Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ КВЕСТ-КІМНАТ»

Виконав: студент IV курсу, групи 1ІСТ-21б
спеціальності 126 – Інформаційні системи та технології
Чумак О. В.
Керівник: д.т.н., професор каф. АІІТ
Квєтний Р. Н.
Консультант: к.т.н., доцент каф. АІІТ
Богач І. В.
«11» 06 2025 р.
Рецензент: д.т.н., професор каф. КН
Іванчук Я. В.
«11» 06 2025 р.

Допущено до захисту
зав. кафедри АІІТ
«12» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 126 – Інформаційні системи та технології
Освітньо-професійна програма Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
_____ д.т.н., проф. Олег БІСІКАЛО
«12» 06 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Чумаку Олексію Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка мобільного застосунку для бронювання квест-кімнат» керівник роботи Квєтний Роман Наумович, д. т. н., проф., проф. каф. АІІТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «20» березня 2025 року №97
2. Термін подання студентом роботи 10.06.2025 р.
3. Вихідні дані до роботи:
• Перелік квест-кімнат, що включає: назву, опис, місто, рівень складності, кількість гравців, ціни;
• Графік доступних слотів для бронювання, з вказанням дати та часу;
• Дані користувачів, отримані під час реєстрації (ім'я, email, номер телефону, роль);
• Результати автентифікації через Firebase, які включають унікальний ID;
• Фотоматеріали кімнат (URL зображень);
• Тексти інтерфейсу для локалізації, розділені по мовах (UA, EN).
4. Зміст текстової частини
Аналіз предметної області. Проєктування мобільного застосунку для бронювання квест-кімнат. Реалізація клієнтського застосунку.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Діаграма взаємодії при створенні бронювання. Діаграма взаємодії при автентифікації користувача. Діаграма взаємодії адміністратора при редагуванні кімнати.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-3	Богач І. В., доц. каф. АІІТ		

7. Дата видачі завдання 20.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вибір, узгодження та затвердження теми бакалаврської роботи	07.03.2025	20.03.2025	вик.
2	Аналіз літературних джерел, сучасного стану задачі та формулювання цілей	21.03.2025	30.04.2025	вик.
3	Аналіз аналогів та визначення ключових критеріїв	01.04.2025	12.04.2025	вик.
4	Дослідження інструментів для реалізації мобільного застосунку	13.04.2025	26.04.2025	вик.
5	Реалізація MVP застосунку з базовими функціями бронювання	06.05.2025	20.05.2025	вик.
6	Тестування, усунення помилок, оптимізація UI та UX	21.05.2025	24.05.2025	вик.
7	Підготовка пояснювальної записки, UML-діаграм, додатків та скріншотів застосунку	25.05.2025	01.06.2025	вик.
8	Попередній захист, рецензування, доопрацювання кваліфікаційної роботи	02.06.2025	10.06.2025	вик.
9	Захист бакалаврської роботи перед ЕК	16.06.2025	17.06.2025	вик.

Студент _____ Чумак О. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Квєтний Р. Н.

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 86 сторінок формату А4 включаючи додатки, на яких є 35 рисунки, список використаних джерел містить 25 найменувань.

У бакалаврській кваліфікаційній роботі розглянуто процес розробки мобільного застосунку для бронювання квест-кімнат. Метою роботи є створення зручного та функціонального клієнтського застосунку для платформи iOS із підтримкою реєстрації, авторизації, перегляду доступних кімнат, фільтрації, вибору дати та часу, здійснення бронювання, а також реалізації функціоналу адміністрування – додавання, редагування та видалення квест-кімнат і записів про бронювання.

Застосунок реалізовано з використанням мови програмування Swift та фреймворку SwiftUI з дотриманням архітектурного підходу MVVM. Для аутентифікації користувачів інтегровано Firebase, а обробка запитів до бази даних здійснюється через REST API, реалізоване на серверному фреймворку Vapor. Система підтримує розмежування ролей користувачів (адміністратор і клієнт) з відповідним обмеженням доступу до функціоналу. Результати роботи мають прикладне значення та можуть бути використані для впровадження в організаціях, що надають послуги у сфері розваг, зокрема командних квест-кімнат.

Ключові слова: Swift, SwiftUI, Vapor, REST API, Firebase, мобільний застосунок, автентифікація, бронювання квест-кімнат.

ABSTRACT

The Bachelor's degree qualification work consists of 86 pages of A4 format, including appendices, which contain 35 figures, and the list of references consists of 25 titles.

The bachelor’s qualification paper examines the development process of a mobile application for booking escape questRooms. The main objective is to create a convenient and functional iOS client application that supports user registration, authentication, browsing available questRooms, filtering, selecting date and time, booking, as well as administrative functionality – including adding, editing, and deleting escape questRooms and bookings.

The application is implemented using the Swift programming language and the SwiftUI framework, following the MVVM architectural pattern. Firebase is integrated for user authentication, and the application interacts with the database through a REST API developed using the Vapor server-side framework. The system supports user role differentiation (administrator and client) with appropriate access restrictions. The developed solution has practical value and can be used in organizations that provide entertainment services, particularly team-based escape room experiences.

Keywords: Swift, SwiftUI, Vapor, REST API, Firebase, mobile application, authentication, escape room booking.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Загальні відомості про квест-кімнати 7
1.2 Типологія квест-кімнат 8
1.3 Особливості та проблеми бронювання 9
1.3.1 Універсальні SaaS-платформи для бронювання 10
1.3.2 Власні веб-сайти з простими формами 11
1.3.3 Мобільні застосунки-агрегатори 11
1.4 Потреби користувачів системи 12
1.5 Аналіз існуючих рішень (аналогів) на ринку України 13
1.5.1 Аналог 1: QIMNATA 13
1.5.2 Аналог 2: ЗАМКНЕНІ 14
1.5.3 Аналог 3: QuestRoom 15
1.5.4 Визначення ключових критеріїв 17
1.6 Обґрунтування необхідності розробки мобільного застосунку 18
1.7 Аналіз відповідно до визначених ключових критеріїв 19
1.8 Висновки до розділу 1 21
2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ КВЕСТ-КІМНАТ 22
2.1 Постановка задачі 22
2.2 Вибір технологій для виконання поставленої задачі 24
2.2.1 Вибір платформи розробки 24
2.2.2 Вибір фреймворку інтерфейсу 25
2.2.3 Вибір архітектурного патерну 25
2.2.4 Планування взаємодії з хмарним бекендом 26
2.2.5 Підсумок вибору технологій 27
2.3 Роробка архітектури клієнтського застосунку 28
2.4 UML-діаграми клієнтської частини 30
2.4.1 Component Diagram 30

2.4.3 Class Diagram 32
2.4.2 Use Case-діаграма 32
2.5 Висновки до розділу 2 34
3 РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ КВЕСТ-КІМНАТ 35
3.1 Реалізація основного функціоналу 35
3.1.1 Автентифікація, реєстрація та збереження токена 35
3.1.2 Система ролей 36
3.1.3 Перегляд квест-кімнат 36
3.1.4 Перегляд деталей і створення бронювання 36
3.1.5 Перегляд бронювань 37
3.1.6 Адміністративні функції 37
3.2 Користувацький інтерфейс (UI) 38
3.3 Локалізація інтерфейсу 45
3.3.1 Формат локалізації 45
3.3.2 Інтерфейс локалізації в Xcode 45
3.3.3 Інтеграція з SwiftUI 48
3.3.4 Приклади локалізованого інтерфейсу 49
3.3.5 Переваги обраного підходу 51
3.4 Тестування мобільного застосунку та порівняння з аналогами 51
ВИСНОВКИ 54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 56
ДОДАТКИ 59
Додаток А (обов’язковий) Технічне завдання на бакалаврську кваліфікаційну роботу 61
Додаток Б (обов’язковий) ІЛЮСТРАТИВНА ЧАСТИНА 63
Додаток В (обов’язковий) Лістинг основних функцій застосунку 70
Додаток Г (обов’язковий) ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ 85
Додаток Д (довідниковий) Акт впровадження результатів роботи в ТОВ «СПІЛЬНА СПРАВА» 86

ВСТУП

Актуальність роботи. У сучасному світі цифрові технології все глибше інтегруються в різні сфери життя, включаючи розважальну індустрію. Одним із популярних видів активного відпочинку останніх років є квест-кімнати – тематичні ігрові простори, де учасники розв’язують завдання в обмежений час. Зі зростанням кількості таких закладів та розширенням їхньої аудиторії виникає потреба у сучасних ІТ-рішеннях для автоматизації ключових процесів, зокрема – процесу бронювання.

На сьогодні більшість бронювань у квест-кімнатах здійснюється через дзвінки або веб-форми, що часто супроводжується помилками, дублюваннями, або вимагає постійної участі адміністратора. Такий підхід ускладнює масштабування бізнесу, не дозволяє ефективно управляти слотами та не відповідає очікуванням сучасного користувача, орієнтованого на мобільні технології.

Метою роботи є покращення сервісів бронювання квест-кімнат за рахунок розробки модифікованого програмного забезпечення, що, на відміну від існуючих, дозволить спростити взаємодію між клієнтами та адміністраторами, забезпечить автоматизовану обробку бронювань, зменшить ризики помилок і підвищить рівень сервісу.

Для досягнення мети роботи, потрібно **розв’язати наступні задачі**:

1. Провести аналіз предметної області та існуючих мобільних рішень у сфері бронювання квест-кімнат.
2. Дослідити сучасні програмні засоби та технології, які доцільно використати для реалізації мобільного застосунку для бронювання квест-кімнат.
3. Спроєктувати мобільний застосунок для бронювання квест-кімнат.
4. Реалізувати мобільник застосунок для бронювання квест-кімнат.
5. Створити інтерфейс користувача з використанням SwiftUI та архітектурного патерну MVVM.
6. Протестувати розроблений мобільний застосунок.

Об’єктом дослідження є процес взаємодії користувача з інформаційною системою бронювання квест-кімнат у мобільному середовищі.

Предметом дослідження є методи та засоби розробки мобільного програмного забезпечення, орієнтованого на ефективну взаємодію між клієнтами та адміністраторами квест-кімнат, з використанням архітектури MVVM, сучасних фреймворків iOS та хмарних технологій.

Методи дослідження включають:

– аналіз предметної області та існуючих аналогів;

– синтез функціональних вимог і архітектурних рішень;

– підбір та використання сучасних методів мобільної розробки.

Практичний результат роботи полягає в створенні мобільного застосунку для мобільної платформи iOS, що на відміну від існуючих аналогів спростить взаємодію між клієнтами та адміністраторами, забезпечить автоматизовану обробку бронювань та зменшить ризики помилок і підвищить рівень сервісу.

Створений мобільний застосунок підтримує:

– авторизацію користувачів із розподілом на ролі (адміністратор / клієнт),

– збір інформації про користувачів для подальшого використання в цілях ведення бізнесу;

– перегляд та керування квест-кімнатами,

– вибір дати й часу для бронювання,

– виведення для перегляду та редагування інформації про користувачів, кімнати та бронювання,

– зручний адаптивний локалізований інтерфейс на основі SwiftUI.

Реалізація виконана мовою програмування Swift із використанням фреймворків SwiftUI та UIKit для розробки клієнтської частини, яка працює з Firebase і Vapor + PostgreSQL серверної частини.

Апробація результатів дослідження. Результати роботи було представлено на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2025) [1] та на

Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» [2].

Впровадження результатів роботи. Результати роботи планується використати в розробках ТОВ «СПІЛЬНА СПРАВА» (Додаток Д) та підготовлена заявка на отримання свідоцтва на авторське право.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні відомості про квест-кімнати

Квест-кімнати – це інтерактивні розважальні простори, в яких група учасників виконує завдання та логічні головоломки у тематично оформленому середовищі з метою досягнення конкретної мети (вихід із кімнати, розкриття таємниці тощо). Гра обмежена за часом і передбачає тісну взаємодію між учасниками, що робить квести популярними серед компаній друзів, пар, родин і навіть корпоративних клієнтів [3].

Характерною рисою індустрії квест-кімнат є попереднє бронювання сеансів. На відміну від кінотеатрів чи інших масових розваг, квест-кімнати зазвичай обслуговують одну команду за раз і мають фіксований розклад ігор. Як правило, тривалість одного квесту орієнтовно складає 60 хвилин, після чого потрібен час на підготовку кімнати для наступних гравців. Тому компанії впроваджують таймслоти (часові інтервали) для початку ігор, і клієнти повинні резервувати конкретний слот заздалегідь та вказати відповідну кількість гравців, а саме від 2 до максимальної запропонованої кількості. Спонтанні відвідування без бронювання майже не практикуються – за даними індустрії, більшість кімнат повністю заповнені на кілька днів вперед, тож відвідувачам рекомендовано бронювати гру заздалегідь. Це означає, що ефективна система бронювання – критично важливий компонент бізнесу квест-кімнат.

Традиційно на ранніх етапах розвитку індустрії бронювання приймалися вручну – по телефону, через повідомлення або просту форму на сайті, після чого адміністратор вносив запис у журнал чи електронну таблицю. Такий підхід має суттєві недоліки: високий ризик помилок (людський фактор), незручність для клієнтів (необхідність здійснювати дзвінки у робочий час), відсутність миттєвого підтвердження, складність відстеження та зміни бронювання. З розвитком інформаційних технологій з’явилися автоматизовані системи онлайн-бронювання, що значно спрощують процес. Сучасний клієнт очікує, що він

зможе самостійно переглянути доступні часи та забронювати гру цілодобово через інтернет. Для власників квестів автоматизація теж дає переваги: календар заявок оновлюється автоматично, виключаючи подвійне бронювання, клієнт одразу отримує підтвердження та інструкції, а адміністратор – сповіщення про нове бронювання. Крім того, система може зберігати дані гравців, а саме контакти та дату дня народження, які вони надають особисто, що відкриває можливості для маркетингу (повторні запрошення, акції тощо) [4].

З початку 2010-х років ця форма дозвілля набула великої популярності в багатьох країнах світу, зокрема й в Україні. Особливо активно ринок розвивався у великих містах – Києві, Львові, Харкові, Одесі. За останні роки відзначається поява нових форматів: VR-квести, квести на відкритому повітрі, дитячі квести тощо [5].

1.2 Типологія квест-кімнат

Квест-кімнати (escape rooms) стали популярною формою розваг, що поєднує елементи гри, логіки та командної роботи. Залежно від тематики, технологічного оснащення та цільової аудиторії, квест-кімнати поділяються на кілька основних типів, кожен з яких має свої особливості бронювання та організації гри. З найпопулярніших видів та форматів квест-кімнат можна виділити наступні:

1. Класичні (сюжетні) квести – це традиційні квест-кімнати з логічними загадками, де гравці повинні розгадати послідовність завдань, щоб вийти з кімнати. Тематики варіюються від детективних історій до пригодницьких сюжетів. Бронювання зазвичай здійснюється на фіксований час для групи 2–6 осіб.

2. Квести з акторами – у таких квестах задіяні актори, які взаємодіють з гравцями, створюючи більш глибоке занурення в сюжет. Це вимагає додаткової координації та часто має вищу вартість. Бронювання потребує уточнення щодо наявності акторів на обраний час.

3. VR-квести (віртуальна реальність) – квести, що використовують технології віртуальної реальності, дозволяють гравцям зануритися в цифрове середовище. Такі квести можуть бути як індивідуальними, так і командними. Бронювання включає перевірку наявності необхідного обладнання та технічної підтримки.

4. Дитячі квести – розроблені спеціально для дітей, ці квести мають спрощені завдання та безпечне середовище. Бронювання часто враховує вік учасників та потребує присутності дорослих.

5. Тематичні квести – квести, присвячені певним темам або франшизам (наприклад, “Гаррі Поттер”, “Пірати Карибського моря”), приваблюють фанатів відповідних сюжетів. Бронювання може бути обмежене через популярність тематики [6].

1.3 Особливості та проблеми бронювання

Процес бронювання квест-кімнат залежить від їх типу та специфіки:

- Гнучкість часу. Деякі квести пропонують фіксовані часові слоти, інші — більш гнучкий графік.
- Кількість учасників. Обмеження за кількістю гравців можуть варіюватися залежно від розміру кімнати та складності завдань.
- Попередня підготовка. Деякі квести вимагають попереднього ознайомлення з правилами або проходження інструктажу.
- Додаткові послуги. Можливість замовлення додаткових послуг, таких як фотозйомка, подарункові сертифікати тощо.

У таблиці 1.1 наведено порівняння типів квест-кімнат та особливостей, які присутні під час бронювання.

На ринку доступні різноманітні рішення для онлайн-бронювання квестів – від універсальних платформ для бронювання послуг до спеціалізованих систем, розроблених саме під escape room. Розглянемо основні типи та їхні проблемні аспекти.[7-8]

Таблиця 1.1 – Порівняння типів квест-кімнат з особливостями бронювання

Тип квесту	Тривалість (орієнтовна)	Кількість гравців	Особливості бронювання
Класичний	60 хв	2–6	Стандартне бронювання за графіком
З акторами	60–90 хв	4–8	Потрібна попередня домовленість
VR-квест	30–60 хв	1–4	Перевірка наявності обладнання
Дитячий	45–60 хв	4–10	Врахування вікових обмежень
Тематичний	60 хв	2–6	Можливі обмеження через популярність теми або її специфічність

1.3.1 Універсальні SaaS-платформи для бронювання

Багато квест-кімнат користуються зовнішніми сервісами бронювання “як послуга” (Software-as-a-Service). Resova, Bookeo, FareHarbor, SimplyBook, EasyWeek – саме такі закордонні системи. Такі системи зазвичай працюють за підпискою або комісією: бізнес сплачує або щомісячний тариф, або відсоток від кожного бронювання. Головна перевага SaaS – швидкий старт (не треба самим розробляти сайт чи додаток) та наявність готового функціоналу (календар, повідомлення, оплати). [9]

Однак є кілька недоліків, такі як висока вартість або комісії, адже деякі платформи беруть значні комісійні з кожної транзакції. Наприклад, FareHarbor працює за комісійною моделлю ~6% з кожного бронювання (без фіксованої абонплати), що при великому обсязі продажів становить суттєву суму. Інші пропонують тарифні плани з лімітами: базові пакети Resova обмежують до 600 бронювань на місяць, далі потрібен дорожчий корпоративний план. Такі витрати не завжди підйомні для малого бізнесу.

Також, нерідко присутня зайва складність і «загальність». Універсальні рішення орієнтовані на широкий спектр індустрій (від салонів краси до спортзалів), тому інтерфейс і функції можуть бути недостатньо оптимізовані під специфіку квест-кімнат. Наприклад, Bookeo позиціонується як мультигалузевий інструмент, тому певні його опції зайві або незручні саме для сценарію квестів.

Інколи надлишковий функціонал ускладнює користування: за відгуками, власники квест-кімнат скаржаться на складність налаштування систем та перенасиченість інтерфейсу, що потребує часу на освоєння.

Варто зауважити, що присутні вимоги до мінімального масштабу бізнесу. Деякі провайдери орієнтовані на середній і великий бізнес. Наприклад, щоб стати клієнтом FareHarbor, річний дохід компанії має перевищувати 200 тис. євро. У цьому випадку дрібні провайдери не проходять поріг.

