ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку путівника для легшого подорожування Японією мовою Dart з використанням фреймворку Flutter»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артем СТЕПАНЧЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Артем СТЕПАНЧЕНКО

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Степанченку Артему Олеговичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку путівника для легшого подорожування Японією мовою Dart з використанням фреймворку Flutter» керівник кваліфікаційної роботи доктор філософії (PhD), доцент кафедри ІПЗ Олесь ДІБРІВНИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки мобільних додатків, опис методів інтеграції картографічних сервісів та систем рекомендацій, технічна документація з описом бібліотек для обробки даних та побудови користувацького інтерфейсу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих застосунків, що мають подібний функціонал.
2. Проектування застосунку для легшого подорожування Японією мовою Dart з використанням фреймворку Flutter.

3. Програмна реалізація та опис функціонування застосунку для легшого подорожування Японією мовою Dart з використанням фреймворку Flutter.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. User Flow діаграма.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Проектування застосунку для легшого подорожування Японією мовою Dart з використанням фреймворку Flutter.	14.03-13.04.2024	
4	Програмна реалізація та опис функціонування застосунку для легшого подорожування Японією мовою Dart з використанням фреймворку Flutter.	14.04-18.04.2024	
5	Проектування діаграм	19.04-22.04.2024	
6	Проведення тестування мобільного застосунку	23.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Артем СТЕПАНЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 1 табл., 50 рис., 20 джерел.

Мета роботи – спрощення процесу подорожування Японією, знаходження інформації про місця та планування подорожі.

Об'єкт дослідження – процес подорожування Японією, знаходження інформації про місця та планування подорожі.

Предмет дослідження – мобільний застосунок путівник для підтримки процесу легшого подорожування Японією.

Короткий зміст роботи: В роботі проаналізовано алгоритми та методи розробки мобільних додатків для підтримки подорожей. Проаналізовано існуючі рішення на ринку мобільних додатків, зокрема: Вся Україна, Redigo, Путівник - World Explorer, Sygic Travel Maps Trip Planner, Токіо путівник. Було визначено їх переваги та недоліки, створено порівняльну таблицю для цих застосунків. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: пошук інформації про місця, відображення місць на карті, додавання до вподобаних та рейтинги місць. Проведено функціональне та модульне тестування додатку. В роботі використано мову програмування Dart, фреймворк Flutter для розробки кросплатформенних застосунків, Firebase для зберігання та обробки даних, а також Google Maps API для відображення карт.

Сферою використання застосунку є туризм та подорожування, спрощення процесу планування подорожей Японією, а також знаходження інформації про цікаві місця для мандрівників.

КЛЮЧОВІ СЛОВА: ПУТІВНИК, МОБІЛЬНИЙ ЗАСТОСУНОК, ЯПОНІЯ, ПЛАНУВАННЯ ПОДОРОЖІ, FLUTTER, FIREBASE.

ЗМІСТ

ВСТУП 8
1 ТЕОРЕТИЧНА ЧАСТИНА 10
1.1 Аналіз предметної області 10
1.2 Аналіз програмного забезпечення для легшого подорожування 11
1.3 Таблиця порівняння 18
1.4 Огляд засобів реалізації додатку для легшого подорожування Японією 20
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 23
2.1 Технічне завдання 23
2.2 Засоби розробки 23
2.3 Функціональні вимоги 26
2.4 Нефункціональні вимоги 27
2.5 Діаграма класів 28
2.6 Діаграма варіантів використання 30
2.7 User Flow діаграма 31
3 ОПИС РОЗРОБКИ ЗАСТОСУНКУ 33
3.1 Опис сторінок застосунку 33
3.2 Створення та управління базою даних 35
3.3 Створення компонентів 40
3.4 Створення карточки місця та сторінки місця 48
3.5 Головна сторінка 52
3.6 Створення мапи 55
3.7 Тестування програмного забезпечення 57
ВИСНОВКИ 60
ПЕРЕЛІК ПОСИЛАНЬ 61
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 63
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ 71

