

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ
ОБЛІКУ БРОНІЮВАНЬ В ЗАКЛАДАХ ХАРЧУВАННЯ
SOFTWARE IMPLEMENTATION OF AN APPLICATION
FOR TRACKING FOOD ESTABLISHMENT RESERVATIONS

Виконав: студент(ка) 4 курсу, групи КНТ-128

Спеціальності 121 Інженерія програмного

забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Дем'янова Д.К.

(прізвище та ініціали)

Керівник Федорончак Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Дем'янової Дар'ї Костянтинівни

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація застосунку для обліку бронювань в закладах харчування Software Implementation of an Application for Tracking Food Establishment Reservations

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

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 13 червня 2022 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис програми. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Федорончак Т.В., доцент		
Нормоконтроль	Белова А.В., асистент		

7. Дата видачі завдання “ 28 ” лютого 2020 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

_____ Дем'янова Д.К.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

_____ Федорончак Т.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 92 с., 9 табл., 48 рис., 3 дод., 13 джерел.

FLUTTER, DART, MOBILE, ОБЛІК, БРОНЮВАННЯ, РЕСТОРАН, КЛІЄНТИ, CLEAN ARCHITECTURE, BLOC.

Об'єкт дослідження – автоматизація обліку бронювань.

Предмет дослідження – мобільний застосунок “Free Table”, для бронювань в закладах харчування.

Мета роботи – розробка мобільного застосунку, що забезпечить зручне створення бронювань в закладах харчування.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування Dart, фреймворк Flutter, сервіс Firebase, персональний комп'ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 10.

Результати. Створено мобільний застосунок, у якому є функціонал бронювання для клієнтів та керування бронюваннями для адміністраторів.

Висновки. Розроблено програмну систему, яка автоматизує та прискорює процес бронювання столиків в закладах харчування.

Галузь використання – бронювання в закладах харчування.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 92 pages, 48 figures, 9 tables, 3 appendixes, 13 sources.

FLUTTER, DART, MOBILE, ACCOUNTING, BOOKING, RESTAURANT, CLIENTS, CLEAN ARCHITECTURE, BLOC.

Object of study is a software for the accounting bookings.

Subject of study is a mobile application for reservations in restaurants.

The purpose of the work is to create a mobile application that will provide convenient creation of reservations in food establishments.

Materials, methods, and tools: object-oriented programming, Microsoft Kinect technology and device, programming Dart, framework Flutter, service Firebase, IBM-compatible personal computer with an Intel Core i5 processor under the control of the Microsoft Windows 10 operating system.

Results. The mobile application is created having booking functionality for customers and booking management for administrators

Conclusions. A software system that automates and accelerates the process of booking tables in food establishments has been developed.

The field of use is booking tables in food establishments.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Огляд існуючих методів вирішення завдання.....	10
1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання.....	12
1.3 Постановка завдання.....	17
2 Опис програми.....	18
2.1 Вибір інструментарію розробки програмного забезпечення	18
2.2 Проектування архітектури	21
2.3 Вибір інструментарію для бази даних	23
2.4 Модулі системи, що розроблюється	24
2.5 Проектування функціональних вимог до програмного забезпечення ..	25
2.6 Висновки за розділом 2	26
3 Основні рішення щодо реалізації компонентів системи.....	27
3.1 Опис модулів системи	27
3.2 Алгоритм функціонування розробленої програми.....	32
3.3 Опис розроблених інтерфейсів	36
3.4 Опис бази даних	43
3.5 Висновки за розділом 3	44
4 Експлуатація, тестування та експериментальне дослідження програми	45
4.1 Призначення та умови застосування програми	45
4.2 Виконання програми.....	45
4.3 Тестування програми	55
4.4 Висновки за розділом 4	58
Висновки	59
Перелік джерел посилань	60
Додаток А Технічне завдання	61

Додаток Б Текст програми	65
Додаток В Слайди презентації	85

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
BLoC	– Business Logic Component;
IDE	– Integrated development environment;
JSON	– JavaScript Object Notation;
NoSQL	– Not only SQL;
SQL	– Structured Query Language;
UI	– User Interface;
БД	– База даних;
ПП	– Програмний продукт;
ПЗ	– Програмне забезпечення;
ПК	– Персональний комп'ютер.

ВСТУП

Системи обліку є невід’ємною частиною багатьох закладів харчування, а також в деякій мірі повинні бути об’єднані із системами для клієнтів. Задля спрощення роботи та точності збереження даних використовуються відповідні застосунки.

Більшість користувачів в таких випадках надають перевагу мобільним застосункам через зручність керування даними та незалежність від браузерів та сумісність з ними. А застосунок, що розроблюється за допомогою однієї кодової бази одразу для декількох платформ, значно скорочує час на повну розробку в тій мірі, щоб задовольнити максимальну кількість користувачів.

Актуальність роботи полягає у створенні крос-платформенного застосунку для адміністратора та клієнтів закладів харчування задля ведення обліку бронювань та доступності до функціоналу в будь-який момент для обох типів користувачі. Цей застосунок буде поєднувати в собі декілька призначень, які зазвичай виконують різні застосунки, а також дозволяє скоротити час на пошук закладу і виявлення вільних місць на певний час.

Існуючі мобільні або веб-застосунки задовольняють потреби користувачів не в повній мірі або не є доступними на всіх платформах. Тому було вирішено створити програмний продукт, який забезпечить функціоналом бронювання клієнтів закладів і функціоналом обліку бронювань та складення зон розташування столиків для адміністраторів.

Всі необхідні дані будуть зберігатися у базі даних (БД), що зробить адміністрування більш зручним та надасть більший арсенал засобів для захисту персональних даних та використання їх у роботі програми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Мобільні застосунки все частіше стають звичними і бажаними речами для перегляду будь-яких сервісів, соцмереж, магазинів тощо. Маючи майже безперервний доступ до встановлених застосунків, користувачі частіше дивляться той чи інший контент, роблять замовлення, переглядають рекламу – тобто дуже зростають потецнійні споживачі тих чи інших послуг в інфопросторі. Мобільний застосунок є потужним інструментом для досягання бізнес-цілей та збільшення охоплень користування.

Предметна область даної дипломної роботи полягає в створенні такого застосунку для бронювання столиків. Предметом дослідження є мобільний застосунок для обліку бронювань столиків у закладах харчування.

Переваги цього застосунку полягає в доступності Інтернету, можливості в будь-який момент переглянути актуальні заклади харчування, вільні столики для бронювання на певний час. Часто у людей є бажання переглянути завантаженість закладів та існування вільних столиків без додаткового дзвінка, а у закладів є необхідність зберігати та переглядати заявки на бронь. За допомогою даного застосунку з'являється можливість вести облік закладів, їхній розклад бронювань, щоб відвідувачі мали доступ і могли зробити бронювання, а для адміністратора закладу спрощується робота ведення обліку бронювань і розташування столиків.

Потенційними користувачами та цільовою аудиторією є адміністратори закладів харчування та клієнти таких закладів.

1.1 Огляд існуючих методів вирішення завдання

Для веденні обліку найчастіше створюються мобільні застосунки або вебсайти для обслуговування клієнтів шляхом онлайн бронювання столиків, встановлення і поліпшення бізнес-процесів і подальшого аналізу результатів.

Мобільний застосунок є різновидом програмного забезпечення, призначеного для запуску на мобільному телефоні, наприклад, смартфоні або планшетному ПК (персональному комп'ютеру). Мобільні застосунки часто служать для надання клієнтам адміністрування, порівнянню з тими, які доступні на ПК. Програми – це загалом невеликі окремі програмні блоки з обмеженою ємністю. Таке використання програмного забезпечення спочатку підтримувалося Apple Inc., також її App Store, який пропонує величезну кількість застосувань для iPhone, iPad та iPod Touch. За всіх рівних умов кожне універсальне застосування надає роздільну та обмежену корисність [1].

Існуючі в настільних застосунках мобільні застосунки були програмними одиницями з обмеженою кількістю функціональності. З появою смартфонів, за ініціативи Apple Inc., мобільні застосунки з простих застосунків що імітують програми, написані для настільних комп'ютерів, тепер стали застосунками дуже складними з точки зору використання функцій, доступних виключно за допомогою мобільного телефону. Функції на основі розташування, чутливість сенсорного екрану, подвійні камери, голосова аналітика відкрили абсолютно новий вимір застосунків. З експоненційним зростанням продажів смартфонів і безперервний попит з боку людей, з'явилися численні студії, які розробляли застосунки для ігри, покупки в Інтернеті, банківські справи, бронювання подорожей і готелів, пошук на основі GPS і карт, огляди сайтів і соціальні мережі. На даний момент найпопулярніші платформи для смартфонів підтримують мобільні застосунки Android, iOS та Windows Mobile. Застосунки в першу чергу розповсюджується через App Stores, що належать постачальникам платформ, таким як Google Play, Apple App Store і Windows Phone Store. Більше того, багато популярних програм тепер попередньо завантажено в пристрої виробниками. Для не користувачів iOS також існує кілька альтернатив Магазини застосунків, такі як Amazon App Store або Vodafone App Store [2].

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

Так як кількість подібних за функціоналом застосунків виявилось не багато, були розглянуті аналоги веб-застосунків та мобільних застосунків для обслуговування закладів харчування. Щоб бути більш об'єктивними, було окреслено такі основні критерії для порівняння програмних засобів:

- можливість реєстрації облікового запису;
- зберігання даних про власні бронювання;
- доступ адміністратора до створення акаунту закладу;
- можливість скласти приблизну схему розташування місць;
- можливість переглядати і редагувати заявки;
- доречність дизайну;
- адаптивність до різних девайсів;
- доступність на різних платформах;
- додаткові можливості.

1.2.1 thestol

«thestol.» - це мобільний застосунок для перегляду різних закладів та бронювання у Києві (рис. 1.1). З основних переваг він має можливість переглянути основні характеристики закладів, фільтрація даних для перегляду ресторанів за різними кухнями, перегляд рейтингу та відгуків, легкий та зрозумілий інтерфейс користувача. Головною метою є бронювання столиків на конкретний час у закладі.

З основних недоліків – це те, що застосунок доступний тільки для завантаження в AppStore, тобто недоступний для телефонів с ОС Android. Також немає змоги передивитись розташування столиків в різних зонах закладу [3].

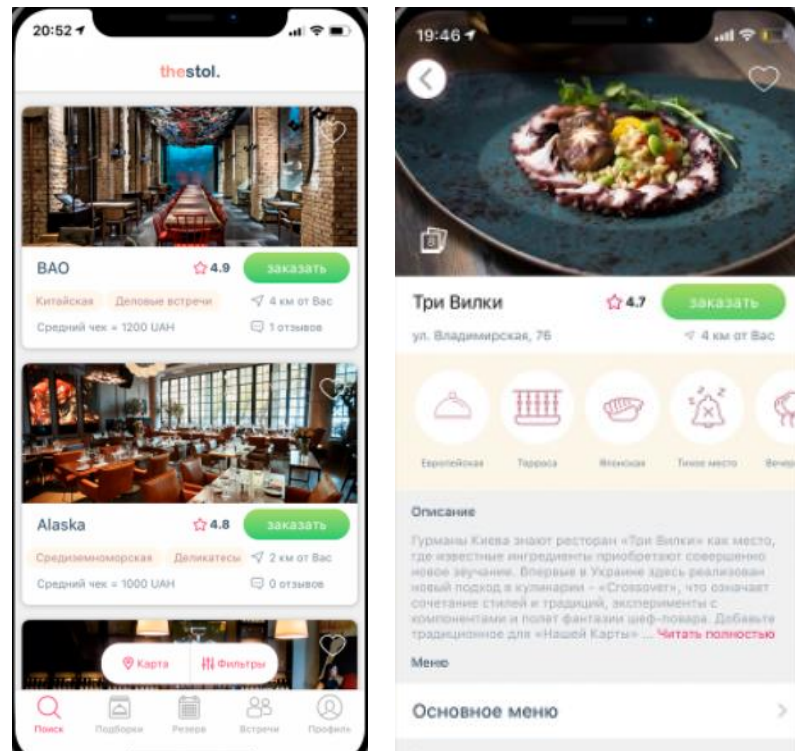


Рисунок 1.1 – Мобільний застосунок thestol (головна сторінка та сторінка закладу) [3]

1.2.2 RestOn

Наступним застосунком є RestOn (рис. 1.2). За допомогою нього, щоб замовити столик у будь-якому закладі, достатньо мати під рукою будь-який пристрій із виходом до Інтернету. Розібратися у зручній навігації сайту зможе будь-який користувач. Власники пристроїв iPhone, телефонів та планшетів на платформі Android зможуть встановити собі мобільний застосунок RestOn та використовувати його абсолютно безкоштовно у будь-який час у будь-якому місці [4].

У застосунку RestOn представлена зручна форма пошуку, завдяки якій ви зможете підібрати заклад, що відповідає вашому формату зустрічі, вашим особистим уподобанням та бюджету. Вибравши заклад, що підходить найбільше, можна забронювати стіл. Для користувача є можливість реєстрації, але

немає авторизації адміністраторів закладу. Додатково цей застосунок пропонує функцію додавання у вибране закладів та перегляд меню.

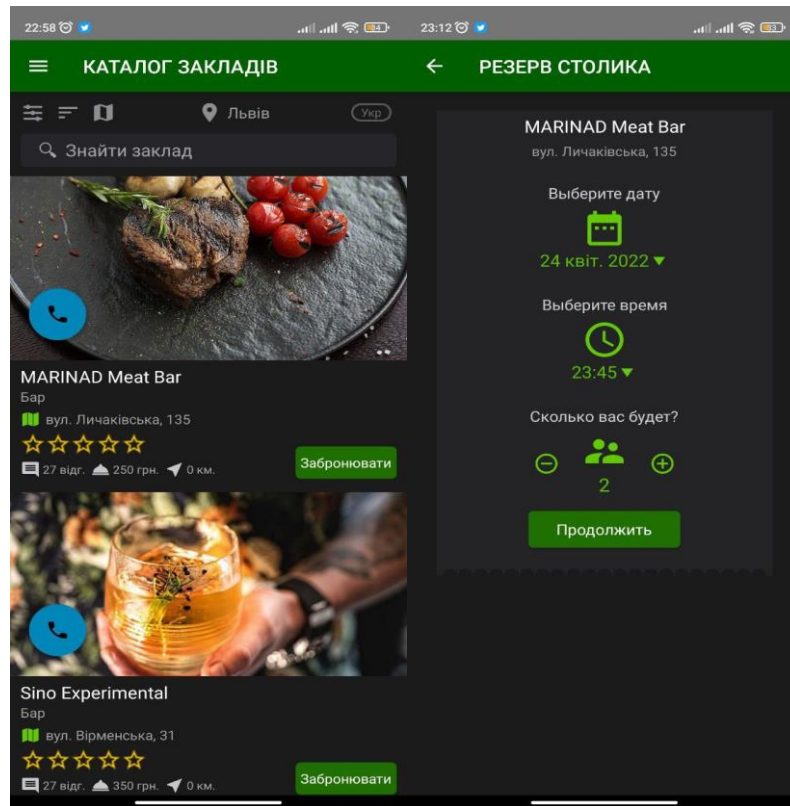


Рисунок 1.2 – Мобільний застосунок RestOn (головна сторінка та сторінка закладу) [4]

1.2.3 Ласун

Вебзастосунки – це програми, доступ до яких здійснюється за допомогою веббраузера через мережу та розроблені за допомогою мов, що підтримуються браузером (наприклад, HTML, JavaScript). Використання вебпрограм залежить від веббраузерів і включає в себе багато знайомих програм, таких як онлайн роздрібні продажі, онлайн-аукціони та вебпошта. Вебзастосунки необхідні у сфері взаємодії між бізнесом через мережі, наприклад, для закордонних компаній, які передають проекти один одному на аутсорсинг. Впровадження інфраструктури вебзастосунків може забезпечити такі життєво важливі процеси, як переказ коштів та оновлення інформації про ціни [5] .

Ласун – це вебзастосунок, який надає багато різного функціоналу для закладів харчування, наприклад: перегляд інформації, новин про заклади, підбірки по різних святах, різноманітні статті та інтерв'ю (рис. 1.3, рис. 1.4). Застосунок також має адмін-панель для адміністраторів, які хочуть розмістити свій заклад для доступу. Бронювання столиків також є одним з аспектів, які складають функціонал. Можна обрати час, кількість відвідувачів і навіть зону (наприклад, літня тераса).

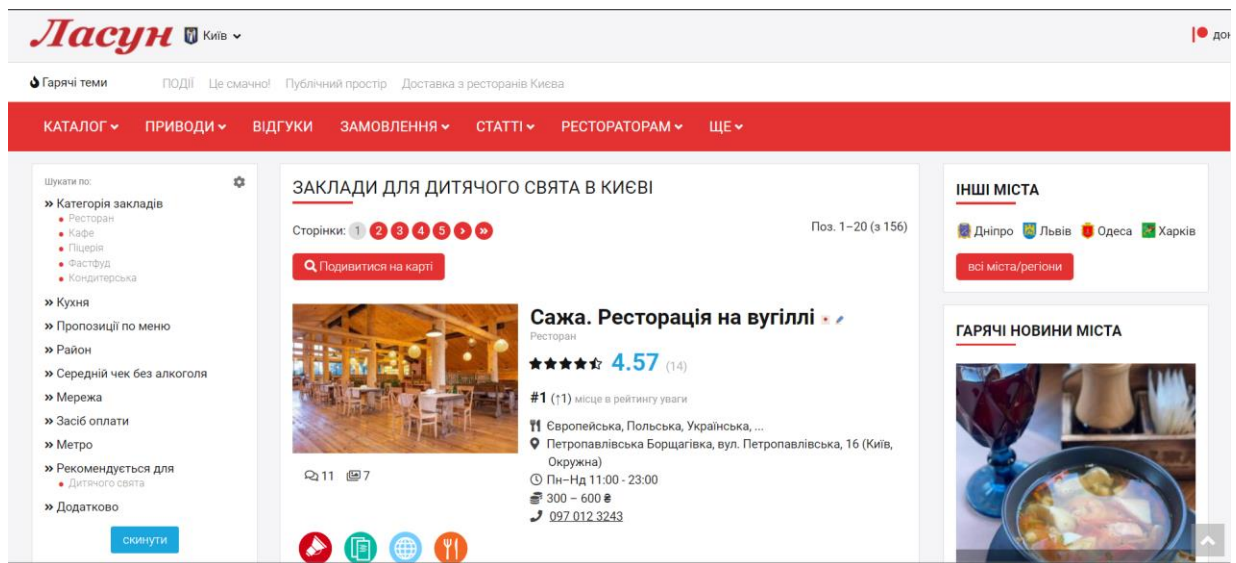


Рисунок 1.3 – Головна сторінка вебзастосунку Ласун [6]

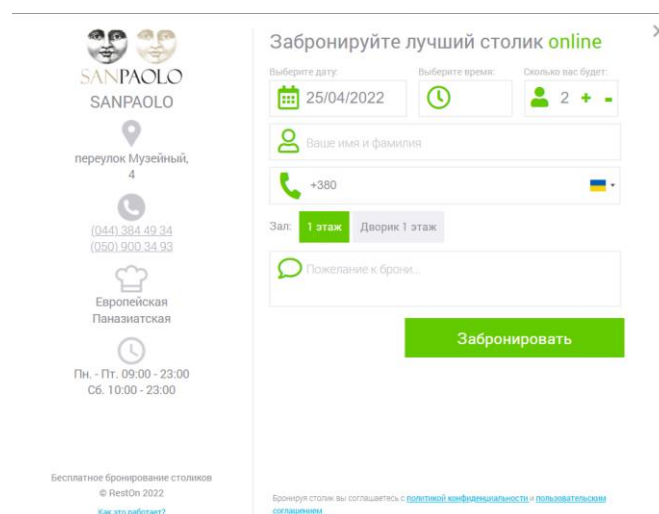


Рисунок 1.4 – Створення бронювання [6]

1.2.4 Порівняльна таблиця програмних рішень

Вище було наведено найбільш популярні на сьогоднішній день програми, які дозволяють вести облік бронювань столиків в закладах. Але крім основних переваг, усі вони мають і недоліки. Наприклад, в деяких програмах відсутній доступ до адміністрування закладів, і не всі програми є кросплатформеними.

