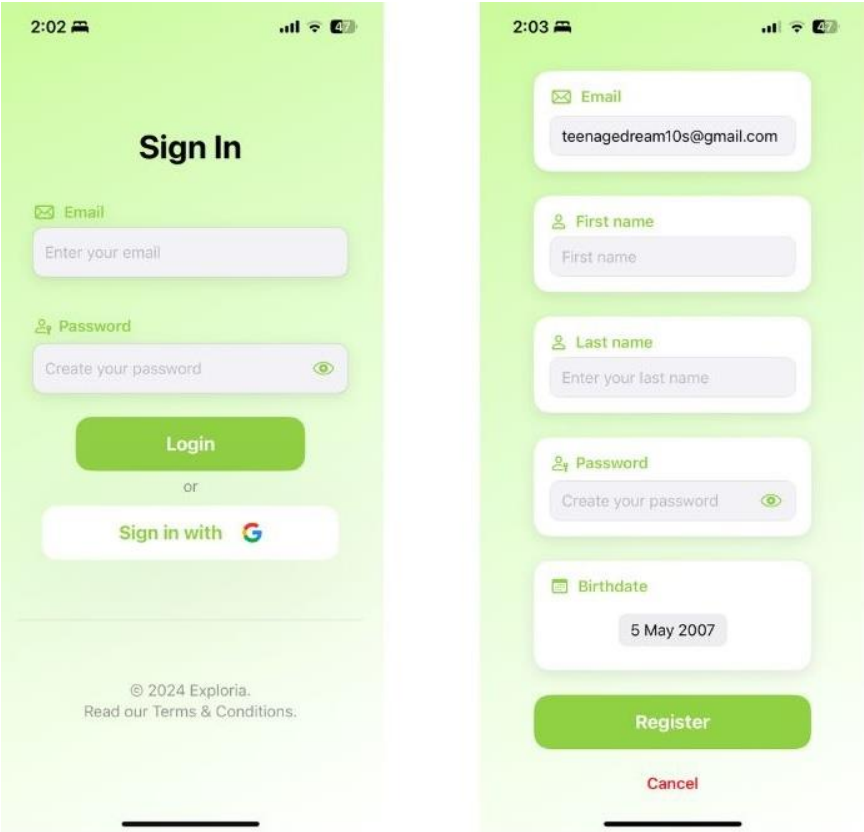

Рисунок 2.1 – Форми для авторизації користувача та реєстрації

Для застосунку було реалізовано вхід через Google-акаунт, а також більш традиційний спосіб – за логіном і паролем. Такий підхід забезпечує зручність використання: користувачі можуть увійти через Google-акаунт, а за відсутності облікового запису – швидко зареєструватися і отримати доступ до основного функціоналу додатку.

Безпека відіграє важливе значення, особливо коли йдеться про зберігання особистих даних. Firebase відповідає сучасним стандартам інформаційної безпеки, тобто всі чутливі дані шифруються та передаються через захищені канали.

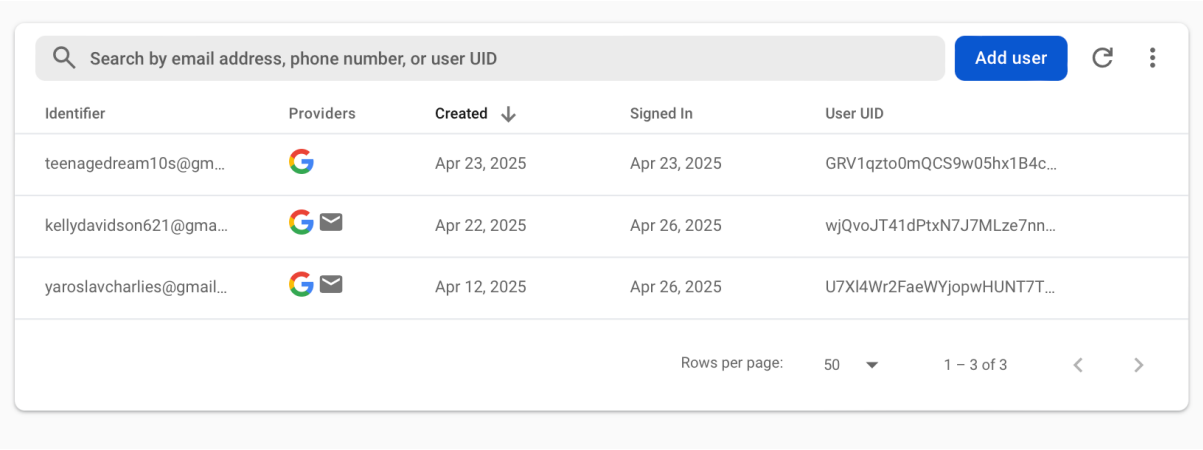

Рисунок 2.2 – Список користувачів

Для зберігання інформації про користувачів було використано Firestore Database. Вибір зберігання через це середовище можна обґрунтувати тим, що Firestore забезпечує автоматичну синхронізацію даних між клієнтом та сервером, що дозволяє переглядати та редагувати інформацію в режимі онлайн. Усі зміни відразу зберігаються в кеші, а при відновленні підключення до інтернету, всі дані автоматично передаються на сервер. Зазначені характеристики гарантують зручність та безперебійну роботу додатку.

Ця хмарна база даних використовується для швидкого доступу до потрібних атрибутів і оптимізованої синхронізації даних. Firestore Database є NoSQL сховищем, що організовує інформацію у вигляді колекцій документів, забезпечуючи гнучку зберігання даних про зареєстрованих користувачів.

У нашому випадку для зберігання інформації про користувачів була створена колекція документів під назвою “users”, де кожен документ містить дані про окремого користувача. Окремий документ містить як базову інформацію, таку як ім’я, пароль, дата народження, так і спеціальні атрибути, пов’язані з активністю користувача в додатку: список резервацій, вподобані готельні комплекси та номери, а також залишені відгуки.

Рисунок 2.3 – Приклад структури даних користувача

Отже, використання Firebase у поєднанні з Firestore забезпечує розробку потужного додатку з надійною автентифікацією та швидкодійною базою даних, що є ключовими чинниками для стабільної та ефективної роботи додатку.

2.4 Розробка серверної частини за допомогою Django

Для правильної реалізації отримання інформації про готелі та їхні номери було вирішено розробити API з використанням фреймворку Django, написаного мовою Python.

Django – це фреймворк для розробки веб-додатків на мові програмування Python. В рамках цього дослідження було використано такий інструмент як Django REST Framework - розширення для створення серверної частини. Рішення використовувати даний інструмент можна обґрунтувати тим, що в Django дуже легко створювати API, а головне дуже швидко. Django REST Framework надає потужні засоби для створення RESTful API, зокрема інструменти для серіалізації даних, контролю доступу та реалізації пагінації.

Основні компоненти для реалізації архітектури серверної частини:

- Моделі:
  Було створено кілька моделей даних, які описують ключові сутності системи. Зокрема, модель Hotel містить базову інформацію про готелі, таку як назва, опис, розташування, рейтинг тощо.

```
6 usages  Yaroslav Baranivskyy
class Hotel(models.Model):
    name = models.CharField(max_length=100)
    location = models.CharField(max_length=200)
    detailed_location = models.CharField(max_length=200, default="")
    rating = models.PositiveSmallIntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(5)],
        default=1
    )
    description = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    image = models.URLField(default="")
    amenities = models.JSONField(default=list, blank=True)

    Yaroslav Baranivskyy
    def __str__(self):
        return self.name
```

Рисунок 2.4 – Створення сутності Hotel

- Серіалізатори:
  Для того, щоб перетворити моделі даних у формати, які будуть придатні для правильної передачі через API, реалізовано серіалізатори. Вони дозволяють визначати, які поля моделі будуть доступні через API.

```
5 usages   Yaroslav Baranivskyy
class HotelSerializer(serializers.ModelSerializer):
    Yaroslav Baranivskyy
    class Meta:
        model = Hotel
        fields = '__all__'
```

Рисунок 2.5 – Створення серіалізатора HotelSerializer

- Контролери запитів
  Обробка запитів здійснюється за допомогою views, які виконують роль контролерів в архітектурі REST API. Вони, у свою чергу, взаємодіють з моделями та серіалізаторами, що забезпечує правильну обробку вхідних запитів і формування відповідей у форматі, придатному для подальшого використання.

```
2 usages   Yaroslav Baranivskyy
class HotelListAPIView(APIView):
    Yaroslav Baranivskyy
    def get(self, request):
        hotels = Hotel.objects.all()
        serializer = HotelSerializer(hotels, many=True)
        return Response(serializer.data)

    Yaroslav Baranivskyy
    def post(self, request):
        serializer = HotelSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Рисунок 2.6 – Створення API Endpoint для роботи з даними про готельні комплекси

- URLs
  Для маршрутизації запитів до views були налаштовані відповідні URLs, що дозволяють коректно передавати запити та отримувати інформацію.