Присутній сервіс EasyWeek, що орієнтований на український ринок, який пропонує CRM з онлайн-бронюванням і має безкоштовний тариф для невеликих клієнтів (до 1 локації та 1 співробітника), що привабливо для стартапів. Але для росту бізнесу доведеться переходити на платні пакети, і наразі EasyWeek менше відомий у сфері квестів порівняно з глобальними гравцями.

Кожна з цих незручностей створює значні перешкоди для розвитку індустрії розваг, а саме ігрових квест-кімнат.

1.3.2 Власні веб-сайти з простими формами

Деякі підприємці, зокрема українські, ідуть шляхом розробки свого сайту з формою заявки. Це вирішує проблему контролю даних та унікального брендингу, але не автоматизує процес повністю: адміністратор все одно вручну підтверджує наявність, оновлює календар на сайті. Без інтеграції з календарем високий ризик, що декілька користувачів можуть одночасно надіслати заявку на один час. Таким рішенням часто бракує миттєвого підтвердження – клієнт відправив форму і чекає, поки з ним зв’яжуться, що знижує задоволеність, або навіть не гарантує підтвердження бажаного часу бронювання кімнати.

1.3.3 Мобільні застосунки-агрегатори

Існують спроби створити агрегатори, де користувач може переглядати і бронювати квести від різних компаній (наприклад, сервіс Morty у США).

Переваги – єдиний каталог, зручність для користувача, але бізнес-моделі таких агрегаторів теж зазвичай передбачають комісію з кожного бронювання або плату за розміщення. До того ж, на локальному рівні (місто або країна) агрегатори можуть не містити всіх провайдерів, і малі компанії не завжди готові приєднуватися.

З огляду на ці проблеми, багато операторів квест-кімнат шукають баланс між вартістю і функціональністю системи бронювання.

1.4 Потреби користувачів системи

Для підвищення ефективності взаємодії між клієнтами та адміністраторами квест-кімнат, необхідно врахувати ключові потреби обох сторін. Для клієнта потрібно реалізувати:

- можливість перегляду вільних кімнат і доступних слотів;
- зручний інтерфейс бронювання (без дзвінків);
- перегляд історії своїх бронювань;
- розрахунок вартості бронювання у реальному часі;
- можливість скасування бронювання.

Для адміністратора:

- створення, редагування та видалення кімнат.
- управління часовими слотами.
- перегляд та керування бронюваннями клієнтів.
- контроль над завантаженістю кімнат.

Додатково, адміністратор зберігає за собою весь функціонал, який доступний клієнтам.

1.5 Аналіз існуючих рішень (аналогів) на ринку України

На поточному етапі на ринку України доступно кілька веб-застосунків, які надають функціональність бронювання квест-кімнат. Було проаналізовано три приклади підприємств в Україні, які надають можливість приймати участь в іграх у квест-кімнатах: QIMNATA, ЗАМКНЕНІ та QuestRoom.

1.5.1 Аналог 1: QIMNATA

Розглянемо Аналог 1 – QIMNATA (рис. 1.1):

Назва: QIMNATA;

Архітектура: 3-tier web-application;

Інструменти реалізації: Не вказано, скоріш за все HTML, CSS, JavaScript для сайту та PHP/Java/Python для серверу. [10]

Перелік функцій:

- бронювання квест-кімнат;
- робота у реальному часі з актуальними даними;
- можливість обрати мову веб-сторінки;
- зворотній зв’язок.

Переваги:

- зручний доступ до вибору бажаної кімнати;
- можливість обрати англійську мову веб-сторінки;
- миттєве бронювання кімнат.

Недоліки:

- відсутність інтеграції з календарем;
- відсутність реєстрації користувачів;
- погана верстка веб-сторінки.

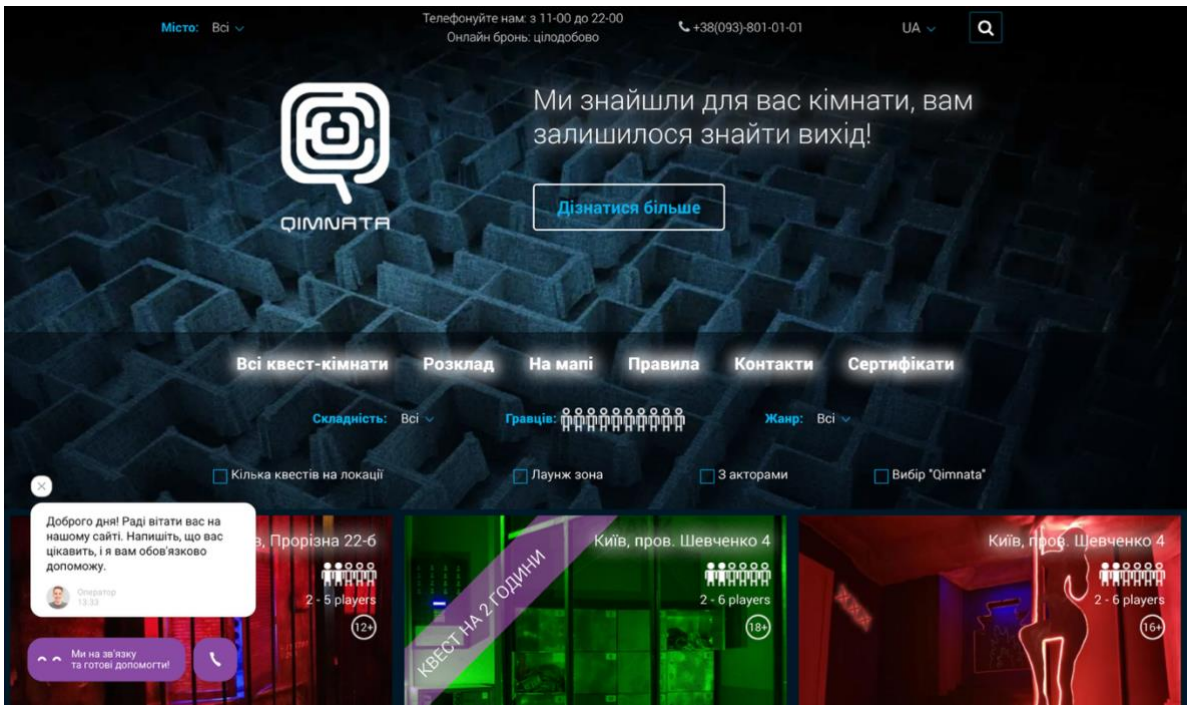


Рисунок 1.1 – Головна сторінка веб-сайту QIMNATA

1.5.2 Аналог 2: ЗАМКНЕНІ

Розглянемо Аналог 2 – ЗАМКНЕНІ (рис. 1.2):

Назва: ЗАМКНЕНІ.

Архітектура: 3-tier web-application.

Інструменти реалізації: Не вказано, скоріш за все HTML, CSS, JavaScript для сайту та PHP/Java/Python для серверу. [11]

Перелік функцій:

- бронювання квест-кімнат;
- робота у реальному часі з актуальними даними;
- можливість обрати мову веб-сторінки;
- зворотній зв’язок.

Переваги:

- зручний доступ до вибору кімнат;
- миттєве бронювання кімнат;
- вказано складність кожної з квест-кімнат;
- знижки на день народження;

- можливість обрати англійську мову веб-сторінки.

Недоліки:

- відсутність інтеграції з календарем;
- відсутність реєстрації користувачів;
- відсутність можливості розрахуватися на сайті.

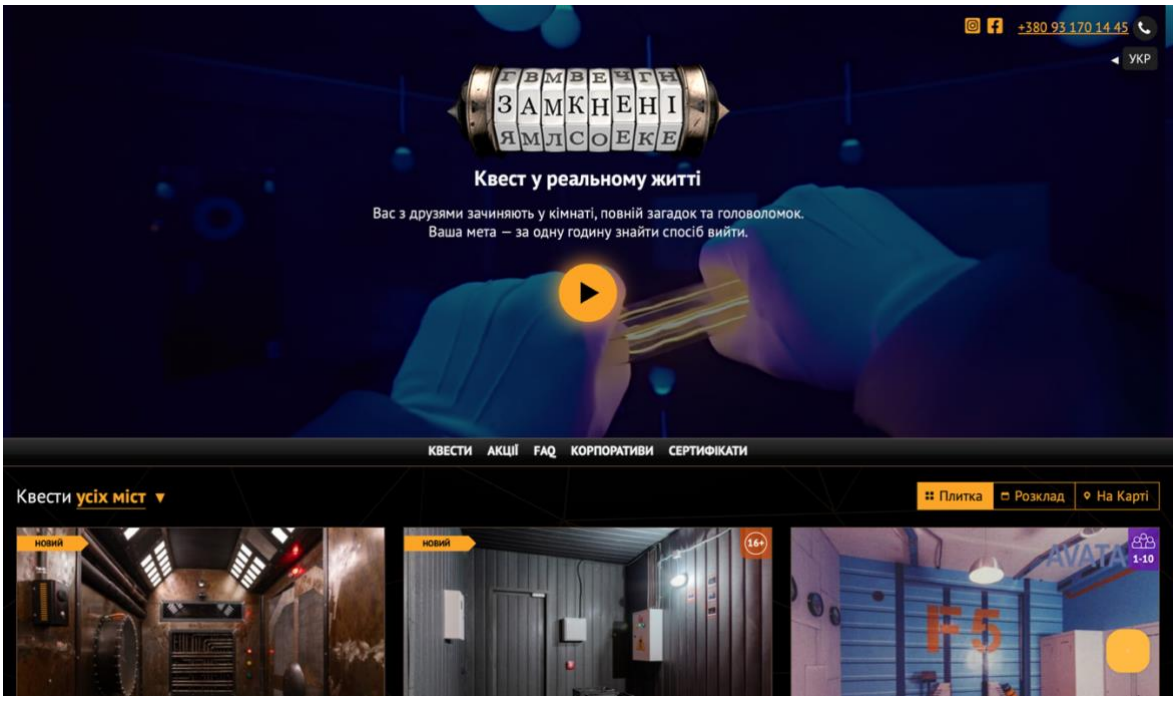


Рисунок 1.2 – Головна сторінка веб-сайту ЗАМКНЕНІ

1.5.3 Аналог 3: QuestRoom

Розглянемо Аналог 3 – QuestRoom (рис. 1.3):

Назва: QuestRoom;

Архітектура: 3-tier web-application;

Мова реалізації: Не вказано, скоріш за все HTML, CSS, JavaScript для сайту та PHP/Java/Python для серверу. [12]

Перелік функцій:

- бронювання квест-кімнат;
- робота у реальному часі з актуальними даними;
- можливість оплати бронювання;

- реєстрація та автентифікація користувача;
- можливість обрати мову веб-сторінки;
- зворотній зв’язок.

Переваги:

- зручний доступ до вибору кімнат;
- присутність оплати картами Visa та Mastercard.
- миттєве бронювання кімнат;
- вказано складність кожної з квест-кімнат;
- можливість обрати англійську мову веб-сторінки;
- знижки на день народження.

Недоліки:

- відсутність інтеграції з календарем;
- відсутність можливості розрахуватися на сайті.

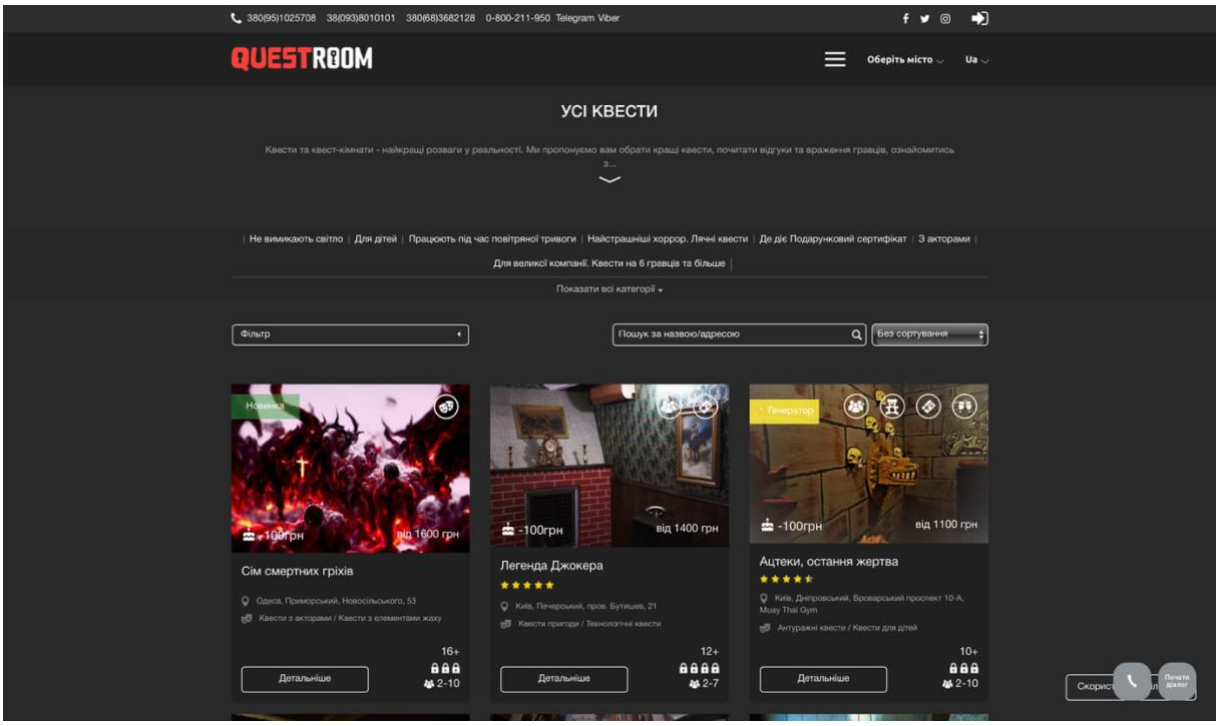


Рисунок 1.3 – Головна сторінка веб-сайту QuestRoom

1.5.4 Визначення ключових критеріїв

Для об’єктивного порівняння функціональних можливостей аналогічних систем було визначено перелік ключових критеріїв, які безпосередньо впливають на зручність використання, ефективність адміністрування та потенціал масштабування майбутнього мобільного застосунку. До таких критеріїв належать:

– архітектура системи, що визначає її технічну побудову та принципи взаємодії клієнта з сервером;

– автентифікація користувачів – необхідна для персоналізації доступу й забезпечення безпеки;

– інтеграція з календарем – дозволяє уникнути накладок у бронюваннях і покращує взаємодію з мобільними пристроями;

– онлайн-оплата – суттєво підвищує зручність користувача та конверсію;

– миттєве бронювання – забезпечує оперативність і зменшує навантаження на персонал;

– реєстрація користувачів – дозволяє зберігати історію взаємодії та формувати аналітику;

– знижки/акції – підвищують лояльність клієнтів;

– відображення складності квестів – спрощує вибір для користувача;

– багатомовність – важлива для охоплення ширшої аудиторії, включно з туристами;

– зворотній зв’язок – дозволяє отримувати відгуки та покращувати сервіс;

– підтримка мобільних платформ – критично важлива для сучасного користувача, який переважно використовує смартфон;

– якість верстки – впливає на досвід використання користувачем, адаптивність і загальне сприйняття інтерфейсу;

Зазначені критерії дозволяють не лише зіставити функціональні можливості систем, а й чітко визначити, в чому майбутній мобільний застосунок перевершуватиме наявні аналоги.

1.6 Обґрунтування необхідності розробки мобільного застосунку

У сучасних умовах розвиток цифрових рішень для малого та середнього бізнесу є не лише бажаним, а й необхідним. Ринок мобільних застосунків зростає стрімкими темпами: згідно з App Annie [13], кількість завантажень мобільних застосунків перевищила 250 мільярдів на рік, що свідчить про зростання довіри користувачів до мобільних платформ.

Користувачі очікують швидкої та безперебійної взаємодії з бізнесом – без необхідності телефонних дзвінків або ручного підтвердження. У контексті квест-кімнат мобільний застосунок дає можливість суттєво покращити користувацький досвід, надаючи клієнтам зручний інтерфейс для перегляду вільних слотів, вибору часу та оформлення бронювання за кілька секунд.

З боку адміністратора застосунок дозволяє:

- централізовано керувати квест-кімнатами та доступними слотами;
- оперативно переглядати бронювання клієнтів;
- вносити зміни до розкладу в реальному часі;
- контролювати заповнюваність кімнат і проводити аналітику.

У порівнянні з класичним підходом, наприклад: телефонне бронювання, Excel-таблиці або неадаптовані веб-форми), мобільний застосунок забезпечує наступні переваги наведені в таблиці 1.2.

Вибір розробки саме мобільного застосунку обумовлений ще й тим, що:

– понад 75% користувачів інтернету в Україні заходять у мережу саме через смартфон [14];

– нативні застосунки дозволяють використовувати можливості пристрою (push-сповіщення, геолокацію, камеру).

Таблиця 1.2 – Порівняння класичного підходу з мобільним застосунком

Критерій	Класичне рішення	Мобільний застосунок
Доступність	Лише в робочі години	24/7 з будь-якого пристрою
Швидкість оформлення	Залежить від адміністратора	Декілька кліків
Надійність даних	Часто ручне введення	Верифікація та контроль конфліктів
UX/інтерфейс	Несистемна взаємодія	Адаптивний дизайн, зручна навігація
Можливість масштабування	Обмежена	Висока (додавання філій, кімнат тощо)

Враховуючи наведені фактори, можна зробити висновок, що розробка мобільного застосунку є доцільним і ефективним способом цифровізації процесу бронювання квест-кімнат. Це дозволить оптимізувати як внутрішні процеси компанії, так і зовнішню взаємодію з клієнтом.

1.7 Аналіз відповідно до визначених ключових критеріїв

Розглянуті аналоги систем бронювання квест-кімнат (QIMNATA, ЗАМКНЕНІ та QuestRoom) реалізують базову функціональність бронювання квест-кімнат через веб-інтерфейси. Проте всі вони мають спільні системні обмеження:

- відсутність нативного мобільного застосунку, що обмежує зручність користувача, особливо для мобільної аудиторії.
- нереалізована гнучка рольова модель, що не дозволяє адміністратору гнучко керувати контентом, а клієнтам – отримувати персоналізований функціонал.

- відсутність адаптації інтерфейсу під IOS та використання специфічних можливостей платформи (наприклад, push-сповіщень, інтеграції з календарем, appstorage).

- не підтримується динамічне управління часовими слотами, що робить систему вразливою до людських помилок або дублювання бронювань.

Більш деталізоване порівняння аналогів було наведено в таблиці 1.3.