Також дійсно існують переваги характерні для усіх програм: добре структурована інформація про самі заклади, їхню кухню, популярність, можливість обрати дату і час для бронювання, створення власного акаунту.

У ході виконання дипломної роботи буде реалізовано програмний продукт (ПП), що буде мати наступні переваги, на відміну від існуючих аналогів: кросплатформеність, доступ адміністраторів до застосунку, можливість користувача переглянути власні бронювання, вибір місця і зони розташування столиків.

Розглянувши кілька існуючих аналогів, було створено таблицю порівняння (таблиця 1.1).

Таблиця 1.1 – Порівняльна таблиця

Критерій оцінювання	Назва застосунку		
	thestol	RestOn	Ласун
Обліковий запис	+	+	+
Дані про власні бронювання	+	+	-
Створення акаунту адміністратора	-	-	+
Схема розташування місць	-	-	+
Можливість переглядати і редагувати заявки	-	-	-
Адаптивність до різних девайсів/платформ	-	+	+-
Облік заявок з боку адміністратора (підтвердження/видалення)	-	-	+

1.3 Постановка завдання

За результатом аналізу аналогів було поставлене завдання роботи - це розробка кросплатформенного мобільного застосунку для бронювання столиків. ПП повинен забезпечувати виконання нижче перерахованих функцій:

- створення профіля користувача;
- перегляд інформації про заклади харчування;
- онлайн бронювання столиків;
- перегляд власних бронювань;
- облік адміністратором бронювань;
- створення адміністратором зон та схем розташування столиків
- редагування адміністратором інформації заклад.

2 ОПИС ПРОГРАМИ

2.1 Вибір інструментарію розробки програмного забезпечення

Існує багато доступних фреймворків для розробки мобільного застосунку. Для Android є нативний фреймворк, заснований на мові Java, а iOS надає фреймворк на основі мови Objective-C/Swift. Однак дуже популярною стає саме кросплатформенна розробка, щоб не робити два застосунки різними мовами і подолати цю складність.

Коли розробники навчилися працювати в мобільній екосистемі, вони дотримувались подібних шаблонів проектування та ідей рамок, які були створені для роботи з веб-екосистемою. Подібно до ускладнень розробки коду для кількох браузерів, у мобільній екосистемі виникла проблема розробки коду, який може працювати як на пристроях iOS, так і на Android, при цьому мрія завжди полягати в тому, щоб мати єдину кодову базу, яка працювала б на всіх пристроях. Існують мобільні фреймворки, які підтримують обидві ОС [7].

2.1.1 Огляд особливостей мови програмування

Flutter є простим і високопродуктивним фреймворком на основі Dart. Наведемо основні властивості цього фреймворку:

- забезпечує високу продуктивність, відтворюючи інтерфейс користувача безпосередньо на полотні операційної системи, а не через нативні компоненти. Flutter також пропонує багато готових до використання віджетів (UI) для створення сучасного застосунка. Ці віджети оптимізовані для мобільного середовища та проектування програми за допомогою віджетів;
- програма Flutter сама по собі є віджетом. Віджети Flutter також підтримують анімацію та жести;
- логіка програми заснована на реактивному програмуванні. Віджет за бажанням може мати стан. Змінюючи стан віджета, Flutter порівню-

ватиме стан віджета (старий і новий) та відображатиме віджет лише з необхідними змінами замість повторного відтворення всього віджета [8].

Наведемо основні переваги мови програмування Dart:

- компіляція Dart. Dart є достатньо гнучким, щоб забезпечити різні способи виконання коду, тому Flutter використовує завчасну компіляцію Dart Ahead Of Time (AOT) з урахуванням продуктивності під час компіляції випускної версії програми, а також використовує тільки Just In Time (JIT) з посекундною компіляцією коду під час розробки, з метою швидкого зворотного зв'язку для змін коду;
- збір сміття: Flutter використовує потік з короткоживучими об'єктами, а це означає багато короточасних виділень пам'яті;
- легко освоїти: Dart — це гнучка, надійна, сучасна та просунута мова. Ця мова була адаптована, оскільки Flutter став популярнішим, з великою кількістю синтаксичного цукру та фундаментальними змінами дизайну, які дійсно допомагають у створенні програми Flutter;
- декларативний інтерфейс користувача: у Flutter ви використовуєте декларативний стиль для розміщення віджетів, що означає, що віджети є незмінними і є лише легкими «кресленнями» [7].

2.1.2 Огляд особливостей обраного середовища розробки

Інтегроване середовище розробки (IDE) — це програмне середовище, яка надає програмістам комплексні засоби для розробки програмного забезпечення. IDE зазвичай складається принаймні з редактора вихідного коду, засобів автоматизації збірки та налагоджувача [9].

Для того, щоб обрати потрібне IDE було порівняно VS Code, Android Studio та IntelliJ.

VS Code — це легкий редактор, який має чудову підтримку Flutter, хорошу розширюваність і широко використовується в спільноті розробників.

Переваги використання VS Code як IDE для Flutter полягають у тому, що він:

- запускається майже на всіх платформах;
- має чудову продуктивність ;
- має величезну базу користувачів, тож у разі виникнення проблем

можна легко отримати допомогу.

Але ви також повинні знати деякі мінуси, а саме:

- він не може відокремлювати вкладки на окремі вікна, і це не зміниться найближчим часом;
- розширення (extensions) можуть викликати проблеми з продуктивністю.

Android Studio, безумовно, більш важкий ПП, ніж VS Code. Але додаткові функції та набір функцій можуть спростити роботу з вашими програмами. Крім того, можливість відокремлення вкладок IDE на окремі вікна покращує розробку. Однак, якщо ви просто потребуєте мінімальних вимог, Android Studio, ймовірно, буде повільно використовуватися, а іноді й не реагувати. У цьому випадку вам, ймовірно, буде краще мати легшу IDE, як-от VS Code.

IntelliJ IDEA — ще один хороший варіант для розробки на Flutter. Однак, оскільки він має ту саму основу, що й Android Studio (IntelliJ IDEA), вам, ймовірно, краще використовувати Android Studio. Іноді певні виправлення або проблеми вирішуються в IntelliJ IDEA до того, як вони вирішуються в Android Studio, тому ви можете розглянути можливість їх використання, якщо у вас виникла певна проблема, яку ще не вирішено в Android Studio. Ще одна причина, по якій ви можете розглянути цей варіант, — це якщо у вас уже встановлено цю IDE для роботи над проектами іншими мовами [10].

Підсумовуючі вище згадані переваги й недоліки IDE, було обрано VS Code через гарну продуктивність та швидкий відклик.

Для розробки мобільних застосунків треба обрати не тільки IDE, але й засоби для запуску застосунку – емулятори або реальні телефони. Так як роз-

робка ведеться не з операційної системи MacOS – компіляція програми для IOS та запуск її на симуляторі iPhone не можлива. Тому залишаються варіанти IDE для розробки й тестування на Android. Було обрано відлагодження та тестування на реальному телефоні, так як це полегшує розуміння реальних масштабів об'єктів на екрані і дозволяє повноцінно використовувати весь функціонал мобільного пристрою. Також це не зобов'язує додатково завантажувати емулятори телефонів, що значно економить час та обсяг пам'яті.

2.2 Проектування архітектури

За основу проекту була обрана архітектура, яка називається Clean Architecture. Вона передбачає розподіл всього застосунку на такі шари: domain, data, presentation. Таким чином бізнес-логіка і use cases відділені від преставлення та доступу до даних. Наприклад, за допомогою репозиторіїв UI (user interface) отримує дані для відображення, але звідки і як зв'язались дані – репозиторій не повинен це знати. Доступ до даних можуть встановлювати відповідні сервіси, наприклад application programming interface (API), або локальна БД. Для репозиторіїв важливо тільки отримання даних, а не ресурс. Так само і сервіс із архітектурного шару data не має уявлення про те, як і де будуть відображені дані. Зберігається так звана модульність застосунку. Така архітектура дозволяє швидко шукати необхідні сервіси та інше, є гнучкою до змін та доповнень і легкою для сприйняття та опанування іншими розробниками.

2.2.1 Керування станом

Стан є основним будівельним блоком вашої програми. Він визначає загальну поведінку вашої програми від початку вашого шляху користувача до того часу, коли користувач закриває/завершує вашу програму. Кожна зміна, яку бачить користувач, є власним станом, і вам потрібно переконатися,

що користувач бачить те, що ви збираєтеся показати. Вам також потрібно переконатися, що при кожній можливій взаємодії з користувачем програма показує дійсні стани. Вона не повинна показувати те, що ви або користувач не очікуєте побачити (виняток, червоний екран, неочікувана чи необроблена помилка тощо). Управління станом — це просто техніка або кілька методів, які використовуються для керування змінами, що відбуваються у вашій програмі. Коли ваша програма стає більшою, вам потрібно застосувати належний підхід до управління станом, щоб мати можливість відстежувати кожну зміну у вашій програмі та змусити застосунок відповідати користувачеві.

Організація state management може включати наступне:

- відповідь на взаємодії з користувачем;
- відстеження даних на різних екранах програми;
- зміна точок даних у програмі в одному місці та обробка відповідей в інших точках програми.

Існують різні методи керування станами у вашій програмі Flutter, починаючи від простих способів, таких як `setState` (внесення змін до інтерфейсу користувача за допомогою виклику функції), до просунутих підходів, таких як `BLoC` (компонент бізнес-логіки) та `Provider` (методика, що широко використовується серед різноманітних застосунків).

`BLoC` означає компонент бізнес-логіки (Business Logic Component). Ця техніка широко використовується у багатьох програмах завдяки її здатності відокремити вашу основну бізнес-логіку від інтерфейсу користувача, мережі або класів репозиторію. `BLoC` працює з потоками і дозволяє надсилати події з вашого інтерфейсу користувача та отримувати стани в інтерфейс користувача, щоб оновлювати зміни на основі взаємодії з користувачем. У середині управління станом `BLoC` є три основні ключові слова: сам блок - `Bloc` (вся ваша бізнес-логіка лежить у цьому класі), стан - `State` (він вказує інтерфейсу користувача, коли його оновлювати), подія - `Event` (ви запускаєте події з класу інтерфейсу користувача, щоб щось зробити всередині вашого блоку) [11].

На діаграмі представлена взаємодія BLoC із іншими шарами архітектури (рис. 2.1).

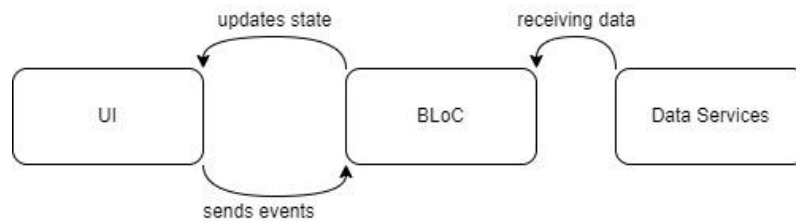


Рисунок 2.1 – Схема роботи BLoC

2.3 Вибір інструментарію для бази даних

Окрім широко вживаних SQL (Structured Query Language) баз даних існують також NoSQL (Not only SQL). Вони характеризуються зберіганням значень по ключу і дуже зручні та підходять для нескладних застосунків. Одним із представників різних сервісів для застосунків, в тому числі NoSQL БД, є Firebase. Це платформа, розроблена Google, яка має Cloud Firestore та Realtime Database – бази для зберігання даних. База даних Firebase Realtime — це хмарна база даних. Дані зберігаються у форматі JSON і синхронізуються в режимі реального часу з кожним підключеним клієнтом. Коли ви створюєте міжплатформні програми з нашими платформами Apple, Android і JavaScript SDK, усі ваші клієнти спільно використовують один екземпляр бази даних реального часу й автоматично отримують оновлення з найновішими даними. Для застосунку “Free Table” була обрана Realtime Database через зручний спосіб зберігання даних у форматі JSON та через швидкий обмін інформацією в real-time [12].

2.4 Модулі системи, що розроблюється

Обравши відповідну архітектуру та інструментарій, треба визначитись із розроблюваними модулями в застосунку (рис. 2.2). Presentation – архітектурний шар із UI, який містить в собі сторінки аутентифікації, сторінку профіля, домашню сторінку і бронювання (останні відрізняються для адміністратора закладу і клієнта). Для бізнес-логіки буде створено ZonesBloc – для створення розташування столиків у зонах, та BookingsBloc – для керування бронюваннями. А також додається загальний Bloc для всього застосунку (наприклад, з налаштуваннями, зміною мови тощо). У Data Layer будуть додані різні сервіси для доступу до бази даних через API або через вбудовані бібліотеки.

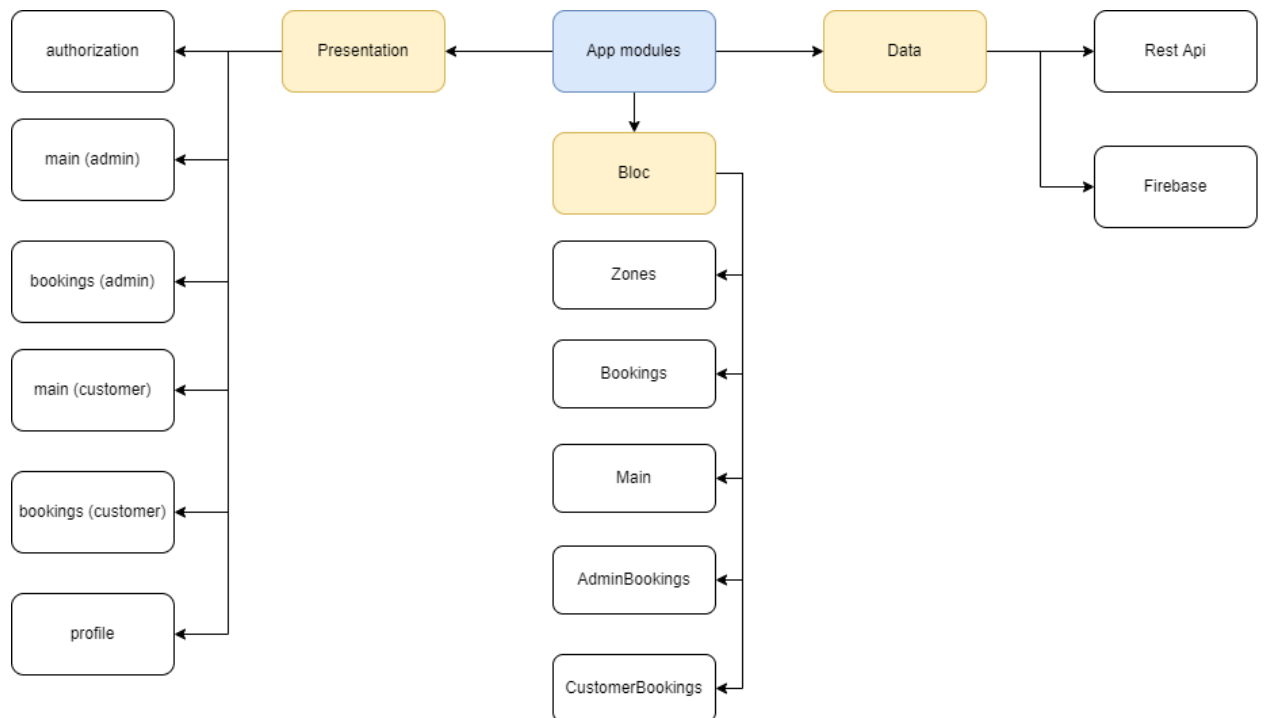


Рисунок 2.2 – Модулі системи

2.5 Проєктування функціональних вимог до програмного забезпечення

Після аналізу предметної області та завдання на диплом, було виділено наступні функціональні вимоги до розроблюваного ПЗ (програмного забезпечення):

- авторизація;
- реєстрація;
- створення зон розташування столиків;
- створення бронювань;
- керування бронюваннями;
- редагування профілю.

Також було розроблено функціональну схему за цими вимогами (рис.2.3).

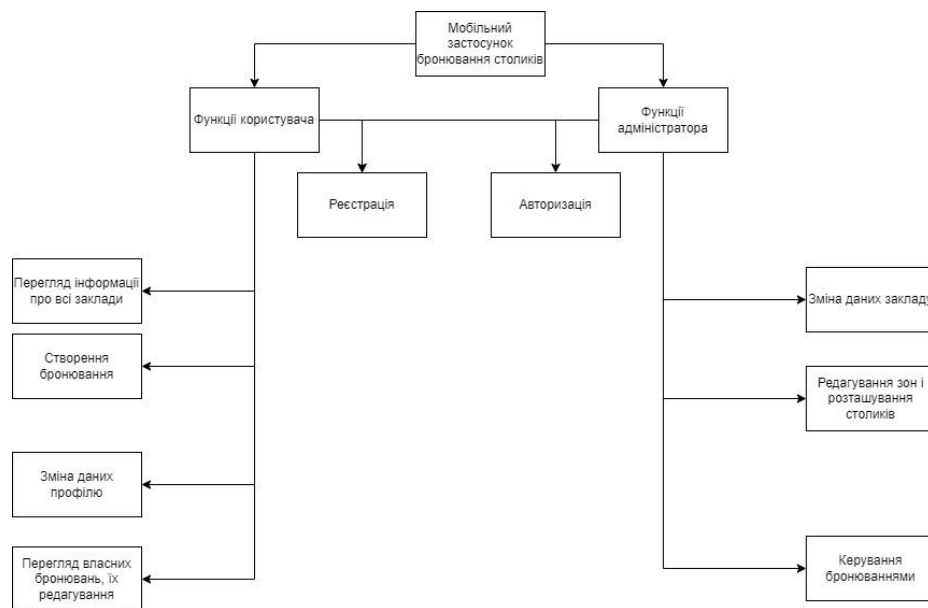


Рисунок 2.3 – Функціональна схема

Користувачами застосунку є адміністратори та клієнти закладів. Адміністратори мають функціонал щодо керування бронюваннями закладу.

Клієнти мають функціонал для бронювання столиків. Діаграма прецедентів зображена на рис. 2.4.

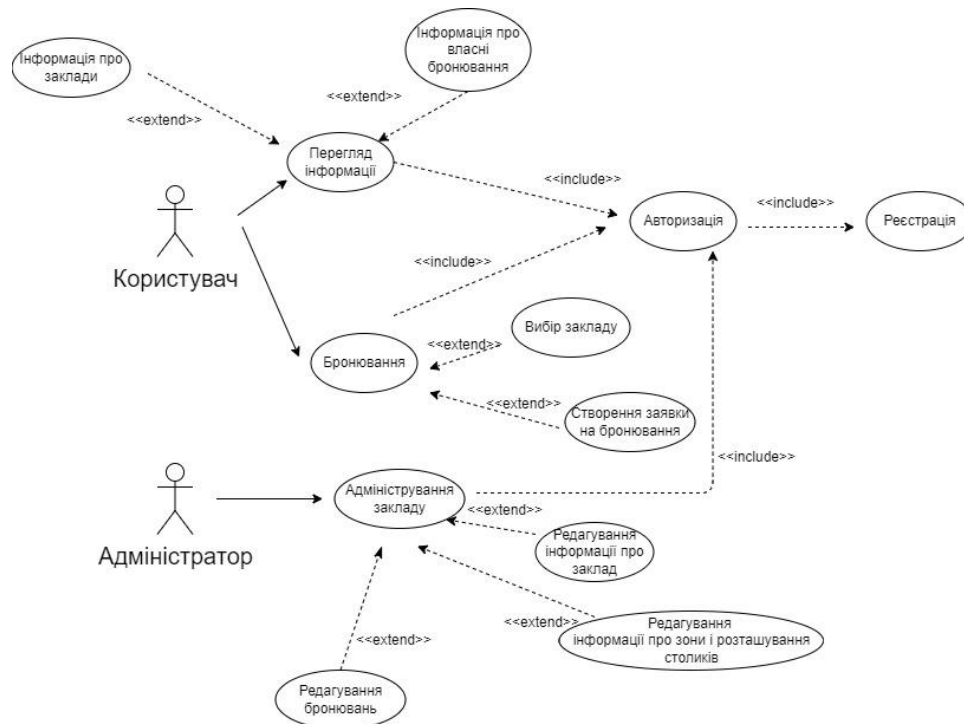


Рисунок 2.4 – Діаграма прецедентів

2.6 Висновки за розділом 2

В даному розділі були розглянуті основні переваги обраної мови програмування. Були порівняні основні середовища для розробки подібного ПЗ та обрана найкраща за всіма характеристиками. Були також обрані бібліотека для керування станом та NoSQL БД. На основі обраного підходу програмування була створена функціональна схема ПЗ та описані основні модулі розроблюваної системи.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Опис модулів системи

В розділі опису ПЗ було наведено схему модулів. Вона реалізована розділенням на три категорії. Модуль domain був розділений на такі складові:

- AdminBookings;
- Booking;
- CustomerBookings;
- Main;
- Zone.