ВСТУП

Обгрунтування вибору теми та її актуальність: У сучасному світі люди почали дуже часто подорожувати та цікавитися іншим країнами. Також кожного дня використовують свої мобільні пристрої. На використання мобільного пристрою людина витрачає майже п’ять годин свого часу. Та оскільки розвивається ринок мобільних застосунків й розробка застосунків для подорожування, можна зробити висновок, що людям цікаво та зручно мати декілька застосунків для легшого подорожування країнами. Оскільки, після пандемії відкрилися кордони Японії для туристів, можемо помітити, що велика кількість туристів хоче відвідати цю країну. Саме тому для допомоги мандрівникам буде актуальним застосунок путівник для легшого подорожування Японією, що допоможе їм спланувати свою мандрівку та знайти інформацію про цікаві місця Японії.

Об’єкт дослідження – процес подорожування Японією, знаходження інформації про місця та планування подорожі.

Предмет дослідження – мобільний застосунок путівник для підтримки процесу легшого подорожування Японією.

Мета роботи – спрощення процесу подорожування Японією, знаходження інформації про місця та планування подорожі.

Методи дослідження – методологія передачі, зберігання та обробки даних, розробки та тестування програмного забезпечення.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати існуючі аналогічні програмні забезпечення, виявити їх переваги та недоліки. Та як результат створити порівняльну таблицю.
2. Розробити функціональні та нефункціональні вимоги до застосунку, опираючись на висновки раніше проаналізованих аналогів.
3. Дослідити технології, інструменти та програмні засоби для вирішення поставленої задачі.

4. Розробити застосунок для спрощення процесу подорожування Японією, знаходження інформації про місця та планування подорожі з урахуванням прийнятих раніше технологій та поставлених заздалегідь вимог.

5. Провести тестування застосунку.

6. Пройти апробацію на науково-технічних конференціях.

Практичне значення результатів: Даний застосунок використовується для пошуку інформації про місця та спрощує процес планування подорожей користувача.

Для розробки застосунку використовувалися, як мова програмування Dart, фреймворк Flutter, сервіс Firebase та середовище розробки Visual Studio Code.

Практична значущість результатів: Розроблений застосунок спрощує процес подорожування користувача Японією надаючи йому можливість дізнатися інформацію про місця та знайти їх на карті.

Галузь використання – туризм, подорожування

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Аналіз предметної області

Подорожування стало невід'ємною частиною людського життя. Сфера туризму сильно розвивається разом з технологіями для створення застосунків у цьому напрямку. Саме тому потрібно проаналізувати дану сферу, використання застосунків та розібратися в їх перевагах та недоліках.

За даними взятими у німецької компанії Statista, що надає доступ до статистики та досліджень, можемо зробити такий висновок, що за 2020 та 2021 рік серед прихильників подорожувати найбільше припадало на користувачів мобільних пристроїв. А якщо точніше, то на 2021 рік припадало 60% усього трафіку на мобільні пристрої і лише 38% на користування комп'ютером.

Більшість користувачів надає перевагу мобільним застосункам ніж мобільним веб-сайтам, також половина мандрівників має встановленим принаймні один туристичний застосунок.

Оскільки, сфера туризму дуже розвинена, то можемо побачити велику різноманітність туристичних застосунків. Можна виділити такі види застосунків:

1. Застосунки для купівлі авіаквитків - ці застосунки допомагають мандрівникам шукати та купувати авіаквитки, надають інформацію про рейси в режимі реального часу.

2. Застосунки для бронювання житла - надають змогу знайти та забронювати готелі, хостели, апартаменти та інші види житла. Також надають інформацію про житло, відгуки та рейтинг від інших мандрівників.

3. Транспортні застосунки - створені для надавання інформації про місцевий транспорт, а також можуть мати можливість для оплати транспортних послуг.

4. Навігаційні застосунки - надають карти та інформацію як дістатися до відповідних місць, кафе, пам'ятки, магазини та інші місця.