Таблиця 1.3 – Порівняння існуючих рішень на ринку відповідно до ключових критеріїв

Критерій	**QIMNATA**	**ЗАМКНЕНІ**	**QuestRoom**
Архітектура	3-tier web-application	3-tier web-application	3-tier web-application
Автентифікація користувачів	–	–	+
Інтеграція з календарем	–	–	–
Оплата онлайн	–	–	Частково (Visa / MasterCard)
Миттєве бронювання	+	+	+
Реєстрація користувачів	–	–	+
Знижки / акції	–	+ (на день народження)	+ (на день народження)
Вказано складність кімнат	–	+	+
Багатомовність	UA, EN	UA, EN, RU	UA, EN, RU
Зворотній зв’язок	+	+	+
Підтримка мобільних платформ	– (веб)	– (веб)	– (веб),
Якість верстки	Низька	Середня	Середня
Посилання	qimnata.com	vzaperti.com.ua	questroom.com.ua

1.8 Висновки до розділу 1

У результаті проведеного аналізу предметної області було встановлено, що ринок квест-кімнат в Україні активно розвивається та потребує впровадження сучасних цифрових рішень для оптимізації процесу взаємодії з клієнтами. Системи бронювання, що використовуються на практиці, здебільшого є ручними або лише частково автоматизованими, що створює ризики помилок, перевантаження персоналу та зниження рівня обслуговування.

Було визначено ключові потреби обох груп користувачів – клієнтів і адміністраторів – які повинні враховуватись при розробці інформаційної системи. Зокрема, йдеться про швидкий доступ до вільних слотів, зручне бронювання, керування кімнатами та можливість аналізу заповнюваності.

Обґрунтовано доцільність створення мобільного застосунку як оптимального інструменту цифровізації процесів бронювання. Порівняння з традиційними методами підтвердило значну перевагу мобільних рішень за такими критеріями, як доступність, надійність, масштабованість та зручність користувача.

Таким чином, розробка мобільного застосунку для бронювання квест-кімнат є актуальним завданням, що відповідає потребам ринку та створює основу для подальших досліджень і практичної реалізації в межах кваліфікаційного проєкту.

2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ КВЕСТ-КІМНАТ

2.1 Постановка задачі

У межах поставленого завдання було поставлено задачу розробити мобільний застосунок для бронювання квест-кімнат. Після проведеного аналіз предметної області зрозуміло, що застосунок повинен забезпечити:

- авторизацію користувачів за ролями (адміністратор / клієнт);
- можливість створення, редагування та перегляду квест-кімнат;
- бронювання доступних слотів у режимі реального часу;
- захист від дублювання;
- інтуїтивно зрозумілий адаптивний інтерфейс.

Також, для вирішення цієї задачі необхідно провести огляд існуючих аналогічних рішень, визначити критерії оцінювання ефективності, а також обґрунтувати вибір технологій та архітектури.

Приступаючи до реалізації, необхідно визначитися з технологічним стеком і архітектурою програмного забезпечення. Вибір буде базуватися на специфіці завдання (мобільний застосунок для кінцевих користувачів + бекенд, який використовується для зберігання даних та їх підготовкою для клієнтів), а також на сучасних тенденціях розробки.

Відповідно до проведеного аналізу в попередніх розділах роботи можна сформувати наступні чіткі функціональні вимоги і надати їм критерій важливості (таблиця 2.1).

Таблиця 2.1 – Функціональні вимоги до мобільного застосунку

№	Функція	Важливість	Роль користувача	Коментар / Примітка
1	Автентифікація та реєстрація	Обов’язкова	Усі	Через Firebase, з перевіркою UID і збереженням ролі
2	Рольова модель доступу	Обов’язкова	Усі	Доступ до функцій залежно від ролі (admin/client)
3	Перегляд списку квест-кімнат	Обов’язкова	Усі	Фільтрація за локацією, складністю
4	Перегляд деталей кімнати	Обов’язкова	Усі	З описом, зображеннями, рівнем складності
5	Перегляд доступних слотів за обраною датою	Обов’язкова	Усі	Динамічна підвантаження слотів з сервера
6	Створення нового бронювання	Обов’язкова	Клієнт	Вибір кількості учасників, автоматичний розрахунок вартості
7	Перегляд списку своїх бронювань	Обов’язкова	Клієнт	Сортування за датою
8	Скасування або редагування бронювання	Обов’язкова	Клієнт, Адміністратор	Можливість змінити слот або кількість гравців
9	Створення нової квест-кімнати	Обов’язкова	Адміністратор	З додаванням текстових даних та зображень
10	Редагування/видалення кімнати	Обов’язкова	Адміністратор	CRUD-операції з перевіркою прав
11	Завантаження зображень кімнат	Обов’язкова	Адміністратор	–
12	Перегляд користувачів	Бажана	Адміністратор	Для управління ролями та перегляду даних
13	Зміна ролі користувача	Бажана	Адміністратор	Наприклад, підвищення до адміністратора
14	Push-сповіщення про бронювання	Бажана	Усі	–

Продовження таблиці 2.1

15	Інтеграція з календарем	Бажана	Усі	Автоматичне додавання нагадувань
16	Зміна особистих даних	Обов’язкова	Усі	Ім’я, телефон, дата народження

2.2 Вибір технологій для виконання поставленої задачі

2.2.1 Вибір платформи розробки

Розглянемо, чи доцільно створювати крос-платформний додаток (який працював би і на iOS, і на Android) чи зосередитися на нативній розробці під одну платформу. Крос-платформні фреймворки (Flutter, React Native тощо) дозволяють з однієї кодової бази отримати додатки для різних ОС, що потенційно розширює аудиторію. Наприклад, Flutter успішно застосовується для подібних завдань і має високий рівень продуктивності та гарний UI. Однак у нашому випадку було вирішено реалізовувати проєкт як нативний iOS додаток з таких міркувань:

– робота з мовою програмування Swift, яка є досить легкою в опануванні та нативною для розробки на пристрої, які працють на системах IOS, iPadOS, macOS та інші з екосистеми компанії Apple. [15]

– продуктивність та плавність UI на високому рівні при нативній розробці.

– нативний застосунок краще інтегрується з можливостями платформи: push-сповіщення, Apple ID Auth, Apple Pay для оплати тощо – усе це можна реалізувати “з коробки”.

– сервер, який обробляє запити клієнтського застосунку, також розроблений на мові програмування Swift з використанням фреймворку Vapor.

Отже, платформою обрано Apple IOS, а основною мовою програмування – Swift, яка є сучасною мовою від Apple для розробки під iOS/macOS.

2.2.2 Вибір фреймворку інтерфейсу

Apple у 2019 році представила революційний UI-фреймворк SwiftUI, що спрощує і прискорює створення інтерфейсів завдяки декларативному підходу. SwiftUI дозволяє розробнику описувати інтерфейс у вигляді станів і реактивно оновлювати його при зміні даних. Це значно скорочує код і зменшує ймовірність помилок у логіці відображення. За словами Apple, SwiftUI “робить побудову потужних інтерфейсів простішим, ніж будь-коли” і автоматизує багато аспектів (адаптивна верстка, підтримка темної теми, локалізація тощо).

Даний підхід прекрасно підходить для розробки застосунку, оскільки інтерфейс включає динамічні списки (список квест-кімнат, розклад слотів) та форми введення, адже SwiftUI забезпечує їхнє легке двобічне зв’язування зі станом (даними). Також важливо, що SwiftUI є спільним фреймворком для iOS, iPadOS, macOS – тобто в майбутньому можна відносно легко адаптувати застосунок для iPad або навіть створити настільну версію, користуючись тією ж кодовою базою. Крім того, SwiftUI заохочує використання сучасних архітектурних патернів. [16]

Також, він залишає підтримку для попереднього фреймворку UIKit [17], де не лише можна додавати елементи до наявних представлень на SwiftUI, а й повністю збирати представлення з UIKit елементів.

2.2.3 Вибір архітектурного патерну

Для структурування внутрішньої логіки програми обрано патерн MVVM, який чудово поєднується зі SwiftUI. Ідея MVVM – розділити відповідальність між: модель (дані предметної області), view (відображення/інтерфейс) та viewModel (модель представлення, що містить бізнес-логіку і зв’язує модель з view) . У SwiftUI фактично відсутній класичний контролер (як в MVC), тому роль посередника виконує ViewModel, а View бере на себе тільки відображення та обробку базових подій UI. Переваги MVVM для реалізації проєкту:

– підтримуваність: view не містить бізнес-логіки, вона зосереджена у ViewModel. Це робить код більш організованим і простим для змін. При зростанні додатку (додаванні нових функцій) легше масштабувати, маючи чітко розділені компоненти.

– зручність тестування: оскільки основна логіка знаходиться у ViewModel і не залежить від UI, її можна покривати модульними тестами ізольовано (імітувати модель і перевіряти вихідні дані) . Це підвищує надійність коду.

– реактивність: SwiftUI прив’язує View до ViewModel через дані-стани (property wrappers @State, @ObservedObject тощо). Це значить, що зміна стану у ViewModel автоматично оновить View. Таким чином MVVM дуже природно реалізується – ViewModel виступає “джерелом правди” для інтерфейсу.

MVVM вже став де-факто стандартом при розробці зі SwiftUI , бо сприяє чистій архітектурі. У проєкті буде реалізовано для кожного основного екрану відповідний view model, який інкапсулює всю логіку. [18]

2.2.4 Планування взаємодії з хмарним бекендом

Клієнтська частина застосунку взаємодіє із серверною, реалізованою за допомогою Vapor – сучасного серверного фреймворку на мові Swift. Цей фреймворк забезпечує створення повноцінного RESTful API, що дозволяє ефективно обробляти запити від мобільного застосунку, керувати структурою бази даних, реалізувати розподіл доступу за ролями та забезпечити масштабованість рішення. [19-20]

У якості системи керування базою даних використовується PostgreSQL [], з підключенням через ORM Fluent, що дозволяє описувати моделі, зв’язки та міграції на рівні коду Swift без необхідності писати сирі SQL-запити. [21-22]

Для автентифікації користувачів інтегровано Firebase Authentication. Кожен запит до захищених маршрутів проходить перевірку за допомогою middleware, яке валідовує ID токен, отримує UID користувача з Firebase, а далі –

витягує деталі користувача з локальної бази PostgreSQL, зокрема його роль (client або admin). Це забезпечує надійне розмежування прав доступу на рівні API. [23]

Серверна логіка реалізує такі функції:

– створення, редагування та видалення квест-кімнат;

– керування слотами для бронювання;

– обробку заявок на бронювання із врахуванням ролей;

– скасування бронювань;

– збереження та обробку зображень кімнат – як у вигляді посилань на хмарні сховища, так і у вигляді локальних файлів у Docker-томах або S3-сумісних сервісах.

Увесь серверний стек працює у контейнеризованому середовищі Docker, що забезпечує ізоляцію компонентів, стабільність розгортання і зручність у супроводі. Компоненти – база даних, API-сервер та інші сервіси – запускаються в межах Docker Compose, що дозволяє легко масштабувати проєкт, додавати нові сервіси (аналітика, черги завдань, пошук тощо) та керувати всією інфраструктурою централізовано.

2.2.5 Підсумок вибору технологій

Для реалізації мобільного застосунку було обрано сучасний та узгоджений між собою стек технологій, що забезпечує високу продуктивність, гнучкість у розробці, масштабованість і зручність у подальшому супроводі.

Мобільний клієнт: клієнтська частина реалізується під платформу iOS з використанням мови програмування Swift та фреймворку SwiftUI. Визначено застосування архітектурного патерну MVVM (Model–View–ViewModel), який дозволяє чітко розділити відповідальність між відображенням інтерфейсу, бізнес-логікою та моделлю даних. Такий підхід забезпечує легкість супроводу, спрощує тестування та дає змогу реалізувати реактивну поведінку інтерфейсу через механізми спостереження (@State, @ObservedObject, @Published).

Серверна частина та база даних: у якості бекенду використовується власний сервер на базі фреймворку Vapor (Swift) у поєднанні з базою даних PostgreSQL. Це рішення дозволяє реалізувати повноцінне REST API для взаємодії з клієнтським застосунком, підтримує ролі користувачів (адміністратор / клієнт), забезпечує CRUD-функціонал для кімнат, слотів та бронювань. Сервер працює в Docker-контейнерах, що дозволяє легко розгортати систему локально або в хмарному середовищі. Для ідентифікації користувачів інтегровано Firebase Authentication, що дає змогу використовувати надійні механізми авторизації (email/password, Apple ID, Google тощо). Сервер перевіряє дійсність Firebase ID-токена через middleware, а потім асоціює його з локальним користувачем у PostgreSQL для визначення ролі та доступу.

Отже, технологічний стек з яким відбувається робота забезпечує оптимальний баланс між швидкістю розробки, функціональністю, контролем над інфраструктурою та можливістю подальшого масштабування. Він підходить для забезпечення належного виконання функціональних вимог загаданих у таблиці 2.1. Використання нативного iOS-інтерфейсу, який працює разом з потужним серверним фреймворком Vapor, надійною базою PostgreSQL і перевіреною автентифікацію від Firebase, що робить розроблювану систему гнучкою, надійною та зручною як для кінцевого користувача, так і для адміністратора.

2.3 Роробка архітектури клієнтського застосунку

Клієнтську частину мобільного застосунку реалізовано на платформі iOS з використанням мови Swift і фреймворку SwiftUI. Архітектурна модель MVVM (Model–View–ViewModel) забезпечує поділ відповідальності між UI, логікою обробки даних та структурами. Це дозволяє досягти високої модульності, зменшення зв’язності та легшого тестування.

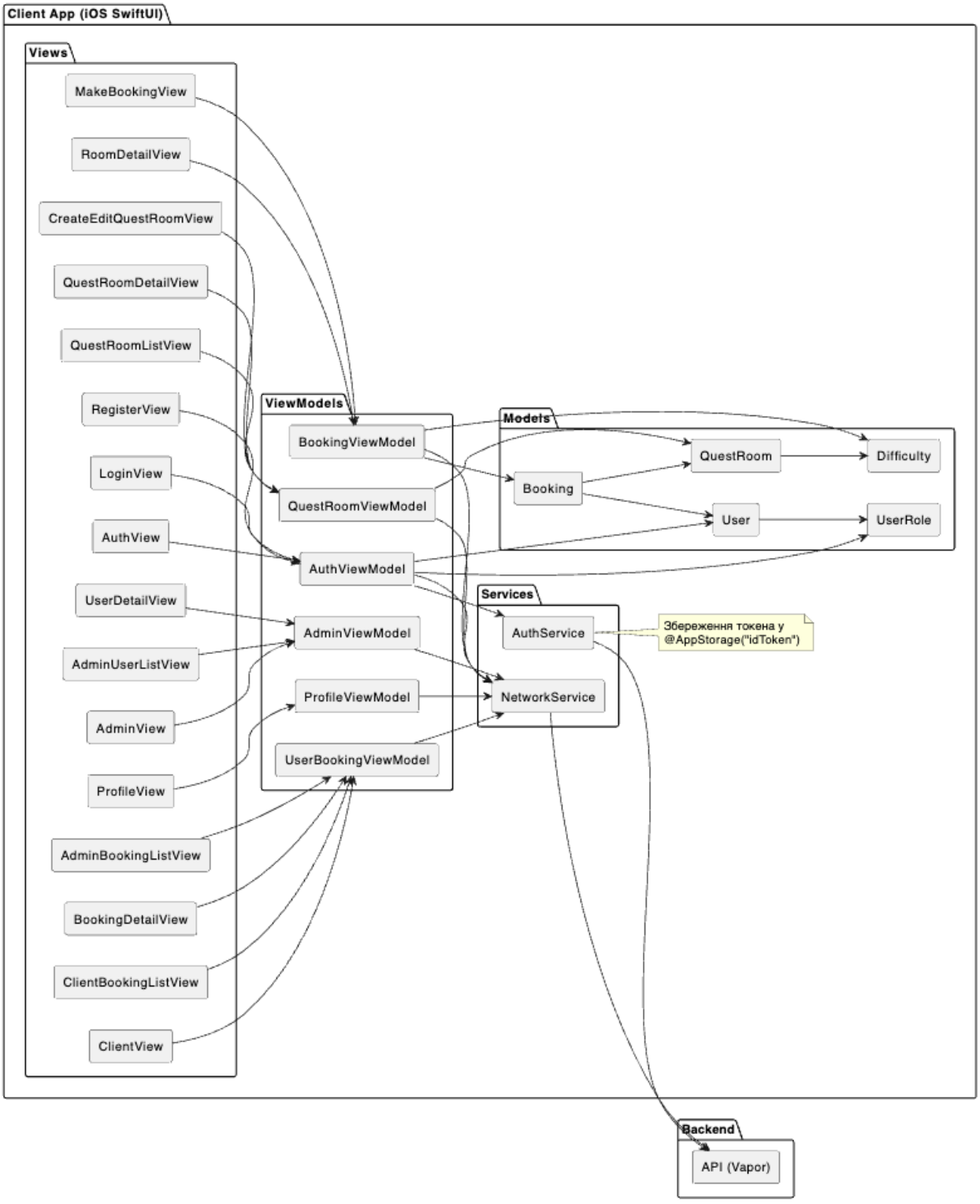


Рисунок 2.1 – Архітектурна структуру клієнтської частини

Модулі клієнтського застосунку:

- Models – описують основні структури даних: User, QuestRoom, Booking, Difficulty, UserRole. Всі підтримують Codable та Identifiable. Ці моделі представляють дані, отримані з API або локально збережені.

- ViewModels – реалізують логіку взаємодії інтерфейсу з моделями. Кожен ViewModel (AuthViewModel, QuestRoomViewModel, BookingViewModel) має властивості з @Published для реактивної прив’язки до SwiftUI.
- Views – усі екрани (SwiftUI-компоненти) відображаються на основі даних з ViewModel. Наприклад: LoginView, QuestRoomListView, MakeBookingView.
- Services – інкапсулюють взаємодію з сервером через HTTP-запити (NetworkService) або обробку авторизації (AuthService). Токен авторизації зберігається в @AppStorage("idToken").

На рисунку 2.1 зображено архітектурну структуру клієнтської частини у вигляді компонентної діаграми.

2.4 UML-діаграми клієнтської частини

UML-діаграми дають змогу наочно представити архітектуру, функціональність і взаємодію компонентів клієнтського застосунку. У даному підрозділі розглянуто три основні типи діаграм: Use Case, Class Diagram і Component Diagram.

2.4.1 Component Diagram

На рисунку 2.2 подано компонентну діаграму архітектури застосунку відповідно до моделі MVVM.