Кожний модуль містить свій Bloc, який наслідує клас IBloc, BlocEvent, який наслідує IBlocEvent. Принцип реалізації класів показаний на прикладі AdminBookings (рис. 3.1).

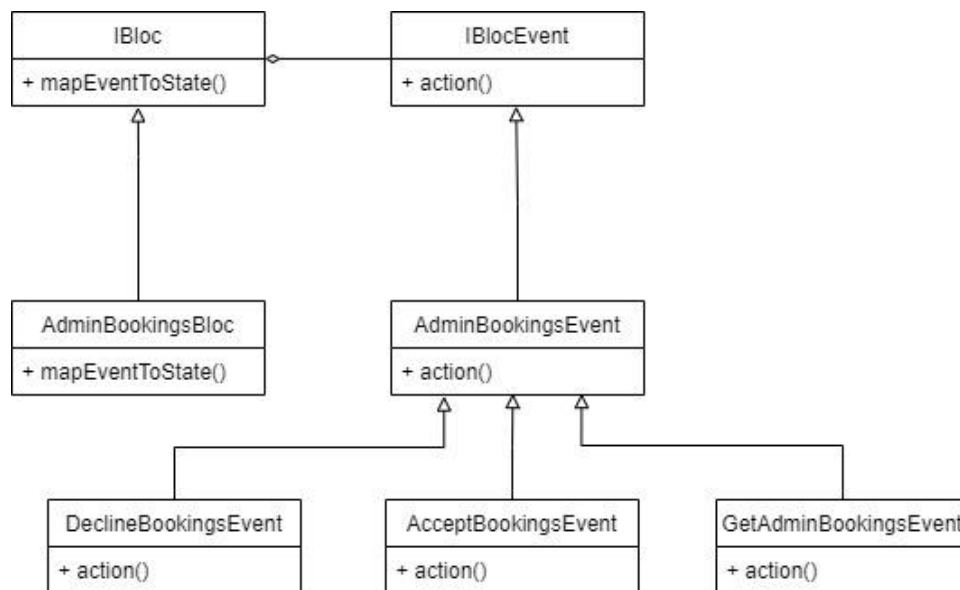


Рисунок 3.1 – Діаграма класів для блоку та івентів

Модуль AdminBookings відповідає за керуванням інформацією щодо бронювань закладу для адміністратора. Він містить AdminBookingsBloc, який керує станом. AdminBookingsState – клас стану, який зберігає в собі дані, що наведені в таблиці.

Таблиця 3.1 – Поля AdminBookingsState

Назва змінної	Тип структури	Значення за замовчуванням	Призначення
currentBookings	List<BookingInfoDTO?>	[]	Список всіх актуальних бронювань закладу
status	LoadingStatus	LoadingStatus.pending	Статус завантаження бронювань
reloadStatus	LoadingStatus	LoadingStatus.pending	Статус оновлення бронювань

AdminBookingsBloc має наступні події (events), які керують станом AdminBookingsState:

- AcceptBookingEvent – прийняти заявку на бронь (дати їй статус accepted);
- DeclineBookingEvent – відхилити заявку на бронь (статус declined);
- GetAdminBookingsEvent – завантажити з серверу всі бронювання в даному закладі.

Модуль Booking відповідає за керування інформацією щодо бронювань столику з боку клієнта. Він містить BookingBloc, який керує станом. BookingState – клас стану, який зберігає в собі дані, що наведені в табл. 3.2.

Таблиця 3.2 – Поля BookingState

Назва змінної	Тип структури	Значення за замовчуванням	Призначення
currentRestBookings	List<BookingInfoDTO>	[]	Список всіх актуальних бронювань закладу
status	LoadingStatus	LoadingStatus.pending	Статус завантаження ресторанів
restaurants	LoadingStatus	LoadingStatus.pending	Список існуючих ресторанів
currentRestZones	List<ZoneDTO>	[]	Список зон по обраному закладу
currentBooking	BookingInfo?	-	Поточне бронювання
placesCount	int?	1	Кількість місць для бронювання

Booking має наступні події (events), які керують станом BookingState:

- AddBookingEvent – відправляє дані про бронювання на сервер;
- AddCountEvent – значення кількості місць збільшується на 1;
- ClearBookingEvent – тимчасові значення про бронювання видаляються зі стану;
- GetRestBookingsEvent – завантажуються все бронювання, які є в обраному ресторані;
- GetRestsEvent – завантажуються список всіх ресторанів;
- GetZonesEvent – завантажуються список всіх зон;
- RemoveCountEvent – значення кількості місць зменшується на 1;
- SetBaseBookingEvent – встановлюються базові дані про бронювання (дані клієнта та телефон ресторану);
- SetDateEvent – встановлення дати бронювання;
- SetTableBookingEvent – встановлення номеру столика та назви зони в бронюванні;
- SetTimeEvent – встановлення часу бронювання.

Модуль CustomerBookings відповідає за керування інформацією щодо всіх поточних бронювань клієнта. Він містить CustomerBookingsBloc, який керує станом. CustomerBookingsState – клас стану, який зберігає в собі дані, що наведені в табл. 3.3.

Таблиця 3.3 – Поля CustomerBookingsState

Назва змінної	Тип структури	Значення за замовчуванням	Призначення
currentBookings	List<BookingInfoDTO?>	[]	Список всіх актуальних бронювань закладу
status	LoadingStatus	LoadingStatus.pending	Статус завантаження бронювань

CustomerBookings має наступні події (events), які керують станом CustomerBookings:

- DeleteBookingEvent – видалення бронювання;

– GetCustomerBookingsEvent – завантаження всіх бронювань користувача.

Модуль Main відповідає за керуванням всього застосунку, основними його властивостями. Він містить MainBloc, який керує станом. MainState – клас стану, який зберігає в собі дані, що наведені в табл. 3.4.

Таблиця 3.4 – Поля MainState

Назва змінної	Тип структури	Значення за замовчуванням	Призначення
appLoaded	Completer	-	Показує статус відкриття програми
currentTab	int	0	Поточний відкритий таб

Main має наступні події (events), які керують станом MainState :

- ChangeTabEvent - змінює поточний головний таб в bottom bar;
- CheckPhoneEvent - перевірка чи існує введений телефон в базі зареєстрованих користувачів;
- CheckTypeEvent - отримання типу користувача;
- InitAppEvent - ініціалізація застосунку;
- LogoutEvent - вихід з акаунту;
- SaveTokenEvent - збереження токена користувача на девайс;
- SaveTypeEvent - збереження типу користувача на девайс.

Модуль Zone відповідає за керуванням інформацією створення зон розташування столиків. Він містить ZoneBloc, який керує станом. ZoneState – клас стану, який зберігає в собі дані, що наведені в табл. 3.5.

Таблиця 3.5 – Поля ZoneState

Назва змінної	Тип структури	Значення за замовчуванням	Призначення
1	2	3	4
zonesWithTables	List<ZoneDTO>	[]	Показує статус відкриття програми

Продовження таблиці 3.5

1	2	3	4
editZone	Map<int, int>	{}	Поточний відкритий таб
currentNum	int	2	кількість місць столика
status	LoadingStatus	LoadingStatus.pendi ng	статус завантаження zones

Zone має наступні події (events), які керують станом ZoneState :

- ChangeTableSeatsEvent - зміна кількості місць за столиком (для створення/редагування зони);
- CreateEditZoneEvent - копіювання зони в тимчасову змінну для редагування;
- CreateNewZoneEvent - створення нової зони;
- DeleteNewZoneEvent - видалення нової зони (якщо не була збережена);
- EditNewZoneEvent - зміни щодо нової зони;
- EditZoneEvent - зміни щодо зони, яка редагується;
- FetchZones - завантаження всіх зон закладу;
- SaveEditableZoneEvent - збереження зони, що редагувалась;
- SetNameEvent - встановлення назви для зони.

В модулі Data відбувається зв'язок з сервером, обмін даними, збереження даних на девайс. Він поділяється на такі складові:

- Auth;
- Booking;
- Zone.

Всі стореджи для зберігання інформації на девайсі реалізовані за допомогою HiveStorage. Всі запити на сервер здійснюються за допомогою пакета Dio та Retrofit і за допомогою технології RestApi, що дозволяє зручно відправляти запити з указанням всіх потрібних параметрів.

Auth відповідає складається з наступних сервісів:

- TokenStorage – сторедж для зберігання токена;

- TypeStorage – сторедж для зберігання типу користувача;
- AuthApi – апи сервіс для відправлення dio-запитів.

Booking складається з наступних сервісів:

- BookingService – для обробки dio-запитів;
- BookingApi – апи сервіс для відправлення dio-запитів під час авторизації.

Zone складається з сервісу ZoneService – API сервіс для відправлення dio-запитів під час створення зон розташування.

3.2 Алгоритм функціонування розробленої програми

В процесі розробки функціоналу застосунку “Free Table” було виділено декілька основних алгоритмів роботи ПП.

3.2.1 Алгоритм авторизації

Крок 1. Для авторизації користувача як адміністратора або клієнта зчитується введений в відповідному текстовому полі номер телефону.

Крок 2. Введений номер перевіряється на існування в RealtimeDatabase через подію CheckPhoneEvent.

Крок 3. Якщо номер існує - далі за цим номером відбувається авторизація за допомогою FirebaseAuth через функцію FirebaseAuth.instance.verifyPhoneNumber().

Крок 4. Для успішної авторизації далі треба ввести в відповідне поле код із смс, яка прийшла на вказаний номер.

Крок 5. Цей код разом з отриманим verificationId потрібні для завершення авторизації через функцію FirebaseAuth.instance.signInWithCredential(credential).

Крок 6. В AuthBloc викликається подія CheckTypeEvent для перевірки в базі даних типу користувача.

Крок 7. Якщо цей тип `admin` – після збереження токена на девайс відбувається перенаправлення на головну сторінку адміністратора, якщо `customer` – клієнта.

3.2.2 Алгоритм реєстрації

Крок 1. Для реєстрації користувача зчитуються введені в відповідні текстові поля номер телефону та ім'я.

Крок 2. Введений номер перевіряється на існування в `RealtimeDatabase` через подію `CheckPhoneEvent`.

Крок 3. Якщо номер ще не існує - далі за цим номером відбувається реєстрація за допомогою `FirebaseAuth` через функцію `FirebaseAuth.instance.verifyPhoneNumber()`.

Крок 4. Для успішної авторизації далі треба ввести в відповідне поле код із смс, яка прийшла на вказаний номер.

Крок 5. Також треба обрати тип користувача: адміністратора або клієнта.

Крок 6. Код із смс разом з отриманим `verificationId` потрібні для завершення авторизації через функцію `FirebaseAuth.instance.signInWithCredential(credential)`.

Крок 7. В `AuthBloc` викликається подія `SendTypeEvent` для збереження типу користувача.

Крок 8. Якщо цей тип `admin` – після збереження токена на девайс відбувається перенаправлення на головну сторінку адміністратора, якщо `customer` – клієнта.

3.2.3 Алгоритм додавання зон з розташуванням столиків

Крок 1. Алгоритм створення зони розпочинається з переходу на екран NewZone. Сама зона є мапом Map<int,int>, що зберігає як ключ - id столика, і як значення – кількість місць за ним.

Крок 2. При переході викликається CreateNewZoneEvent, який створює порожній мап для нової зони. На ній розташовані кнопки з кількістю місць та поле для заповнення.

Крок 3. При натисканні кнопки з кількістю – викликається ChangeTableSeatsEvent, який змінює в стані ZoneState кількість місць за столиком для вибору.

Крок 4. Обираючи комірку в полі зони (метод onTap), можна викликати EditZoneEvent, що редагує створену нову зону.

Крок 5. Після збереження даних викликається SaveEditableZoneEvent, що зберігає цей мап в стані ZoneState, методом context.showPopUp (nameDialog(context)) відкривається діалогове вікно для вводу назви створеної зони.

Крок 6. Далі через подію SetNameEvent ця назва зберігається в тому ж ZoneState.

Крок 7. Робиться виклик POST запиту для відправлення даних у RealtimeDatabase.

3.2.4 Алгоритм редагування зон

Алгоритм редагування зони дуже схожий на алгоритм створення.

Крок 1. Редагування розпочинається з переходу на екран NewZone (використовується той самий шаблон представлення, щоб зменшити кількість повторюваного коду).

Крок 2. При переході викликається CreateEditZoneEvent, який копіює дані зони в новий тимчасовий мап для редагування зони.

Крок 3. При натисканні кнопки з кількістю – так само викликається `ChangeTableSeatsEvent`.

Крок 4. При редагуванні самого поля також викликається `EditZoneEvent`, що редагує саме цю тимчасову зону.

Крок 5. Після збереження даних викликається `SaveEditableZoneEvent`, що зберігає цей мап в стані `ZoneState`, відкривається діалогове вікно для зміни назви створеної зони.

Крок 6. Далі аналогічно створенню зони відбувається `SetNameEvent`.

Крок 7. Відправка даних на сервер.

3.2.5 Алгоритм керування заявками

Створена заявка на бронювання має 3 стани: `Accepted` - прийнята, `Declined` - відхилена, `Processing` – ще знаходиться в обробці.

Крок 1. Щоб прийняти нову або вже відхилену заявку, треба викликати `AcceptBookingEvent` та передати повну інформацію про бронювання.

Крок 2. Далі робиться виклик PATCH запиту для оновлення даних у `RealtimeDatabase`.

Такий само алгоритм для відміни нової чи прийнятої заявки через `DeclineBookingEvent`.

3.2.5 Алгоритм бронювання столиків клієнтами

Крок 1. Для бронювання столика в закладі обирається спочатку заклад із списку.

Крок 2. По цьому закладу через подію `GetZonesEvent` стягуються з серверу всі доступні зони та розташування столиків в них.

Крок 3. Через `GetRestBookingsEvent` стягуються дані про бронювання, що вже є в закладі (щоб можна було визначити вільні столики).

Крок 4. Далі обираються параметри бронювання та зберігаються у відповідний BookingState. Дата обирається через нативний календар та зберігається через SetDateEvent, при чому початкова дата встановлюється поточна та вибрати можна тільки наступні дати у межах двох тижнів.

Крок 5. Час встановлюється також через нативний годинник і зберігається через SetTimeEvent.

Крок 6. Кількість людей можна змінити відповідними кнопками і за допомогою AddCountEvent і RemoveCountEvent, які відповідно додають або віднімають значення.

Крок 7. На полях, відведених для зон розташування обирається вільний столик (та зону) через подію SetTableBookingEvent.

Крок 8. Встановлюються базові дані про бронювання (дані про клієнта та адміністратора ресторану).

Крок 9. Дані відправляються на сервер після підтвердження через AddBookingEvent.

Крок 10. Очищуються всі тимчасові дані у ClearBookingEvent.

3.3 Опис розроблених інтерфейсів

Для зручного використання програми інтерфейс має бути лаконічним, не перевантаженим елементами, простий та зрозумілий. Користування програмою має бути зрозумілим одразу через загально відомі і використовувані компоненти (так як кнопки, текстові поля, іконки т.д.). Інтерфейси були розроблені з використанням 2 основних кольорів, шрифт був обраний Google Fonts – Manrope. Основними екранами неавторизованого користувача є екран авторизації (рис. 3.2) та екран реєстрації (рис. 3.3).

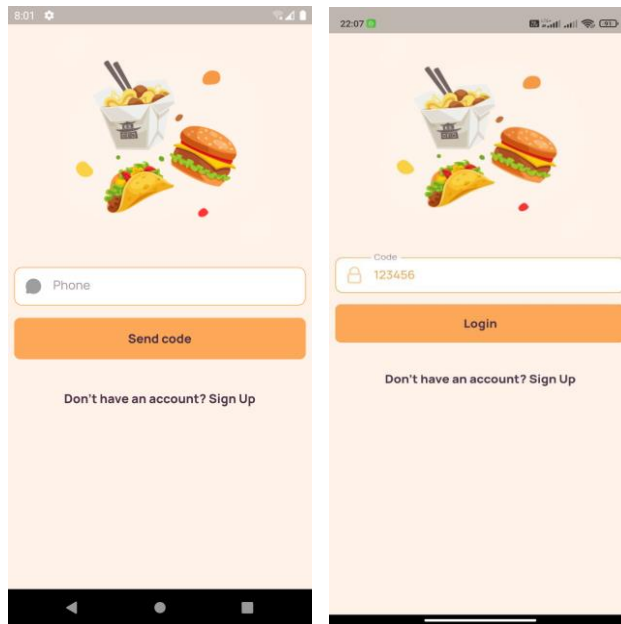


Рисунок 3.2 – Екран авторизації, введення телефону та коду

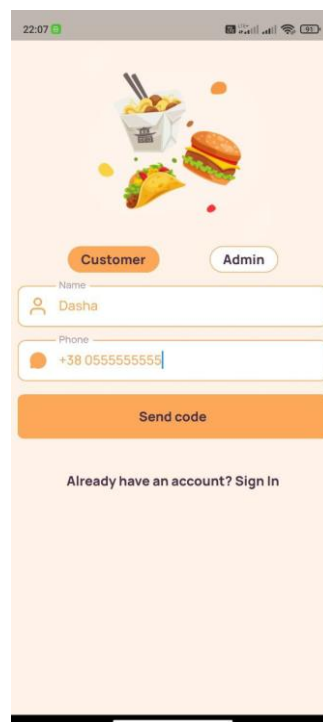


Рисунок 3.3 – Екран реєстрації

Основними екранами авторизованого клієнта є головна сторінка (рис. 3.4), сторінка актуальних бронювань (рис. 3.5) і сторінка профілю клієнта (рис. 3.6).

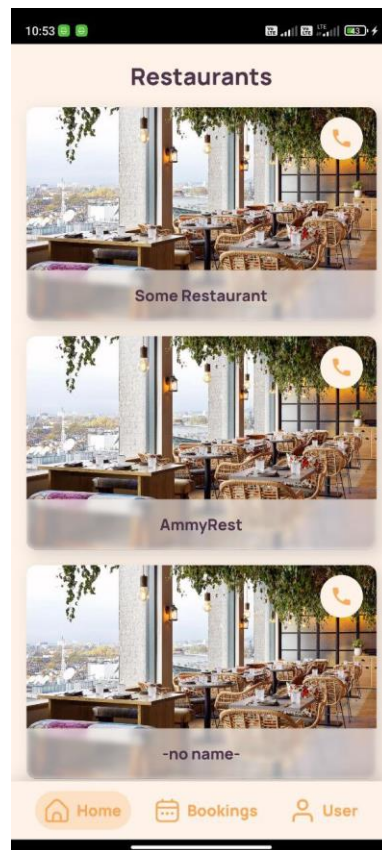


Рисунок 3.4 – Голова сторінка клієнта

Основним навігаційним елементом є bottom bar із кнопками для переключення між вкладками (основними екранами). Більшість елементів були розроблені кастомно за дизайном (тобто не використовувались вбудовані елементи flutter, на кшталт ElevatedButton, Card). На головній сторінці -Home - розташовані всі ресторани, доступні для бронювання. На сторінці Booking – список актуальних бронювань, з указанням всієї необхідної інформації про них та статусом заявки. Статус додатково відмічається кольором для швидкого розрізнення. Сторінка профілю – User – має ім'я користувача, номер телефону та кнопку для виходу з акаунту.