```
urlpatterns = [
    # Готелі
    path('hotels/', HotelListAPIView.as_view(), name='hotel-list'),
    path('hotels/<int:pk>/', HotelDetailsAPIView.as_view(), name='hotel-details'),

    # Окремі номера
    path('rooms/', RoomListAPIView.as_view(), name='room-list'),
    path('rooms/<int:pk>/', RoomDetailsAPIView.as_view(), name='room-details'),

```

Рисунок 2.7 – Створення URLs для правильної маршрутизації

Отже, результатом цього підпункту стала реалізована серверна частина, яка ефективно обробляє інформацію та забезпечує надійну взаємодію між клієнтським додатком і сервером. Система містить усі необхідні компоненти, реалізовані для стабільної роботи з запитами та базою даних.

2.5 Платформа Render для розгортання серверної частини додатку

Для розгортання серверної частини було обрано таку хмарну платформу як Render. Цей сервіс надає інструменти для розгортання та масштабування веб-сервісів, баз даних тощо. Render має зручний інтерфейс та широкий функціонал, що дозволяє розгортати додатки навіть користувачам з мінімальним досвідом у роботі з хмарними технологіями. Render вважається відносно новою хмарною платформою на ринку, проте швидко набуває популярності завдяки простому використанню та активному розвитку. Після ознайомлення з офіційною документацією, робота з Render виявляється не такою складною, як це може здатися на перший погляд.

Render пропонує простий процес розгортання веб-сервісу безпосередньо з GitHub репозиторію, що значно полегшує оновлення проєкту. Будь-які зміни в коді автоматично запускають повторне розгортання сервера без додаткового втручання. Платформа самостійно завантажує код сервісу чи веб-додатку, потім встановлює необхідні залежності і після цих кроків, додаток буде вже успішно запущено. У результаті для розгорнутого сервісу буде надано власний URL адрес, за якою він стає доступним для користувачів.

В контексті цього дослідження було розгорнуте API, в якому зберігається інформацію про готелі та готельні номера, а також реалізовано невеликий сервіс, який надсилає листи-підтвердження про успішне бронювання номеру.

Таким чином, Render є ефективним інструментом для швидкого розгортання та підтримки серверної частини проєкту, надаючи стабільний доступ до функціоналу через автоматично згенеровану URL-адресу.

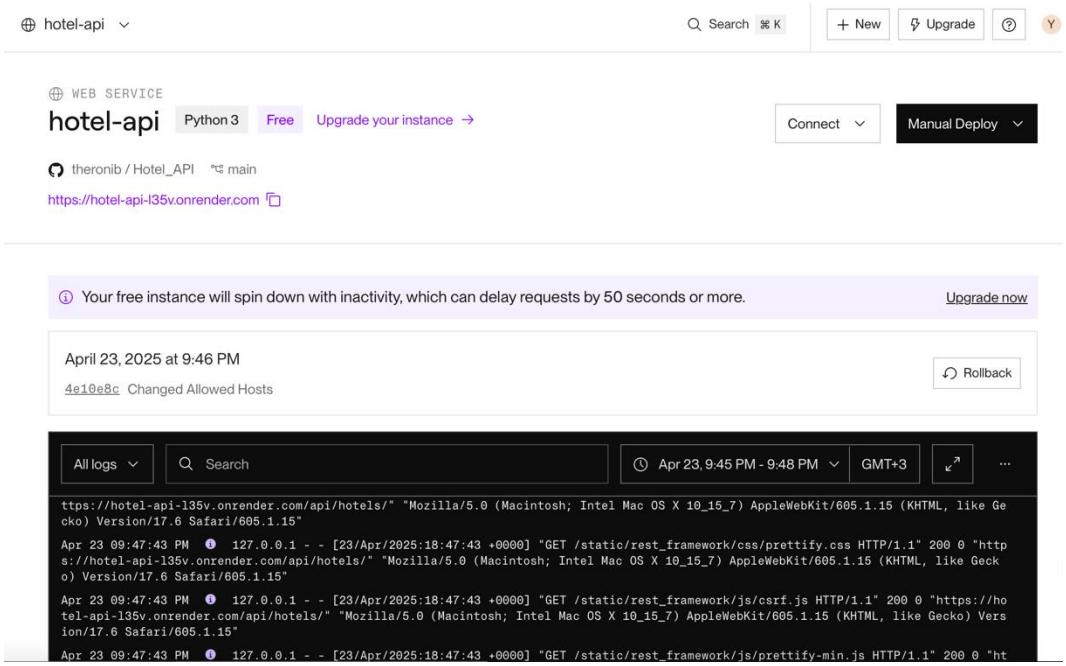

Рисунок 2.8 – Розгортання API

РОЗДІЛ 3
АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ДОДАТКУ

3.1 Загальна архітектура додатку

Для розробки даного додатку було обрано такий архітектурний шаблон проєктування як MVVM. Ця архітектура є однією з найсучасніший на сьогодні і дуже популярна серед iOS розробників, адже завдяки їй код можна легше організувати, підтримувати, а згодом масштабувати додаток у майбутньому.

Основна перевага полягає в тому, що ми можемо правильно розділити відповідальність між різними частинами застосунку. Тобто розробник буде краще орієнтуватися в коді, оскільки логіка обробки даних зберігається окремо від того, як ця інформація подається в інтерфейсі. Це є важливим фактором, бо з часом додаток буде розширюватися, до нього буде додано нові можливості та екрани, і завжди потрібно буде щось змінити або

підправити – саме тоді наша архітектура дозволяє робити це без зайвих складнощів, не зачіпаючи решту структуру програми.

Ще однією перевагою є те, що архітектура MVVM дуже добре поєднується зі SwiftUI, який був використаний в під час розробки цього додатку. SwiftUI побудований на реактивному підході, тож інтерфейс буде автоматично оновлюватися при будь-якій зміні даних у ViewModel. Це значно спрощує роботу розробників, адже зменшує кількість ручної роботи та знижує ризик помилок. Крім того, ViewModel у MVVM є досить універсальною, тому її можна використовувати повторно на декількох екранах, якщо логіка взаємодії з даними є спільною.

Суть цього шаблону полягає в тому, що логіка програми розділяється на три частини. Model – компонент, який відповідає за дані, такі як структури користувача, готелів, готельних номерів тощо. ViewModel – компонент, який забезпечує правильну обробку даних, а також займається обробкою запитів до API та бази даних. View – це інтерфейс користувача. Для створення інтерфейсу було використано SwiftUI, тому екрани працюють охайно та акуратно, адже вони підписані на ViewModel та автоматично реагують на зміну даних через @ObservedObject або @StateObject. View лише відображають готові дані на екрані.