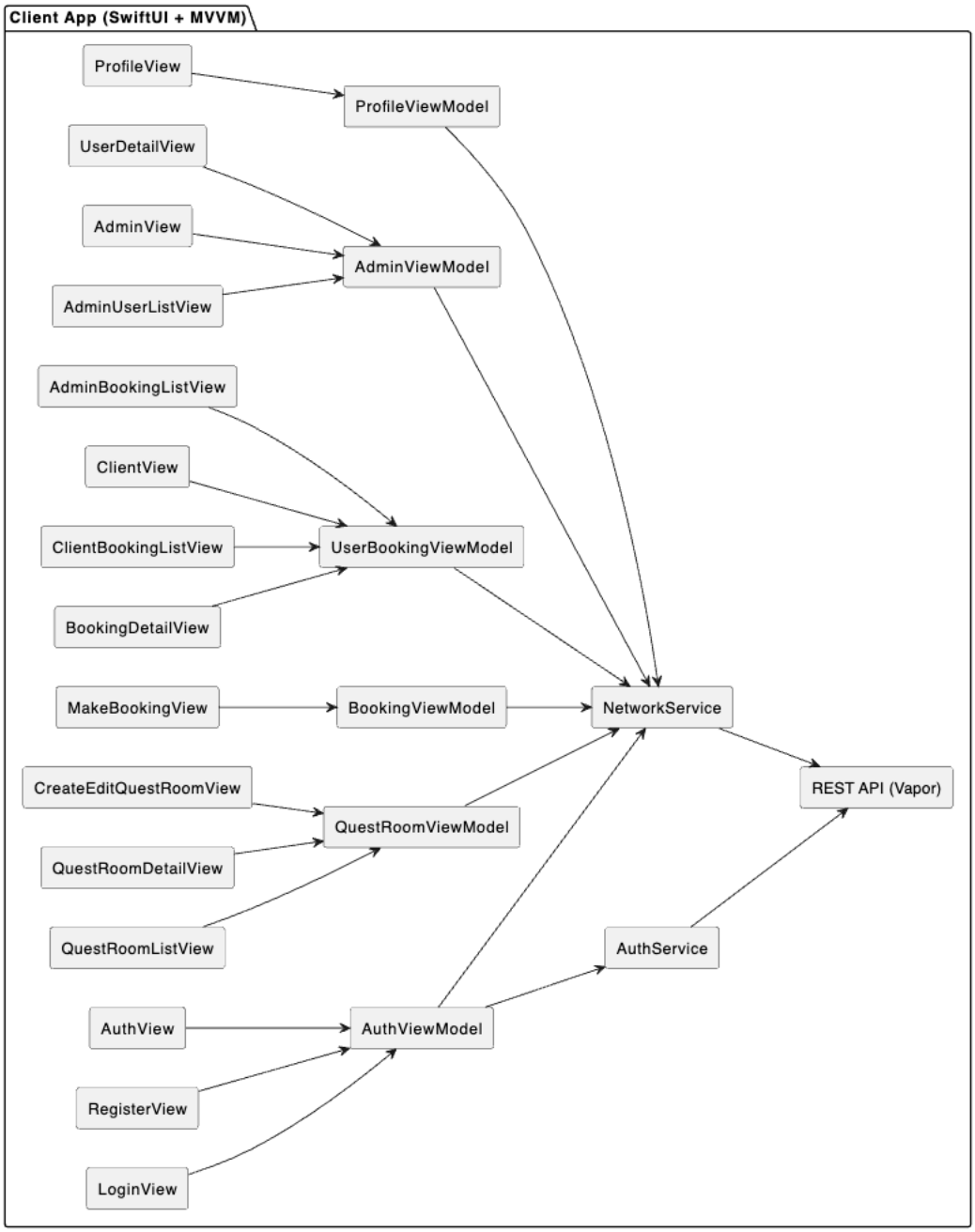


Рисунок 2.2 – Діаграма компонентів

Нижче наведені всі відповідні елементи:

- Views: LoginView, RegisterView, QuestRoomListView, QuestRoomDetailView, MakeBookingView, CreateEditQuestRoomView, AdminView, ClientView, ProfileView
- ViewModels: AuthViewModel, BookingViewModel, QuestRoomViewModel, AdminViewModel, ProfileViewModel, UserBookingViewModel

- Services: AuthService, NetworkService

Компоненти Views взаємодіють лише з ViewModel, які інкапсулюють бізнес-логіку і викликають відповідні сервіси для обміну даними з сервером. Це забезпечує слабке зв'язування і спрощує тестування та підтримку застосунку.

2.4.3 Class Diagram

На рисунку 2.3 зображено класову діаграму клієнтської частини. Вона включає основні моделі:

- User (id, name, email, role, phoneNumber, birthday, firebaseUID)
- QuestRoom (id, title, description, location, difficulty, maxPlayers, maxPrice, imageURLs)
- Booking (id, date, timeSlot, numberOfPlayers, price, user, questRoom)

Кожна з моделей реалізує протоколи Codable та Identifiable, що забезпечує зручну роботу з API та SwiftUI. Між об'єктами Booking і QuestRoom/User існують асоціації, які відображають реальні зв'язки в предметній області.

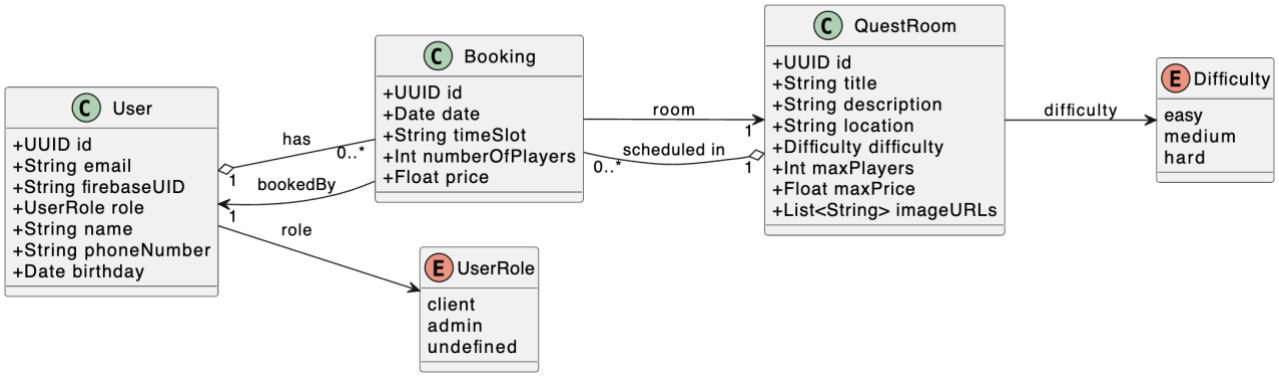


Рисунок 2.3 – Діаграма класів

2.4.2 Use Case-діаграма

На рисунку 2.4 зображено UML-діаграму прецедентів використання системи. Вона ілюструє взаємодію основних акторів (Client та Admin) із

застосунком. Кожен прецедент відображає ключову функціональність, доступну для відповідної ролі:

- Client: перегляд списку кімнат, перегляд деталей кімнати, створення бронювання, перегляд особистих бронювань.
- Admin: керування кімнатами (створення, редагування, видалення), перегляд усіх бронювань, перегляд користувачів. Також, залишає за собою всі можливості ролі Client.

Ця діаграма дозволяє визначити функціональні вимоги до системи з погляду користувача.

Рисунок 2.4 – Діаграму прецедентів використання системи

Таким чином, UML-діаграми є важливим елементом технічної документації, що дозволяє візуалізувати структуру, функціональність і архітектуру клієнтського застосунку у чіткій і зрозумілій формі.

2.5 Висновки до розділу 2

В другому розділі було розроблено архітектуру клієнтської частини застосунку з урахуванням ролей користувачів (клієнт / адміністратор), спроєктувано структуру взаємодії з серверною частиною та побудовано UML-діаграми клієнтської частини.

3 РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ КВЕСТ-КІМНАТ

3.1 Реалізація основного функціоналу

Клієнтський застосунок реалізує повний цикл взаємодії користувача з системою бронювання квест-кімнат: авторизацію, реєстрацію, вибір ролі, перегляд кімнат, створення і перегляд бронювань, а для адміністратора – ще й управління контентом. Уся логіка побудована на архітектурі MVVM, де “View” відповідає за відображення інтерфейсу, “ViewModel” – за бізнес-логіку, а “Service” – за взаємодію з API.

3.1.1 Автентифікація, реєстрація та збереження токена

Вхід користувача реалізовано через екран “LoginView”, який передає введені дані (email, пароль) до “AuthViewModel”, що в свою чергу викликає метод “signIn(...)” у “AuthService”. У разі успішної відповіді сервер повертає ID-токен, роль користувача та іншу інформацію, яка зберігається в базі даних. Данні про токен на роль зберігаються та поширюються між виглядами застосунку за допомогою “@AppStorage("idToken")” і “@AppStorage("userRole")”, що дозволяє автоматично підтримувати стан авторизації між сесіями і адаптувати кожен вигляд під дійсного авторизованого користувача. Якщо токен дійсний, застосунок автоматично показує головний екран без повторного входу. Крім того, токен передається як Bearer-заголовок при кожному запиті до API.

Реєстрація реалізована у “RegisterView”, де користувач вводить ім’я, email, пароль, номер телефону та дату народження. Ці дані передаються до “AuthViewModel”, який викликає метод “signUp(...)” у “AuthService”. У разі успіху користувач одразу авторизується, отримуючи токен і роль, як і при звичайному вході. Реєстрація можлива лише з роллю клієнта.

3.1.2 Система ролей

Кожен користувач має роль: "client" або "admin". Під час входу сервер визначає роль і повертає її у відповіді, після чого вона зберігається у “@AppStorage("userRole")”. Від цієї ролі залежить логіка навігації та доступ до функцій: клієнт бачить лише загальнодоступний функціонал (бронювання, профіль), а адміністратор – додаткові панелі керування (керування кімнатами, користувачами, всіма бронюваннями). Усі перевірки ролі здійснюються як на рівні View (“if role == "admin"“), так і на рівні логіки (“guard role == "admin" else { return }”).

3.1.3 Перегляд квест-кімнат

Після входу користувач потрапляє на “QuestRoomListView”, де відображено всі доступні квест-кімнати. Завантаження даних здійснюється через “QuestRoomViewModel”, який викликає метод “fetchQuestRooms()” у “NetworkService”. Отримані кімнати зберігаються у властивості “@Published var questRooms: [QuestRoom]”, що автоматично оновлює UI. Кожна картка кімнати містить загальну інформацію: назву, зображення, рівень складності, ціну, кількість гравців та локацію. Пошук за назвою та фільтрація за містом реалізовані через функції ViewModel і окремі елементи керування у View.

3.1.4 Перегляд деталей і створення бронювання

При натисканні на кімнату відкривається “QuestRoomDetailView”, де виводиться повний опис, галерея зображень, складність, ціна та кнопка "Забронювати". Після її натискання користувач потрапляє у “MakeBookingView”, де за допомогою “BookingViewModel”:

1) обирає дату (“DatePicker”),

2) отримує доступні слоти через “fetchAvailableSlots(...)”,

3) вводить кількість гравців,

4) натискає кнопку для створення бронювання через “createBooking(...)”.

ViewModel перевіряє, чи слот доступний, та виконує POST-запит. У разі успіху з’являється підтвердження, а слот стає зайнятим.

3.1.5 Перегляд бронювань

Користувачі можуть переглядати свої бронювання у “ClientBookingListView”, який використовує “UserBookingViewModel” та метод “loadUserBookings(...)”. Кожна бронь включає дату, час, назву кімнати та статус. У разі потреби користувач може скасувати бронювання або змінити слот. Для адміністратора реалізовано окремий екран “AdminBookingListView”, який отримує всі бронювання (за допомогою “fetchAllBookings(...)”) і дозволяє переглядати їх по кімнатах та користувачах.

3.1.6 Адміністративні функції

Користувач із роллю адміністратора має доступ до “AdminView”, де:

- може створювати нові кімнати (“CreateEditQuestRoomView”),
- редагувати існуючі кімнати (префілінг даних),
- видаляти кімнати (з підтвердженням),
- переглядати список користувачів (“AdminUserListView”) та змінювати їхні ролі (“promoteUser(...)”).

Використовується “QuestRoomViewModel” для CRUD-операцій над кімнатами, а всі запити надсилаються через “NetworkService”. Для завантаження/вивантаження зображень реалізовано окрему функцію “uploadImage(...)”, яка передає зображення на сервер і повертає відносне посилання.

Таким чином, реалізація функціоналу охоплює всі основні сценарії – від реєстрації й входу до керування користувачами – з чітким розділенням відповідальностей між компонентами, реактивністю інтерфейсу та відповідністю сучасним практикам SwiftUI-розробки.

3.2 Користувацький інтерфейс (UI)

Інтерфейс клієнтського застосунку розроблено на основі SwiftUI з використанням декларативного підходу, який дозволяє швидко та зручно створювати динамічні екрани з підтримкою реактивного оновлення стану. Усі представлення (Views) максимально розділено відповідно до функціональних блоків системи.

Інтерфейс охоплює такі основні загальні екрани:

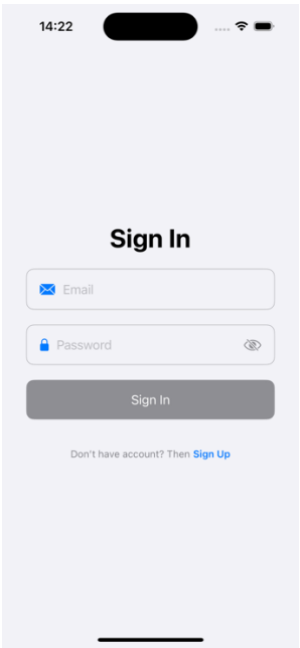


Рисунок 3.1 – LoginView

- LoginView – вхід у систему, поля для email і пароля, кнопка входу та навігація до реєстрації (рис. 3.1).

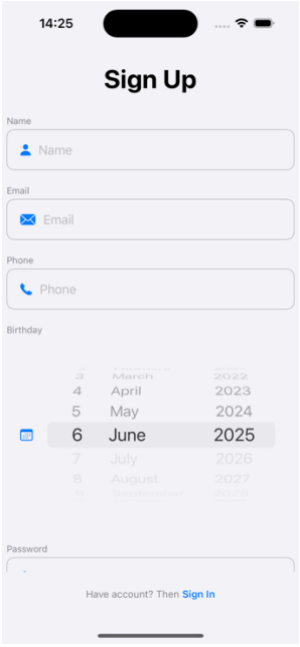


Рисунок 3.2 – RegisterView

- RegisterView – форма створення облікового запису з розширеним переліком полів: ім’я, email, телефон, день народження, пароль (рис. 3.2).

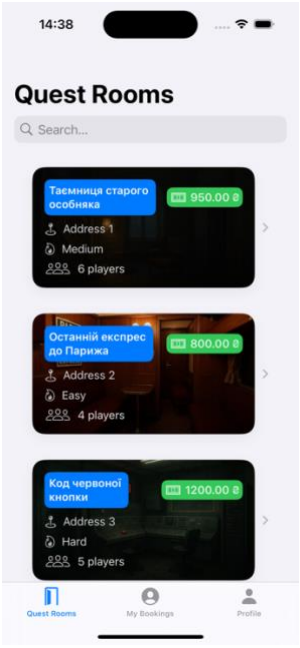


Рисунок 3.3 – QuestRoomListView

- QuestRoomListView – головний екран зі списком квест-кімнат, фільтрація за містом, підтримка пошуку та прокручування (рис. 3.3).

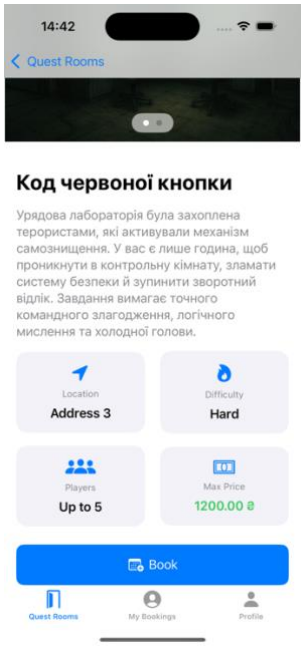


Рисунок 3.4 – QuestRoomDetailView

- QuestRoomDetailView – деталізація кімнати: фото, опис, складність, максимальна вартість та кількість учасників, кнопка «Забронювати» (Рисунок 3.4).

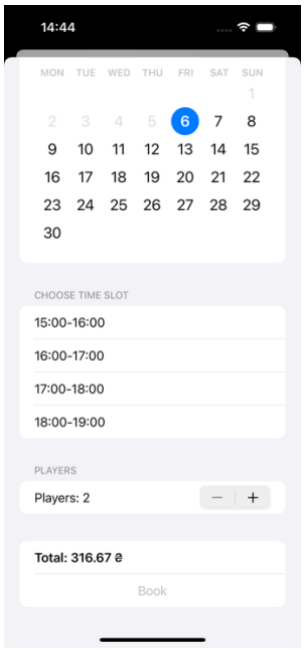


Рисунок 3.5 – MakeBookingView

- MakeBookingView – екран створення бронювання: вибір дати, слоту, кількості учасників, кнопка підтвердження (рис. 3.5). Після натискання користувач отримує сповіщення про результат бронювання у вигляді Alert.
- ProfileView – перегляд і редагування особистої інформації користувача (рис. 3.6).

Далі, перейдемо до екранів, які вже відображатимуться по різному для користувачів з ролями Адміністратор та Клієнт. Виконується це з метою збільшення зручності використання графічним інтерфейсом і обмеженням доступу до певних функції.

Наприклад, користувач з роллю Клієнт не має можливості побачити інструменти додавання або редагування кімнат і, також, можливості переглядати інформацію інших користувачів системи. Повертаючись до адміністратора: йому доступні всі інструменти звичайного користувача, але у видозміненій формі.

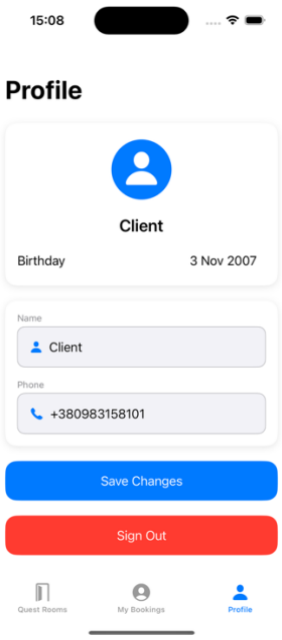


Рисунок 3.6 – ProfileView

Розглянемо детальніше специфіковані екрани:

- ClientBookingListView – списки бронювань клієнта, поділяються на минулі бронювання та майбутні (рис. 3.7).

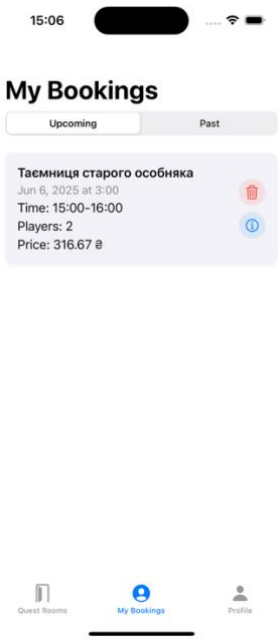


Рисунок 3.7 – ClientBookingListView

- AdminBookingListView – списки бронювань адміністратора, відображення усіх бронювань по кімнатам (рис. 3.8).

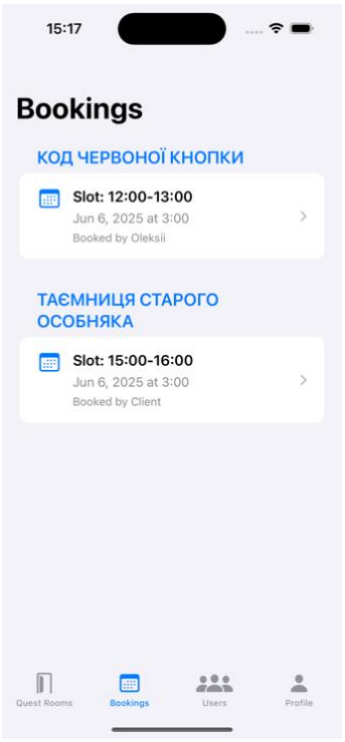


Рисунок 3.8 – AdminBookingListView

- CreateEditQuestRoomView – адміністративна форма створення/редагування квест-кімнат, підтримка завантаження фото (рис. 3.9).