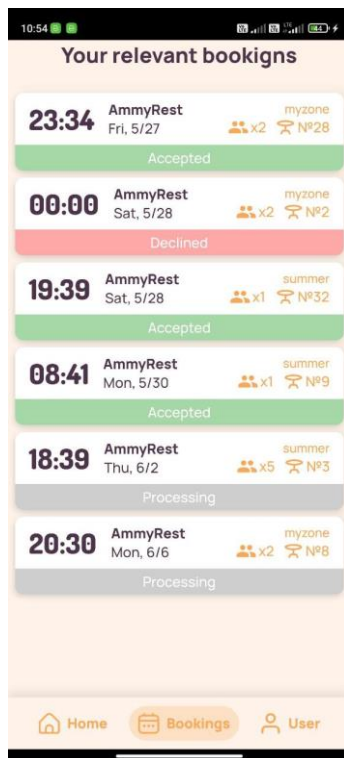


Рисунок 3.5 – Сторінка актуальних бронювань

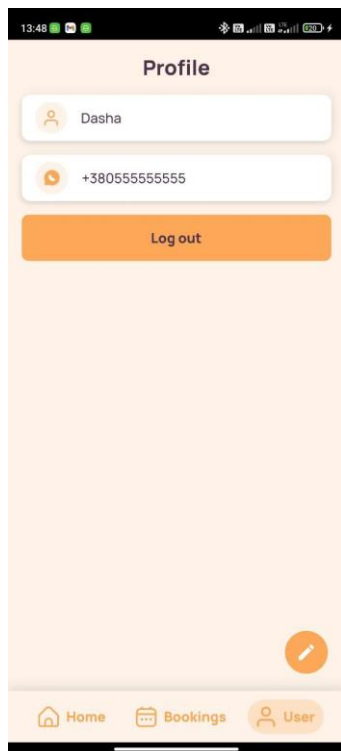


Рисунок 3.6 – Екран профіль клієнта

Для створення бронювання також доданий відповідний екран (рис. 3.7). На ньому розташовані елементи вибору кількості місць, виклики

нативних календаря та годинника для встановлення дати й часу, підказка щодо значень комірок у полі якоїсь зони закладу та безпосередньо самі зони із столиками.

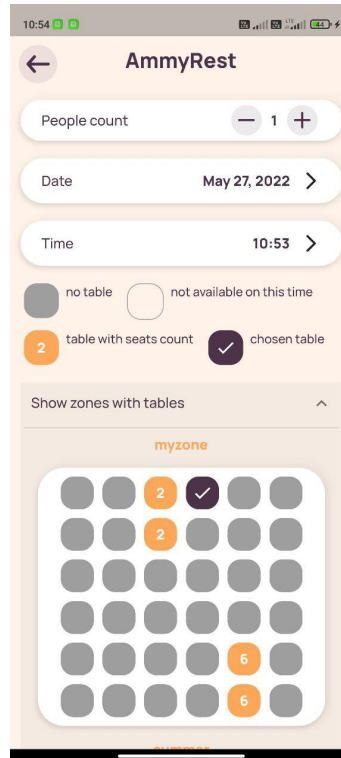


Рисунок 3.7 – Сторінка створення бронювання

Основними екранами авторизованого адміністратора є головна сторінка (рис. 3.8), сторінка актуальних бронювань (рис. 3.10) і сторінка профілю закладу (рис. 3.9). На головній сторінці розташовані всі зони закладу у вигляді карток. Їх можна видалити, редагувати, додати. Екран профілю схожий на профіль клієнта, тільки додається фотографія закладу.

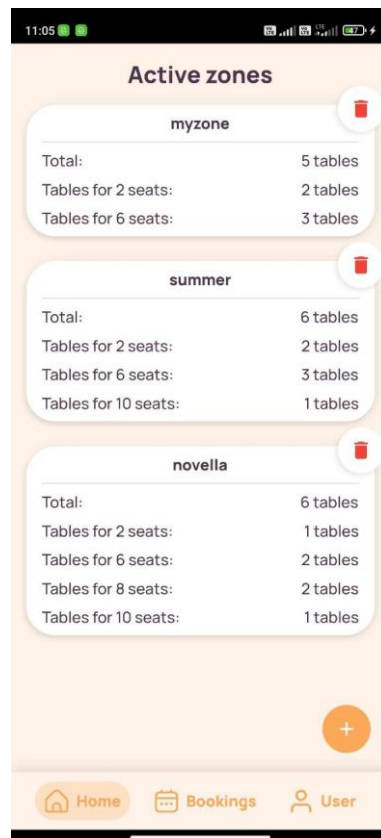


Рисунок 3.8 – Головна сторінка адміністратора

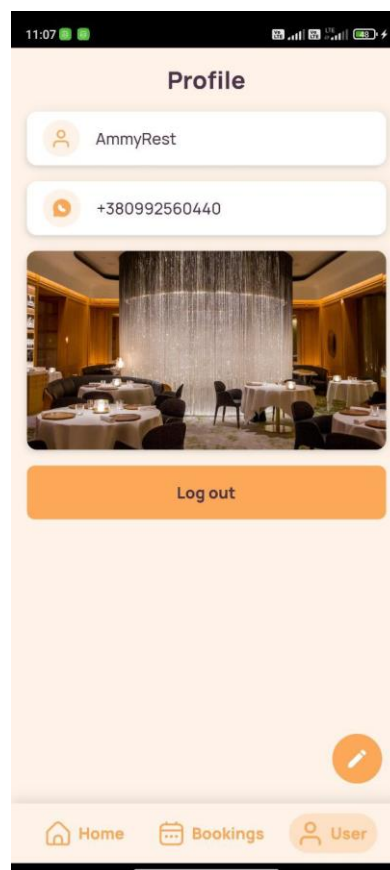


Рисунок 3.9 – Екран профілю закладу

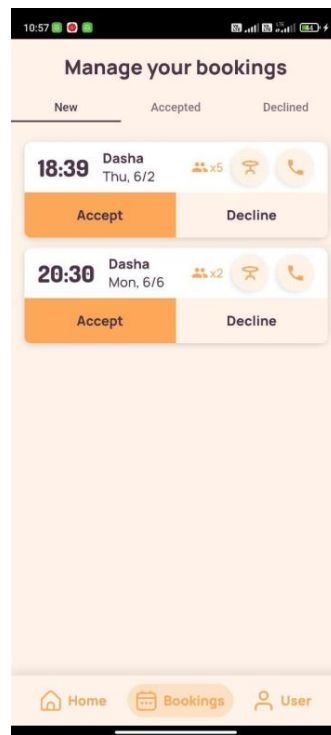


Рисунок 3.10– Екран бронювань адміністратора

Щоб редагувати або додавати зону, створено екран зони (рис. 3.11). Також серед головних вкладок є сторінка бронювань із трьома внутрішніми вкладками для різних типів, на кожній вкладці список заявок у вигляді карток.

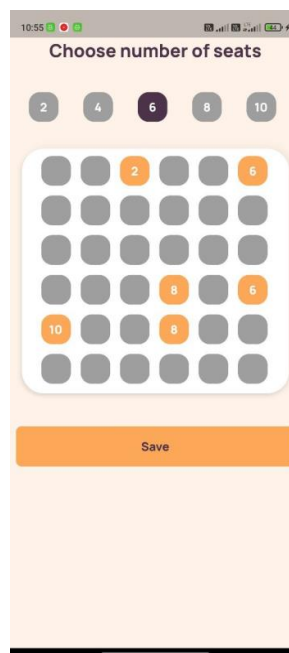


Рисунок 3.11 – Створення зони закладу

3.4 Опис бази даних

Так як обрана база даних є NoSql то сутності, що у ній містяться представляють собою формат JSON. В БД є основні три сутності: bookings, user-types, zones.

Таблиця bookings зберігає в собі створені бронювання. Вони зберігаються у мапі, ключем до кожного бронювання є унікальний код (id). Модель бронювання представлена в табл. 3.6

Таблиця 3.6 – Поля моделі bookings

Назва поля	Тип змінної	Призначення
accepted	bool	Чи прийнята заявка
customer	String	Ім'я клієнта
dateTime	DateTime	Дата й час заявки
peopleCount	Int	Кількість місць
phone	String	Телефон клієнта
restaurantPhone	String	Телефон закладу
tableId	Int	Номер столика
zoneId	String	Назва зони

Таблиця user-types зберігає в собі інформацію про зареєстрованих користувачів. Вони зберігаються у мапі, ключем до кожного користувача є унікальний код (id). Модель користувача представлена в табл. 3.7.

Таблиця 3.7 – Поля моделі user-types

Назва поля	Тип змінної	Призначення
name	String	Ім'я користувача
phone	String	Телефон клієнта
type	String	Тип користувача

Таблиця zones зберігає в собі інформацію про зони по кожному закладу. Вони зберігаються у мапі, ключем до кожного закладу є номер телефону, за яким він зареєстрований, значення – це мап із ключем унікальним кодом та моделлю зони, яка представлена в табл. 3.8.

Таблиця 3.8– Поля моделі zone

Назва поля	Тип змінної	Призначення
title	String	Телефон клієнта
tablesMap	Map<int, int>	Мап зони, ключ – номер столика, значення - кількість місць

3.5 Висновки за розділом 3

В даному розділі були розглянуті основні створені модулі ПП та керування станом через Bloc. На основі функціональної схеми були виділені основні алгоритми в роботі програми. Беручи за основу алгоритми, були створені основні інтерфейси застосунку, враховуючи 2 типи користувачів. Також був представлений тип БД в Firebase та були описані її основні таблиці.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення та умови застосування програми

Призначенням мобільного застосунку є керування бронюваннями в закладах харчування. За допомогою ПП можна створити акаунт клієнта або адміністратора, створити зони розташування столиків, редагувати дані про бронювання та розташування столиків, робити бронювання, редагувати дані акаунту.

Для нормального функціонування програми, повинні виконуватися наступні програмно-технічні умови:

- ОС Windows 10 або вище, MacOS не нижче 10.15;
- оперативна пам'ять не менше 2 ГБ;
- вільна пам'ять комп'ютера не менше 4 ГБ;
- частота процесора не менше 300 МГц;
- встановлений VSCode, Android Studio або IntelliJ;
- підключення до інтернету;
- встановлений Flutter не нижче 2.10.2.

4.2 Виконання програми

Для успішного запуску програми спочатку треба завантажити все пакети й плагіни за допомогою команди:

```
flutter pub get
```

Далі щоб оновити файли автогенерації треба викликати команду:

```
flutter pub run build_runner build --delete-conflicting-outputs
```

Для запуску програми в режимі release треба викликати наступну команду:

```
flutter run --release lib/main_prod.dart
```

Першим відкритим екраном буде екран авторизації. На екрані авторизації є валідація поля телефону, а на екрані реєстрації - полей телефону та імені. Якщо значення телефону введено неправильно, то виводиться відповідне повідомлення. Якщо ім'я введено неправильно – виводиться також відповідне повідомлення (рис. 4.1).

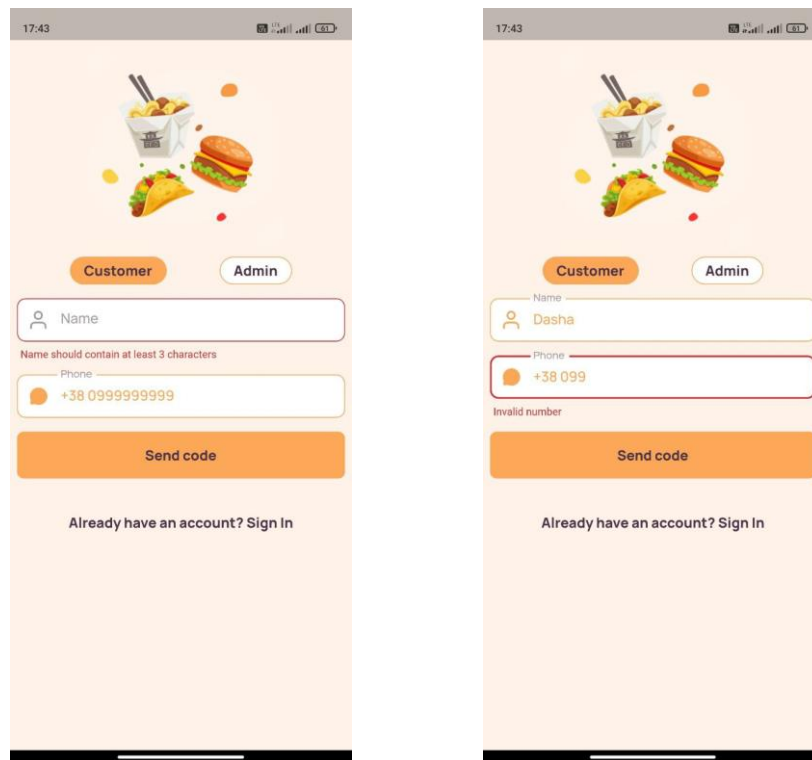


Рисунок 4.1 – Валідація імені та номеру телефону

Щоб зареєструватись або авторизуватись, треба спочатку пройти валідацію, а далі натиснути відповідну кнопку. Якщо авторизація відбувається за номером, який не зареєстровано – з'являється наступне повідомлення (рис. 4.2). Якщо реєстрація відбувається за номером, який вже зареєстровано –

з'являється наступне повідомлення (рис. 4.3). У поля для коду треба ввести код із смс і натиснути кнопку Login/ SignUp.

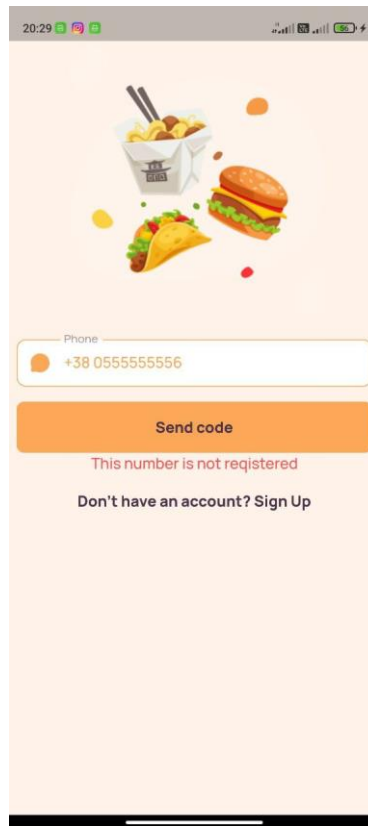


Рисунок 4.2 – Помилка серверу (акаунт не існує)

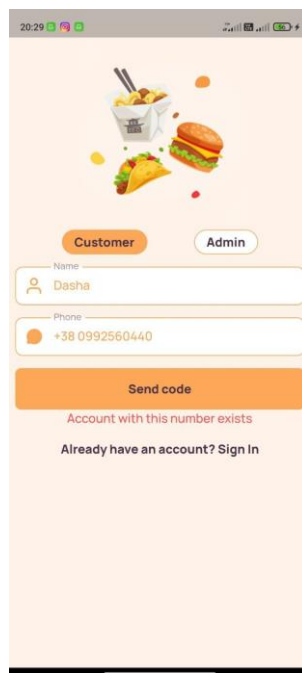


Рисунок 4.3 – Помилка серверу (акаунт вже існує)

Якщо користувач є клієнтом, щоб створити замовлення треба натиснути на ресторан із списку. Кнопка Book залишається неактивною, доки не будуть заповнені всі поля. Для створення бронювання треба заповнити дані (рис. 4.4): кнопками – і + вибрати кількість місць, із діалогового вікна обрати дату, а потім час. У полі зони треба обрати столик. Далі натискаючи кнопку бронювання, ми відсилаємо запит на сервер та повертаємось на головну сторінку.

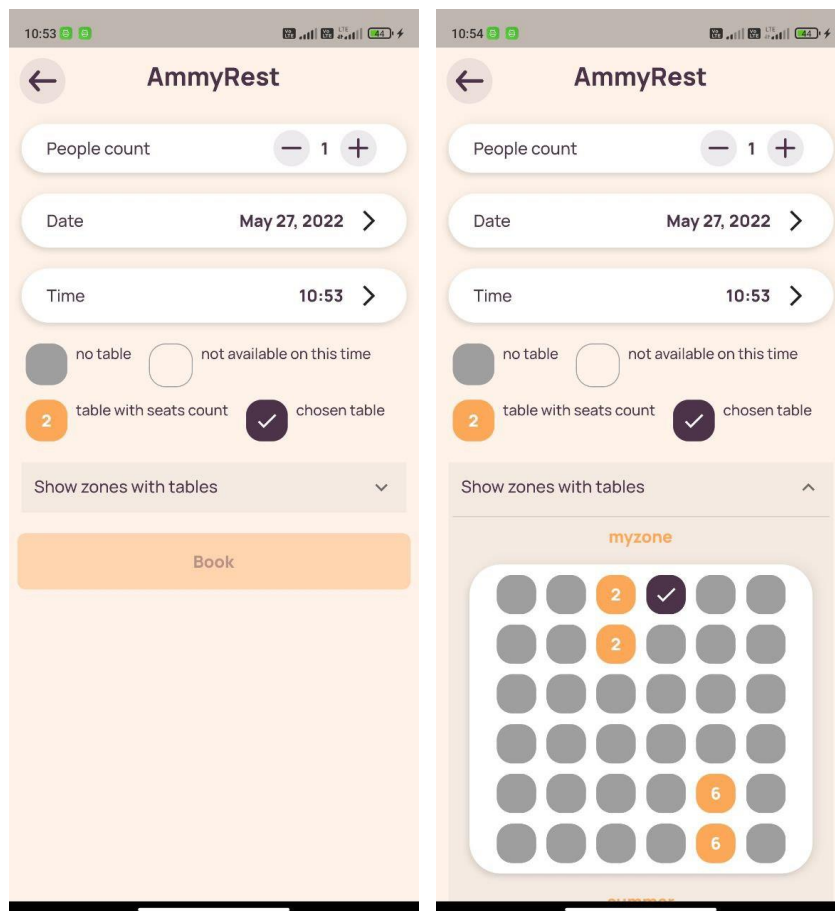


Рисунок 4.4 – Екран бронювання в закритому і відкритому режимі зон

На сторінці бронювань можна видаляти їх. Щоб це зробити, треба свайпнути справа наліво, де буде показана іконка видалення. Далі відкривається вікно із запитанням про підтвердження видалення (рис. 4.5). Після натискання Yes вікно закривається, а бронювання видаляється. Після натискання Cancel скасовується видалення.

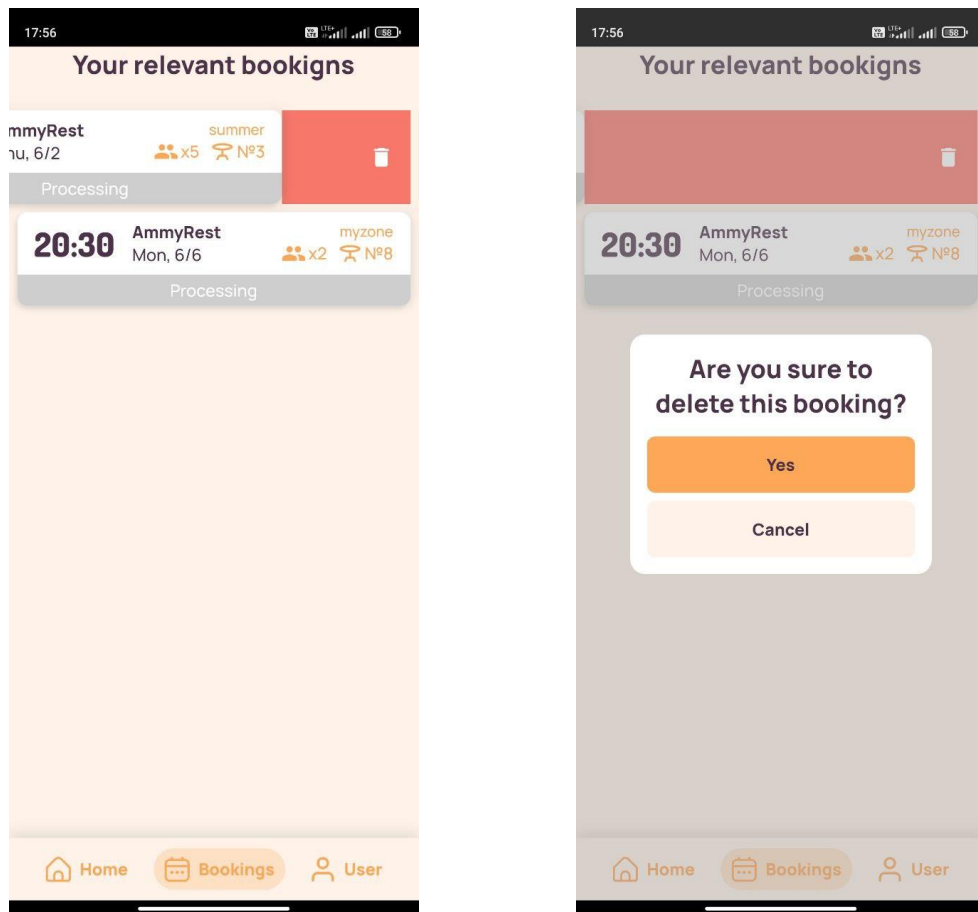


Рисунок 4.5 – Видалення бронювання із підтвердженням

Щоб редагувати профіль на відповідній сторінці треба натиснути на кнопку з іконкою редагування. У відкритому вікні є текстове поле, де можна змінити своє ім'я для відображення (рис. 4.6). Після натискання Save вікно закривається, а ім'я користувача зберігається. Після натискання Cancel скасовується редагування.