5. Застосунки путівники - пропонують інформацію про місцеві визначні пам’ятки, містять оцінки та відгуки від інших користувачів.

Також є застосунки, що комбінують декілька видів туристичних застосунків, наприклад можуть одночасно допомагати знайти та забронювати авіаквитки і житло.

Отже, цільовою аудиторією для застосунку путівника будуть мандрівники різного віку та різного досвіду у подорожуваннях, які шукають зручний спосіб для легшого та комфортного подорожування Японією. Серед основних вимог користувачів є інтуїтивний простий інтерфейс, достовірна та актуальна інформація.

Для вирішення цієї задачі буде використовуватися технологія кросплатформенної розробки для створення застосунку відразу на двох операційних системах IOS та Andorid. Кросплатформенна розробка - це властивість/можливість програмного забезпечення працювати на декількох апаратних чи програмних платформах. Серед популярних технологій, що використовуються для створення кросплатформлених застосунків є два фреймворки з відкритими вихідними кодами від компанії Google - Flutter та від Microsoft - Xamarin. Використання цих технологій допоможе зробити простий та інтуїтивний інтерфейс користувача.

1.2 Аналіз програмного забезпечення для легшого подорожування

Для задоволення потреб мандрівників вже були створені застосунки путівники для легшого подорожування. Давайте розглянемо існуючі застосунки, що надають інформацію про пам’ятки, визначимо їхні переваги та недоліки.

Український застосунок “Вся Україна”, що розроблявся для допомоги мандрівникам під час чемпіонату Європи по футболу у 2012 році. За допомогою цього застосунку мандрівник міг дізнатися інформацію про різні місця в Україні та знайти їх на карті. У застосунку можна користуватися такими функціями:

– Курс валют

Користувач у застосунку обирає необхідну йому валюту та вводить її певну суму та в результаті отримує відповідь з переведеною у іншу валюту суму.

– Прогноз погоди

Можливість переглянути прогноз погоди для міст.

– зробити закладки місць

У застосунку після відкриття інформації про певне місце, користувач може додати його до закладки та потім щоб знову не шукати інформацію про певне місце, може натиснути на головному екрані вкладку закладки та переглянути інформацію про вже збережені ним місця.

На даний момент застосунок не розвивається та не має нових оновлень, а також був видалений з Google Play Market, тому стару версію можемо завантажити лише з інтернету.

Переваги застосунку:

– має багато категорій для пошуку потрібного місця;

– достатньо великий вибір міст для ознайомлення;

– має додатковий функціонал (курс валют, прогноз погоди);

– безкоштовний застосунок.

Недоліки застосунку:

– російськомовна локалізація;

– немає детального опису;

– застосунок можна завантажити тільки на Android;

– підтримка застосунку закінчилася.

Рис. 1.1 Логотип застосунку “Вся Україна”

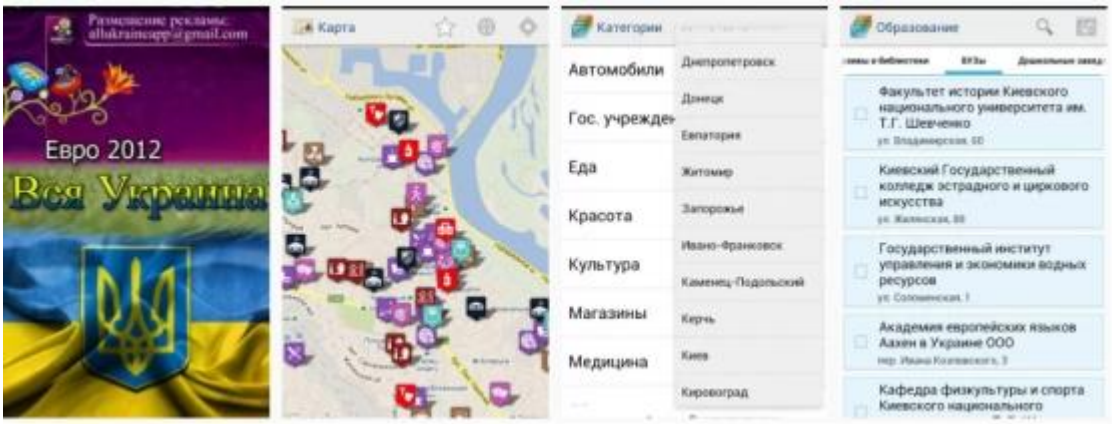


Рис. 1.2 Приклади екранних форм застосунку “Вся Україна”