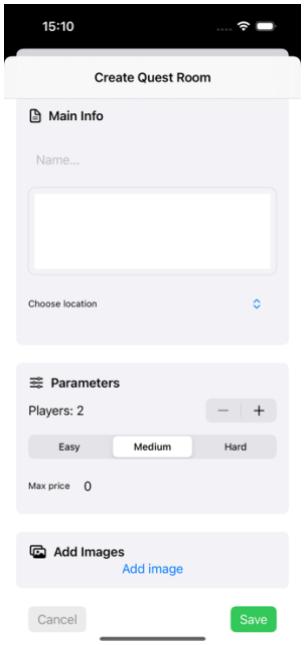


Рисунок 3.9 – CreateEditQuestRoomView

- AdminUserListView – список користувачів у системі (рис. 3.10) з якого є можливість переходу до перегляду деталей користувача (рис. 3.11).

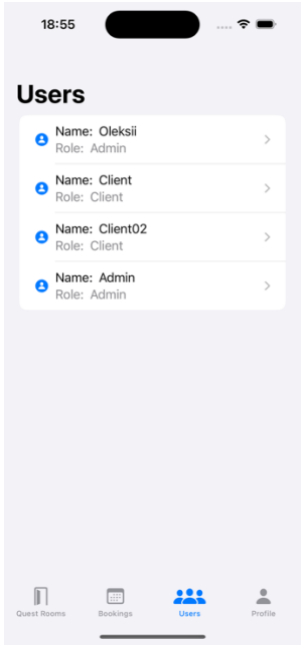


Рисунок 3.10 – AdminUserListView

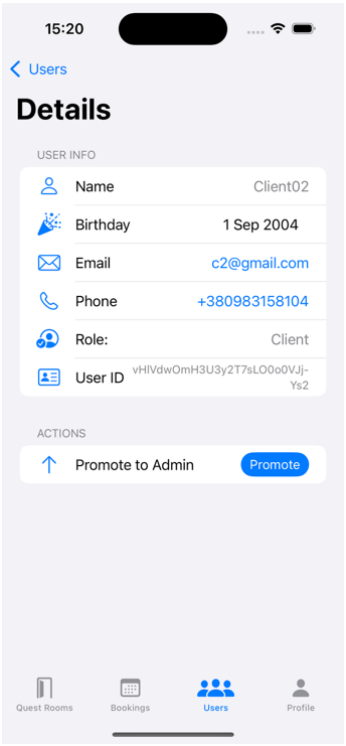


Рисунок 3.11 – UserDetailView

- UserDetailView – перегляд інформації про користувача з можливістю редагування ролі, а саме підвищення клієнта до ролі “Адміністратора” (рис. 3.11).

ClientView та AdminView складаються з попередніх переглядів відповідно до ролей.

Кожен екран має логічну структуру з використанням Form, List, Section, NavigationStack або ScrollView для кращої взаємодії користувача. Кнопки, текстові поля та селектори стилізовані у відповідності до системного вигляду iOS.

Також реалізовано підтримку темної теми (dark mode) – усі кольори інтерфейсу автоматично адаптуються до системного режиму. Текст, фонові кольори та іконки підтримують динамічні стилі (.foregroundStyle(.primary) тощо). Переглянути їх можна у додатку Б.

Завдяки реактивному підходу SwiftUI, усі зміни в даних автоматично відображаються в інтерфейсі без потреби в ручному оновленні. Це забезпечує

плавний та інтуїтивно зрозумілий досвід для користувача незалежно від його ролі в системі.

3.3 Локалізація інтерфейсу

Застосунок реалізовано з підтримкою багатомовного інтерфейсу, що дозволяє користувачам взаємодіяти українською та англійською мовами. У цьому розділі описано застосовані технології, структуру локалізації, інтеграцію з інтерфейсом SwiftUI, приклади використання та переваги обраного підходу.

3.3.1 Формат локалізації

Замість традиційного “Localizable.strings”, застосовано сучасний формат Localizable.xcstrings, що підтримується в Xcode 15 і новіших. Основні переваги:

- зберігання всіх перекладів у структурованому вигляді;
- зручне редагування через інтерфейс Xcode;
- підтримка множинних мов у єдиному каталозі;
- вбудована система статусів (NEW, NEEDS REVIEW, TRANSLATED).

Формат “xcstrings” забезпечує:

- плюралізацію: автоматичне відображення різних форм залежно від кількості (напр., "1 гравець", "2 гравці");
- варіативність: можливість задавати різні варіанти перекладу для різних пристроїв (наприклад, iPhone / iPad);
- вставки змінних: підтримка форматованих рядків із параметрами;
- автоматичне виявлення застарілих або незаповнених рядків.

3.3.2 Інтерфейс локалізації в Xcode

Xcode надає вбудовані інструменти для управління локалізацією застосунків, зокрема через формат String Catalog. Після створення .xcstrings

файлу, Xcode автоматично створює зручний інтерфейс, у якому можна додавати переклади для кожного ключа (рис. 3.12).

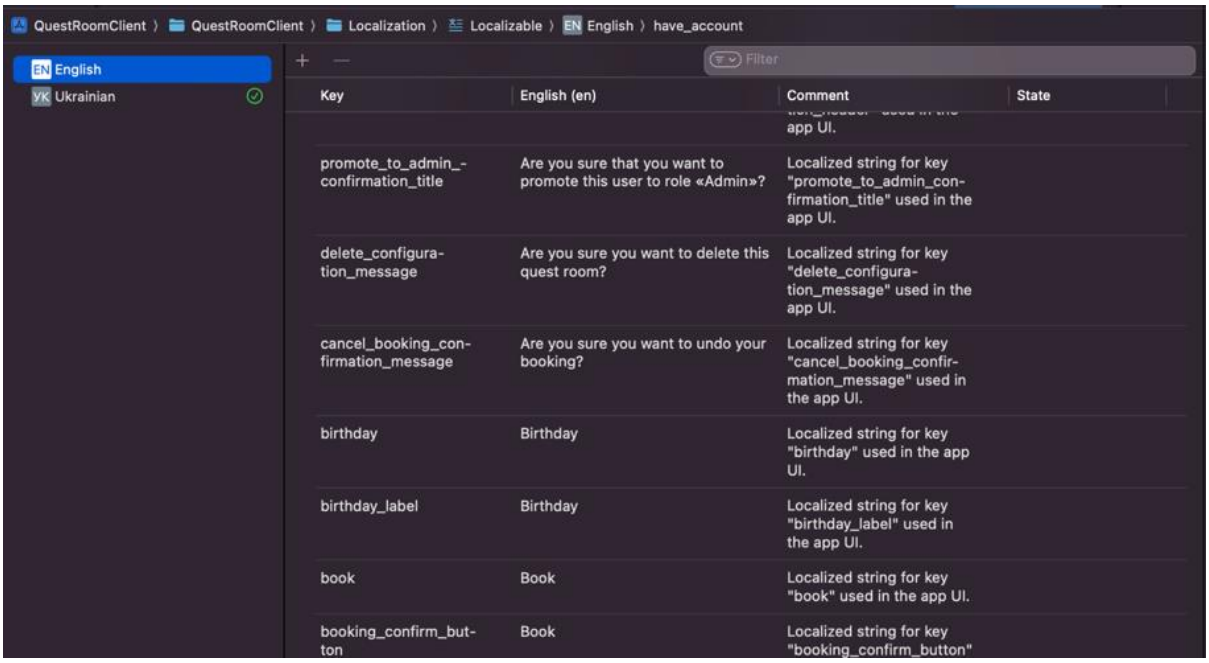


Рисунок 3.12 – Інтерфейс локалізації англійської мови (основна мова)

Цей інтерфейс розділено на колонки:

- Key — унікальний ідентифікатор для рядка;
- Value — базовий текст (наприклад, англійською);
- Translation — переклад для кожної мови, що підтримується проєктом;
- Comment — контекст для перекладачів.
- State — відображає статус перекладу (наприклад, "translated", "untranslated", "needs review") (рис. 3.13).

Користувач також може додавати нові ключі вручну або за допомогою інструменту автозаповнення з коду. Xcode дозволяє миттєво переглядати помилки (наприклад, відсутність перекладу для певної мови) та підтримує функціонал pluralization та варіантів локалізованого тексту залежно від параметрів. Так як основною мовою застосунку є англійська, то вона пропонується як оригінальний переклад і може використовуватися тимчасовий

переклад, поки не буде здійснена повна локалізації каталогу на іншу мову, наприклад, українську (рис. 3.14).

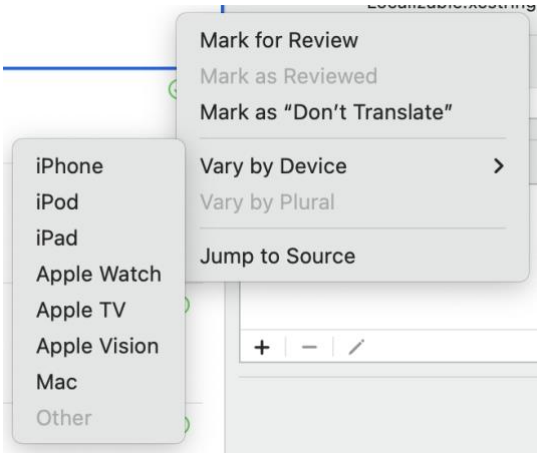


Рисунок 3.13 – Опції поля State

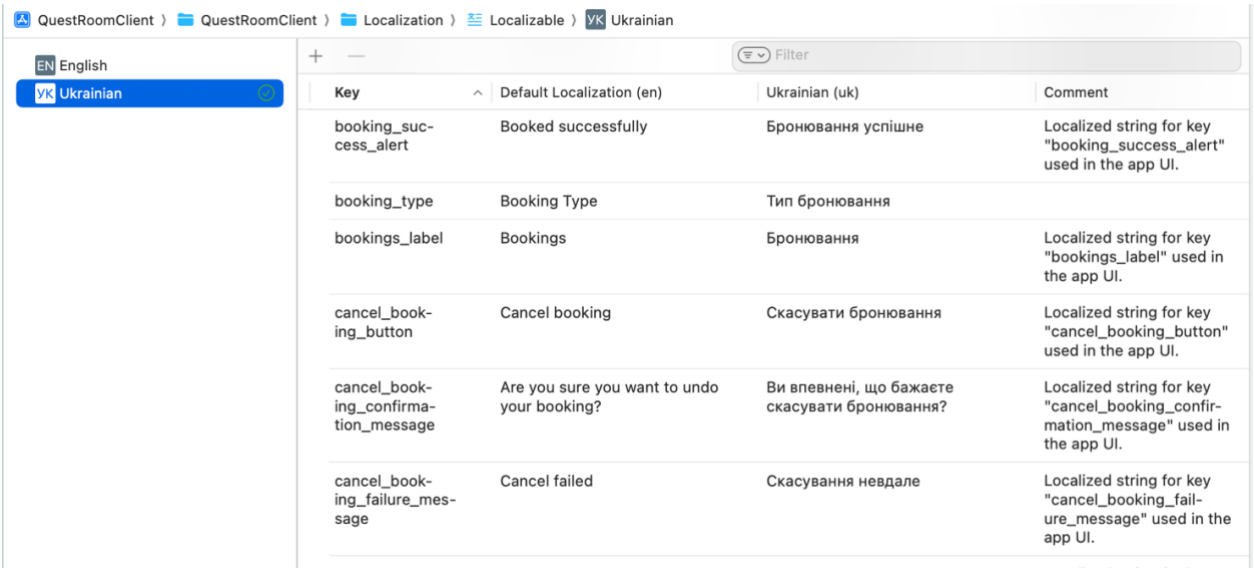


Рисунок 3.14 – Інтерфейс локалізації української мови

3.3.3 Інтеграція з SwiftUI

Завдяки використанню типу “LocalizedStringKey”, SwiftUI автоматично підхоплює ключі з каталогу. На рисунку 3.15, видно, що в полях, де мав би бути присутній текст, вкладені ключі локалізації.

```
Picker(
    "booking_type",
    selection: $selectedTab
) {
    Text("upcoming_bookings_label").tag(Tab.upcoming)
    Text("past_bookings_label").tag(Tab.past)
}
.pickerStyle(.segmented)
.padding(.horizontal)
```

Рисунок 3.15 – Приклад використання ключів локалізації “booking_type”, “upcoming_bookings_label” та “past_bookings_label” під час розробки графічного інтерфейсу

Ці ключі автоматично підключаються до локалізованих значень без додаткового коду. Якщо ж це не працює автоматично, то можна використати NSLocalizedString("…", comment: "…"), який примусово додає ключ локалізації до каталогу і підставлятиме його значення, а саме локалізований варіант у це місце відповідно.

Для перегляду локалізованого інтерфейсу під час розробки застосовують .environment(\.locale, Locale(identifier: "…")) для передперегляду Preview, де вказують потрібний каталог локалізацій. Приклад зображено на рисунку 3.16.

```
#Preview {
    RegisterView()
        .environment(\.locale, Locale(identifier: "uk"))
}
```

Рисунок 3.16 – Приклад виклику української локалізації для передперегляду локалізованого вигляду RegisterView

Це дозволяє розробникам бачити, як виглядатиме інтерфейс конкретною мовою прямо в Preview, що суттєво пришвидшує процес перевірки локалізації, виявлення обрізаних елементів або неперекладених рядків ще до запуску застосунку на пристрої.

3.3.4 Приклади локалізованого інтерфейсу

На рисунку 3.17 представлено екран реєстрації в застосунку, локалізований англійською мовою. Користувач бачить перекладені поля введення, кнопки та повідомлення відповідно до встановленого мовного середовища. Відповідно адаптуються і системні елементи: такі як DatePicker, який пропонує обрати дату з відповідними перекладами назви місяців.

На рисунку 3.18 показано той самий екран, але з активованою українською локалізацією, що ілюструє динамічне перемикання мов та коректне відображення інтерфейсу.

Можна й надалі розширювати набір локалізації застосунку.

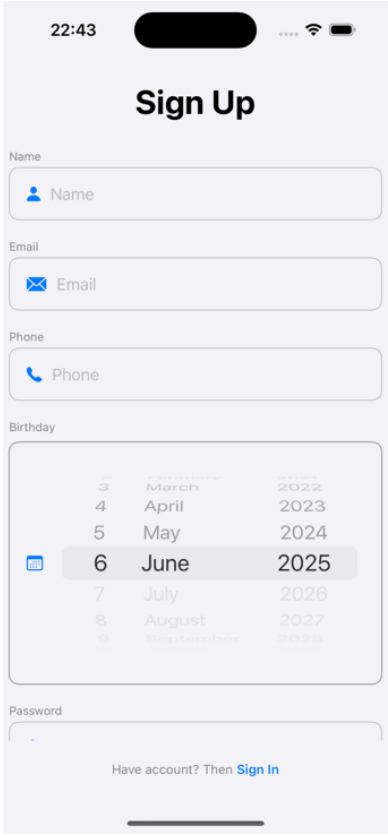


Рисунок 3.17 – RegisterView з англійською локалізацією

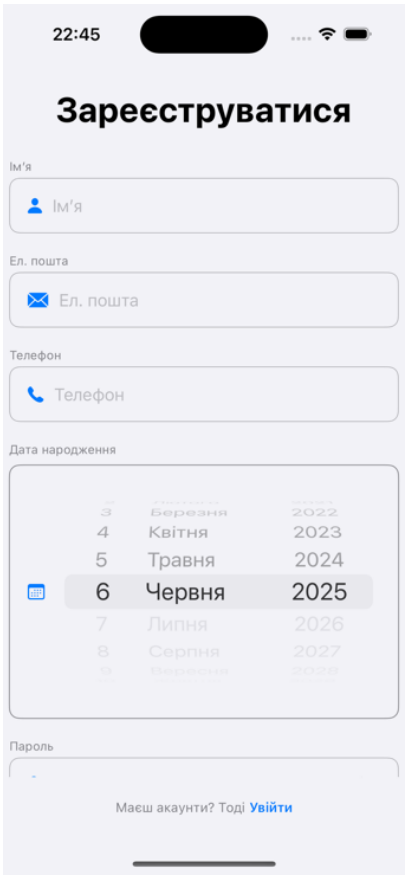


Рисунок 3.18 – RegisterView з українською локалізацією

3.3.5 Переваги обраного підходу

Можна виокремити наступні переваги такого підходу щодо локалізації застосунку:

- масштабованість для нових мов без зміни коду;
- зменшення кількості помилок форматування;
- зручна робота в команді з перекладачами;
- полегшення тестування і перегляду інтерфейсу різними мовами.

У підсумку, використання “Localizable.xcstrings” підвищує якість інтерфейсу, забезпечує надійність і знижує технічний борг під час додавання нових мов.

3.4 Тестування мобільного застосунку та порівняння з аналогами

На завершальному етапі розробки мобільного застосунку було проведено тестування основних функціональних блоків, зокрема авторизації, перегляду кімнат, створення бронювання та локалізації інтерфейсу. Тестування охоплювало як стандартні сценарії використання, так і обробку помилок — наприклад, при втраті інтернет-з’єднання або спробі створення повторного бронювання на зайнятий слот, яке розроблений застосунок пройшов успішно.

З метою об’єктивної оцінки досягнутого результату, було проведено порівняльний аналіз реалізованого мобільного застосунку з трьома популярними аналогами на українському ринку: QIMNATA, ЗАМКНЕНІ та QuestRoom. Порівняння здійснювалося за критеріями, які були визначені на попередньому етапі аналізу: архітектура, підтримка авторизації, інтеграція з календарем, багатомовність, якість верстки, підтримка мобільних платформ тощо, наведний на таблиці 3.1.

Таблиця 3.1 – Порівняння реалізованого застосунку з конкурентами-аналогами

Критерій	**QIMNATA**	**ЗАМКНЕНІ**	**QuestRoom**	**Реалізований застосунок**
Архітектура	3-tier web-application	3-tier web-application	3-tier web-application	MVVM (mobile)
Автентифікація	–	–	+	+
Інтеграція з календарем	–	–	–	+
Оплата онлайн	–	–	Частково (Visa / MC)	– (але можливе)
Миттєве бронювання	+	+	+	+
Реєстрація користувачів	–	–	+	+
Знижки / акції	–	+ (на день народження)	+ (на день народження)	– (можна додати у майбутньому)
Вказано складність кімнат	–	+	+	+
Багатомовність	UA, EN	UA, EN, RU	UA, EN, RU	UA, EN
Зворотній зв’язок	+	+	+	+
Підтримка мобільних платформ	– (веб)	– (веб)	– (веб)	iOS (SwiftUI)
Якість верстки	Низька	Середня	Середня	Висока

Згідно з порівняльною таблицею, розроблений застосунок демонструє ряд переваг над існуючими рішеннями:

- єдиний, хто реалізував повноцінну підтримку мобільної платформи iOS з використанням архітектури MVVM;
- інтеграція з календарем пристрою через EventKit для автоматичних нагадувань;
- гнучка та адаптивна локалізація інтерфейсу на основі сучасного формату .xcstrings;
- висока якість верстки, реалізована засобами SwiftUI з підтримкою темної теми;

- повноцінна автентифікація та реєстрація користувача з інтеграцією Firebase.