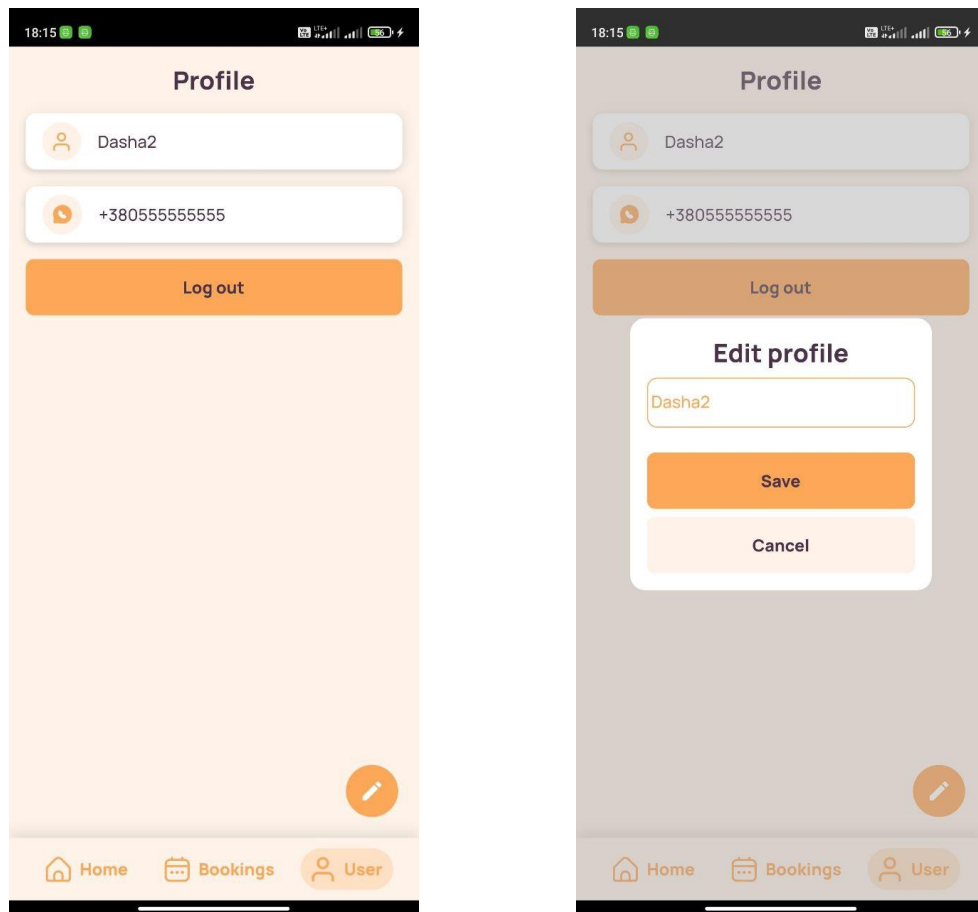


Рисунок 4.6 – Екран профілю і режим редагування

Якщо користувач є адміністратором, то щоб додати зону треба натиснути кнопку з іконкою +, а щоб редагувати – просто натиснути на обрану зону. Для видалення зони треба натиснути на іконку видалення поряд із зоною. У відкритому вікні обрати Yes для видалення, Cancel – щоб просто закрити це вікно (рис. 4.7). Для створення/редагування зони розташування треба заповнити поле столиками (рис. 4.8). Далі натискаючи кнопку Save, ми відсилаємо запит на сервер та повертаємось на головну сторінку. Якщо ж треба скасувати редагування або нову зону – треба натиснути нативну кнопку назад, у відкритому вікні обрати Yes для скасування режиму редагування, Cancel – щоб просто закрити це вікно.

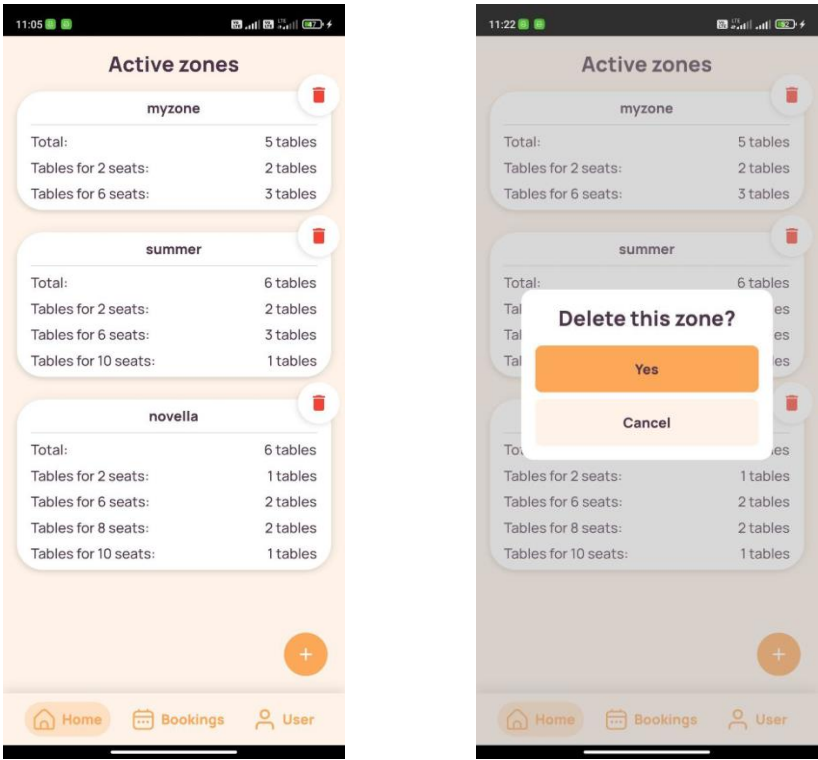


Рисунок 4.7 – Процес видалення зони

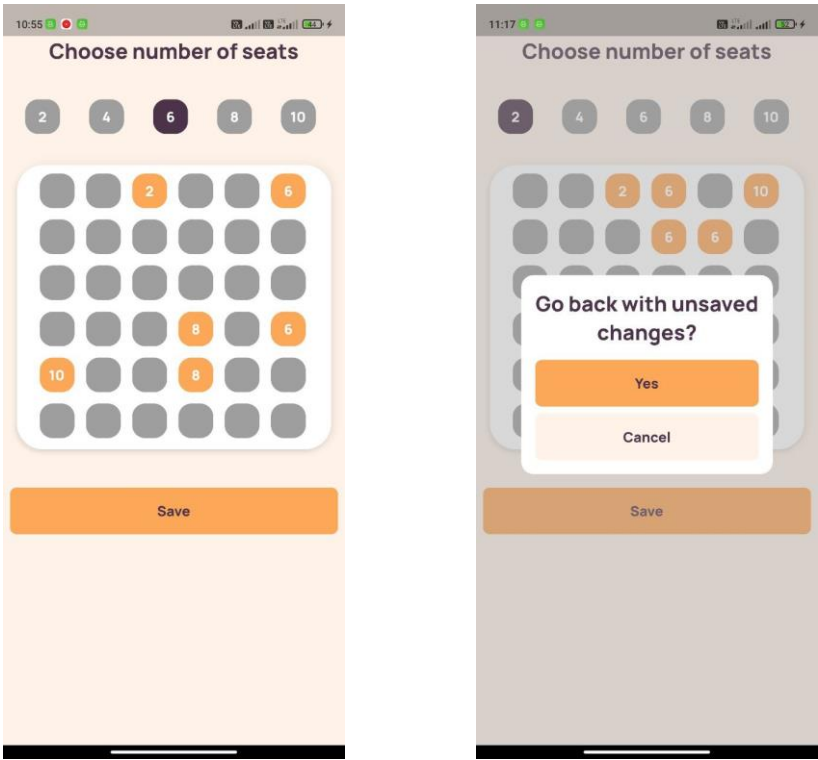


Рисунок 4.8 – Редагування / створення зони

На сторінці бронювань є три вкладки: New – бронювання, що не підтверджені ще, Asserpted – прийняті бронювання, Declined – скасовані. Для пе-

реміщення між вкладками можна використовувати свайп вправо-вліво або натискати на заголовки вкладок. Щоб прийняти скасоване або нове бронювання треба натиснути на кнопку Асепт і у відкритому вікні підтвердити кнопкою Yes. Щоб скасувати прийняте або нове бронювання треба натиснути на кнопку Decline і у відкритому вікні підтвердити кнопкою Yes (рис. 4.9).

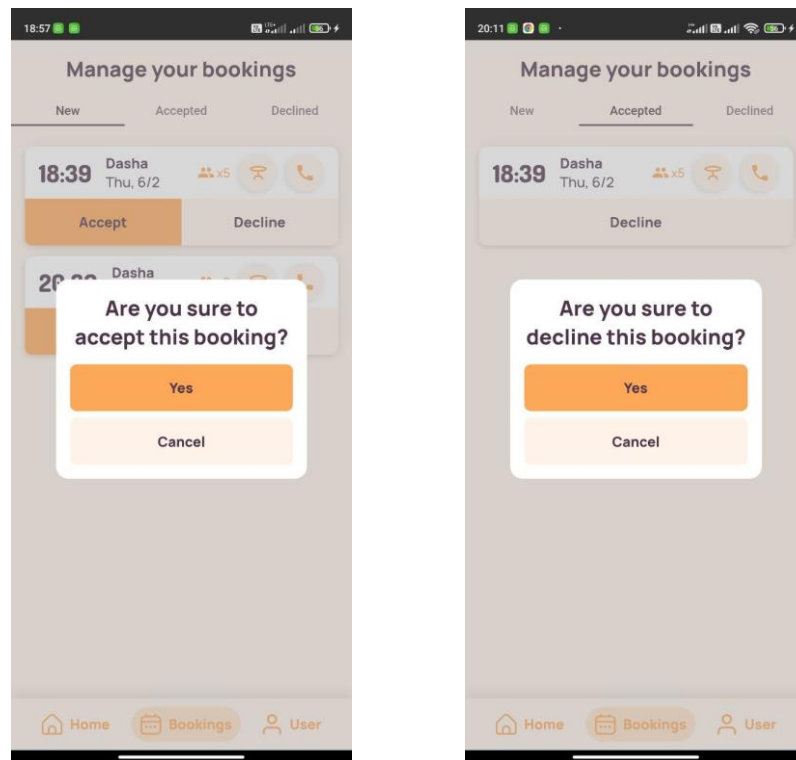


Рисунок 4.9 – Прийняття або скасування заявки

Для перегляду інформації про розташування столика треба натиснути на відповідну іконку. У відкритому вікні буде показано розташування (рис. 4.10).

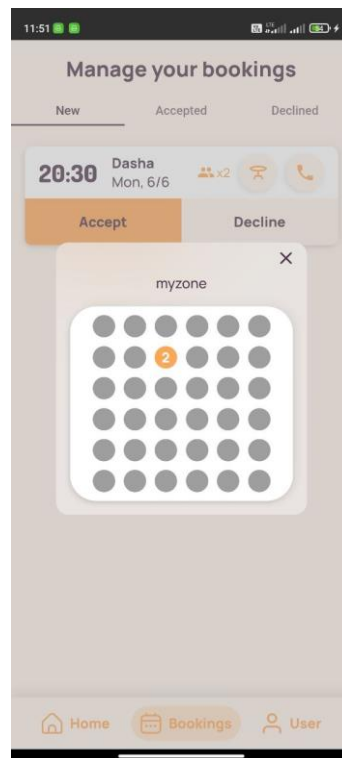


Рисунок 4.10 – Розташування столика в бронюванні

Для того, щоб подзвонити клієнту треба натиснути на іконку телефону, відкриється системний телефон (рис. 4.11).

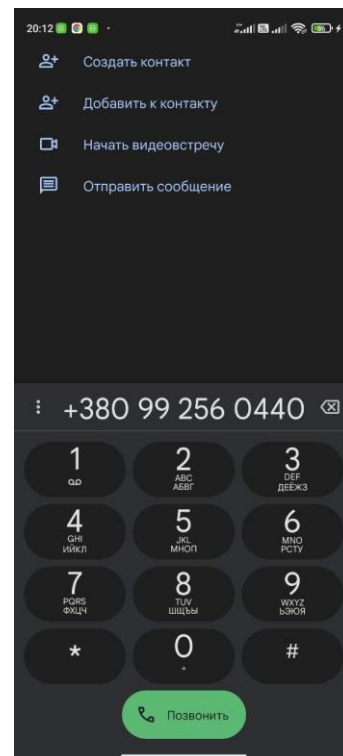
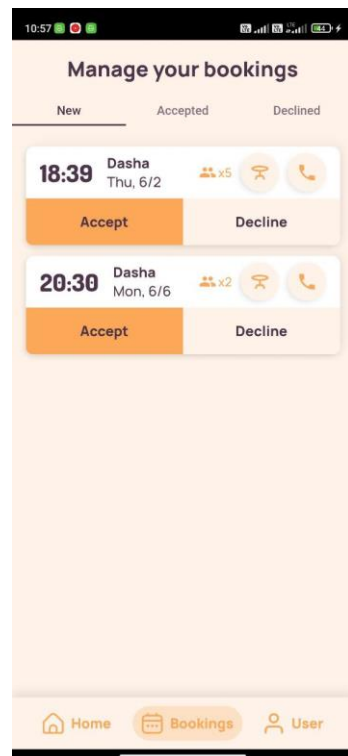


Рисунок 4.11 – Перенаправлення в телефон для дзвінка

Зміна даних профілю майне аналогічна до клієнта, тільки додається фотографія закладу. У вікні редагування під текстовим полем знаходиться фотографія і кнопка для зміни. Після натискання на кнопку з'являється нове модальне вікно з вибором джерела файлу. Обране з галереї чи камери фото встановлюється замість попереднього. Щоб зберегти всі дані, треба натиснути Save, в протилежному випадку – Cancel (рис. 4.12).

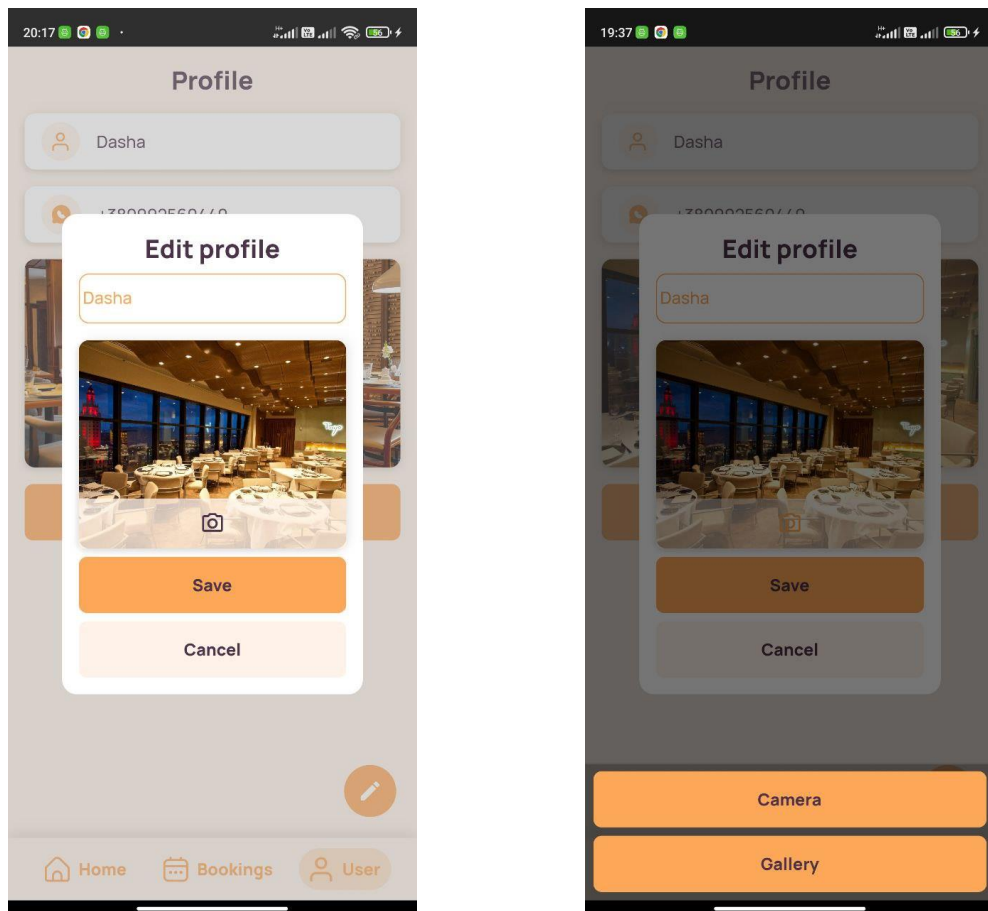


Рисунок 4.12 – Редагування профілю адміністратора

Якщо під час роботи застосунку зникає зв'язок з інтернетом – з'являється наступне повідомлення (рис. 4.13).

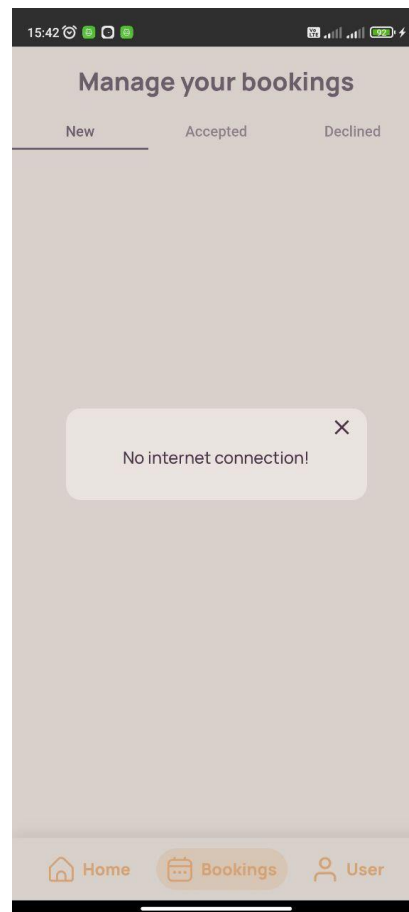


Рисунок 4.13 – Попередження відсутності зв’язку

4.3 Тестування програми

Чим більше функцій має ПП, тим важче його перевірити вручну. Автоматичні тести допомагають переконатися, що застосунок працює правильно, перш ніж ви його публікуєте, зберігаючи при цьому ваш функціонал та швидкість виправлення помилок. Тестування у застосунку Flutter може бути трьох типів:

- unit test;
- widget test;
- integration test.

Unit тест перевіряє одну функцію, метод або клас. Мета unit тесту полягає в тому, щоб перевірити правильність одиниці логіки за різноманітних умов. Модульні тести, як правило, не зчитуються з диска і не записуються на диск, не відображаються на екрані та не отримуються дії користувача поза

процесом, який виконує тест. Widget test перевіряє один віджет. Метою widget test є перевірка того, що інтерфейс віджета виглядає та взаємодіє, як очікувалося. Тестування віджета включає кілька класів і вимагає тестового середовища, яке забезпечує відповідний контекст життєвого циклу віджета. Integration test тестує повну програму або велику частину програми. Мета інтеграційного тесту полягає в тому, щоб переконатися, що всі віджети та служби, які тестуються, працюють разом, як очікувалося. Крім того, ви можете використовувати інтеграційні тести, щоб перевірити ефективність вашої програми [13].