Застосунок “Redigo”, це застосунок путівник, що містить інформацію про велику кількість місць по всьому світі. Застосунок пропонує обрати країну зі списку та певне місто і переглядати місця, що є в застосунку. У застосунку є можливість завантажити інформацію про всю країну чи певне місто в ній на пристрій. Також в застосунку є такий функціонал, як:

– можливість додати місця в обрані, користувач обирає певне вподобане місце та потім в іншій вкладці знайти його;

– розмовник для декількох іноземних мов, користувач обирає на якій мові хоче отримати переклад слів, фраз та також може обрати категорію для якої події хоче використати фразу, наприклад при відвідуванні ресторану.

– може показати пам’ятки, що знаходяться поруч, функція працює якщо мандрівник має доступ до інтернету та увімкнену геолокацію пристрою.

Переваги застосунку:

– достатньо великий вибір країн та міст для ознайомлення;

– можливий пошук по назві країни, міста, пам’ятки;

– має додатковий функціонал (розмовник);

– має можливість завантажити путівник по країні пристрій;

– безкоштовний застосунок.

Недоліки застосунку:

– російськомовна локалізація;

– застарілий та місцями незручний інтерфейс;

– застосунок можна завантажити тільки на Android.

Рис. 1.3 Логотип застосунку “Redigo”

Рис. 1.4 Приклади екранних форм застосунку “Redigo”

Застосунок “Путівник - World Explorer”, що позиціонує себе застосунком путівником, який надає список пам’яток, що знаходяться поруч. Насправді застосунок відображає список місць, що знаходяться поруч в не залежності чи це пам’ятка чи ще щось, вся інформація у цьому застосунку про ці місця виключно з вікіпедії. Розібратися в застосунку важко, розібратися в функціоналі застосунку також. Застосунок постійно пропонує придбати преміум версію.

Протестувавши даний застосунок можна виділити лише одну перевагу, а саме те, що справді відображає велику кількість місць ,що знаходяться поруч.

Недоліки застосунку:

– незрозумілий функціонал;

– велика кількість не зрозумілих місць через баги застосунку;

– застосунок можна завантажити тільки на Android;

– реклама та постійна пропозиція купити преміум версію.

Рис. 1.5 Логотип застосунку “Путівник - World Explorer”

Рис. 1.6 Приклади екранних форм застосунку “Путівник - World Explorer”

Наступний застосунок можна завантажити на пристрій IOS та Android, сам застосунок безкоштовний, але має платну підписку. “Sygic Travel Maps Trip Planner” надає мандрівнику великий функціонал для якісного подорожування.

Можна виділити такі функції, що пропонує цей застосунок:

– запланувати поїздку

Користувач обирає місто куди хочу відправитися, робить назву для цієї подорожі та дати й таким чином запланувати її у застосунку

– знайти пам’ятки та додати їх в обране

На карті користувач може обрати місце та додати його до обраного, але щоб отримати інформацію про це місце потрібно мати преміум версію застосунку.

– знайти екскурсії

Застосунок відображає список екскурсій, ціну та тривалість екскурсії, а при натисканні певної екскурсіє перенаправляє на сайт з більш детальною інформацією про екскурсію

– дізнатися прогноз погоди

– переглянути карту метро

Більшість функціоналу застосунку можна використовувати тільки через преміум. Сам застосунок зручний у користуванні.