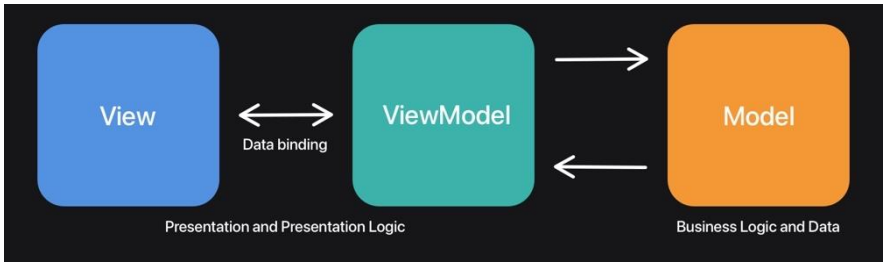

Рисунок 3.1 – Архітектура MVVM [5]

У моєму застосунку ця архітектура нада можливість логічно організувати екрани, які пов’язані з авторизацією, переглядом готельних комплексів, переглядом готельних номерів тощо. Для кожного з цих модулів використовується власна ViewModel, яка відповідає за обробку даних та взаємодію з Firebase. Це допомогло зробити структуру застосунку чистою та охайною.

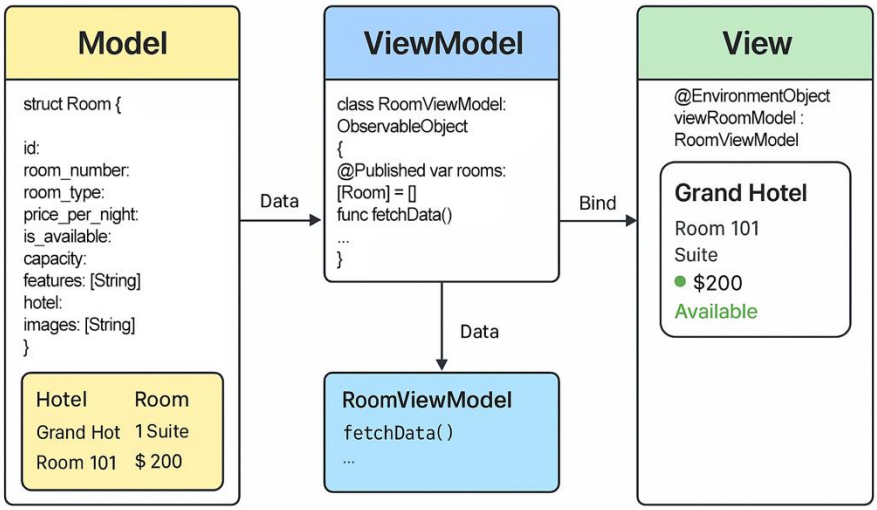

Рисунок 3.2 – Приклад реалізації MVVM у SwiftUI для роботи з даними про готельні номери

Таким чином, можна побачити, що архітектурний шаблон проєктування MVVM значно полегшує розробку додатка. Завдяки чіткому розподілу відповідальності між компонентами, працювати з кодом стає набагато легше. В результаті використання даного шаблону, код програми вийшов структурованим та інтуїтивно зрозумілим.

3.2 Проектування бази даних

У рамках даного проєкту було цікаво трішки поексперементувати з різними підходами до обробки даних, тому було прийнято рішення використовувати комбіновану структуру зберігання даних.

Для зберігання та обробки інформації про зареєстрованих користувачів застосунку використовується Firestore Database, а для зберігання даних про готелі та готельні номери було реалізовано власне RESTful API за допомогою Django REST Framework, де усі дані обробляються безпосередньо всередині API.

У Firestore Database кожен користувач представлений у вигляді окремого об’єкту з такими ключовими полями:

- email,firstname, lastname – основна інформація про користувача;
- password_hashed – захешований пароль;
- birthdate – дата народження у форматі timestamp;

- uid – унікальний ідентифікатор користувача;
- favorites hotels – масив з вподобаними готелями;
- favorite rooms – масив з вподобаними готельними кімнатами;
- reservations – масив бронювань;
- reviews – масив відгуків.

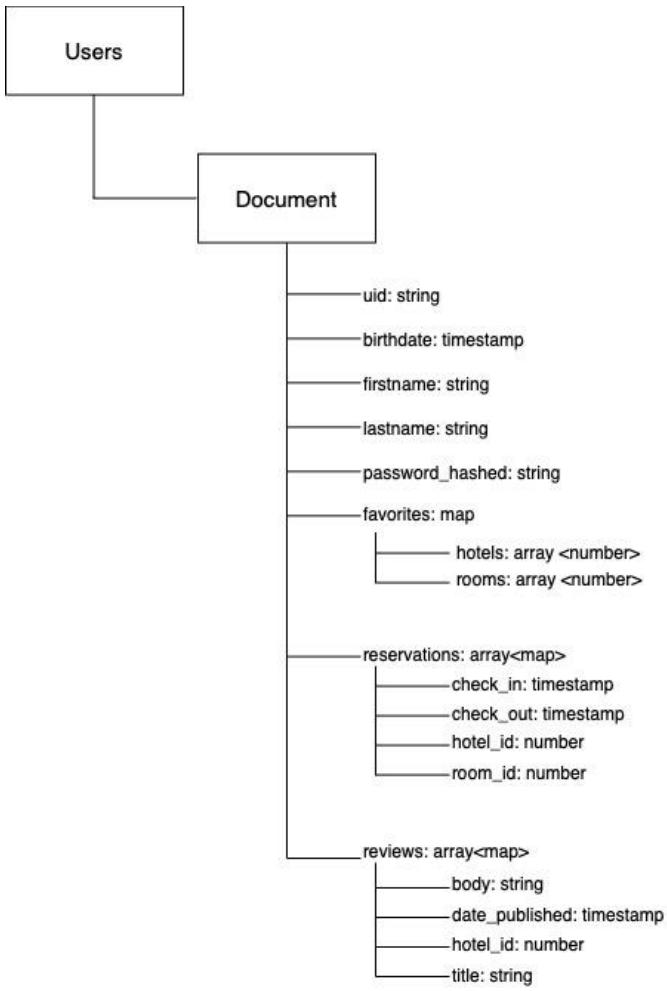

Рисунок 3.3 – Структура документа користувача в Firebase Database

У Django REST Framework кожен готель та готельний номер зберігається у вигляді окремих об’єктів. Для кожного готелю та номера визначено відповідні поля, що описують їхні характеристики. Готель містить такі поля, як: ID готелю, назва готелю, місцезнаходження, рейтинг, опис, зручності та зображення готелю. Готельний номер включає такі поля,

як: номер кімнати, тип номера, ціна за ніч, доступність, місткість, зручності та зображення номера.

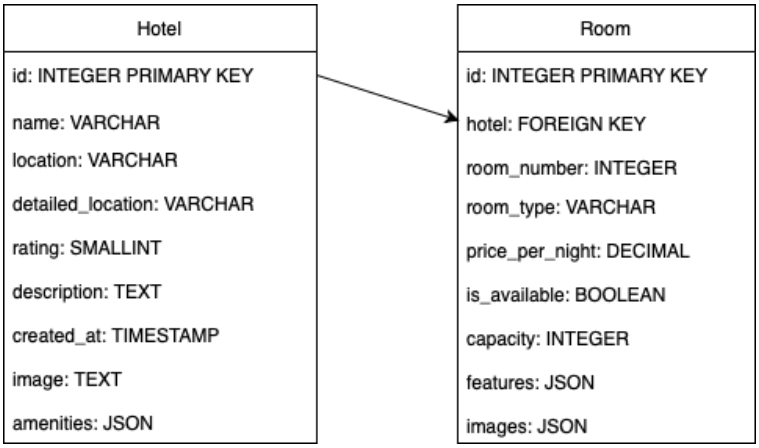

Рисунок 3.4 – Структура зберігання даних для готелів та номерів

Даний підхід до розділення дозволяє ефективно зберігати дані, полегшує їх обробку та управління, забезпечуючи зручність масштабування і підтримки додатку у майбутньому.

3.3 Проектування інтерфейсу користувача

Інтерфейс є одним із найголовніших компонентів додатку, адже саме він виступає своєрідною “обкладинкою” додатку. Саме цей елемент формує перші враження користувача, визначаючи, наскільки зручно, зрозуміло та приємно взаємодіяти з додатком. Основним завданням успішно розробленого інтерфейсу є не лише справити гарні враження на користувача, а й забезпечити доволі зрозумілу навігацію, логічну структуру розділів та зрозумілу подачу інформації. Інтерфейс має бути легким для користування, як для новачків, так і для досвідчених користувачів. Важливо не забувати і про візуальну частину, тобто про добре підібрану кольорову гаму, іконки, стилізацію шрифтів – усе це має велике значення для загального досвіду користування додатком.

Під час розробки даного додатку, значна увага приділялась простоті використання, мінімалізму в дизайні та логічному розташуванню елементів. Інтерфейс був створений з акцентом на те, щоб забезпечити простоту сприйняття, уникнення візуального перевантаження елементами та, насамперед, логічне розташування, залишаючи тільки дійсно необхідні компоненти для найкращого користувацького досвіду.

Запорукою успішного інтерфейсу є його ефективна взаємодія з функціоналом додатку, тобто має бути забезпечення швидкого та зручного доступу до всіх основних функцій – бажано не більше ніж у два кліки.