Для тестування застосунку були створені unit тести. На прикладі Places count тесту показаний алгоритм виконання тесту. Він складається з таких кроків.

Крок 1. Налаштування середовища для тестування.

```
initDependencies(MockConfig());
final bloc = BookingBloc.byGetIt();
```

Крок 2. Виконання передумов тесту.

```
expect(bloc.state.placesCount, 1);
```

Крок 3. Виконання кроків тесту.

```
bloc.add(AddCountEvent());
await Future.delayed(Duration(milliseconds: 50));
```

Крок 4. Порівняння очікуваного результату з реальним.

```
expect(bloc.state.placesCount, 2);
```

Список написаних тестів розташований на рис. 4.14. Були протестовані основні події блоків AdminBookings, Booking, CustomerBookings, Zone.

```

✓ (✓) test\bloc\admin_booking_test.dart 2/2 passed: 310ms
  ✓ (✓) Admin Booking Bloc Tests 2/2 passed: 310ms
    ✓ (✓) get bookings test 125ms
    ✓ (✓) accept/decline bookings test 185ms
✓ (✓) test\bloc\booking_bloc_test.dart 4/4 passed: 419ms
  ✓ (✓) Booking Bloc Tests 4/4 passed: 419ms
    ✓ (✓) places count test 159ms
    ✓ (✓) datetime test 138ms
    ✓ (✓) table/zone test 61ms
    ✓ (✓) base booking info test 61ms
✓ (✓) test\bloc\customer_bookings_bloc_test.dart 2/2 passed: 252ms
  ✓ (✓) Customer Booking Bloc Tests 2/2 passed: 252ms
    ✓ (✓) get bookings test 116ms
    ✓ (✓) delete bookings test 136ms
✓ (✓) test\bloc\zone_bloc_test.dart 3/3 passed: 412ms
  ✓ (✓) Zone Bloc Tests 3/3 passed: 412ms
    ✓ (✓) new zone test 167ms
    ✓ (✓) fetch zones test 59ms
    ✓ (✓) edit zones test 186ms

```

Рисунок 4.14 – Розроблені тести і їх розташування в файлах

Результати тестування показані на рис. 4.15.

```

✓ Admin Booking Bloc Tests get bookings test
✓ Admin Booking Bloc Tests accept/decline bookings test
✓ Booking Bloc Tests places count test
✓ Booking Bloc Tests datetime test
✓ Booking Bloc Tests table/zone test
✓ Booking Bloc Tests base booking info test
✓ Customer Booking Bloc Tests get bookings test
✓ Customer Booking Bloc Tests delete bookings test
✓ Zone Bloc Tests new zone test
✓ Zone Bloc Tests fetch zones test
✓ Zone Bloc Tests edit zones test
Exited

```

Рисунок 4.15 – Виконання unit-тестів

4.4 Висновки за розділом 4

В даному розділі були розглянуті умови запуску застосунку, призначення ПП. Також було розглянуто повністю користування застосунком та відгуки програми на взаємодію з користувачем, обробка помилок користувача. ПЗ було протестовано за допомогою автоматизованого тестування через unit тести.

ВИСНОВКИ

У ході виконання дипломної роботи проведено детальне ознайомлення із предметною областю – бронювання в закладах харчування. Було ознайомлення із найбільш важливими питаннями цієї області та найвагомішими рішеннями: ведення обліку бронювань, створення профілю користувача, створення бронювань через застосунок.

Під час роботи із першим розділом були розглянуті існуючі аналоги мобільних та веб-застосунків для бронювання столиків. Розглянуті програмні продукти були проаналізованими за критеріями порівняно з розроблюваним проектом. Також у першому розділі були виділені основні переваги, які треба реалізувати, та недоліки, яких треба уникнути при розробці.

У другому розділі розглянуто мову програмування, її основні переваги при створенні мобільних застосунків. Більш того, було приділено увагу розгляду різних IDE та обрано VS Code. Була обрана система керування станів в застосунку – BloC. Також на основі поставленої задачі була сформована функціональна схема та виявлені основні модулі системи.

У третьому розділі була розглянута логіка функціонування програми, алгоритми та опис модулів. Була також описана структура сутностей обраної БД.

В решті-решт, було розроблено ПП, який реалізовував усі поставлені задачі та функціональні вимоги. Були також передбачені різні повідомлення користувачу під час роботи, при введенні неправильних значень чи при зміні інформація (видалення, збереження, тощо). В подальшому можливо модернізувати ПП в напрямку збільшення функціоналу, наприклад, додавання більше інформації про заклад, фільтрація та сортування списку закладів за критеріями тощо.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Le Pham Mobile application: definition, technology types and examples 2022 [Електрон. ресурс]. – Режим доступу: <https://magenest.com/en/mobile-application/>
2. Shetty, D. Mobile App Ecosystem: An app developer's path to Success: thesis ... master degree/ D. Shetty. – Massachusetts Institute of Technology, 2014. – 68 p.
3. TheStol [Електрон. ресурс]. – Режим доступу: <https://thestol.com>
4. RestOn [Електрон. ресурс]. – Режим доступу: https://reston.ua/kyiv_catalog/.
5. Al-Fedaghi S. Developing Web Applications [Text] / S. Al-Fedaghi // International Journal of Software Engineering and Its Applications. – 2011. – Vol. 5 No. 2, April, 2011 – P. 57-68.
6. Ласун [Електрон. ресурс]. – Режим доступу: <https://lasoon.net/ukr/>
7. Bailey T. Flutter For Beginners [Text] / T. Bailey, A. Biessek. – Packt Publishing, 2021. – 370 p.
8. Siddhartha S. Flutter Tutorial For Beginners With Dart [Text] / S. Siddhartha. – Tutorials Point, 2019. – 185 p.
9. Integrated Development [Електрон. ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Integrated_development_environment
10. Cianci L. Best IDEs for Flutter in 2022 [Електрон. ресурс]. – Режим доступу: <https://blog.logrocket.com/best-ides-flutter-2022/>
11. Ashad W. Managing State in Flutter Pragmatically [Text] / W. Ashad. – Packt Publishing, 2021. – 227 p.
12. Firebase Realtime Database [Електрон. ресурс]. – Режим доступу: <https://firebase.google.com/docs/database>
13. Testing Flutter apps [Електрон. ресурс]. – Режим доступу: <https://docs.flutter.dev/testing>

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.1 Назва програми

Мобільний застосунок “Free Table”.

A.1.2 Призначення та область застосування

Програмне забезпечення розробляється для використання в закладах харчування. Автоматизація ведення обліку бронювань та онлайн бронювання столиків є дуже важливою для закладів. Розроблення застосунку з доступом користувача в свою чергу позитивно впливає на ріст кількості клієнтів та збільшення прибутків.

A.2 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація застосунку для обліку бронювань в закладах харчування» затверджене наказом номер 102 від 27 квітня 2022 р. по Національному університету "Запорізька політехніка".

A.3 Призначення розробки

Програмне забезпечення призначене для обліку бронювання, пошуку закладів, створення бронювань, керування заявками.

A.4 Вимоги до програми

A.4.1 Вимоги до функціональних характеристик

Програмне забезпечення «Free Table» має виконувати наступні функціональні задачі:

- реєстрація користувача;
- авторизація користувача;
- додавання, видалення та редагування даних про зони розташування столиків;
- додавання, видалення бронювань;
- зміна статусу бронювань;
- перегляд закладів;
- зміна даних профілю.

A.4.2 Вимоги до надійності

Система повинна розпізнавати ситуації нерозуміння системою користувача (користувачем програми), а також мати обробку збійних ситуацій. Наприклад, якщо користувач вводить телефон або ім'я, що не відповідають формату – то повинно виводитись необхідне повідомлення. Також якщо не введені всі поля бронювання, то кнопка, що додає бронювання в БД має бути неактивною. Якщо користувач вводить дані входу, які вже є в БД, повинно виводитись повідомлення, що такий користувач існує. Перед тим, як видалити якийсь об'єкт або змінити статус заявки, повинно виводитись повідомлення про підтвердження.

A.4.3 Вимоги до технічних та програмних засобів

Для забезпечення коректного функціонування ПЗ у користувача має бути смартфон з такими характеристиками:

- ОС Android 9.0 і вище;
- оперативну пам'ять об'ємом, 1Гігабайт, не менше;
- вільну пам'ять 100 Мб, не менше.

A.5 Стадії і етапи розробки

Розробка повинна бути проведена в чотири стадії.

Розробка технічного завдання.

Стадія «Розробка технічного завдання» складається з наступних етапів:

- розробка технічного завдання;
- узгодження технічного завдання;
- затвердження технічного завдання.

Робоче проектування.

Стадія «Робоче проектування» складається з наступних етапів:

- розробка програми;
- розробка програмної документації;
- випробування програми.

Тестування програми.

Здача роботи в експлуатацію.

На цій стадії повинен бути виконаний етап підготовки і передачі програми.

A.6 Порядок контролю та прийняття

Дипломна робота повинна відповідати вимогам, які зазначені в пункті A.4 та повинна бути перевірена керівником роботи.

Порядок прийняття роботи відповідає вимогам прийняття бакалаврських дипломних робіт.

ДОДАТОК Б
Текст програми

LoginScreen.dart

```
import 'package:flutter/material.dart';
import 'package:free_table/presentation/screens/auth/login/mixins/login_screen_mixin.dart';
import 'package:free_table/presentation/shared/layouts/main_layout.dart';
import 'package:free_table/presentation/shared/widgets/app_button.dart';
import 'package:free_table/presentation/shared/widgets/app_text_button.dart';
import 'package:free_table/presentation/shared/widgets/app_text_field.dart';
import 'package:free_table/resources/app_paths.dart';
import 'package:free_table/resources/assets/app_png_assets.dart';
import 'package:free_table/resources/assets/app_svg_assets.dart';
import 'package:free_table/resources/theme/app_text_styles.dart';

class LoginScreen extends StatefulWidget {
  const LoginScreen({Key? key}) : super(key: key);

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> with LoginMixin {
  @override
  void initState() {
    super.initState();
    codeController = TextEditingController();
    phoneController = TextEditingController();
  }

  @override
  void dispose() {
    codeController.dispose();
    phoneController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return MainLayout.withFocusLayout(
      resizeToAvoidBottomInset: true,
      child: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 8.0),
          child: Form(
            key: formKey,
            child: Column(
              children: [
                _image,
                spacer,
                Visibility(
                  visible: !isCodeEntering,
                  child: Padding(
                    padding: EdgeInsets.only(bottom: 16.0),
                    child: phoneField(),
                  ),
                ),
                Visibility(
                  visible: isCodeEntering,
                  child: Padding(
                    padding: EdgeInsets.only(bottom: 16.0),
                    child: codeField(),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

```

        ),
      ),
      loginButton(),
      Text(
        errorText,
        style: AppTextStyles.m16_500(color: Colors.redAccent),
      ),
      spacer,
      signUpButton(),
      spacer
    ],
  ),
),
),
),
);
}

AppTextButton signUpButton() {
  return AppTextButton(
    label: "Don't have an account? Sign Up",
    onTap: () {
      router.replaceNamed(AppPaths.signupScreen);
    },
  );
}

AppButton loginButton() {
  return AppButton(
    label: !isCodeEntering ? 'Send code' : 'Login',
    onTap: onTap,
  );
}

AppTextField codeField() {
  return AppTextField(
    textController: codeController,
    prefixIcon: AppSvgAssets.password,
    labelText: 'Code',
    onChanged: (value) {
      formKey.currentState!.validate();
    },
    validator: (value) {
      if (value != null && value.isEmpty) {
        return 'Enter code';
      }
      return null;
    },
  );
}

AppTextField phoneField() {
  return AppTextField(
    textController: phoneController,
    prefixIcon: AppSvgAssets.phone,
    keyboardType: TextInputType.phone,
    prefix: Text(
      '+38 ',
      style: AppTextStyles.m16_500(),
    ),
    labelText: 'Phone',
    onChanged: (value) {

```

```

        formKey.currentState!.validate();
      },
      validator: (value) {
        if (value != null && value.length != 10) {
          return 'Invalid number';
        }
        return null;
      },
    );
  }
}

Widget get spacer => SizedBox(height: 16.0);

Widget get _image {
  return Image.asset(
    AppPngAssets.food,
    height: 300.0,
  );
}
}

```

LoginScreenMixin.dart

```

import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:free_table/config/injection_config.dart';
import 'package:free_table/presentation/screens/auth/login/login_screen.dart';
import 'package:free_table/resources/app_paths.dart';
import 'package:free_table/src/auth/entity/dto/user_type_body/user_type_dto.dart';
import 'package:free_table/src/main/domain/bloc/app_bloc.dart';
import 'package:free_table/src/main/domain/bloc/events/check_phone_event.dart';
import 'package:free_table/src/main/domain/bloc/events/check_type_event.dart';
import 'package:free_table/src/main/domain/bloc/events/save_token_event.dart';
import 'package:free_table/src/main/domain/bloc/events/save_type_event.dart';

mixin LoginMixin on State<LoginScreen> {
  late final TextEditingController phoneController;
  late final TextEditingController codeController;
  FirebaseAuth auth = FirebaseAuth.instance;

  final formKey = GlobalKey<FormState>();

  AppRouter get router => getIt<AppRouter>();

  bool isCodeEntering = false;

  String verificationID = '';

  String errorText = '';

  void onTap() async {
    final AppBloc appBloc = BlocProvider.of<AppBloc>(context);
    if (isCodeEntering) {
      final PhoneAuthCredential credential = PhoneAuthProvider.credential(
        verificationId: verificationID,
        smsCode: codeController.text,
      );
      await auth.signInWithCredential(credential);
      final token = await auth.currentUser?.getIdTokenResult(true);
    }
  }
}

```

```

appBloc.add(SaveTokenEvent(token: token?.token));
String? type;
appBloc.add(CheckTypeEvent(
  phone: '38' + phoneController.text,
  onComplete: (userType) {
    type = userType?.toLowerCase();
    if (type == UserType.admin.name) {
      router.replaceNamed(AppPaths.tabScreenAdmin);
    } else if (type == UserType.customer.name) {
      router.replaceNamed(AppPaths.tabScreenCustomer);
    }

    appBloc.add(SaveTypeEvent(type: type));
  }));
}
if (formKey.currentState!.validate()) {
  FocusScope.of(context).unfocus();

  appBloc.add(CheckPhoneEvent(
    phone: '38' + phoneController.text,
    onComplete: (isPhoneExist) async {
      if (isPhoneExist) {
        errorText = '';
        setState(() {});
        await auth.verifyPhoneNumber(
          phoneNumber: '+38' + phoneController.text,
          verificationCompleted: _onVerificationComplete,
          codeSent: _onCodeSent,
          verificationFailed: (exception) {
            print('failed $exception');
          },
          codeAutoRetrievalTimeout: (verificationId) {
            print('timeout');
          },
        );
      } else {
        errorText = 'This number is not registered';
        setState(() {});
      }
    }));
}
}

void _onCodeSent(verificationId, resentToken) async {
  verificationID = verificationId;
  isCodeEntering = true;
  setState(() {});
}

void _onVerificationComplete(credentials) async {
  await auth
    .signInWithCredential(
      credentials,
    )
    .then((value) {
      print('completed $credentials');
      print('value $value');
    });
}
}

```

TabScreenAdmin.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:free_table/presentation/screens/main_admin/bookings_admin/admin_bookings_screen.dart';
import 'package:free_table/presentation/screens/main_admin/home/home_screen.dart';
import 'package:free_table/presentation/shared/layouts/main_layout.dart';
import 'package:free_table/presentation/shared/profile/profile_screen.dart';
import 'package:free_table/presentation/shared/screen_bars/bottom/bottom_bar.dart';
import 'package:free_table/resources/assets/app_svg_assets.dart';
import 'package:free_table/src/main/domain/bloc/app_bloc.dart';

class TabScreenAdmin extends StatelessWidget {
  final tabScreens = [
    HomeScreenAdmin(),
    AdminBookingsScreen(),
    ProfileScreen(),
  ];

  TabScreenAdmin({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: BlocBuilder<AppBloc, AppState>(
        builder: (context, state) {
          return MainLayout(
            bottomBar: BottomBar(
              items: [
                BottomBarItemModel(
                  title: 'Home',
                  icon: AppSvgAssets.home,
                  index: 0,
                ),
                BottomBarItemModel(
                  title: 'Bookings',
                  icon: AppSvgAssets.bookings,
                  index: 1,
                ),
                BottomBarItemModel(
                  title: 'User',
                  icon: AppSvgAssets.user,
                  index: 2,
                ),
              ],
            ),
            child: tabScreens[state.currentTab],
          );
        },
      ),
    );
  }
}
```

BottomBarItem.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
```

```

import 'package:flutter_svg/flutter_svg.dart';
import 'package:free_table/resources/theme/app_colors.dart';
import 'package:free_table/resources/theme/app_text_styles.dart';
import 'package:free_table/src/main/domain/bloc/app_bloc.dart';
import 'package:free_table/src/main/domain/bloc/events/change_tab.dart';

class BottomBarItem extends StatelessWidget {
  final String title;
  final String icon;
  final int index;
  final int currentIndex;

  const BottomBarItem({
    required this.title,
    required this.icon,
    required this.index,
    required this.currentIndex,
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return InkWell(
      onTap: () => BlocProvider.of<AppBloc>(context).add(ChangeTabEvent(index)),
      child: AnimatedContainer(
        duration: Duration(milliseconds: 300),
        height: 40,
        alignment: Alignment.center,
        margin: EdgeInsets.symmetric(horizontal: 5.0),
        padding: EdgeInsets.symmetric(horizontal: 10.0, vertical: 8.0),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(20.0),
          color: currentIndex == index
            ? AppColors.orange.withOpacity(0.25)
            : AppColors.lightOrange,
        ),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            iconWidget(),
            Padding(
              padding: const EdgeInsets.only(left: 8.0),
              child: titleWidget(),
            ),
          ],
        ),
      ),
    );
  }

  Text titleWidget() {
    return Text(
      title,
      style: AppTextStyles.m16_800(),
    );
  }

  SvgPicture iconWidget() {
    return SvgPicture.asset(
      icon,
      color: AppColors.orange,
      height: 25.0,
    );
  }

```

```

        width: 25.0,
      );
    }
  }
}

```

AcceptedBookingsList.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:free_table/presentation/screens/main_admin/bookings_admin/components/booking.dart';
import 'package:free_table/src/admin_bookings/domain/bloc/admin_bookings_bloc.dart';
import 'package:free_table/src/booking/entity/booking_info_dto/booking_info_dto.dart';

class AcceptedBookingsList extends StatelessWidget {
  const AcceptedBookingsList({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<AdminBookingsBloc, AdminBookingsState>(
      builder: (context, state) {
        final List<BookingInfoDTO> list = state.currentBookings
          .where((element) =>
            element?.info?.accepted != null &&
            element!.info!.accepted! &&
            element.info!.dateTime!.difference(DateTime.now()).inMinutes > 0)
          .toList();

        list.sort((infoA, infoB) => infoA!.info!.dateTime!.compareTo(infoB!.info!.dateTime!));

        return ListView.builder(
          padding: EdgeInsets.symmetric(horizontal: 16.0, vertical: 24.0),
          itemCount: list.length,
          itemBuilder: (context, index) => Padding(
            padding: const EdgeInsets.only(bottom: 12.0),
            child: Booking(
              info: list.elementAt(index),
              acceptButton: false,
            ),
          ),
        );
      },
    );
  }
}

```

Booking.dart

```

import 'package:flutter/material.dart';
import 'package:free_table/presentation/screens/main_admin/bookings_admin/mixins/booking_mixin.dart';
import 'package:free_table/presentation/shared/widgets/app_icon_button.dart';
import 'package:free_table/presentation/shared/widgets/app_button.dart';
import 'package:free_table/presentation/shared/widgets/app_card.dart';
import 'package:free_table/resources/theme/app_colors.dart';
import 'package:free_table/resources/theme/app_text_styles.dart';