Переваги застосунку:

– зручний інтерфейс;

– достатньо великий вибір країн та міст для ознайомлення;

– можливий пошук по назві країни, міста, пам’ятки;

– має додатковий функціонал;

– має можливість запланувати подорож;

– застосунок можна завантажити на всі ОС.

Недоліки застосунку:

– більшість необхідного функціоналу доступна через підписку;

Рис. 1.7 Логотип застосунку “Sygic Travel Maps Trip Planner”

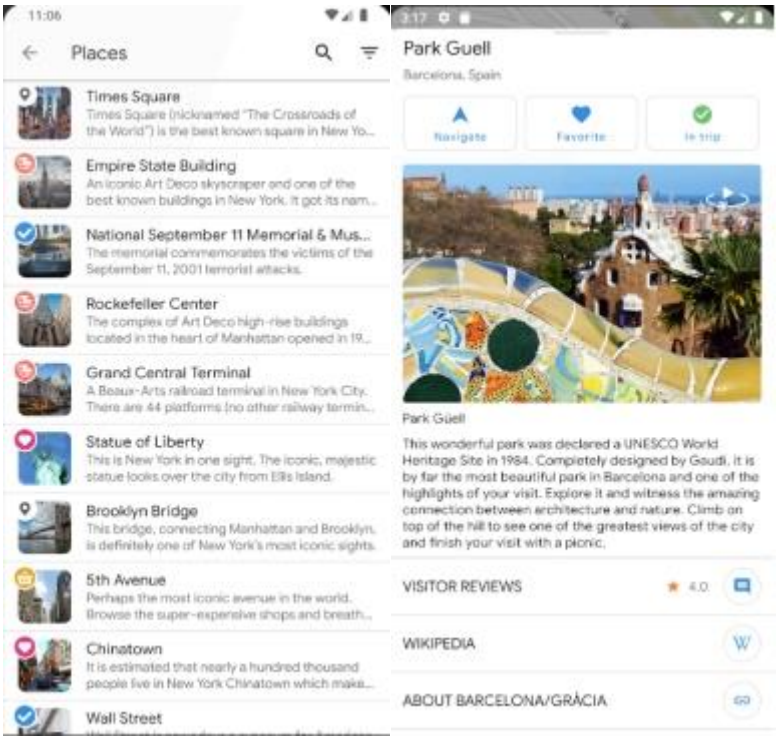


Рис. 1.8 Приклади екранних форм застосунку “Sygic Travel Maps Trip Planner”

Застосунок “Токіо путівник” допомагає мандрівникам дізнатися про місця визначні та популярні місця в Токіо. Застосунок має велику кількість цікавих місць для ознайомлення. Пам’ятки можна додати в обрані, а також побачити на карті. Також в застосунку можна прочитати багато інформації про саме місто, а саме: його історію, географію, транспорт та іншу цікаву інформацію. Є цікавий AR функціонал, що при наведенні через камеру на певну пам’ятку запропонує прочитати про неї інформацію.

Переваги застосунку:

– зручний та простий інтерфейс;

– достатньо великий вибір пам’яток для ознайомлення;

– великий вибір додаткової цікавої інформації;

– незвичний AR функціонал;

Недоліки застосунку:

– проблеми з перекладом;

Рис. 1.9 Логотип застосунку “Токіо путівник”