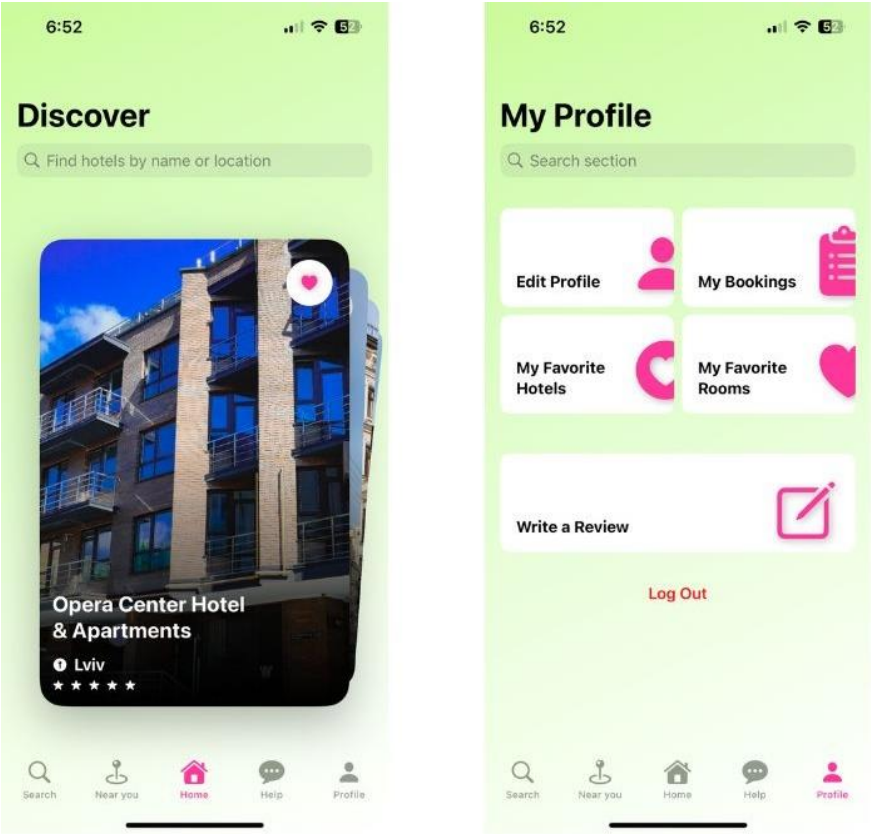

Рисунок 3.5 – Інтерфейс додатку

Усі екрани додатку були виконані в єдиному стилі для того, щоб забезпечити послідовність та зручність у користуванні, адже єдиний зовнішній вигляд інтерфейсу допомагає користувачеві швидше орієнтуватися в структурі та функціональних можливостях додатку.

Кольори також відіграють певну роль в оформленні користувацького інтерфейсу, оскільки вони значною мірою впливають на перше враження користувача та загальне сприйняття додатка в цілому.

Для даного додатку були обрані два основні кольори, а саме яскраво-рожевий (#ff389a) та світло-зелений (#cffca3). Рожевий колір використовується для кнопок, іконок, а також для відображення улюблених готелів, номерів, особистих відгуків і резервацій, тоді як зелений колір здебільшого використовується для загального фону додатку.

Комбінація таких кольорів виражає свіжість, енергійність і динамічність, а також надає відчуття легкості, що є важливим аспектом нашого інтерфейсу, адже ми прагнемо, щоб користувачі почували себе комфортно під час роботи в додатку.

Рисунок 3.6 – Основна палітра кольорів, яка була використана для дизайну інтерфейсу

Як можна побачити, інтерфейс відіграє ключову роль у процесі розробки додатку. Йому слід приділяти велику увагу навіть до найменших деталей, адже саме з них формується цілісне та якісне рішення. Завдяки дотриманню єдиного стилю, доцільному розташуванню елементів і ретельно підібраній кольоровій палітрі вдалося створити інтерфейс, що забезпечить приємний і комфортний користувацький досвід.

3.4 Механізми взаємодії користувача з додатком.

Як було зазначено раніше, основною метою є не лише функціональність додатку, а й простота цього функціоналу у використанні. Тому було прийнято рішення реалізувати такий інтерфейс, де кожна вкладка має своє чітке призначення та доступ до необхідних функцій у два кліки. Це дозволить користувачеві швидко орієнтуватися в додатку та без будь-яких труднощів використати ті чи інші функції відповідно до своїх потреб.

Для навігації в межах додатку було використано такий компонент, як TabView. Це зручний інструмент, що дозволяє створити переміщення між основними вкладками за допомогою нижньої панелі. Основна його ідея полягає в тому, що кожен з цих розділів є незалежним один від одного, тому користувач може швидко й без проблем переключатися між ними.

Мій додаток було поділено рівно на п’ять основних вкладок: Search, Near You, Home, Help, Profile. Кожна з цих вкладок є унікальною і виконує своє певне призначення.

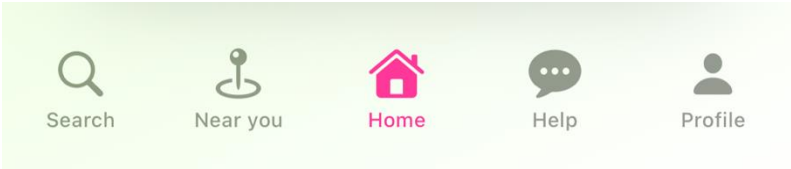

Рисунок 3.7 – Нижня навігаційна панель, реалізована за допомогою TabView

Розпочнемо огляд з найголовнішої вкладки цього додатку – Home. Вкладка Home відповідає за головну сторінку додатку. На цій сторінці можна знайти функціонал у вигляді пошуку готелів за назвою чи містом, а також невеличкі картки, створені за допомогою компонента BigUIPaging. Для реалізації таких карток було використано CardDeckPageViewStyle, що може здатися дуже схожим до системи карточок, подібні до тих, що є в додатку Tinder, де також можна робити свайпи. Втім, в нашому випадку картки використовуються лише для перегляду наявних готелів, що є в базі даних у цей момент. Також у верхньому кутку кожної картки можна помітити іконку сердечка. Натискання на цю іконку дозволяє додати готель до списку улюблених. При переході на сторінку конкретного готелю можна переглянути його основні характеристики, а також ознайомитися з відгуками інших користувачів.

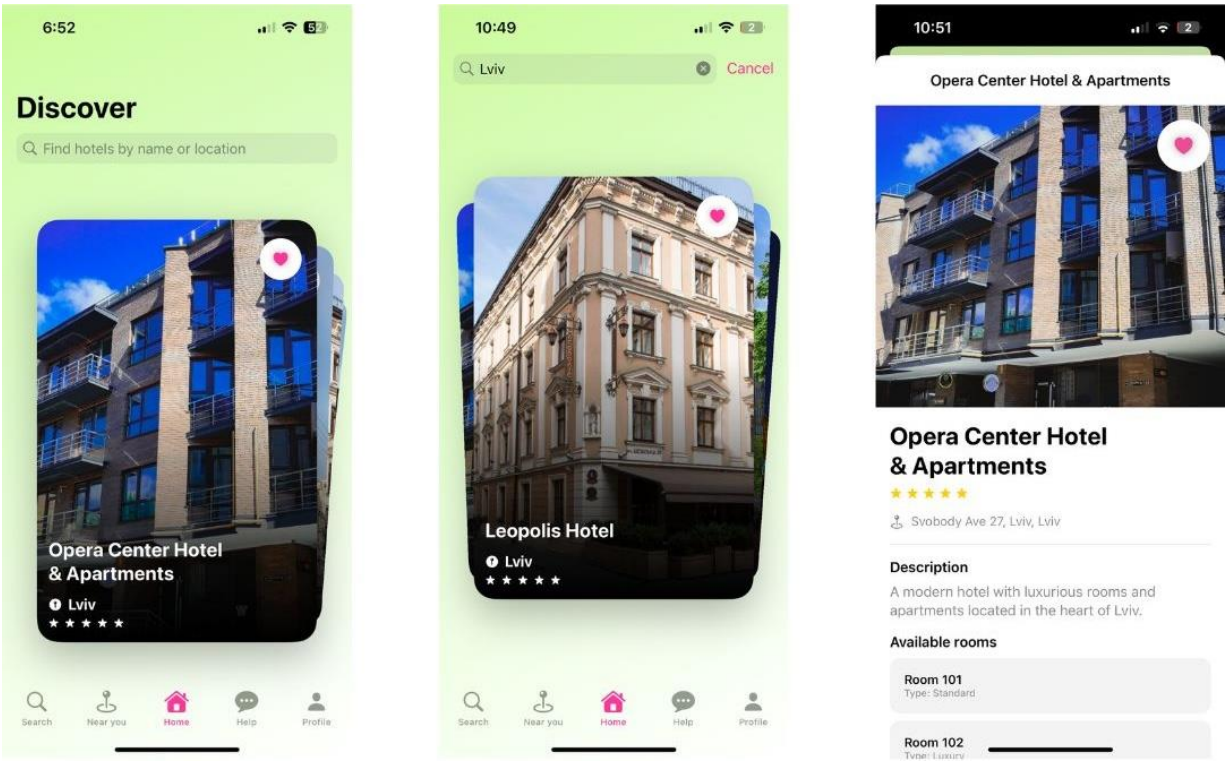

Рисунок 3.8 – Головна сторінка (Home) з пошуковою системою та можливістю перегляду детальної інформації про окремий готель

Наступною вкладкою є вкладка під назвою Help. Основне призначення цієї вкладки полягає у наданні допомоги та підтримки користувачу, тобто вона виконує роль технічної підтримки, де можна запитати, наприклад, інформацію про готель або дізнатися, як дістатися до готельного комплексу. Розмову з користувачем здійснює вбудований штучний інтелект.

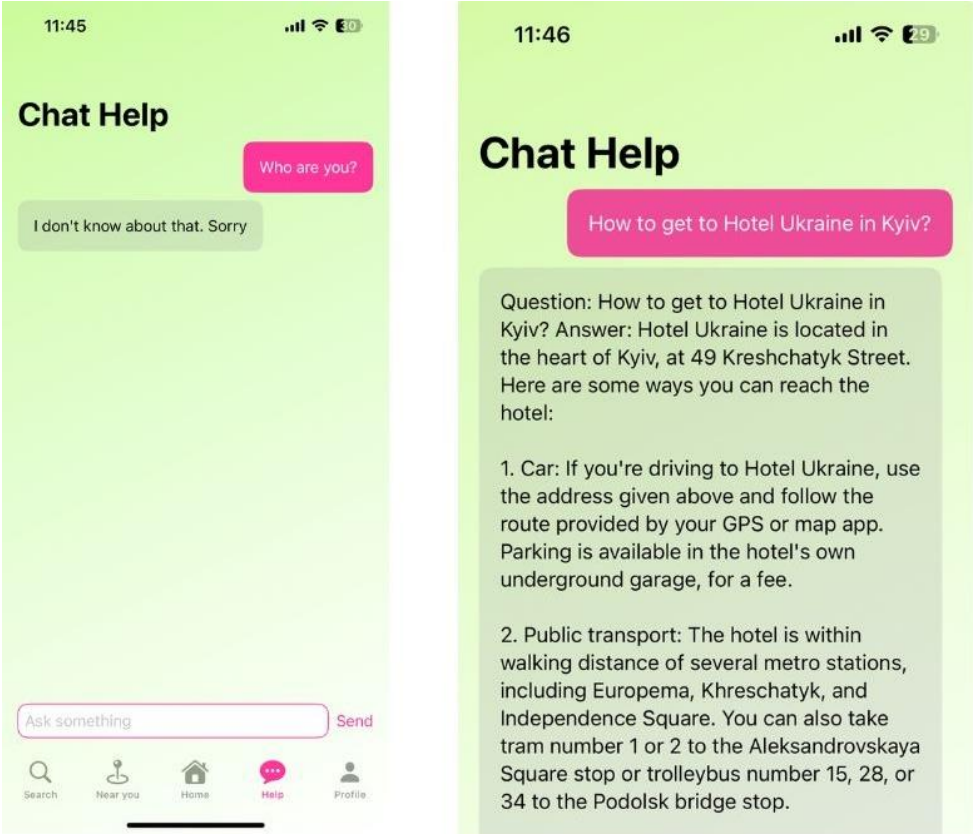

Рисунок 3.9 – Сторінка для спілкування з чат-ботом