Попри те, що у застосунку ще не реалізовано функціонал онлайн-оплати або знижок до дня народження, його функціональна база значно перевершує типові можливості конкурентів, зосереджених переважно на веб-платформах.

Таким чином, результати тестування підтверджують працездатність та переваги розробленого клієнтського застосунку у контексті обраної предметної області.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було проведено повноцінне дослідження, присвячене аналізу проблематики бронювання квест-кімнат та створенню мобільного рішення для її подолання. Зростаюча популярність escaperoom-індустрії в Україні та у світі зумовлює потребу в цифровізації основних процесів взаємодії між клієнтами та адміністраторами, зокрема — автоматизації бронювання, управління слотами та персональними даними.

Проведений аналіз предметної області показав, що більшість існуючих моделей бронювання залишаються ручними або лише частково автоматизованими, що створює значне навантаження на персонал, підвищує ризик людських помилок і обмежує можливість масштабування бізнесу. Крім того, виявлено низький рівень інтеграції мобільних рішень у цій сфері, що суперечить сучасним очікуванням користувачів, які віддають перевагу інтуїтивним мобільним застосункам із миттєвим доступом до послуг.

У другому розділі були досліджені існуючі рішення на українському ринку. Виявлено, що більшість платформ або не підтримують мобільний застосунок узагалі, або мають обмежену функціональність, не враховуючи специфіки escape room-бізнесу. Також було розглянуто закордонні SaaS-рішення, які часто є надмірними за функціональністю або надто дорогими для локального малого бізнесу. Це ще раз підтвердило доцільність створення власного оптимізованого застосунку.

У результаті було сформульовано низку вимог до функціоналу майбутньої системи: наявність авторизації та реєстрації, рольової моделі доступу, інтерфейсу для перегляду та бронювання квест-кімнат, можливості обробки зображень кімнат, а також перегляду історії бронювань.

Розроблений клієнтський застосунок орієнтований на користувачів платформи iOS, реалізований із використанням мови Swift та сучасного фреймворку SwiftUI, що забезпечує високу адаптивність, нативний інтерфейс і простоту підтримки. Застосунок побудовано за архітектурною моделлю MVVM,

яка дозволяє ефективно організувати код, відокремити логіку представлення від візуального інтерфейсу та спростити тестування окремих модулів.

У застосунку реалізовано підтримку авторизації користувачів з розподілом на ролі (адміністратор і клієнт), можливість перегляду списку доступних квест-кімнат, створення нових бронювань із динамічним вибором дати та часового слоту, а також перегляд, редагування та скасування існуючих бронювань. Для адміністратора передбачено додатковий функціонал: створення нових кімнат, редагування інформації про них, завантаження зображень і керування користувачами.

Під час реалізації особлива увага приділялася зручності інтерфейсу, адаптивності до різних сценаріїв використання, а також загальній стабільності роботи застосунку. Було проведено попереднє тестування функціональних компонентів, зокрема форм для введення даних, навігації між екранами та динамічного завантаження зображень. Також були враховані рекомендації щодо UI/UX-дизайну в межах екосистеми Apple.

Підсумовуючи, можна зазначити, що всі поставлені завдання в межах бакалаврської кваліфікаційної роботи були повністю виконані. Створений прототип мобільного застосунку може слугувати прикладом ефективної цифровізації локального сервісного бізнесу та має потенціал для реального впровадження. У майбутньому проєкт може бути доповнений новими функціями, зокрема інтеграцією платіжних сервісів, розширенням під інші платформи (наприклад, Android), а також удосконаленням системи аналітики для бізнесу.

Звіт про виконану роботу було складено відповідно до вимог методичних вказівок, що гарантує ного структурованість, логічність складення матеріалу та відповідність академічним стандартам. [25]

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Порівняння інструментів SwiftUI та UIKit для розробки інтерфейсу користувача для iOS-пристроїв на мові програмування Swift / Чумак О. В., Богач І.В. // LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025), Вінниця, 2025. [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/24313.

2. Локалізація мобільних застосунків на IOS за допомогою SwiftUI та нового формату Localizable.xcstrings / Чумак О. В., Богач І.В. // Молодь в науці: дослідження, проблеми, перспективи (МН-2025), Вінниця, 2025. [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25423.

3. Пашковський О. М. Квест-кімнати: теорія та практика [Текст] : навч. посіб. / О. М. Пашковський. Львів : ЛНУ ім. І. Франка, 2022. 184 с.

4. Коваленко В. І. Цифрові технології в індустрії розваг [Текст] : монографія / В. І. Коваленко. Київ : КНЕУ, 2021. 232 с.

5. Модератори реальності: як відкрити квест-кімнату в Україні // Блог Ощадбанку [Електронний ресурс]. URL: https://www.oschadbank.ua/blog/moderatory-realnosti-yak-vidkryty-kvest-kimnatu-v-ukrayini (дата звернення: 22.05.2025)

6. 16 Types of Escape Rooms. URL: https://escapely.com/types-of-escape-rooms. (дата звернення: 06.06.2025)

7. Іванова Н. В. Автоматизація процесів бронювання в сфері розважальних послуг [Текст] : стаття / Н. В. Іванова // Журнал «ІТ в бізнесі». 2020. № 4. С. 45–53.

8. Brown T., Green M. Escape Room Industry: Trends and Digital Solutions [Текст] : стаття / T. Brown, M. Green // International Journal of Entertainment Technology. 2023. Vol. 10, Iss. 2. P. 101–115.

9. EasyWeek. Онлайн-платформа для бронювання [Електронний ресурс]. URL: https://easyweek.io. (дата звернення: 22.05.2025)

10. QIMNATA [Електронний ресурс]. URL: https://qimnata.com/ua. (дата звернення: 22.05.2025)

11. ZAMKNENI [Електронний ресурс]. URL: https://vzaperti.com.ua. (дата звернення: 22.05.2025)

12. QuestRoom [Електронний ресурс]. URL: https://questroom.com.ua/ua. (дата звернення: 22.05.2025)

13. App Annie. The State of Mobile 2023. URL: https://www.data.ai/en/go/state-of-mobile-2023. (дата звернення: 06.06.2025)

14. Statista. Share of mobile internet traffic worldwide 2012–2024. URL: https://www.statista.com (дата звернення: 06.06.2025)

15. Apple Inc. The Swift Programming Language [Електронний ресурс]. URL: https://swift.org/documentation. (дата звернення: 22.05.2025)

16. Apple Inc. SwiftUI Documentation [Електронний ресурс]. URL: https://developer.apple.com/documentation/swiftui. (дата звернення: 22.05.2025)

17. Apple Inc. UIKit Framework Reference [Електронний ресурс]. URL:https://developer.apple.com/documentation/uikit. (дата звернення: 22.05.2025)

18. MVVM Design Pattern Explained [Електронний ресурс]. URL:https://www.raywenderlich.com/6733535-mvvm-design-pattern-tutorial-for-ios. (дата звернення: 22.05.2025)

19. Vapor Team. Vapor Documentation [Електронний ресурс]. URL:https://docs.vapor.codes. (дата звернення: 22.05.2025)

20. REST API Tutorial [Електронний ресурс]. URL:https://restfulapi.net. (дата звернення: 22.05.2025)

21. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs. (дата звернення: 22.05.2025)

22. Fluent ORM Documentation [Електронний ресурс]. URL: https://docs.vapor.codes/fluent/overview/. (дата звернення: 22.05.2025)

23. Firebase. Get started with Firebase Authentication on iOS [Електронний ресурс]. URL: https://firebase.google.com/docs/auth/ios/start. (дата звернення: 22.05.2025)

24. ДСТУ 8302:2015. URL: https://pdf.lib.vntu.edu.ua/books/2018/ДСТУ%208302%20повний.pdf. (дата звернення: 11.06.2025)

25. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт (проєктів) для студентів спеціальностей: 126 «Інформаційні системи та технології», 151 «Автоматизація та комп'ютерно-інтегровані технології» [Електронний ресурс] / Уклад. К. В. Овчинников, О. В. Бісікало. – Вінниця : ВНТУ, 2022. – 50 с.

ДОДАТКИ

Вінницький національний технічний університет

Кафедра автоматизації та інтелектуальних інформаційних технологій

ЗАТВЕРДЖУЮ

завідувача кафедри АІІТ

д.т.н., професор Олег БІСІКАЛО

_____

(підпис)

«_23_» _березня_ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу

«Розробка мобільного застосунку для бронювання квест-кімнат»

08-31.БКР.021.02.000 ТЗ

Керівник бакалаврської кваліфікаційної роботи

д.т.н., проф. Роман Квєтнгий

_____

(підпис)

«___» _____ 2025 р.

Розробив студент гр. 1ІСТ-21б

Олексій Чумак

_____

(підпис)

«___» _____ 2025 р.

Вінниця ВНТУ – 2025 рік

Додаток А (обов'язковий)

Технічне завдання на бакалаврську кваліфікаційну роботу

1 Підстава для проведення робіт

Підставою для виконання бакалаврської кваліфікаційної роботи на тему: «Розробка мобільного застосунку для бронювання квест-кімнат» є наказ № 97 від 20.03.2025 р.

Термін виконання робіт:

початок 20.03.2025 р.
кінець р.

2 Мета та вихідні дані для проведення робіт

Метою бакалаврської кваліфікаційної роботи є покращення сервісів бронювання квест-кімнат за рахунок розробки модифікованого програмного забезпечення, що, на відміну від існуючих. дозволить спростити взаємодію між клієнтами та адміністраторами, забезпечить автоматизовану обробку бронювань, зменшить ризики помилок і підвищить рівень сервісу.

Вихідними даними для проведення робіт є індивідуальне завдання на бакалаврську кваліфікаційну роботу від 20.03.2025 р.

3 Етапи виконання робіт

Виконавцем всіх перерахованих в даному розділі етапів є: студент групи **1ІСТ-21б Чумак Олексій Віталійович** факультету інтелектуальних інформаційних технологій та автоматизації Вінницького національного технічного університету, а замовником є кафедра автоматизації та інтелектуальних інформаційних технологій.

№ Етапу	Зміст етапу	Строки виконання
Е1	Аналіз предметної області	20.03 – 12.04
Е2	Проєктування мобільного застосунку для бронювання квест-кімнат	13.04 – 05.05
Е3	Реалізація функціоналу клієнтського застосунку	06.05 – 01.06
Е4	Висновки	01.06 – 10.06

4 Призначення і галузь застосування

Мобільний застосунок для бронювання квест-кімнат призначений для спрощення взаємодії між клієнтами та адміністраторами закладів, які надають послуги квест-розваг. Застосунок дозволяє користувачам переглядати доступні кімнати, ознайомлюватися з описами, бронювати зручні слоти, а також додавати події до власного календаря. Для адміністраторів реалізовано функціонал керування контентом, розкладом та користувачами через зручний інтерфейс.

Розроблена система може застосовуватися у сфері розваг для автоматизації процесу бронювання у реальних квест-локаціях. Також вона придатна для використання в навчальних закладах як приклад сучасного мобільного застосунку, а її архітектура дозволяє легко масштабувати рішення під інші типи бронювання (наприклад, студії, події, консультації). Особливо актуальною система є в умовах активного використання мобільних пристроїв і цифрових сервісів у повсякденному житті.

5 Технічні дані

2.5 Платформа розробки – Swift 5.9
2.6 Середовище розробки – Xcode 15.3
2.7 Операційна система – iOS 16.0 або новіша
2.8 Архітектура застосунку – MVVM (Model-View-ViewModel)
2.9 Система автентифікації – Firebase Authentication (email/пароль)
2.10 Локалізація інтерфейсу – .xcstrings (підтримка української та англійської мов)
2.11 Серверна частина – Vapor+PostgreSQL

6 Джерела розробки

6.1 Apple Inc. The Swift Programming Language [Електронний ресурс]. URL:https://swift.org/documentation.
6.2 Apple Inc. SwiftUI Documentation [Електронний ресурс]. URL: https://developer.apple.com/documentation/swiftui.
6.3 MVVM Design Pattern Explained [Електронний ресурс]. URL:https://www.raywenderlich.com/6733535-mvvm-design-pattern-tutorial-for-ios.
6.4 Firebase. Get started with Firebase Authentication on iOS [Електронний ресурс]. URL: https://firebase.google.com/docs/auth/ios/start. (дата звернення: 22.05.2025)

Здобувач ст. гр. 1ІСТ-21б _____ Олексій ЧУМАК

Додаток Б

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ КВЕСТ-КІМНАТ

Діаграма послідовності для створення бронювання

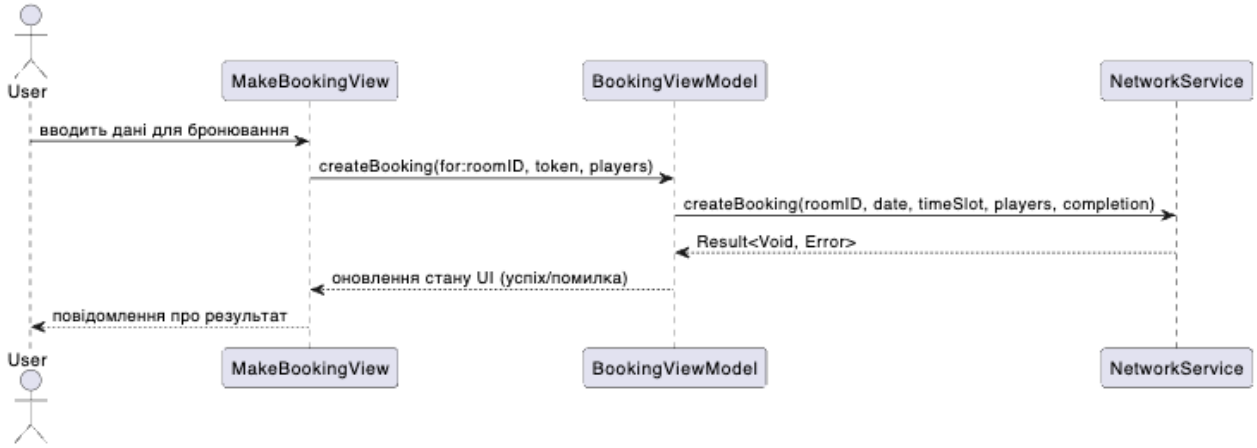


Рисунок Б.1 – Діаграма послідовності для створенні бронювання

Діаграма послідовності для автентифікації користувача

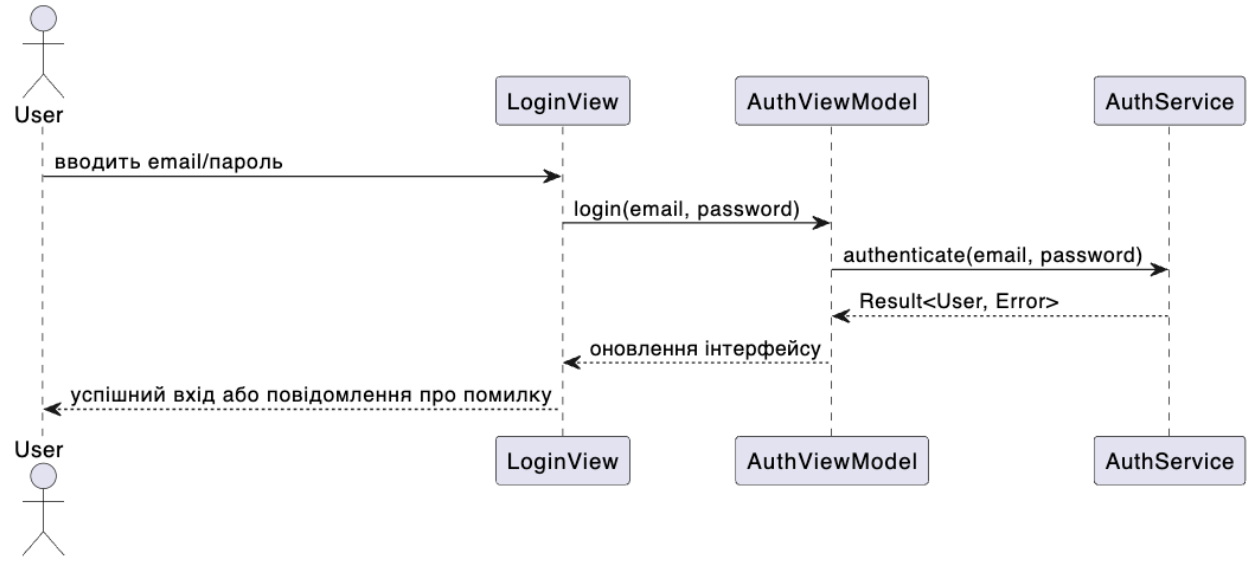


Рисунок Б.2 – Діаграма послідовності для автентифікації користувача

Діаграма послідовності для редагування кімнати

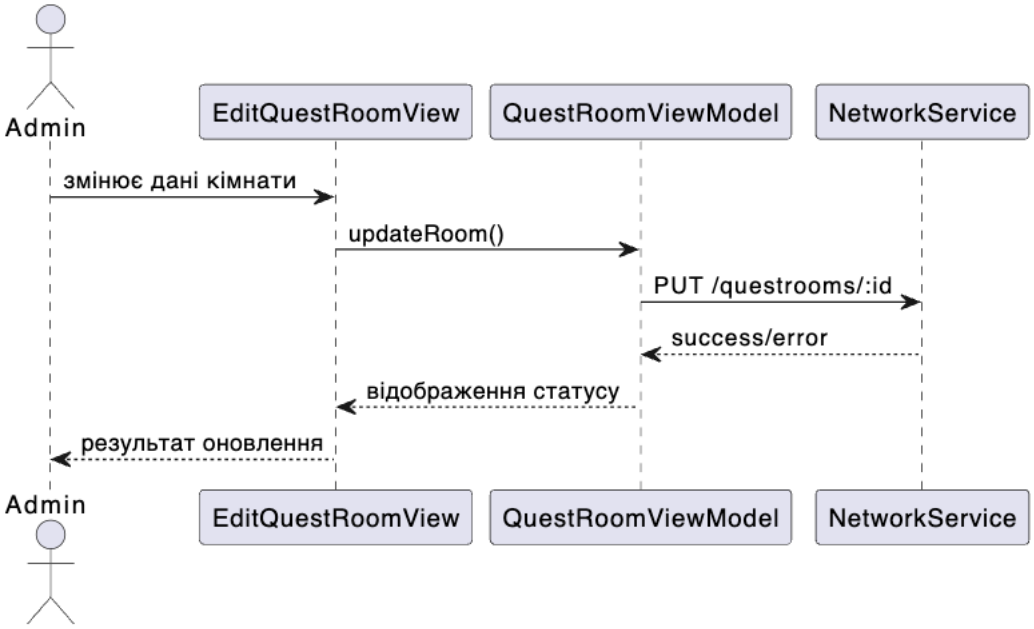


Рисунок Б.3 – Діаграма послідовності для редагуванн кімнати

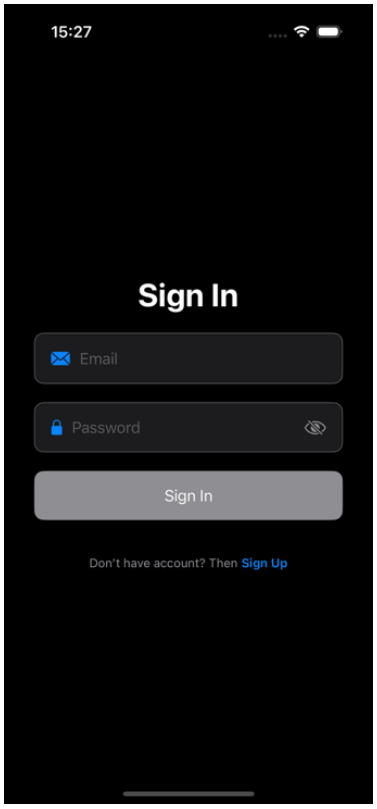


Рисунок Б.4 – LoginView (темний)