```

```

import
age:free_table/src/booking/entity/booking_info_dto/booking_info_dto.dart';
import 'package:intl/intl.dart';
import 'package:url_launcher/url_launcher.dart';

class Booking extends StatelessWidget with BookingMixin {
  final bool acceptButton;
  final bool declineButton;
  final BookingInfoDTO? info;

  const Booking({
    required this.info,
    this.acceptButton = true,
    this.declineButton = true,
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return AppCard(
      padding: EdgeInsets.zero,
      borderRadius: BorderRadius.circular(10.0),
      child: Column(
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16.0),
            child: Row(
              children: [
                if (info?.info?.dateTime != null) time(),
                customerInfo(),
                tableInfo(context),
              ],
            ),
          ),
          SizedBox(
            child: Row(
              children: [
                if (acceptButton)
                  Expanded(
                    child: AppButton(
                      label: 'Accept',
                      borderRadius: acceptBorder,
                      onTap: () => onAccept(context, info),
                    ),
                  ),
                if (declineButton)
                  Expanded(
                    child: AppButton(
                      label: 'Decline',
                      appButtonType: AppButtonType.alternative,
                      borderRadius: declineBorder,
                      onTap: () => onDecline(context, info),
                    ),
                  ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}

```

'pack-

```

Radius get radius => Radius.circular(8.0);

BorderRadius get acceptBorder {
  if (acceptButton && declineButton) return BorderRadius.only(bottomLeft: radius);
  return BorderRadius.only(bottomLeft: radius, bottomRight: radius);
}

BorderRadius get declineBorder {
  if (acceptButton && declineButton) return BorderRadius.only(bottomRight: radius);
  return BorderRadius.only(bottomLeft: radius, bottomRight: radius);
}

Expanded customerInfo() {
  return Expanded(
    child: Padding(
      padding: const EdgeInsets.all(8.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            info?.info?.customer ?? '-',
            style: AppTextStyles.m16_800(color: AppColors.eclipse),
          ),
          if (info?.info?.dateTime != null)
            Text(
              DateFormat.MEd().format(info!.info!.dateTime!),
              style: AppTextStyles.m16_500(color: AppColors.eclipse),
            ),
        ],
      ),
    ),
  );
}

Padding time() {
  return Padding(
    padding: const EdgeInsets.only(right: 8.0),
    child: Text(
      DateFormat.Hm().format(info!.info!.dateTime!),
      style: AppTextStyles.h28_800(color: AppColors.eclipse),
    ),
  );
}

Widget tableInfo(BuildContext context) {
  return Row(
    children: [
      Icon(
        Icons.people,
        color: AppColors.orange,
        size: 20,
      ),
      SizedBox(width: 2.0),
      Text(
        'x' + info!.info!.peopleCount.toString(),
        style: AppTextStyles.m14_500(),
      ),
      SizedBox(width: 10.0),
      AppIconButton(
        onTap: () => onTablesOpen(context, info),
        icon: Icons.table_bar_outlined,
      ),
    ],
  );
}

```

```

        SizedBox(width: 10.0),
        AppIconButton(
          onTap: () => launch('tel: +${info!.info!.restaurantPhone!}'),
          icon: Icons.phone,
        ),
      ],
    );
  }
}

```

BookingTabs.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:free_table/presentation/screens/main_admin/bookings_admin/components/accepted_bookings_list.dart';
import 'package:free_table/presentation/screens/main_admin/bookings_admin/components/declined_bookings_list.dart';
import 'package:free_table/presentation/screens/main_admin/bookings_admin/components/new_bookings_list.dart';
import 'package:free_table/presentation/shared/widgets/loader.dart';
import 'package:free_table/resources/theme/app_colors.dart';
import 'package:free_table/src/admin_bookings/domain/bloc/admin_bookings_bloc.dart';
import 'package:free_table/src/admin_bookings/domain/bloc/events/get_admin_bookings_event.dart';
import 'package:free_table/src/zone/domain/bloc/zone_bloc.dart';

class BookingTabs extends StatefulWidget {
  const BookingTabs({Key? key}) : super(key: key);

  @override
  State<BookingTabs> createState() => _BookingTabsState();
}

class _BookingTabsState extends State<BookingTabs> with SingleTickerProviderStateMixin {
  late final TabController _controller;

  @override
  void initState() {
    _controller = TabController(length: 3, vsync: this);
    BlocProvider.of<AdminBookingsBloc>(context).add(GetAdminBookingsEvent());
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        TabBar(
          overlayColor: MaterialStateProperty.all(AppColors.eclipse),
          controller: _controller,
          labelColor: AppColors.eclipse,
          indicatorColor: AppColors.eclipse,
          labelPadding: EdgeInsets.all(8.0),
          tabs: [
            Text('New'),

```

```

        Text('Accepted'),
        Text('Declined'),
      ],
    ),
    SizedBox(
      height: MediaQuery.of(context).size.height * 0.70,
      child: BlocBuilder<AdminBookingsBloc, AdminBookingsState>(
        builder: (context, state) {
          if (state.status == LoadingStatus.start || state.status == LoadingSta-
tus.pending) return Loader();
          return TabBarView(
            controller: _controller,
            children: [
              NewBookingsList(),
              AcceptedBookingsList(),
              DeclinedBookingsList(),
            ],
          );
        },
      ),
    ),
  ],
);
}
}

```

BookingMixin.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:free_table/config/injection_config.dart';
import 'package:free_table/presentation/pop_ups/general/no_button_dialog.dart';
import 'package:free_table/presentation/pop_ups/general/two_button_dialog.dart';
import 'package:free_table/presentation/pop_ups/pop_up_manager.dart';
import 'package:free_table/presentation/screens/main_admin/new_zone/components/new_zone_area.dart';
import 'package:free_table/src/admin_bookings/domain/bloc/admin_bookings_bloc.dart';
import 'package:free_table/src/admin_bookings/domain/bloc/events/accept_booking_event.dart';
import 'package:free_table/src/admin_bookings/domain/bloc/events/decline_booking_event.dart';
import 'package:free_table/src/booking/entity/booking_info_dto/booking_info_dto.dart';

mixin BookingMixin {
  Future<void> onTablesOpen(BuildContext context, BookingInfoDTO? info) async {
    Map<int, int> zoneMap = Map.fromIterable(
      List.generate(36, (index) => index),
      value: (element) => 0,
    );
    zoneMap[info!.info!.tableId!] = info.info!.peopleCount!;
    await context.showAsyncPopUp(
      NoButtonDialog(
        title: info.info!.zoneId!,
        body: NewZoneArea(
          currentZone: zoneMap,
          onTap: (index) {},
        ),
      ),
    );
  }
}

```

```

void onAccept(BuildContext context, BookingInfoDTO? info) {
  final bloc = BlocProvider.of<AdminBookingsBloc>(context);
  context.showAsyncPopUp(
    TwoButtonDialog(
      title: 'Are you sure to accept this booking?',
      acceptButtonText: 'Yes',
      declineButtonText: 'Cancel',
      onAcceptButtonTap: () {
        bloc.add(AcceptBookingEvent(info));
        context.popRoute();
      },
      onDeclineButtonTap: () {
        context.popRoute();
      },
    ),
  );
}

void onDecline(BuildContext context, BookingInfoDTO? info) {
  final bloc = BlocProvider.of<AdminBookingsBloc>(context);
  context.showAsyncPopUp(
    TwoButtonDialog(
      title: 'Are you sure to decline this booking?',
      acceptButtonText: 'Yes',
      declineButtonText: 'Cancel',
      onAcceptButtonTap: () {
        bloc.add(DeclineBookingEvent(info));
        context.popRoute();
      },
      onDeclineButtonTap: () {
        context.popRoute();
      },
    ),
  );
}
}

```

ZoneElement.dart

```

import 'package:flutter/material.dart';
import 'package:free_table/config/injection_config.dart';
import 'package:free_table/presentation/pop_ups/general/two_button_dialog.dart';
import 'package:free_table/presentation/pop_ups/pop_up_manager.dart';
import 'package:free_table/presentation/shared/widgets/app_card.dart';
import 'package:free_table/resources/theme/app_colors.dart';
import 'package:free_table/resources/theme/app_text_styles.dart';
import 'package:free_table/src/zone/entity/models/zone/zone.dart';

class ZoneElement extends StatelessWidget {
  final Zone zone;
  final VoidCallback? onTap;

  const ZoneElement({
    required this.zone,
    this.onTap,
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Stack(
      clipBehavior: Clip.none,

```

```

children: [
  InkWell(
    onTap: onTap,
    child: Padding(
      padding: const EdgeInsets.only(bottom: 26.0),
      child: AppCard(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            spacer,
            titleWidget(),
            spacer,
            Container(
              height: 1,
              color: AppColors.eclipse.withOpacity(0.2),
            ),
            spacer,
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                textElement('Total:'),
                textElement('${zone.totaleTables} tables'),
              ],
            ),
            ...zone.tablesMap.entries
              .map(
                (e) => Padding(
                  padding: const EdgeInsets.only(top: 8.0),
                  child: Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                      textElement('Tables for ${e.key} seats:'),
                      textElement('${e.value} tables'),
                    ],
                  ),
                ),
              ).toList(),
            spacer,
          ],
        ),
      ),
    ),
  ),
  Positioned(
    right: -10,
    top: -20,
    child: Container(
      decoration: BoxDecoration(
        boxShadow: [
          BoxShadow(color: Colors.black12.withOpacity(0.1), offset: Offset(-2,
2), blurRadius: 4.0),
        ],
        shape: BoxShape.circle,
        color: Colors.white,
      ),
      child: IconButton(
        onPressed: () async {
          await context.showAsyncPopUp(
            TwoButtonDialog(
              title: 'Delete this zone?',
              acceptButtonText: 'Yes',

```

```

        declineButtonText: 'Cancel',
        onAcceptButtonTap: () {
          context.popRoute();
        },
        onDeclineButtonTap: () {
          context.popRoute();
        },
      ),
    );
  },
  icon: Icon(
    Icons.delete,
    color: Colors.red,
  ),
),
),
),
],
);
}

Text textElement(String text) {
  return Text(
    text,
    style: AppTextStyles.m16_500(
      color: AppColors.eclipse,
    ),
  );
}

Text titleWidget() {
  return Text(
    zone.title,
    style: AppTextStyles.m16_800(
      color: AppColors.eclipse,
    ),
  );
}

PreferredSize get spacer => SizedBox(height: 8.0);
}

```

BookingApi.dart

```

import 'package:dio/dio.dart';
import 'package:free_table/src/booking/entity/booking_info/booking_info.dart';
import 'package:free_table/src/main/domain/entity/name/name.dart';
import 'package:retrofit/http.dart';

part 'booking_api.g.dart';

@RestApi()
abstract class BookingApi {
  factory BookingApi(Dio dio, {String baseUrl}) = _BookingApi;

  @POST('/bookings.json')
  Future<Name?> addBooking(
    @Body() BookingInfo body,
    @Query('auth') String? token,
  );
}

```

```

@PATCH('/bookings/{id}.json')
Future<Name?> patchBooking(
  @Path() String id,
  @Body() BookingInfo body,
  @Query('auth') String? token,
);

@DELETE('/bookings/{id}.json')
Future<dynamic> deleteBooking(
  @Path() String id,
  @Query('auth') String? token,
);

@GET('/bookings.json')
Future<dynamic> getBookings({
  @Query('orderBy') required String orderBy,
  @Query('equalTo') required String equalTo,
  @Query('auth') String? token,
});
}

```

GetZonesEvent.dart

```

import 'package:free_table/src/booking/domain/bloc/booking_bloc.dart';
import 'package:free_table/src/zone/domain/bloc/zone_bloc.dart';

class GetZones extends BookingEvent {
  final String phone;

  GetZones({required this.phone});

  @override
  Stream<BookingState> action(covariant BookingBloc bloc) async* {
    yield bloc.state.copyWith(status: LoadingStatus.start);
    try {
      final fetchedMap = await bloc.zoneService.getZones(phone: phone);
      yield bloc.state.copyWith(
        currentRestZones: fetchedMap,
        status: LoadingStatus.success,
      );
    } catch (e) {
      yield bloc.state.copyWith(status: LoadingStatus.failure);
    }
  }
}

```

SetDateEvent.dart

```

import 'package:flutter/material.dart';
import 'package:free_table/src/booking/domain/bloc/booking_bloc.dart';
import 'package:free_table/src/booking/entity/booking_info/booking_info.dart';

class SetDateEvent extends BookingEvent {
  final DateTime? dateTime;

  SetDateEvent(this.dateTime);

  @override
  Stream<BookingState> action(covariant BookingBloc bloc) async* {

```

```

    if (dateTime != null) {
      final BookingInfo bookingInfo = bloc.state.currentBooking ?? BookingInfo();
      if (bookingInfo.dateTime == null) {
        yield bloc.state.copyWith(
          currentBooking: bookingInfo.copyWith(dateTime:
dateTime!.applied(TimeOfDay.now()))),
        );
      } else {
        yield bloc.state.copyWith(
          currentBooking: bookingInfo.copyWith(dateTime: bookingIn-
fo.dateTime?.appliedDate(dateTime!)),
        );
      }
    }
  }
}

```

BookingRepository.dart

```

import 'package:dio/dio.dart';
import 'package:free_table/config/app_config.dart';
import 'package:free_table/config/modules/dio_module_config.dart';
import 'package:free_table/src/auth/data/storages/token_storage.dart';
import 'package:free_table/src/auth/entity/dto/user_type_body/user_type_dto.dart';
import 'package:free_table/src/auth/entity/models/token/token.dart';
import 'package:free_table/src/booking/data/booking_api.dart';
import 'package:free_table/src/booking/entity/booking_info/booking_info.dart';
import 'package:free_table/src/booking/entity/booking_info_dto/booking_info_dto.dart';
import 'package:free_table/src/booking/entity/restaurant_info/restaurant_info.dart';
import 'package:free_table/src/main/domain/entity/name/name.dart';
import 'package:injectable/injectable.dart';

@lazySingleton
class BookingRepository {
  final AppConfig config;
  final TokenStorage tokenStorage;
  final Dio _dio;

  final BookingApi api;

  BookingRepository(
    @Named(unauthorized) Dio dio,
    this.config,
    this.tokenStorage,
  ) : api = BookingApi(dio, baseUrl: config.baseUrl),
      _dio = dio;

  Future<List<BookingInfoDTO>?> getCustomerBookings(String phone) async {
    if (config.useMock) {
      return List.generate(12, (index) => BookingInfoDTO.mock());
    }
    try {
      final Token? token = await tokenStorage.take();
      final response = await _dio.get(
        config.baseUrl +
        '/bookings.json?auth=${token?.accessToken}&orderBy="phone"&equalTo="$phone"',
      );
      final List<BookingInfoDTO> list = [];

      final responseMap = Map.from(response.data);
    }
  }
}

```

```

        responseMap.forEach((key, value) {
            final BookingInfoDTO booking = BookingInfoDTO(id: key, info: BookingIn-
fo.fromJson(value));
            list.add(booking);
            print(booking);
        });

        return list;
    } catch (e) {
        return null;
    }
}

Future<List<BookingInfoDTO>> getAdminBookings(String phone) async {
    if (config.useMock) {
        return List.generate(12, (index) => BookingInfoDTO.mock());
    }
    try {
        final Token? token = await tokenStorage.take();
        final response = await _dio.get(
            config.baseUrl +
            '/bookings.json?auth=${token?.accessToken}&orderBy="restaurantPhone"&equalTo="$phone"
            ',
        );
        final List<BookingInfoDTO> list = [];

        final responseMap = Map.from(response.data);
        responseMap.forEach((key, value) {
            final BookingInfoDTO booking = BookingInfoDTO(id: key, info: BookingIn-
fo.fromJson(value));
            list.add(booking);
            print(booking);
        });

        return list;
    } catch (e) {
        return null;
    }
}

Future<List<BookingInfoDTO>> getRestBookings(String phone) async {
    if (config.useMock) {
        return List.generate(12, (index) => BookingInfoDTO.mock());
    }
    try {
        final Token? token = await tokenStorage.take();
        final response = await _dio.get(
            config.baseUrl +
            '/bookings.json?auth=${token?.accessToken}&orderBy="restaurantPhone"&equalTo="$phone"
            ',
        );
        final List<BookingInfoDTO> list = [];

        final responseMap = Map.from(response.data);

        responseMap.forEach((key, value) {
            final BookingInfoDTO booking = BookingInfoDTO(id: key, info: BookingIn-
fo.fromJson(value));
            if (booking.info?.accepted != null && booking.info!.accepted!)
list.add(booking);
        });
    }
}

```

```

        return list;
    } catch (e) {
        return null;
    }
}

Future<List<RestaurantInfo>> getRests() async {
    if (config.useMock) {
        return List.generate(12, (index) => RestaurantInfo.mock());
    }
    try {
        final Token? token = await tokenStorage.take();
        final List<RestaurantInfo> list = [];

        final response = await _dio.get(
            config.baseUrl + '/user-types.json?auth=${token?.accessToken}',
        );

        final responseMap = Map.from(response.data);
        responseMap.forEach((key, value) {
            final UserDTO user = UserDTO.fromJson(value);
            if (user.type?.toLowerCase() == UserType.admin.name) {
                list.add(RestaurantInfo(
                    name: user.name,
                    phone: user.phone,
                    id: key,
                ));
            }
        });
        return list;
    } catch (e) {
        return null;
    }
}

Future<Name?> addBooking(BookingInfo body) async {
    final Token? token = await tokenStorage.take();

    return api.addBooking(body, token?.accessToken);
}

Future<dynamic> deleteBooking(String id) async {
    if (config.useMock) return;

    final Token? token = await tokenStorage.take();

    return api.deleteBooking(id, token?.accessToken);
}

Future<dynamic> declineBooking(String id, BookingInfo info) async {
    if (config.useMock) return;
    final Token? token = await tokenStorage.take();

    return api.patchBooking(
        id,
        info.copyWith(accepted: false),
        token?.accessToken,
    );
}

Future<dynamic> acceptBooking(String id, BookingInfo info) async {
    if (config.useMock) return;

```

```

    final Token? token = await tokenStorage.take();

    return api.patchBooking(
      id,
      info.copyWith(accepted: true),
      token?.accessToken,
    );
  }
}

```

DeleteBookingEvent.dart

```

import 'package:free_table/src/booking/entity/booking_info_dto/booking_info_dto.dart';
import 'package:free_table/src/customer_bookings/domain/bloc/customer_bookings_bloc.dart';

class DeleteBookingEvent extends CustomerBookingsEvent {
  final BookingInfoDTO? info;

  DeleteBookingEvent(this.info);

  @override
  Stream<CustomerBookingsState> action(covariant CustomerBookingsBloc bloc) async* {
    if (info?.id != null) await bloc.bookingService.deleteBooking(info!.id!);

    final list = bloc.state.currentBookings;
    list.remove(info);

    yield bloc.state.copyWith(currentBookings: list);} }

```

GetCustomerBookingsEvent.dart

```

import 'package:firebase_auth/firebase_auth.dart';
import 'package:free_table/src/customer_bookings/domain/bloc/customer_bookings_bloc.dart';
import 'package:free_table/src/zone/domain/bloc/zone_bloc.dart';

class GetCustomerBookingsEvent extends CustomerBookingsEvent {
  final String? phone;

  GetCustomerBookingsEvent({this.phone});

  @override
  Stream<CustomerBookingsState> action(covariant CustomerBookingsBloc bloc) async* {
    String? userPhone;
    if (phone == null) {
      userPhone = FirebaseAuth.instance.currentUser?.phoneNumber;
    }
    if (userPhone != null || phone != null) {
      final phoneNumber = (phone ?? userPhone)!.replaceAll('+', '');

      final bookings = await bloc.bookingService.getCustomerBookings(phoneNumber) ?? [];

      yield bloc.state.copyWith(
        currentBookings: bookings,
        status: LoadingStatus.success,
      );
    }
  }
}