Для зручного пошуку готелів була реалізована вкладка Near You. Вона відповідає за відображення карти, на якій користувач може побачити багато позначок рожевого кольору, кожна з яких представляє окреме місто, у якому розміщені певні готельні комплекси. Якщо натиснути на один з позначок, відкриється список доступних готелів у цьому місті.

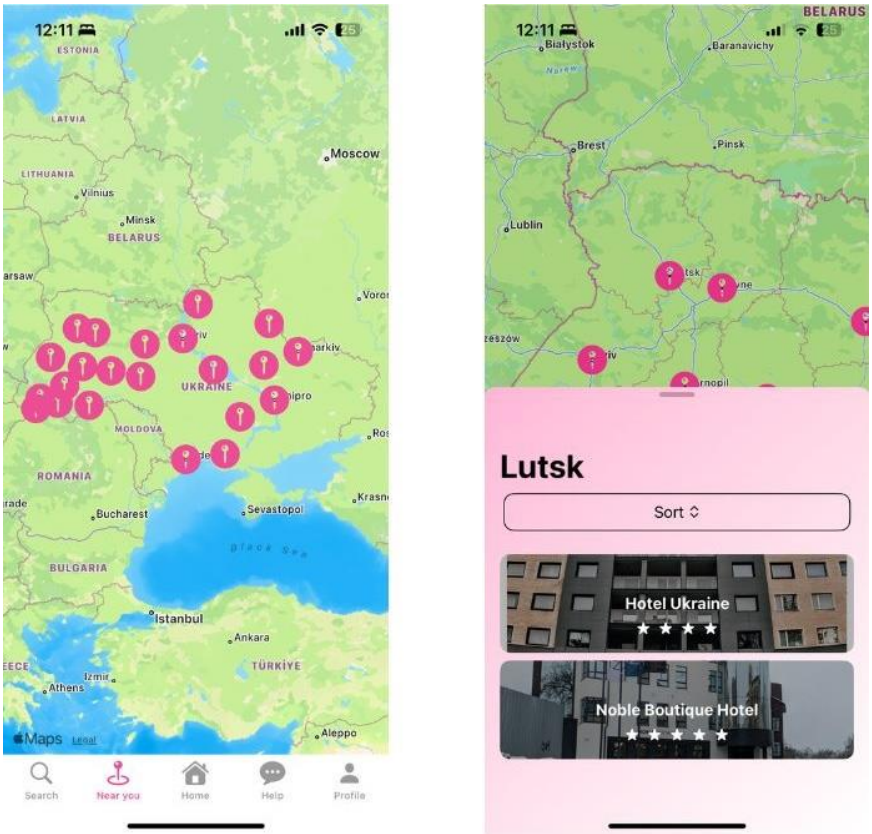

Рисунок – 3.10 – Сторінка, яка відображає міста України, в яких є готелі

Однією із найважливіших вкладок у межах цього додатку є вкладка під назвою Profile. Саме в цій секції користувач працює з особистою інформацією користувача, тобто тут є можливість переглядати та редагувати дані профілю, перегляд свої резервації, вподобані готелі та номери. Крім того, було реалізовано функцію, яка дозволяє користувачу залишити власний відгук про один із готелів, а також переглядати раніше залишені рецензії.

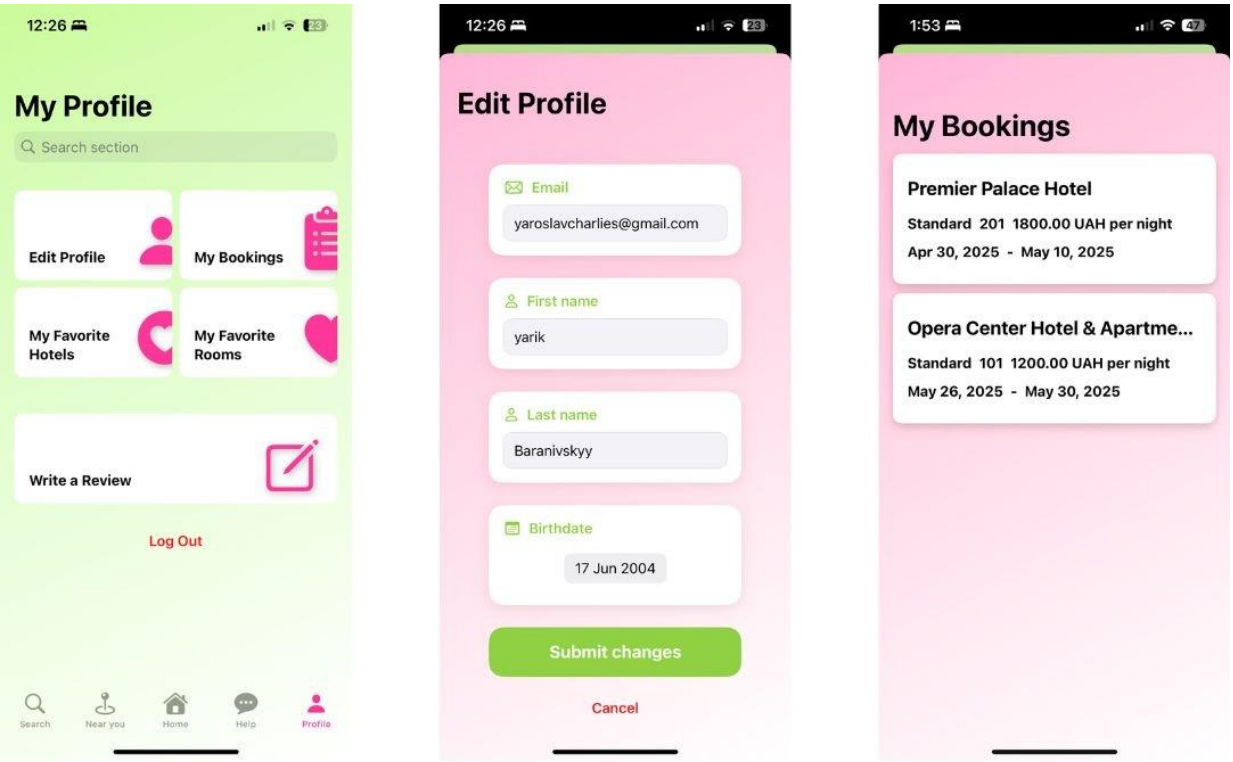

Рисунок – 3.11 – Сторінки для керування особистими даними та перегляду власних резервацій

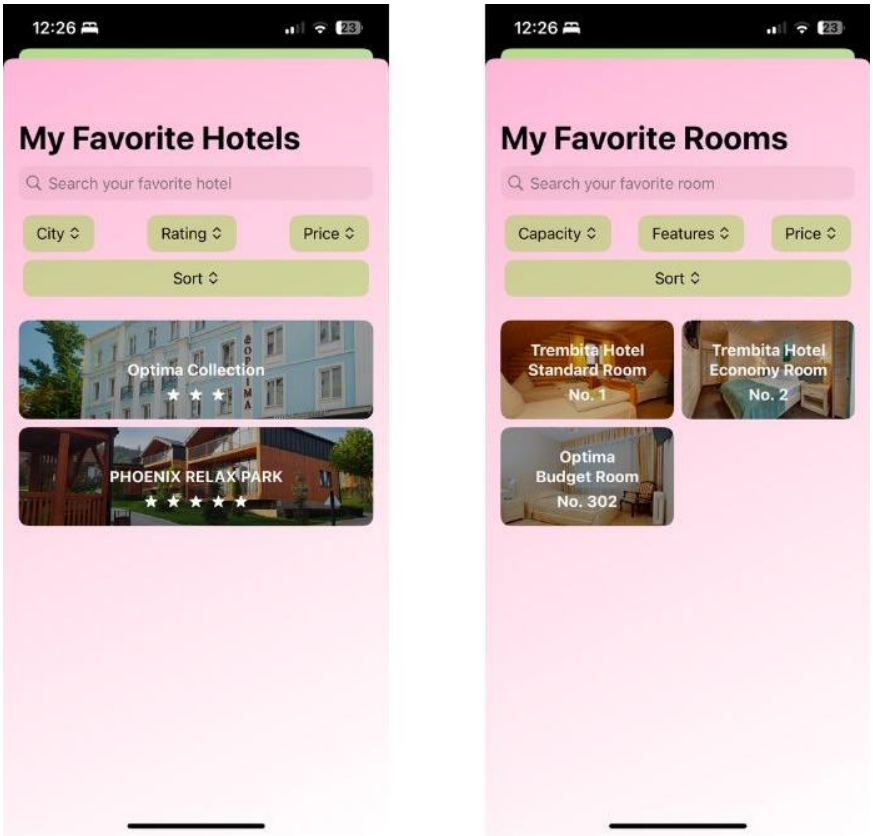

Рисунок – 3.12 – Сторінки для перегляду вподобаних готелів та номерів з можливістю пошуку, фільтрації та сортування

Серцем цього додатку є вкладка під назвою Search. Вона надає користувачеві можливість знайти готель за його назвою, а також відфільтрувати результати за містом, кількістю зірок або середньою ціною за номер. Додатково доступне сортування готелів в алфавітному порядку або за кількість зірок (рейтингом). При переході на сторінку конкретного готелю можна переглянути перелік вільних номерів на даний момент. Ця сторінка також підтримує фільтрацію за місткістю, наявністю певних зручностей, ціною за одну ніч, а також дозволяє сортування списку номерів за їхніми номерами. При переході на сторінку окремого номера, користувач може ознайомитися з його детальною інформацією, додати його до списку вподобаних номерів або здійснити резервацію. Після цього користувача буде перенаправлено на форму підтвердження резервації, де потрібно лише вибрати дату заїзду та дату виїзду, а потім підтвердити бронювання. Після успішного підтвердження з'явиться відповідне повідомлення, а на електронну пошту буде надіслано лист із підтвердженням бронювання номеру.

Рисунок – 3.13 – Сторінка для перегляду готельних комплексів з можливістю пошуку, фільтрації та сортування

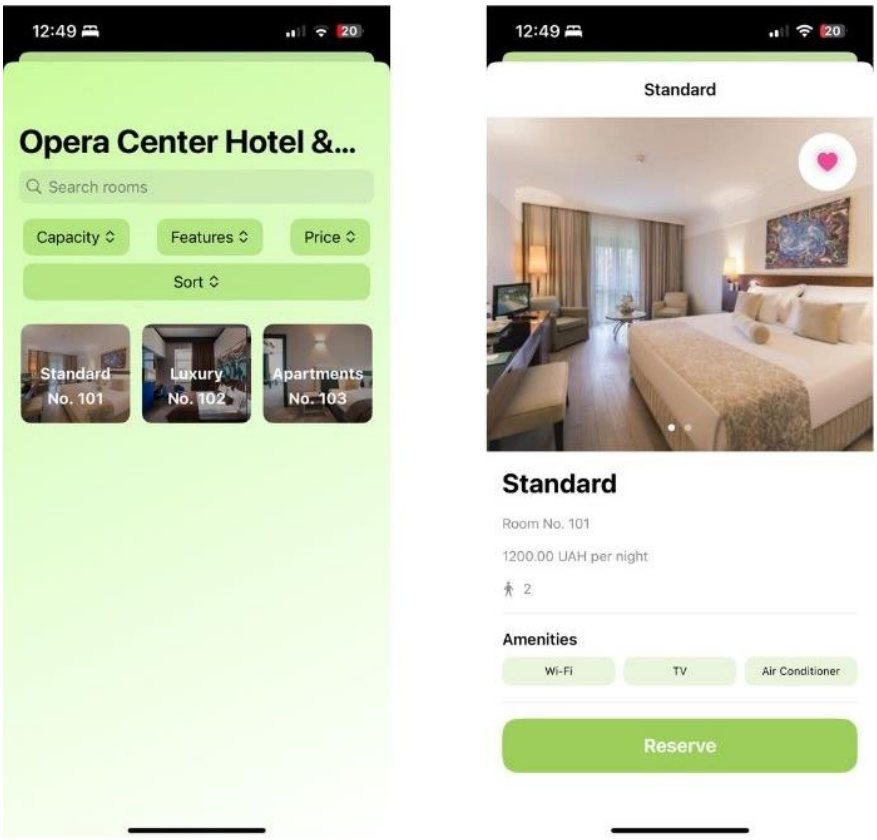

Рисунок – 3.14 – Сторінки для перегляду вільних готельних номерів і детальної інформації про обраний номер

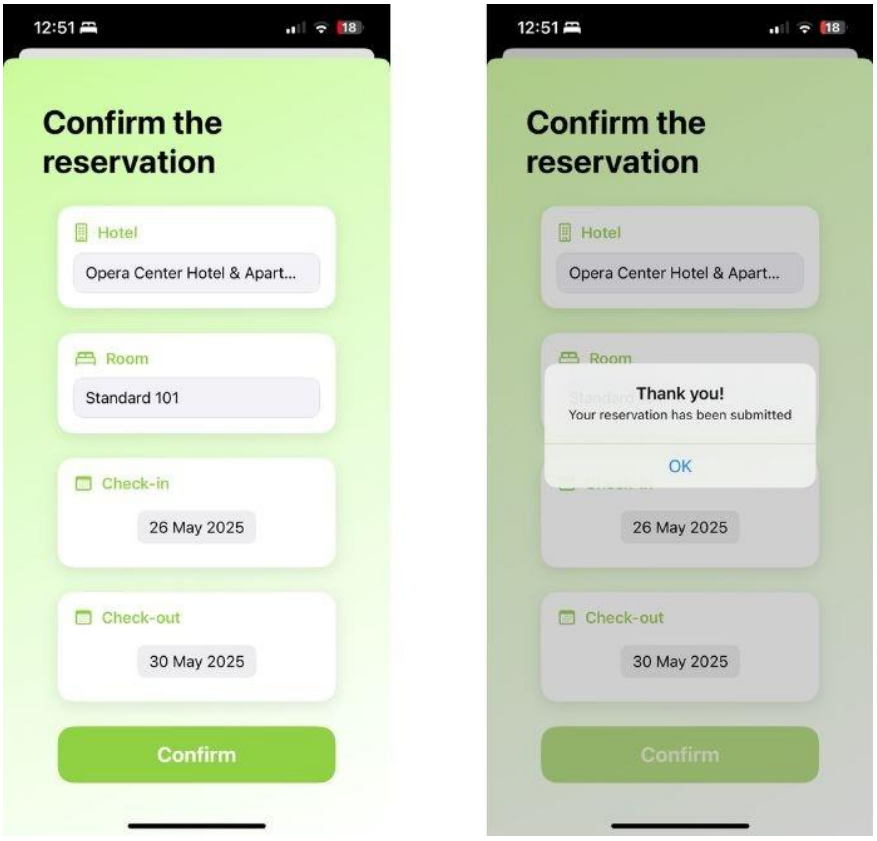

Рисунок – 3.15 – Форма для підтвердження резервації готельного номера

Отже, проаналізувавши структуру додатку, можна зробити висновок, що застосунок є дуже зручним у використанні та буде інтуїтивно зрозумілим для всіх типів користувачів. Це стало можливим завдяки навігації через компонент TabView, який дозволяє поділити програму на окремі логічні секції. Кожна вкладка чітко виконує свої обов'язки відповідно до задуму. Додаток оснащено зручними пошуковими механізмами, різними видами фільтрації та сортування, що дозволяє працювати з інформацією набагато швидше та ефективніше. Такі додаткові функції, як можливість додавати готелі та номери до списку вподобаних, технічна підтримка у вигляді чат-бота та система відгуків, де користувач може залишити власний відгук або переглядати попередні, роблять взаємодію з додатком ще більш комфортною та адаптованою до власних потреб користувача.

РОЗДІЛ 4
ТЕХНІЧНА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ДОДАТКУ

4.1 Робота з Firebase Database і Firebase Authentication

Як зазначалося раніше, для роботи з користувачами було використано інструменти Firebase Database і Firebase Authentication, тоді як для обробки даних про готелі та готельні номери - RESTful API. Кожен з цих компонентів відповідає за зберігання окремих типів даних і працює незалежно один від одного. В рамках цього підпункту буде розглянута автентифікація та реєстрація користувачів.

Реєстрація в додатку доступна лише через обліковий запис Google. Після реєстрації користувач може входити до свого акаунту як через Google акаунт, так і за допомогою електронної пошти та пароля, прив'язаних до цього акаунту.