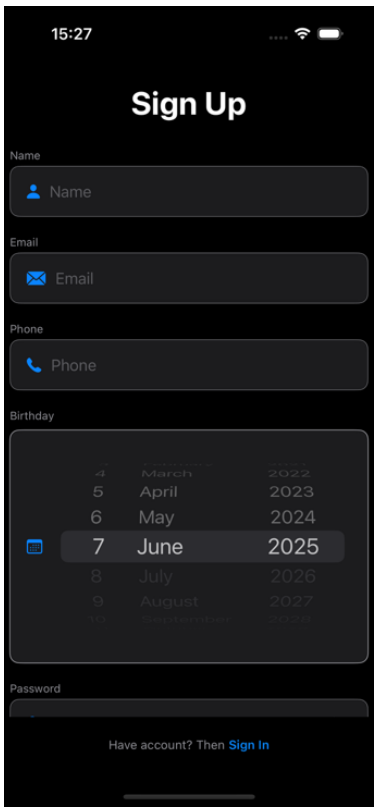


Рисунок Б.5 – RegisterView (темний)

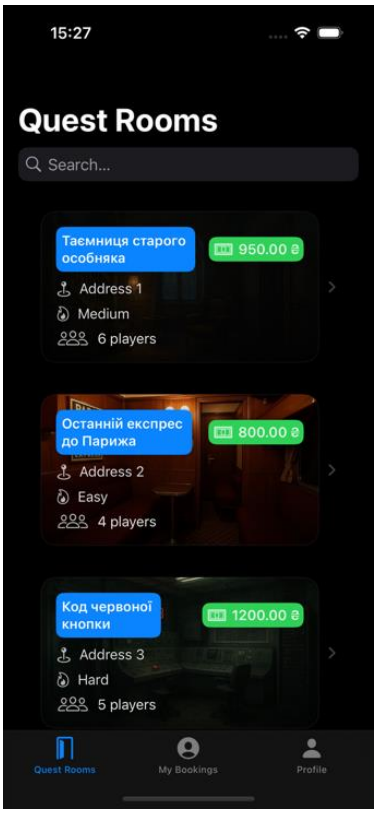


Рисунок Б.6 – QuestRoomListView (темний)

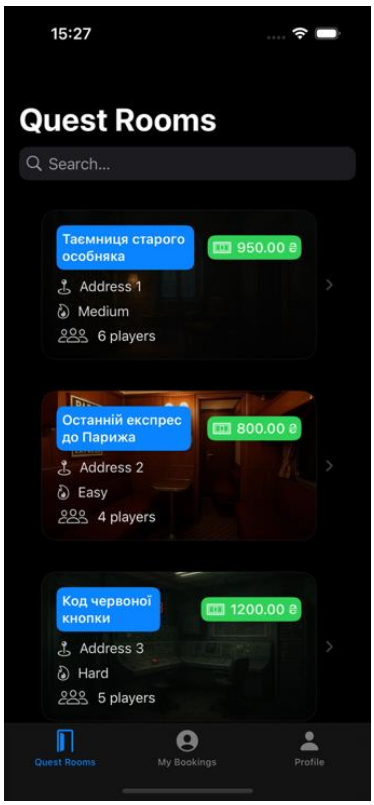


Рисунок Б.7 – RegisterView (темний)

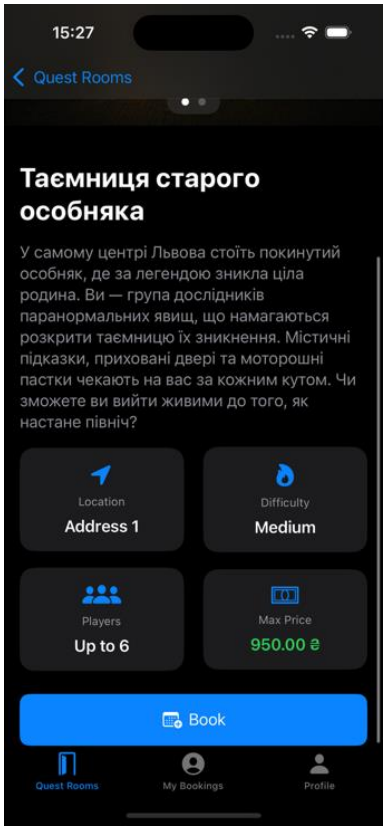


Рисунок Б.8 – QuestRoomDetailView (темний)

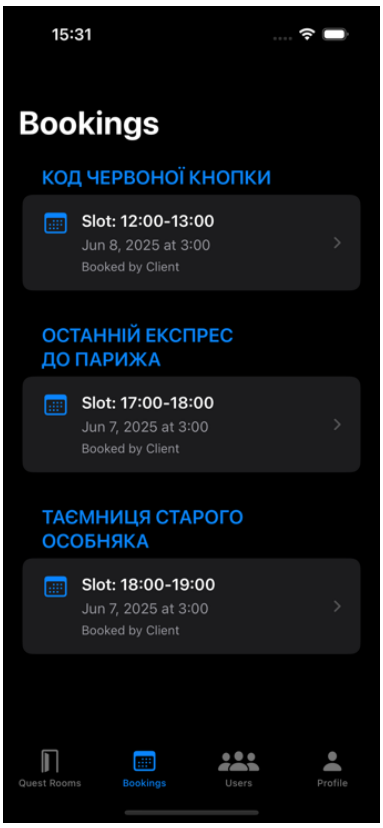


Рисунок Б.9 –AdminBookingListView (темний)

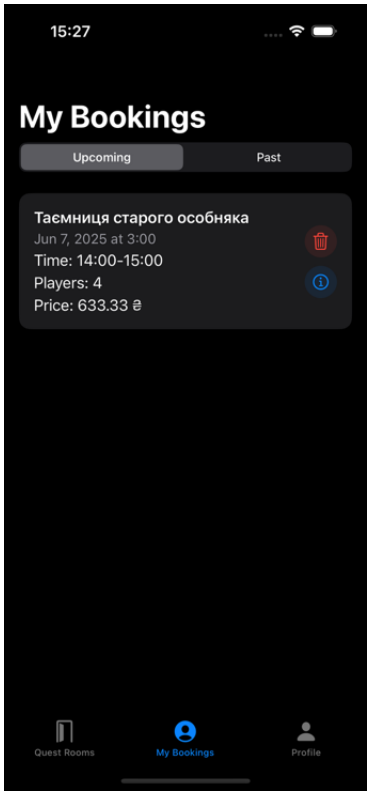


Рисунок Б.10 –ClientBookingListView (темний)

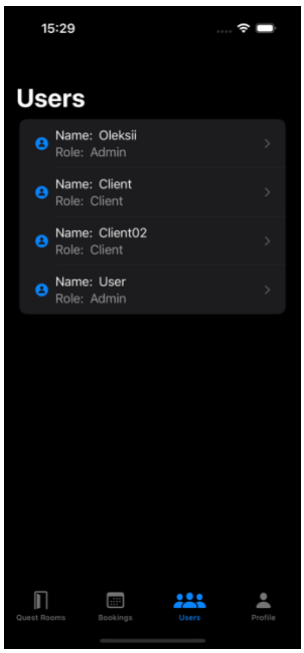


Рисунок Б.11 –AdminUserListView (темний)

Додаток В (обов'язковий)