```

ДОДАТОК В
Слайди презентації

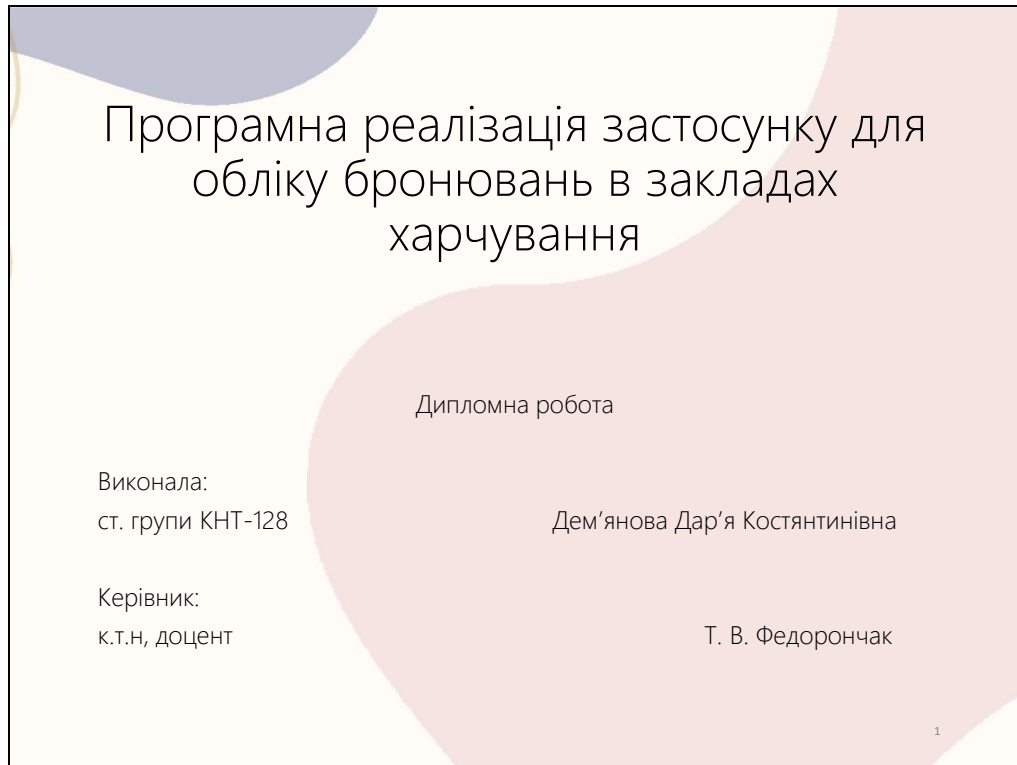


Рисунок В.1 – Слайд 1



Рисунок В.2 – Слайд 2

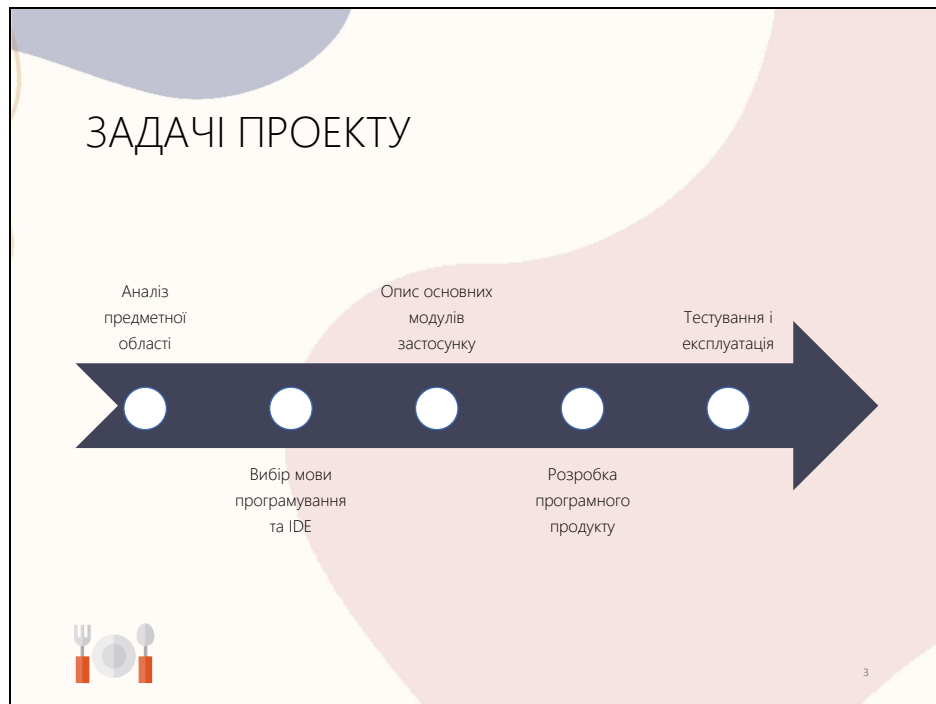


Рисунок В.3 – Слайд 3

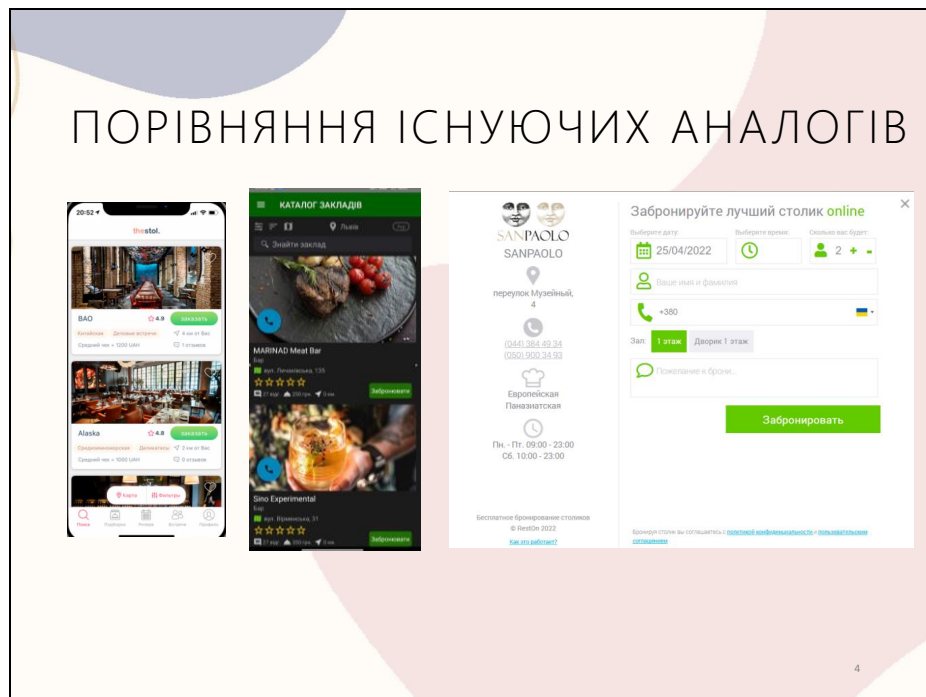


Рисунок В.4 – Слайд 4

ПОРІВНЯННЯ ІСНУЮЧИХ АНАЛОГІВ

Критерій оцінювання	Назва застосунку		
	thestol	RestOn	Ласун
Обліковий запис	+	+	+
Дані про власні бронювання	+	+	-
Створення акаунту адміністратора	-	-	+
Схема розташування місць	-	-	+
Можливість переглядати і редагувати заявки	-	-	-
Адаптивність до різних девайсів/платформ	-	+	+-
Облік заявок з боку адміністратора (підтвердження/видалення)	-	-	+

5

Рисунок В.5 – Слайд 5

ВИБІР МОВИ ПРОГРАМУВАННЯ

Забезпечує високу продуктивність, відтворюючи інтерфейс користувача безпосередньо на полотні операційної системи.

Програма Flutter сама по собі є віджетом.

Логіка програми заснована на реактивному програмуванні.

Ahead Of Time (AOT) та Just In Time (JIT) компіляція

Flutter використовує ефективний збір сміття

Dart — це гнучка, надійна, сучасна та просунута мова.

Декларативний інтерфейс користувача.

Переваги Dart і Flutter

6

Рисунок В.6 – Слайд 6

ПОРІВНЯННЯ СЕРЕДОВИЩ РОЗРОБКИ

Критерій	VS Code	Android Studio	IntelliJ IDEA
Підтримка розширень	+	+	+
Швидкий запуск програми	+	-	+/-
Швидка компіляція	+	-	+/-
Розділення на окремі вікна	-	+	+
Велика кількість розширень для розробки	+	+/-	+/-
Малий обсяг застосунку	+	+/-	+/-

7

Рисунок В.7 – Слайд 7



Рисунок В.8 – Слайд 8

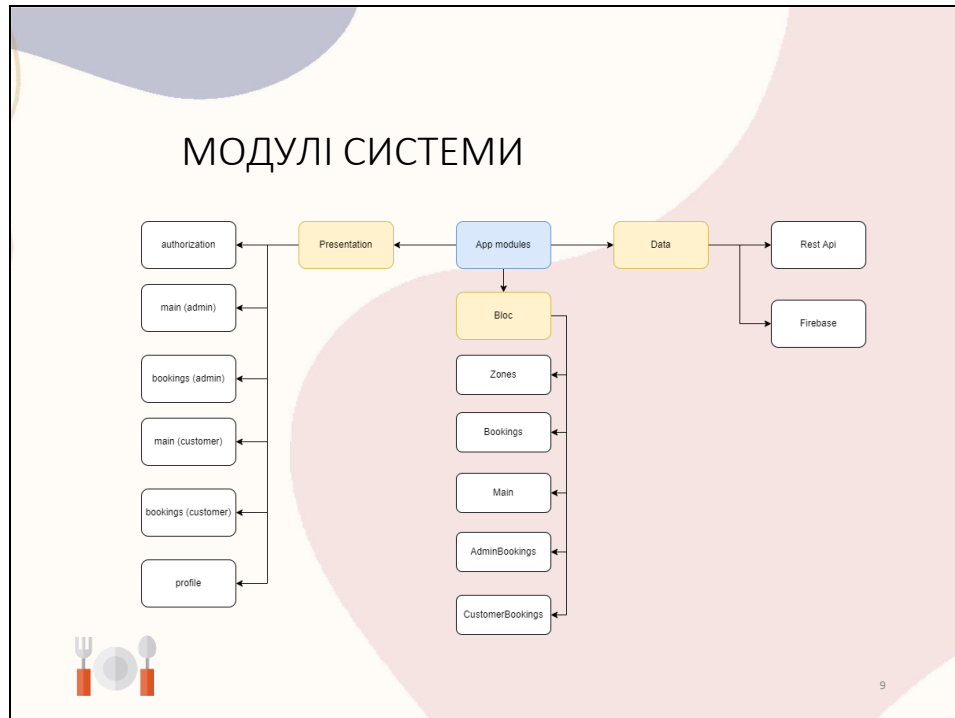


Рисунок В.9 – Слайд 9

ПРОЕКТУВАННЯ БАЗИ ДАНИХ

Firestore – це платформа, розроблена Google, яка має Cloud Firestore та Realtime Database – бази для зберігання даних. База даних **Firestore Realtime** — це хмарна база даних. Дані зберігаються у форматі **JSON** і синхронізуються в режимі реального часу з кожним підключеним клієнтом.

Realtime Database

URL: <https://free-table-89aa6-default-rtdb.firebaseio.com/>

- bookings
- user-types
- zone

URL: <https://free-table-89aa6-default-rtdb.firebaseio.com/>

```

{
  "accepted": true,
  "customer": "Dasha",
  "dateTime": "2022-05-22T01:53:00.000",
  "peopleCount": 3,
  "phone": "380555555555",
  "restaurantPhone": "380992560440",
  "tableId": 3,
  "zoneId": "myzone"
}
          
```

Рисунок В.10 – Слайд 10

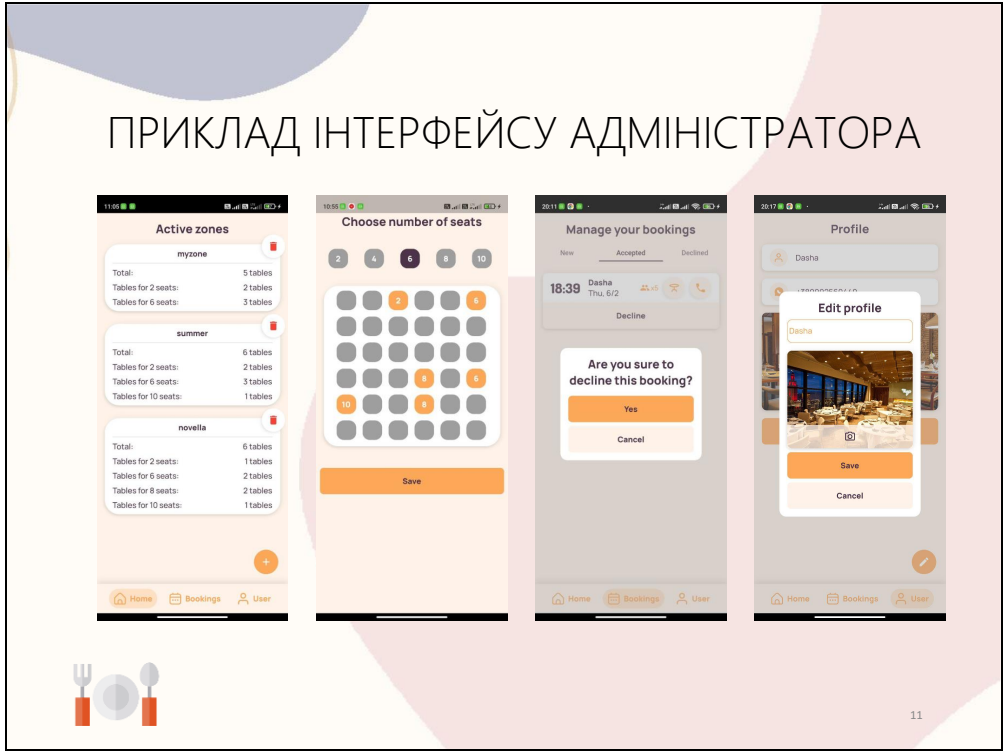


Рисунок В.11 – Слайд 11

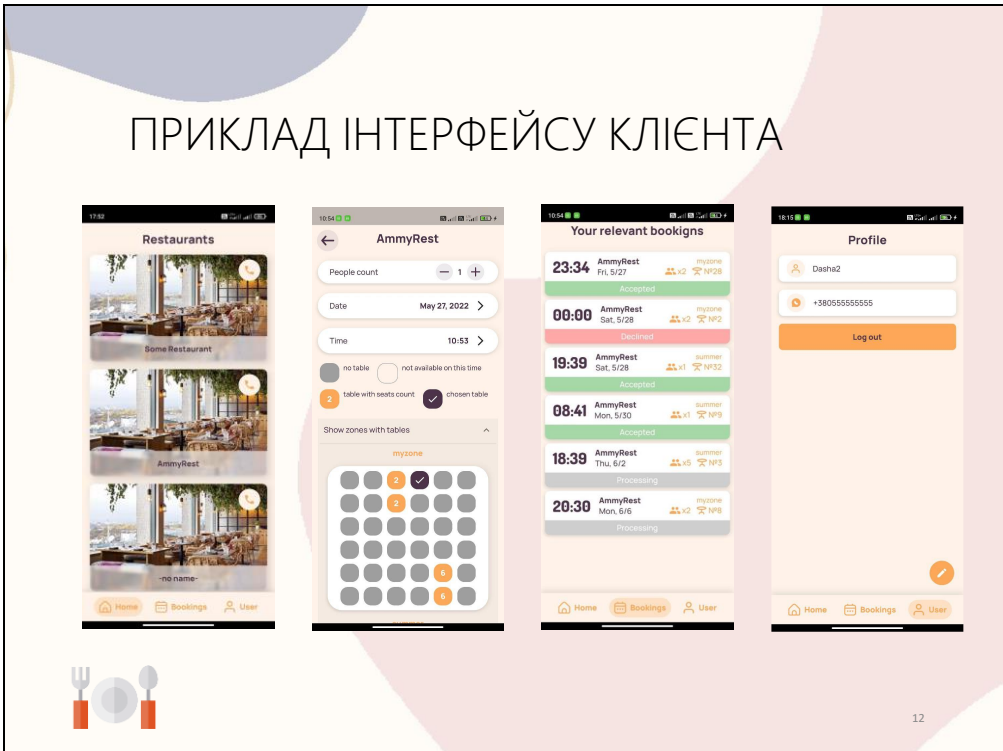


Рисунок В.12 – Слайд 12

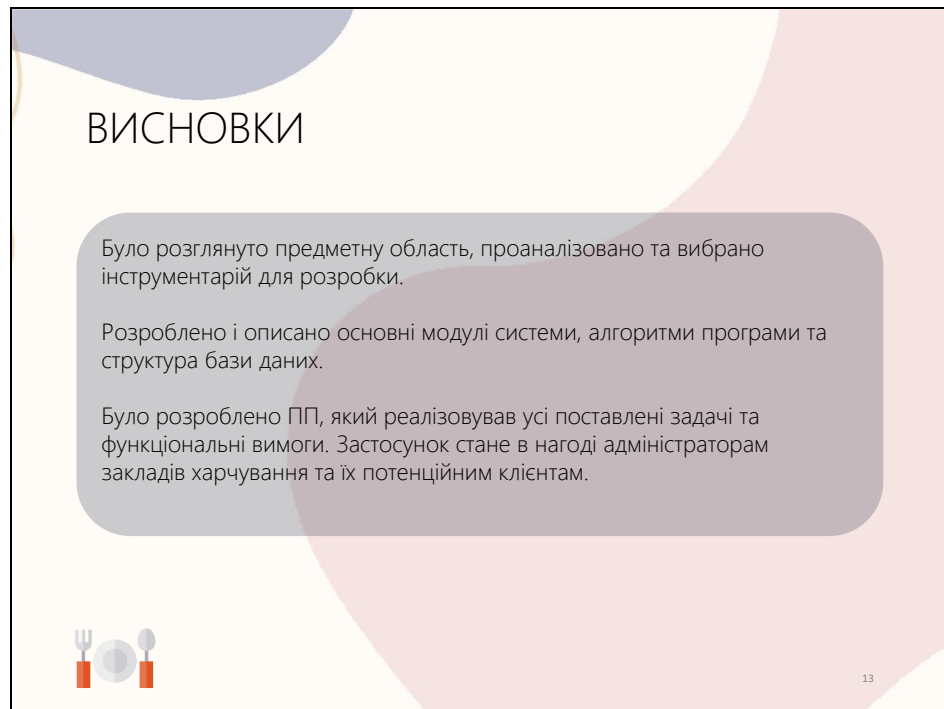


Рисунок В.13 – Слайд 13

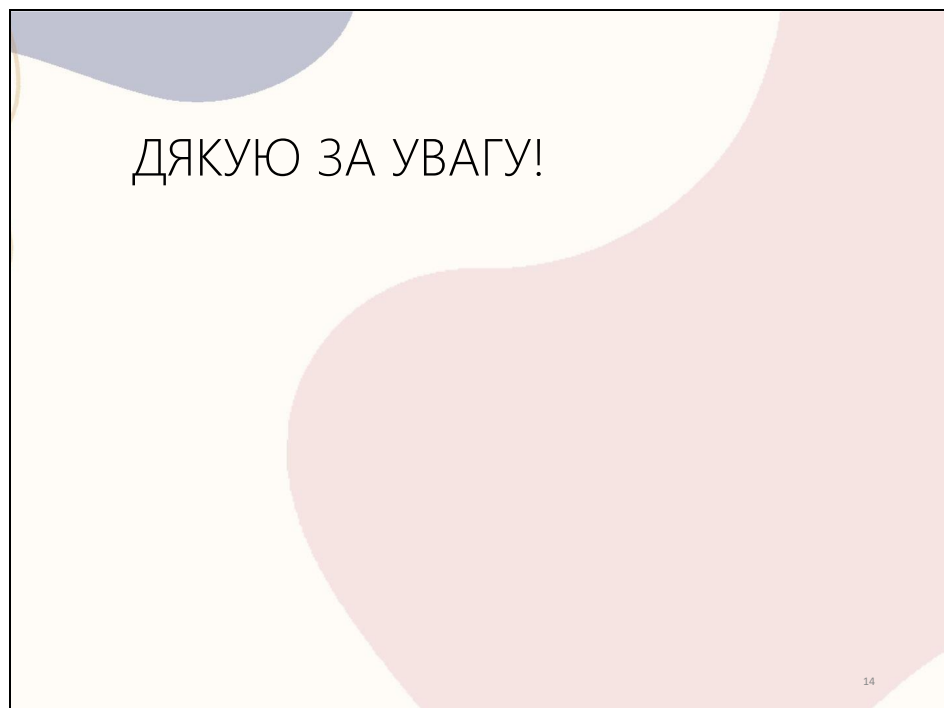


Рисунок В.14 – Слайд 14