```
    func add_user_to_db(email: String, first_name: String, last_name: String, password: String, birthdate: Date) {


        AuthenticationView().linkPasswordToAccount(email: email, password: password)
        print("Linked password to account")

        let hashed_password = RegisterUser.hash_password(pass: password)

        model.addData(firstname: first_name, lastname: last_name, email: email, password: hashed_password, birthdate: birthdate) { success in
            DispatchQueue.main.async {
                if success {
                    registrationSuccess = true
                } else {
                    err = "Failed to register. Please try again."
                }
            }
        }
    }

    static func hash_password(pass: String) -> String{
        let inputPass = Data(pass.utf8)
        let hashed = SHA256.hash(data: inputPass)
        return hashed.compactMap { String(format: "%02x", $0)}.joined()
    }

}
```

Рисунок 4.1 – Частина коду з View під назвою RegisterUser, яка відповідає за повноцінну реєстрацію користувача

Метод add_user_to_db реалізовує процес реєстрації нового користувача в системі. При додаванні користувача, викликається метод linkPasswordToAccount, який створює прив'язку пароля до акаунту користувача через електронну пошту. Потім виконується хешування пароля за допомогою методу hash_password, що створює хеш пароля перед його зберіганням у базі даних. Для цього використано алгоритм SHA256, який гарантує безпечне зберігання пароля у вигляді унікального хешу. Після хешування пароля викликається метод addData, що є кінцевим етапом додавання користувача до бази даних. Передаються основні параметри, такі

як ім’я, прізвище, email тощо. Після виконання цього методу відбувається перевірка на успішність реєстрації. У разі помилки користувач отримає відповідне повідомлення.

```
func addData(firstname: String, lastname: String, email: String, password: String, birthdate: Date, completion: @escaping (Bool) -> Void){
    let db_users = Firestore.firestore()
    guard let currentUser = Auth.auth().currentUser else {return}

    let uid = currentUser.uid

    db_users.collection("users").addDocument(data: ["uid": uid,
                                                    "firstname": firstname,
                                                    "lastname": lastname,
                                                    "email": email,
                                                    "password_hashed": password,
                                                    "birthdate": birthdate,
                                                    "favorites": [
                                                        "hotels": [],
                                                        "rooms": []
                                                    ],
                                                    "reservations": [],
                                                    "reviews": []
                                                   ]) {
                                                        error in

                                                        if error == nil {
                                                            completion(true)
                                                            self.getData()
                                                        }
                                                        else {
                                                            completion(false)
                                                        }
                                                    }
}
```

Рисунок 4.2 – Метод addData, який додає інформацію про користувача до бази даних Firebase Database

Даний метод відповідає за повноцінне додавання користувача до документа в колекцію users, яка містить інформацію про всіх користувачів додатку. Спочатку створюється посилання на базу даних, і отримується екземпляр Firebase. Далі відбувається перевірка, чи користувач увійшов у систему, і отримується унікальний ідентифікатор (uid), який буде занесений у базу даних. Наявність uid потрібна для коректної роботи з резерваціями, вподобаннями та відгуками кожного користувача, оскільки він є ключовим елементом, що забезпечує зв’язок між діями користувача та його обліковим записом. Після цього створюється новий документ у колекції users, і до нього додаються відповідні дані. Дані зберігаються у вигляді нового документа в колекції. При успішному додаванні документа, викликається замикання completion(true), що підтверджує успішне завершення методу. В іншому випадку повертається completion(false), що вказує на помилку при додаванні інформації.

Для автентифікації користувача передбачено два способи: через електронну пошту і пароль, або через Google акаунт. У View під назвою LoginView користувач може здійснити вхід до свого акаунту за допомогою Firebase Authentication.

```
func login() {
    Auth.auth().signIn(withEmail: email, password: userPassword) {
        authResult, error in

        if let error = error {
            loginError = "Wrong email or password. Try again."
            return
        }

        isLoggedIn = true
    }
}
```

Рисунок 4.3 – Метод login, який відповідає за автентифікацію через середовище Firebase

У разі успішного входу змінна isLoggedIn змінюється на true. Якщо вхід не вдалося виконати, користувач отримає повідомлення про помилку.

Другий спосіб автентифікації – через Google акаунт. Користувач обирає опцію входу через Google, за допомогою кнопки “Sign in with Google”. Спочатку для цього методу налаштовується необхідна конфігурація, після чого здійснюється вхід користувача за допомогою Google. У разі успішної реєстрації створюється обліковий запис, але запис в базі створюється лише, якщо користувач ще не був зареєстрований. Якщо акаунт вже є, при спробі входу через Google, користувача відразу перенаправляє на головний екран додатку, оскільки акаунт уже занесений в базу даних.