Лістинг основних функцій застосунку

```
class AuthService: ObservableObject {
    static let shared = AuthService()
    @AppStorage("userRole") var userRole: String = ""
    @AppStorage("isLoggedIn") var isLoggedIn: Bool = false
    @AppStorage("idToken") var idToken: String = ""
    private init() {}
    func signIn(email: String, password: String, completion: @escaping (Result<String, Error>) -> Void) {
        Auth.auth().signIn(withEmail: email, password: password) { result, error in
            if let error = error {
                completion(.failure(error))
                return
            }
            guard let user = result?.user else {
                completion(.failure(NSError(domain: "No user", code: 0)))
                return
            }

            user.getIDToken { token, error in
                guard let idToken = token else {
                    completion(.failure(error ?? NSError(domain: "No token", code: 0)))
                    return
                }

                NetworkService.shared.fetchCurrentUser(token: idToken) { result in
                    switch result {
                    case .success:
                        self.idToken = idToken
                        self.fetchAndSaveRole(token: idToken, completion: completion)
                    case .failure:
                        NetworkService.shared.registerUser(token: idToken, email: email, uid: user.uid, name:
"User", phoneNumber: "+380000000000", birthday: Date()) { result in
                            switch result {
                            case .success:
                                self.fetchAndSaveRole(token: idToken, completion: completion)
                            case .failure(let error):
                                if let urlError = error as? URLError,
                                   urlError.code == .badServerResponse {
                                    print("⚠️ User probably already exists. Proceeding to role fetch.")
                                    self.fetchAndSaveRole(token: idToken, completion: completion)
                                } else {
                                    completion(.failure(error))
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    func refreshIDToken(completion: @escaping (String?) -> Void) {
        guard let user = Auth.auth().currentUser else {
```

```
            completion(nil)
            return
        }
        user.getIDTokenForcingRefresh(true) { token, error in
            if let token = token {
                UserDefaults.standard.set(token, forKey: "idToken")
                completion(token)
            } else {
                print("❌ Failed to refresh token:", error?.localizedDescription ?? "Unknown error")
                completion(nil)
            }
        }
    }
    private func fetchAndSaveRole(token: String, completion: @escaping (Result<String, Error>) -> Void) {
        NetworkService.shared.fetchCurrentUserRole(token: token) { result in
            DispatchQueue.main.async {
                switch result {
                case .success(let role):
                    self.userRole = role.rawValue
                    self.isLoggedIn = true
                    completion(.success(token))
                case .failure(let error):
                    completion(.failure(error))
                }
            }
        }
    }
    func signUp(email: String, password: String, name: String, phoneNumber: String, birthday: Date,
completion: @escaping (Result<String, Error>) -> Void) {
        Auth.auth().createUser(withEmail: email, password: password) { result, error in
            if let error = error {
                completion(.failure(error))
                return
            }
            guard let user = result?.user else {
                completion(.failure(NSError(domain: "No user", code: 0)))
                return
            }
            user.getIDToken { token, error in
                guard let idToken = token else {
                    completion(.failure(error ?? NSError(domain: "No token", code: 0)))
                    return
                }
                NetworkService.shared.registerUser(token: idToken, email: email, uid: user.uid, name: name,
phoneNumber: phoneNumber, birthday: birthday) { result in
                    switch result {
                    case .success:
                        NetworkService.shared.fetchCurrentUserRole(token: idToken) { result in
                            DispatchQueue.main.async {
                                switch result {
                                case .success(let role):
                                    self.userRole = role.rawValue
                                    self.isLoggedIn = true
                                    completion(.success(idToken))
                                case .failure(let error):
                                    completion(.failure(error))
                                }
```

```
                    }
                }
                case .failure(let error):
                    completion(.failure(error))
                }
            }
        }
    }
    func signOut() {
        do {
            try Auth.auth().signOut()
            self.isLoggedIn = false
            self.userRole = ""
        } catch {
            print("Sign out failed: \(error)")
        }
    }
}
class NetworkService {
    static let shared = NetworkService()
    private let baseURL = URL(string: "…")!

    private var idToken: String {
        UserDefaults.standard.string(forKey: "idToken") ?? ""
    }
    private func authorizedDataTask(
        with request: URLRequest,
        retryAction: @escaping () -> Void,
        completion: @escaping (Data?, URLResponse?, Error?) -> Void
    ) {
        URLSession.shared.dataTask(with: request) { data, response, error in
            if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 401 {
                AuthService.shared.refreshIDToken { newToken in
                    guard newToken != nil else {
                        completion(nil, nil, URLError(.userAuthenticationRequired))
                        return
                    }
                    retryAction()
                }
            } else {
                completion(data, response, error)
            }
        }.resume()
    }
    func fetchQuestRooms(completion: @escaping (Result<[QuestRoom], Error>) -> Void) {
        let url = baseURL.appendingPathComponent("quest-rooms")
        var request = URLRequest(url: url)
        request.httpMethod = "GET"
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")
        URLSession.shared.dataTask(with: request) { data, response, error in
            if let error = error {
                completion(.failure(error))
                return
            }
            if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 401 {
                AuthService.shared.refreshIDToken { newToken in
```

```
                guard newToken != nil else {
                    completion(.failure(URLError(.userAuthenticationRequired)))
                    return
                }
                self.fetchQuestRooms(completion: completion)
            }
            return
        }
        guard let data = data else {
            completion(.failure(URLError(.badServerResponse)))
            return
        }
        do {
            let rooms = try JSONDecoder().decode([QuestRoom].self, from: data)
            completion(.success(rooms))
        } catch {
            completion(.failure(error))
        }
    }.resume()
}
func fetchCurrentUser(token: String, completion: @escaping (Result<User, Error>) -> Void) {
    let url = baseURL.appendingPathComponent("/me")
    var request = URLRequest(url: url)
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    URLSession.shared.dataTask(with: request) { data, _, error in
        if let error = error {
            completion(.failure(error))
            return
        }
        guard let data = data else {
            completion(.failure(URLError(.badServerResponse)))
            return
        }
        do {
            let user = try User.decoder.decode(User.self, from: data)
            completion(.success(user))
        } catch {
            completion(.failure(error))
        }
    }.resume()

}

func fetchCurrentUserRole(token: String, completion: @escaping (Result<UserRole, Error>) -> Void) {
    let url = baseURL.appendingPathComponent("/me")

    var request = URLRequest(url: url)
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")

    URLSession.shared.dataTask(with: request) { data, response, error in
        if let error = error {
            completion(.failure(error))
            return
        }
        guard let httpResponse = response as? HTTPURLResponse else {
            completion(.failure(URLError(.badServerResponse)))
            return
```

```
            }
            guard let data = data else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }

            if httpResponse.statusCode != 200 {
                print("❌ Server error:", String(data: data, encoding: .utf8) ?? "Unknown error")
                completion(.failure(URLError(.userAuthenticationRequired)))
                return
            }

            print("📄 /me response: \(String(data: data, encoding: .utf8) ?? "nil")")

            do {
                let decoded = try User.decoder.decode(User.self, from: data)
                completion(.success(decoded.role))
            } catch {
                print("❌ Decoding error:", error)
                completion(.failure(error))
            }
        }.resume()
    }

    func fetchUserBookings(isAdmin: Bool, completion: @escaping (Result<[Booking], Error>) -> Void) {
        let path = "bookings"
        let url = baseURL.appendingPathComponent(path)
        var request = URLRequest(url: url)
        request.httpMethod = "GET"
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")

        URLSession.shared.dataTask(with: request) { data, response, error in
            print(String(data: data ?? Data(), encoding: .utf8) ?? "No data")
            if let error = error {
                completion(.failure(error))
                return
            }
            guard let data = data else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }
            do {
i                let decoder = User.decoder
                let bookings = try decoder.decode([Booking].self, from: data)
                completion(.success(bookings))
            } catch {
                completion(.failure(error))
            }
        }.resume()
    }
    func fetchAllBookings(completion: @escaping (Result<[Booking], Error>) -> Void) {
        fetchUserBookings(isAdmin: true, completion: completion)
    }
    func fetchBookedSlots(for roomID: UUID, date: Date, completion: @escaping (Result<[String], Error>) -
> Void) {
        let formatter = ISO8601DateFormatter()
        let dateString = formatter.string(from: date)
```

```
        guard let url = URL(string: baseURL.absoluteString + "/quest-
rooms/\(roomID)/bookings?date=\(dateString)") else {
            completion(.failure(URLError(.badURL)))
            return
        }
        var request = URLRequest(url: url)
        request.httpMethod = "GET"
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")
        URLSession.shared.dataTask(with: request) { data, _, error in
            if let error = error {
                completion(.failure(error))
                return
            }

            guard let data = data else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }
            do {
                struct BookedQuestRoomShort: Decodable {
                    let timeSlot: String
                }

                let bookings = try JSONDecoder().decode([BookedQuestRoomShort].self, from: data)
                let slots = bookings.map { $0.timeSlot }
                completion(.success(slots))
            } catch {
                completion(.failure(error))
            }
        }.resume()
    }
    func createBooking(roomID: UUID, date: Date, timeSlot: String, numberOfPlayers: Int, completion:
@escaping (Result<Void, Error>) -> Void) {
        let url = baseURL.appendingPathComponent("bookings")
        var request = URLRequest(url: url)
        request.httpMethod = "POST"
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")
        let formatter = DateFormatter()
        formatter.dateFormat = "yyyy-MM-dd"

        let payload: [String: Any] = [
            "questRoomID": roomID.uuidString,
            "date": formatter.string(from: date),
            "timeSlot": timeSlot,
            "numberOfPlayers": numberOfPlayers
        ]

        do {
            request.httpBody = try JSONSerialization.data(withJSONObject: payload)
        } catch {
            completion(.failure(error))
            return
        }

        URLSession.shared.dataTask(with: request) { _, response, error in
            if let error = error {
```

```
            completion(.failure(error))
        } else {
            completion(.success(()))
        }
    }.resume()
}

func registerUser(
    token: String,
    email: String,
    uid: String,
    name: String,
    phoneNumber: String,
    birthday: Date,
    completion: @escaping (Result<Void, Error>) -> Void
) {
    let url = baseURL.appendingPathComponent("users")
    var request = URLRequest(url: url)
    request.httpMethod = "POST"
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")

    let isoFormatter = ISO8601DateFormatter()
    let birthdayString = isoFormatter.string(from: birthday)

    let body: [String: Any] = [
        "email": email,
        "firebaseUID": uid,
        "role": "client",
        "name": name,
        "phoneNumber": phoneNumber,
        "birthday": birthdayString
    ]

    do {
        request.httpBody = try JSONSerialization.data(withJSONObject: body)
    } catch {
        completion(.failure(error))
        return
    }

    URLSession.shared.dataTask(with: request) { _, response, error in
        if let error = error {
            completion(.failure(error))
            return
        }

        guard let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200 else {
            completion(.failure(URLError(.badServerResponse)))
            return
        }

        completion(.success(()))
    }.resume()
}

func updateUserProfile(token: String, name: String, phoneNumber: String, completion: @escaping
```

```
(Result<Void, Error>) -> Void) {
        let url = baseURL.appendingPathComponent("/me")
        var request = URLRequest(url: url)
        request.httpMethod = "PATCH"
        request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")

        let body = ["name": name, "phoneNumber": phoneNumber]

        do {
            request.httpBody = try JSONEncoder().encode(body)
        } catch {
            completion(.failure(error))
            return
        }

        URLSession.shared.dataTask(with: request) { _, response, error in
            if let error = error {
                completion(.failure(error))
            } else if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode != 200 {
                completion(.failure(URLError(.badServerResponse)))
            } else {
                completion(.success(()))
            }
        }.resume()
    }

    // MARK: - Upload Single Image (multipart/form-data)
    func uploadImage(image: UIImage, token: String, completion: @escaping (Result<String, Error>) ->
Void) {
        let url = baseURL.appendingPathComponent("images/upload")
        var request = URLRequest(url: url)
        request.httpMethod = "POST"
        request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
        let boundary = "Boundary-\(UUID().uuidString)"
        request.setValue("multipart/form-data; boundary=\(boundary)", forHTTPHeaderField: "Content-Type")

        var body = Data()
        // Add image part
        if let imageData = image.jpegData(compressionQuality: 0.8) {
            body.append("--\(boundary)\r\n")
            body.append("Content-Disposition: form-data; name=\"image\"; filename=\"upload.jpg\"\r\n")
            body.append("Content-Type: image/jpeg\r\n\r\n")
            body.append(imageData)
            body.append("\r\n")
        }
        body.append("--\(boundary)--\r\n")
        request.httpBody = body

        URLSession.shared.dataTask(with: request) { data, response, error in
            DispatchQueue.main.async {
                if let error = error {
                    completion(.failure(error))
                    return
                }
                guard let httpResponse = response as? HTTPURLResponse, let data = data else {
                    completion(.failure(URLError(.badServerResponse)))
```

```
                return
            }
            if (200...299).contains(httpResponse.statusCode) {
                // Очікуємо відповідь типу {"url": "..."}
                do {
                    if let dict = try JSONSerialization.jsonObject(with: data) as? [String: Any],
                       let url = dict["url"] as? String {
                        completion(.success(url))
                    } else {
                        completion(.failure(NSError(domain: "InvalidResponse", code: -1)))
                    }
                } catch {
                    completion(.failure(error))
                }
            } else {
                let responseText = String(data: data, encoding: .utf8) ?? "No response body"
                print("Server error: Status \(httpResponse.statusCode), Response: \(responseText)")
                completion(.failure(NSError(domain: "ServerError", code: httpResponse.statusCode)))
            }
        }
    }.resume()
}

/// Створення кімнати через JSON (з масивом imageURLs)
func createQuestRoom(title: String, description: String, location: String, difficulty: Difficulty, maxPlayers:
Int, maxPrice: Float, imageURLs: [String], token: String, completion: @escaping (Result<Void, Error>) ->
Void) {
    let url = baseURL.appendingPathComponent("quest-rooms")
    var request = URLRequest(url: url)
    request.httpMethod = "POST"
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    let payload: [String: Any] = [
        "title": title,
        "description": description,
        "location": location,
        "difficulty": difficulty.rawValue,
        "maxPlayers": maxPlayers,
        "maxPrice": maxPrice,
        "imageURLs": imageURLs
    ]
    do {
        request.httpBody = try JSONSerialization.data(withJSONObject: payload)
    } catch {
        completion(.failure(error))
        return
    }
    URLSession.shared.dataTask(with: request) { data, response, error in
        DispatchQueue.main.async {
            if let error = error {
                completion(.failure(error))
                return
            }
            guard let httpResponse = response as? HTTPURLResponse else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }
```

```
            if (200...299).contains(httpResponse.statusCode) {
                completion(.success(()))
            } else {
                let responseText = data.flatMap { String(data: $0, encoding: .utf8) } ?? "No response body"
                print("Server error: Status \(httpResponse.statusCode), Response: \(responseText)")
                completion(.failure(NSError(domain: "ServerError", code: httpResponse.statusCode)))
            }
        }
    }.resume()
}

/// Оновлення кімнати через PATCH JSON (з масивом imageURLs)
func updateQuestRoom(id: UUID, title: String, description: String, location: String, difficulty: Difficulty,
maxPlayers: Int, maxPrice: Float, imageURLs: [String], token: String, completion: @escaping (Result<Void,
Error>) -> Void) {
    let url = baseURL.appendingPathComponent("quest-rooms/\(id)")
    var request = URLRequest(url: url)
    request.httpMethod = "PATCH"
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    let payload: [String: Any] = [
        "title": title,
        "description": description,
        "location": location,
        "difficulty": difficulty.rawValue,
        "maxPlayers": maxPlayers,
        "maxPrice": maxPrice,
        "imageURLs": imageURLs
    ]
    do {
        request.httpBody = try JSONSerialization.data(withJSONObject: payload)
    } catch {
        completion(.failure(error))
        return
    }
    URLSession.shared.dataTask(with: request) { data, response, error in
        DispatchQueue.main.async {
            if let error = error {
                completion(.failure(error))
                return
            }
            guard let httpResponse = response as? HTTPURLResponse else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }
            if (200...299).contains(httpResponse.statusCode) {
                completion(.success(()))
            } else {
                let responseText = data.flatMap { String(data: $0, encoding: .utf8) } ?? "No response body"
                print("Server error: Status \(httpResponse.statusCode), Response: \(responseText)")
                completion(.failure(NSError(domain: "ServerError", code: httpResponse.statusCode)))
            }
        }
    }.resume()
}
func loadImages(from relativePaths: [String], completion: @escaping ([UIImage]) -> Void) {
    let group = DispatchGroup()
```

```
        var images: [UIImage] = []
        let base = self.baseURL
        for path in relativePaths {
            group.enter()
            let fullPath: String
            if path.hasPrefix("http") {
                fullPath = path
            } else if path.hasPrefix("/") {
                fullPath = base.absoluteString + path
            } else {
                fullPath = base.appendingPathComponent(path).absoluteString
            }
            guard let url = URL(string: fullPath) else {
                group.leave()
                continue
            }
            print("🔗 Image URL: \(url.absoluteString)")
            URLSession.shared.dataTask(with: url) { data, _, _ in
                defer { group.leave() }
                if let data = data, let image = UIImage(data: data) {
                    images.append(image)
                }
            }.resume()
        }

        group.notify(queue: .main) {
            completion(images)
        }
    }

    func fullImageURLs(from paths: [String]) -> [URL] {
        return paths.compactMap { baseURL.appendingPathComponent($0) }
    }
    func fetchUsers(completion: @escaping (Result<[User], Error>) -> Void) {
        let url = baseURL.appendingPathComponent("users")
        var request = URLRequest(url: url)
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")
        request.httpMethod = "GET"
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")
        URLSession.shared.dataTask(with: request) { data, response, error in
            if let error = error {
                completion(.failure(error))
                return
            }
            if let httpResponse = response as? HTTPURLResponse,
!(200...299).contains(httpResponse.statusCode) {
                print("❌ Server returned status code: \(httpResponse.statusCode)")
                completion(.failure(URLError(.badServerResponse)))
                return
            }
            guard let data = data else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }
            print("📄 /users response: \(String(data: data, encoding: .utf8) ?? "nil")")
            do {
                let users = try User.decoder.decode([User].self, from: data)
```

```
            completion(.success(users))
        } catch {
            completion(.failure(error))
        }
    }.resume()
}
func deleteQuestRoom(id: UUID, token: String, completion: @escaping (Result<Void, Error>) -> Void) {
    let url = baseURL.appendingPathComponent("quest-rooms/\(id)")
    var request = URLRequest(url: url)
    request.httpMethod = "DELETE"
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    URLSession.shared.dataTask(with: request) { _, response, error in
        if let error = error {
            completion(.failure(error))
            return
        }
        guard let httpResponse = response as? HTTPURLResponse else {
            completion(.failure(NSError(domain: "InvalidResponse", code: -1)))
            return
        }

        if (200...299).contains(httpResponse.statusCode) {
            completion(.success(()))
        } else {
            completion(.failure(NSError(domain: "ServerError", code: httpResponse.statusCode)))
        }
    }.resume()
}

func promoteUser(id: UUID, token: String, completion: @escaping (Result<Void, Error>) -> Void) {
    let url = baseURL.appendingPathComponent("users/\(id)/role")
    var request = URLRequest(url: url)
    request.httpMethod = "PATCH"
    request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")

    let body = ["role": "admin"]
    request.httpBody = try? JSONEncoder().encode(body)

    URLSession.shared.dataTask(with: request) { _, response, error in
        if let error = error {
            completion(.failure(error))
            return
        }

        guard let httpResponse = response as? HTTPURLResponse,
(200...299).contains(httpResponse.statusCode) else {
            completion(.failure(URLError(.badServerResponse)))
            return
        }

        completion(.success(()))
    }.resume()
}


func fetchLocations(completion: @escaping (Result<[String], Error>) -> Void) {
```

```
        let url = baseURL.appendingPathComponent("locations")
        var request = URLRequest(url: url)
        request.httpMethod = "GET"

        if let token = UserDefaults.standard.string(forKey: "idToken") {
            request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
        }

        URLSession.shared.dataTask(with: request) { data, _, error in
            if let error = error {
                completion(.failure(error))
                return
            }

            guard let data = data else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }

            do {
                let locations = try JSONDecoder().decode([String].self, from: data)
                completion(.success(locations))
            } catch {
                completion(.failure(error))
            }
        }.resume()
    }

    func fetchAvailableSlots(for roomID: UUID, date: Date, completion: @escaping (Result<[String], Error>)
-> Void) {
        let formatter = DateFormatter()
        formatter.dateFormat = "yyyy-MM-dd"
        let dateString = formatter.string(from: date)

        var components = URLComponents(string: baseURL.appendingPathComponent("quest-
rooms/\(roomID)/availableSlots").absoluteString)
        components?.queryItems = [URLQueryItem(name: "date", value: dateString)]

        guard let url = components?.url else {
            completion(.failure(URLError(.badURL)))
            return
        }

        var request = URLRequest(url: url)
        request.httpMethod = "GET"
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")

        URLSession.shared.dataTask(with: request) { data, _, error in
            if let error = error {
                completion(.failure(error))
                return
            }

            guard let data = data else {
                completion(.failure(URLError(.badServerResponse)))
                return
            }
```

```
            do {
                let slots = try JSONDecoder().decode([String].self, from: data)
                completion(.success(slots))
            } catch {
                completion(.failure(error))
            }
        }.resume()
    }

    func editBooking(id: UUID, timeSlot: String, numberOfPlayers: Int, completion: @escaping
(Result<Void, Error>) -> Void) {
        let url = baseURL.appendingPathComponent("bookings/\(id)")
        var request = URLRequest(url: url)
        request.httpMethod = "PUT"
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")
        let body: [String: Any] = [
            "timeSlot": timeSlot,
            "numberOfPlayers": numberOfPlayers
        ]
        request.httpBody = try? JSONSerialization.data(withJSONObject: body)

        URLSession.shared.dataTask(with: request) { _, response, error in
            if let error = error {
                completion(.failure(error))
            } else if let httpResponse = response as? HTTPURLResponse,
(200...299).contains(httpResponse.statusCode) {
                completion(.success(()))
            } else {
                completion(.failure(error!))
            }
        }.resume()
    }

    func cancelBooking(id: UUID, completion: @escaping (Result<Void, Error>) -> Void) {
        let url = baseURL.appendingPathComponent("bookings/\(id)")
        var request = URLRequest(url: url)
        request.httpMethod = "DELETE"
        request.setValue("Bearer \(idToken)", forHTTPHeaderField: "Authorization")
        URLSession.shared.dataTask(with: request) { _, response, error in
            if let error = error {
                completion(.failure(error))
            } else if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 204 {
                completion(.success(()))
            } else {
                completion(.failure(error!))
            }
        }.resume()
    }
}
class AdminViewModel: ObservableObject {
    @Published var users: [User] = []
    @Published var errorMessage: String?
    func fetchUsers() {
        NetworkService.shared.fetchUsers() { [weak self] result in
            switch result {
```

```
            case .success(let users):
                DispatchQueue.main.async {
                    self?.users = users
                }
            case .failure(let error):
                DispatchQueue.main.async {
                    self?.errorMessage = error.localizedDescription
                }
            }
        }
    }

    func promoteUser(userID: UUID, token: String) {
        NetworkService.shared.promoteUser(id: userID, token: token) { result in
            switch result {
            case .success:
                self.fetchUsers()
            case .failure(let error):
                self.errorMessage = error.localizedDescription
            }
        }
    }
}
```

85

Додаток Г (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка мобільного застосунку для бронювання квест-кімнат

Тип роботи: бакалаврська кваліфікаційна робота

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ. к. АІІТ, ФІІТА, 1ІСТ-21Б

(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,48 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☑ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Олег БІСІКАЛО, зав. каф. АІІТ (підпис)

(прізвище, ініціали, посада)

Любов ЯНЧУК, секретар. каф. АІІТ (підпис)

(прізвище, ініціали, посада)

Особа, відповідальна за перевірку (підпис) Костянтин ОВЧИННИКОВ

(прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Роман КВЄТНИЙ, д. т. н., проф. каф. АІІТ

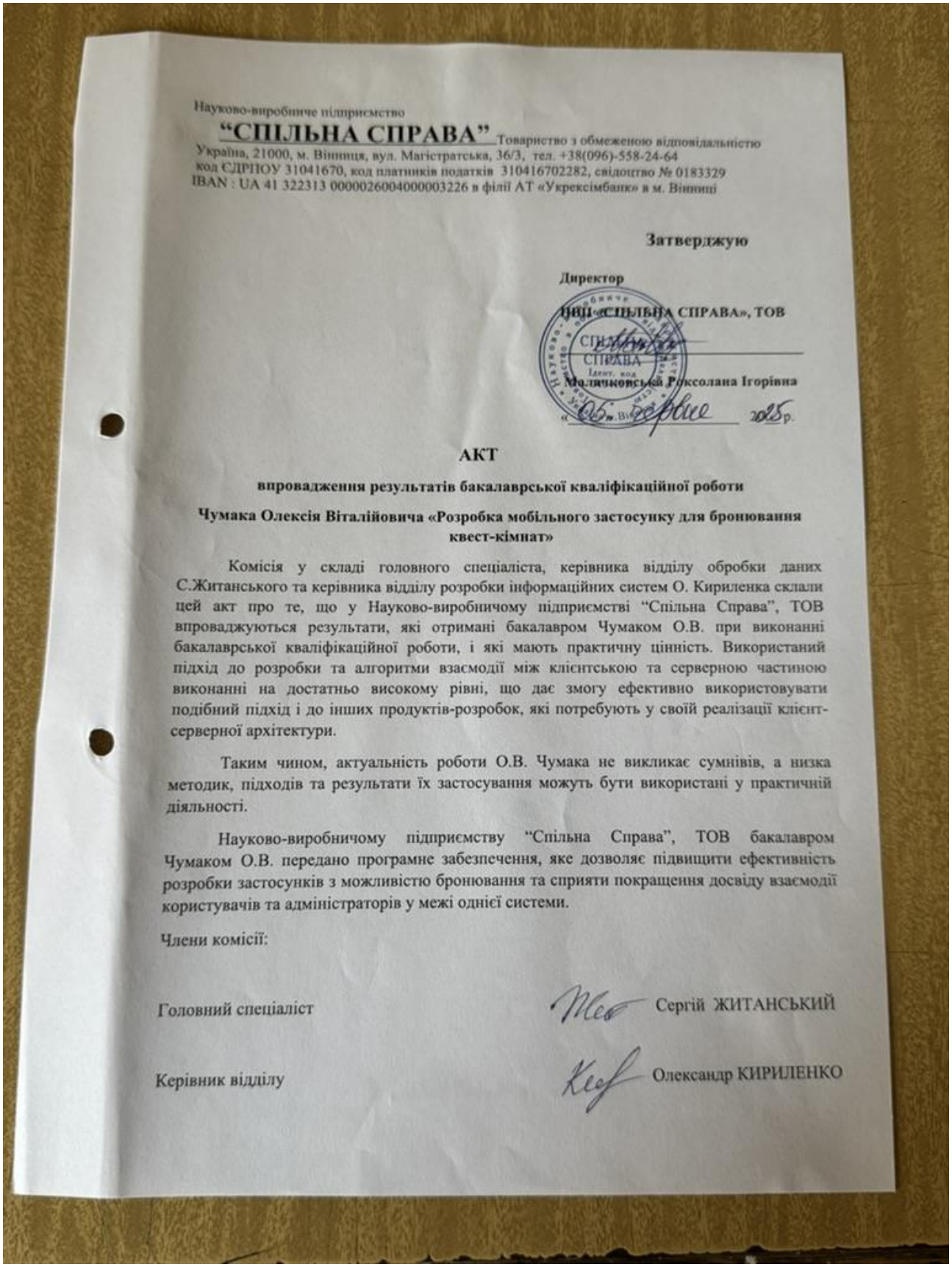
(прізвище, ініціали, посада)

Здобувач (підпис) Олексій ЧУМАК

(прізвище, ініціали)

Додаток Д (довідниковий)

Акт впровадження результатів роботи в ТОВ «СПІЛЬНА СПРАВА»



Науково-виробниче підприємство
"СПІЛЬНА СПРАВА" Товариство з обмеженою відповідальністю
Україна, 21000, м. Вінниця, вул. Магістратська, 36/3, тел. +38(096)-558-24-64
код ЄДРПОУ 31041670, код платників податків 310416702282, свідоцтво № 0183329
IBAN : UA 41 322313 0000026004000003226 в філії АТ «Укрексімбанк» в м. Вінниці

Затверджую

Директор

НВП «СПІЛЬНА СПРАВА», ТОВ

Маличковська Роксолана Ігорівна

«05» вересня 2025 р.

АКТ

впровадження результатів бакалаврської кваліфікаційної роботи

Чумака Олексія Віталійовича «Розробка мобільного застосунку для бронювання квест-кімнат»

Комісія у складі головного спеціаліста, керівника відділу обробки даних С.Житанського та керівника відділу розробки інформаційних систем О. Кириленка склали цей акт про те, що у Науково-виробничому підприємстві "Спільна Справа", ТОВ впроваджуються результати, які отримані бакалавром Чумаком О.В. при виконанні бакалаврської кваліфікаційної роботи, і які мають практичну цінність. Використаний підхід до розробки та алгоритми взаємодії між клієнтською та серверною частиною виконанні на достатньо високому рівні, що дає змогу ефективно використовувати подібний підхід і до інших продуктів-розробок, які потребують у своїй реалізації клієнт-серверної архітектури.

Таким чином, актуальність роботи О.В. Чумака не викликає сумнівів, а низка методик, підходів та результати їх застосування можуть бути використані у практичній діяльності.

Науково-виробничому підприємству "Спільна Справа", ТОВ бакалавром Чумаком О.В. передано програмне забезпечення, яке дозволяє підвищити ефективність розробки застосунків з можливістю бронювання та сприяти покращення досвіду взаємодії користувачів та адміністраторів у межі однієї системи.

Члени комісії:

Головний спеціаліст Сергій ЖИТАНСЬКИЙ

Керівник відділу Олександр КИРИЛЕНКО