```
func signInWithGoogle(){
    guard let clientID = FirebaseApp.app()?.options.clientID else { return }

    let config = GIDConfiguration(clientID: clientID)
    GIDSignIn.sharedInstance.configuration = config

    GIDSignIn.sharedInstance.signIn(withPresenting: Application_utility.rootViewController) {
        user, error in

        if let error = error {
            print(error.localizedDescription)
            return
        }

        guard let user = user?.user, let idToken = user.idToken else { return }

        let accessToken = user.accessToken

        let credential = GoogleAuthProvider.credential(withIDToken: idToken.tokenString, accessToken: accessToken.tokenString)

        Auth.auth().signIn(with: credential) { res, error in
            if let error = error {
                print(error.localizedDescription)
                return
            }

            guard let user = res?.user else { return }
            print(user)
```

Рисунок 4.4 – Метод signInWithGoogle, який виконує вхід через Google акаунт

У структурі InitialView, яка відповідає за відображення того, що бачить користувач на екрані (форму для входу чи головну сторінку додатку), перевіряється, чи занесений акаунт користувача у базу даних. Це

можливо перевірити за допомогою методу checkIfUserExists, який приймає електронну пошту як аргумент і перевіряє, чи є пошта занесеною у колекції users.

```
}.onAppear{
    authListenerHandle = Auth.auth().addStateDidChangeListener { auth, user in
        if let user = user {
            userLoggedIn = true
            userEmail = user.email ?? ""

            ViewModel.checkIfUserExists(email: $userEmail) {
                exists in
                DispatchQueue.main.async {
                    withAnimation {
                        userhasAccount = exists
                        isLoading = false
                    }

                }
            }
        }
        else {
            userLoggedIn = false
            userEmail = ""
            DispatchQueue.main.async {
                withAnimation {
                    userhasAccount = false
                    isLoading = false
                }
            }
        }
    }
}

.onDisappear {
    if let handle = authListenerHandle {
        Auth.auth().removeStateDidChangeListener(handle)
    }
}
```

Рисунок 4.5 – Перевірка наявності акаунту користувача

4.2 Робота з Django REST Framework

Для роботи з інформацією про готелі та готельні номери використовуються моделі Swift, що взаємодіють з RESTful API, реалізованим за допомогою Django REST Framework. Розглянемо приклад на основі роботи з інформацією про готельні номери.

Модель Room відповідає за структуру кожного номера готелю. Вона реалізовує протоколи Codable, Identifiable та Hashable, що забезпечує можливість серіалізації в JSON, підтримку ідентифікаторів за унікальним id, а також використання в колекціях, що потребують хешування.

```
struct Room: Codable, Identifiable, Hashable{
    var id: Int
    var room_number: Int
    var room_type: String
    var price_per_night: String
    var is_available: Bool
    var capacity: Int
    var features: [String]
    var hotel: Int
    var images: [String]
}
```

Рисунок 4.6 – Структура Room, яка репрезентує готельний номер

Для отримання повної інформації про готельні номери використовується клас RoomViewModel, який забезпечує оновлення даних у режимі реальному часу, адже підтримує протокол ObservableObject. Окрім основного методу для отримання даних з API, цей клас також містить методи для отримання списку номерів за ідентифікатором готелю, пошук кімнати за її id, метод для розрахунку середньої ціни номерів в межах одного готелю (використовується для фільтрації при пошуку). Особливо важливим є метод updateRoomAvailability, який оновлює стан номера, наприклад, коли він бронюється або звільняється.

```
class RoomViewModel: ObservableObject {
    @Published var rooms: [Room] = []

    func fetchData(){
        let url = URL(string: "https://hotel-api-l35v.onrender.com/api/rooms/")!

        var request = URLRequest(url: url)
        request.httpMethod = "GET"
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")

        URLSession.shared.dataTask(with: request){ data, response, error in
            if let error = error {
                print("Error while fetching data:", error)
                return
            }

            guard let data = data else {
                return
            }

            do {
                let decodedRooms = try JSONDecoder().decode([Room].self, from: data)
                DispatchQueue.main.async {
                    self.rooms = decodedRooms
                }
                for room in decodedRooms {
                    print("Room ID:", room.id)
                    print("Room hotel: ", room.hotel)
                    print("Room number: ", room.room_number)
                }
            } catch let jsonError {
                print("Failed to decode json", jsonError)
            }

        }
        .resume()
    }
```

Рисунок 4.7 – Метод fetchData для отримання інформації з API про готельні номери

```
@MainActor
func updateRoomAvailability(roomId: Int, available: Bool) async {

    let url = URL(string: "https://hotel-api-l35v.onrender.com/api/rooms/\(roomId)/")!
    print("url to update room: ", url)
    var request = URLRequest(url: url)
    request.httpMethod = "PATCH"
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")

    let body: [String : Any] = ["is_available": available]
    request.httpBody = try? JSONSerialization.data(withJSONObject: body)

    do {
        let (data, response) = try await URLSession.shared.data(for: request)  ⚠ Immu
        if let httpResponse = response as? HTTPURLResponse,
            httpResponse.statusCode == 200 {
                if let index = self.rooms.firstIndex(where: { $0.id == roomId}) {
                    DispatchQueue.main.async {
                        self.rooms[index].is_available = available
                    }

                }
            print("Done updating room")
        }
    }
    catch {
        print("Error: \(error.localizedDescription)")
    }

}
```

Рисунок 4.8 – Метод updateRoomAvailability для зміни статусу доступності готельного номера

Для зручного перегляду всієї інформації про готелі та готельні номери з приємним і сучасним інтерфейсом була реалізована структура HotelRoomsView, яка забезпечує доступ до актуальних даних в режимі реального часу. Завдяки можливостям SwiftUI реалізовано можливості фільтрації та сортування номерів за різними критеріями. Для відображення списку номерів використовується компонент LazyVGrid, особливістю якого є те, що він створює лише ті елементи інтерфейсу, шо знаходяться в області видимості або близько до неї [6]. Це дозволяє ефективніше використовувати пам’ять і ресурси пристрою при роботі з даними додатку.

```
ScrollView {
    LazyVGrid(columns: columns){
        ForEach(fully_filteredRooms.indices, id: \.self){ room in
            let room_seperate = fully_filteredRooms[room]
            SearchRoomCardView(room: room_seperate)
                .environmentObject(viewModel)
                .onTapGesture {
                    selectedRoom = room_seperate
                }
        }
    }
    .padding()
}
.sheet(item: $selectedRoom){
    room in
    RoomDetailsView(room: room)
        .presentationDetents([.large])
        .environmentObject(viewHotelModel)
}
.navigationTitle(hotel.name)
```

Рисунок 4.9 – Виведення списку номерів

Метод filteredRooms відповідає за поступову фільтрацію списку номерів відповідно до параметрів, заданих користувачем. Зокрема, реалізовано пошук за типом або номером кімнати, фільтрацію за ціною за ніч, місткістю, а також за наявними зручностями. Такий підхід допомагає користувачам знайти потрібні варіанти відповідно до їхніх запитів.

```
var filteredRooms: [Room] {
    let rooms = viewModel.getRoomsByNumber(hotel_id: hotel.id)

    var filtered = rooms.filter { room in
        var isMatch = true

        if searchTerm != "" {
            isMatch = isMatch && (room.room_type.localizedCaseInsensitiveContains(searchTerm) || room.room_number ==
                Int(searchTerm))
        }

        if selected_capacity_option != "Capacity"{
            isMatch = isMatch && (room.capacity == capacity_int)
        }

        if selected_features_option != "Features"{
            isMatch = isMatch && (room.features.contains(selected_features_option))
        }

        if selected_price_option != "Price"{
            var price_for_room = Double(room.price_per_night) ?? 0   ⚠ Variable 'price_for_room' was never mutated; consider changing to 'let' const...

            switch selected_price_option{
            case "Up to 500 UAH":
                isMatch = isMatch && (price_for_room < 500.00)
            case "500–1000 UAH":
                isMatch = isMatch && (price_for_room >= 500.00 && price_for_room <= 1000.00)
            case "1000–2000 UAH":
                isMatch = isMatch && (price_for_room >= 1000.00 && price_for_room <= 2000.00)
            case "2000–5000 UAH":
                isMatch = isMatch && (price_for_room >= 2000.00 && price_for_room <= 5000.00)
            case "5000+ UAH":
                isMatch = isMatch && (price_for_room > 5000.00)
```

Рисунок 4.10 – Фрагмент методу filteredRooms, що повертає список відфільтрованих номерів

4.3 Зв’язок між Django REST Framework та Firebase Database

Як було з’ясовано раніше, Firebase Database та Django REST Framework є незалежними один від одного і жодним чином не перетинаються напряму. Відомо, що дані про вподобані готелі, номери зберігаються у Firebase Database у вигляді ідентифікаторів готелів і номерів, тоді як у Django REST Framework зберігається вже детальна інформація про самі готелі та номери. Але постає питання: як поєднати два абсолютно різних середовища, які не мають прямого зв’язку між собою?

Для цього була розроблена логіка, яка забезпечує логічну інтеграцію на рівні клієнтської частини за допомогою ViewModel і механізму SwiftUI під назвою @EnvironmentObject.

@EnvironmentObject – це спеціальна обгортка властивості в SwiftUI, яка дозволяє передавати доступ до спільного об’єкта даних (ViewModel) між кількома View без необхідності явно передавати його через ініціалізатори [7].

```
ProfileView()
    .tabItem{
        Label("Profile", systemImage: "person")
    }
    .tag(3)
    .environmentObject(viewModel)
    .environmentObject(viewUserModel)
    .environmentObject(viewRoomModel)
```

Рисунок 4.11 – Передача ViewModel-об’єктів у ProfileView через .enviornmentObject() для доступу до даних із різних джерел

```
struct ProfileView : View {
    @EnvironmentObject private var viewUserModel: ViewModel
    @EnvironmentObject private var viewHotelModel: HotelViewModel
    @EnvironmentObject private var viewRoomModel: RoomViewModel

    @State private var userEmail = ""
    @State private var searchTerm = ""

    let options : [Option] = [
        Option(id: 1, icon: Image(systemName: "person.fill"), title: "Edit Profile", destination: {
            AnyView(MyProfile())
        }),
        Option(id: 3, icon: Image(systemName: "list.bullet.clipboard.fill"), title: "My Bookings", destination: {
            AnyView(MyReservations())
        }),
        Option(id: 2, icon: Image(systemName: "heart.circle.fill"), title: "My Favorite Hotels", destination: {
            AnyView(MyFavoriteHotels())
        }),
        Option(id: 4, icon: Image(systemName: "heart.fill"), title: "My Favorite Rooms", destination: {
            AnyView(MyFavoriteRooms())
        })
    ]
```

Рисунок 4.12 – Використання @EnvironmentObject для доступу до даних у ProfileView

```
.onAppear {
    viewHotelModel.fetchData()
    viewRoomModel.fetchData()
    viewUserModel.getFavoriteRooms()
    viewUserModel.getFavoriteHotels()
    viewUserModel.getReservations()
    viewUserModel.getReviews()
}
```

Рисунок 4.13 – Завантаження даних при появі ProfileView за допомогою модифікатора onAppear

```
func getFavoriteHotels(){
    if let currentUser = Auth.auth().currentUser {
        let userId = currentUser.uid
        let db_users = Firestore.firestore()

        let user_seperate: Void = db_users.collection("users").whereField("uid", isEqualTo: userId).getDocuments() {
            (querySnapshot, error) in

            if let document = querySnapshot?.documents.first {
                let data = document.data()
                if let favorites = data["favorites"] as? [String: Any],
                   let hotelsArray = favorites["hotels"] as? [Any] {

                    if let hotelIds = hotelsArray as? [Int] {
                        print("ids: ", hotelIds)
                        DispatchQueue.main.async{
                            self.hotels_list = hotelIds.compactMap {
                                hotelId in
                                let hotel = self.viewHotelModel?.getHotelById(hotelId: hotelId)
                                print("VIEWMODEL: ", hotel?.id ?? 0, "", hotel?.name ?? "")
                                return hotel
                            }
                        }
                    }

                }

            }
        }
    }
}
```

Рисунок 4.14 – Отримання списку улюблених готелів користувача

```
ScrollView {
    LazyVGrid(columns: columns) {
        ForEach(filteredFavoriteHotels.indices, id: \.self) { index in
            let hotel = filteredFavoriteHotels[index] ?? Hotel.placeholder
            NavigationLink {
                HotelDetailsView(hotel: hotel)
                    .onAppear {
                        print("Displaying hotel ID: \(hotel.id)")
                    }


            } label: {
                FavoriteHotelCard(hotel: hotel)
                    .onLongPressGesture {
                        showMenuOnLongPress = true
                    }
                    .contextMenu{
                        Button {
                            selectedHotel = hotel

                        } label: {
                            Label("Open Hotel", systemImage: "magnifyingglass")
                        }
                        Button {
                            viewUserModel.removeHotelFromFavorites(hotelId: hotel.id)
                        } label: {
                            Label("Remove from Favorites", systemImage: "heart.slash")
                        }
                    }
            }

        }
    }
    .padding()
}
.navigationTitle("My Favorite Hotels")
```

Рисунок 4.15 – Виведення списку улюблених готелів користувача з можливістю перегляду деталей і керування через контекстне меню

Наприклад, у View під назвою MyFavoriteHotels, до якої підключено ViewModel за допомогою @EnvironmentObject, може бути використано ідентифікатор готелю, який був отриманий з Firebase. Індекс, що надходить із Firebase Database, записується у змінну hotel і передається в структуру FavoriteHotelCard для коректного відображення готелів. Ця структура використовує дані про конкретний готель, передані через властивість hotel. Оскільки hotel є об’єктом, який вже містить усю необхідну інформацію, немає потреби використовувати додаткові методи для її отримання - завдяки використанню @EnvironmentObject усі дані будуть відразу доступні.

ВИСНОВОК

В рамках даного дослідження було розроблено повноцінний мобільний застосунок для операційної системи iOS, основною метою якого є спрощення взаємодії іноземних туристів з українським готельним сервісом та покращення їхнього користувацького досвіду. Робота з готелями охоплює такі ключові можливості, як бронювання номерів, швидкий пошук з фільтрацією, миттєвий доступ до інформації про готелі та готельні номери, актуальність даних, а також публікація та перегляд рецензій від інших туристів. Досягти результату вдалося завдяки детальному аналізу сучасного туристичного ринку в Україні, дослідженню потреб іноземний користувач, а також використанню оптимізованих і сучасних технологічних інструментів для розробки всіх необхідних компонентів додатку. Було пройдено всі основні етапи якісної розробки - від початкового аналізу до повноцінної реалізації всіх запланованих функціональних можливостей додатку.

Початковим етапом розробки став аналіз поточної ситуації на українському туристичному ринку. Завдяки детальному вивченню ситуації вдалося визначити, чого саме прагне цільова аудиторія і з якими труднощами стикається іноземний турист під час подорожей Україною. Була також проведена оцінка конкурентів, що дозволило проаналізувати їхні сильні та слабкі сторони, і на основі цього уникнути подібних помилок, а також запропонувати простіші та ефективніші рішення для розв’язання тієї чи іншої проблеми.

На основі зібраної інформації стало зрозуміло, які функціональні можливості потрібно імплементувати для створення повноцінного додатку. Було прийнято рішення розробити iOS додаток за допомогою середовища Xcode та мови програмування Swift, адже саме вони забезпечують високоякісну та безпечну розробку з урахуванням всіх необхідних аспектів функціонування додатку.

Для правильного зберігання всієї інформації щодо користувачів, надійної системи автентифікації та швидкого доступу до цих даних було використано хмарне середовище Firebase. Окрім того, що це один із найкращих сервісів для роботи з користувацькими даними та автентифікацією, він також забезпечує синхронізацію даних у реальному часі між різними пристроями, що є важливим у контексті додатку, орієнтованого на взаємодію з користувачами. Для роботи з даними про готелі та готельні номери було розроблено власне RESTful API за допомогою потужного інструменту Django REST Framework. Завдяки цьому вдалося побудувати ефективну архітектуру, через яку відбувається отримання даних про готелі та номери.

Для забезпечення стабільного підключення до API та доступу до інформації було використано платформу Render, за допомогою якої вдалося

розгорнути сервісну частину. Усі перелічені компоненти зробили вагомий внесок у створенні стабільного та функціонального додатку, здатного задовільнити всі потреби іноземних туристів.

Показником якісного додатку є зручний та інтуїтивно зрозумілий інтерфейс, який не перевантажує користувачів. Для цього додаток було поділено на п’ять окремих розділів з унікальним функціоналом, перемикатися між якими можна за допомогою навігаційного компонента TabView. Додаток включає такі вкладки, як пошук готелів та готельних номерів, карта для зручної орієнтації, головна сторінка з усіма готелями, технічна підтримка у вигляді чат-боту та розділ для керування особистим профілем. Як було зазначено раніше, кожна вкладка містить певний набір необхідного функціоналу для зручного користування додатком.

Наприклад, вкладка “Search” дозволяє знайти готелі за назвою, а також застосувати фільтрацію за такими параметрами, як місто, кількість зірок та середня ціна номеру за одну ніч. Крім того, користувач може сортувати список готелів та окремо переглядати номери кожного готелю, з можливістю фільтрації та сортування за певними критеріями. Сторінка окремого номера надає користувачу можливість переглянути всю необхідну інформацію, вподобати номер або відразу здійснити бронювання через спеціальну форму, яка заповнюється всього в два кліки. Після успішного бронювання, користувач отримує повідомлення та автоматичний лист-підтвердження на свою електронну пошту.

Підсумовуючи вище сказане, можна стверджувати, що розроблений додаток володіє всіма якостями сучасного та ефективного продукту. Додаток естетично привабливий, оскільки великий акцент був зроблений на підборі гарної кольорової палітри. Він швидко працює, зручний у використанні та інтуїтивно зрозумілий. Цей додаток формує надійну платформу для можливого подальшого розвитку, метою якого є ще більше покращення користувацького досвіду.

Отже, усі поставлені вимоги були повністю виконані, і результатом плідного дослідження став сучасний додаток, здатний максимально полегшити перебування іноземних туристів в Україні, забезпечуючи найкращий туристичний досвід та приємні враження від ефективності українського сервісу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. До бюджету громад у 2024 році надійшло майже 273 млн грн туристичного збору [Електронний ресурс]. – Режим доступу до ресурсу: https://www.tourism.gov.ua/blog/do-byudzhetu-gromad-u-2024-roci-nadiyshlo-mayzhe-273-mln-grn-turistichnogo-zboru (дата звернення: 02.04.2025).

2. StatCounter: Mobile Operating System Market Share in Canada (2024) [Електронний ресурс]. – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/Canada/2024 (дата звернення: 05.04.2025).

3. CIRP: iOS boasts a 39% share in US smartphone market [Електронний ресурс]. – Режим доступу до ресурсу: https://www.gsmarena.com/cirp_ios_boasts_a_39_share_in_us_smartphone_market-news-60287.php (дата звернення: 05.04.2025).

4. iPhone users spend seven times more on apps than Android users [Електронний ресурс]. – Режим доступу до ресурсу: https://9to5mac.com/2023/09/06/iphone-users-spend-apps/ (дата звернення: 02.04.2025).

5. MVVM: An architectural coding pattern to structure SwiftUI Views [Електронний ресурс]. – Режим доступу до ресурсу: https://www.avanderlee.com/swiftui/mvvm-architectural-coding-pattern-to-structure-views/ (дата звернення 08.04.2025)

6. SwiftUI. LazyVGrid [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/documentation/swiftui/lazyvgrid (дата звернення 20.04.2025)

7. SwiftUI. EnvironmentObject [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.apple.com/documentation/swiftui/environmentobject (дата звернення 22.04.2025)