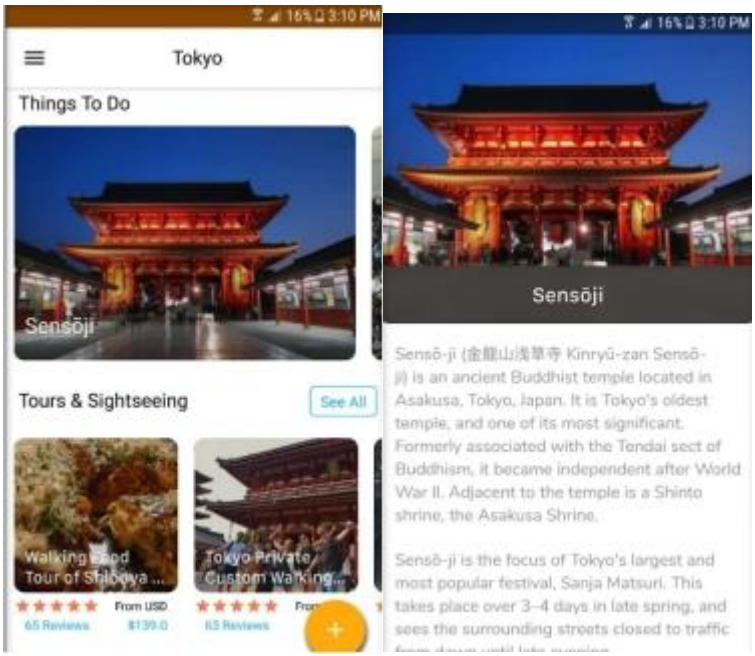


Рис. 1.10 Приклади екранних форм застосунку “Токіо путівник”

1.3 Таблиця порівняння

Після проведеного аналізу застосунків, що мають схожий функціонал, можна зазначити, що вони мають як і спільні переваги, так і недоліки. Саме після вдосконалення зазначених мінусів, ми отримаємо застосунок для легшого подорожування Японією. Заповнимо таблицю (див. табл. 1.1) для порівняння застосунків аналогів з нашим застосунком, де позначка «+» означає, що є наявний функціонал, а позначка «-» відсутній функціонал.

Таблиця 1.1

Порівняння застосунків аналогів з нашим застосунком

Застосунок	Вся Україна	Redigo	Путівник - World Explorer	Sygic Travel Maps Trip Planner	Токіо путівник	ArkTravel
Кросплатформенність	-	-	-	+	+	+
Інтерфейс враховує особливості Японії	-	-	-	-	-	+
Орієнтованість путівника	Вся країна	Весь світ	Весь світ	Весь світ	Місто	Вся країна
Карти місцевості	+	+	+	+	+	+
Рейтинг місць від користувачів	-	-	-	-	-	+
Є додаткова інформація для користувача	-	-	-	-	+	+
Має унікальний додатковий функціонал пов’язаний з країною	-	-	-	-	-	+

1.4 Огляд засобів реалізації додатку для легшого подорожування Японією

Застосунок для легшого подорожування Японією розробляється для використання на мобільних пристроях. Вибір мови програмування та фреймворків залежить від типу застосунка. Застосунки для мобільних пристроїв поділяють на такі типи:

– нативний застосунок - це застосунок, що розробляється під конкретну операційну систему, а саме Android чи IOS. Цей застосунок найбільш оптимізований під конкретну операційну систему, має доступ до всіх можливостей ОС, працюють швидко і стабільно;

– кросплатформенний застосунок - розробляється застосунок відразу для декількох ОС. Даний вид застосунків економить час на розробці застосунку та його подальшому обслуговуванні. Також є дешевшим варіантом у порівнянні нативними застосунками, тому що відразу розробляється під всі необхідні ОС.

– веб-застосунок - це розроблений варіант мобільної версії сайту, що має обмежений функціонал у порівнянні з версією для комп’ютера. Ці застосунки відкриваються у звичайному браузері мобільного пристрою.

В наш час стає популярним кросплатформенна розробка програмного забезпечення, як дізналися раніше, шо це дає можливість створення застосунку відразу для декількох операційних систем. Отже для розробки застосунка для легшого подорожування Японією для операційних систем IOS та Android буде використовуватися кросплатформенний тип застосунків. Саме тому потрібно проаналізувати та обрати необхідну мову програмування та фреймворки. Серед популярних технологій, що використовуються для створення кросплатформлених застосунків є:

– Ionic є одним з популярних фреймворків для розробки кросплатформенних застосунків. Заснований він на AngularJS та надає можливість розробникам отримати доступ до нативних контролерів платформи, завдяки поєднання комбінації мов програмування, як HTML5, JavaScript, CSS та обгортки Cordova.

– React Native – це фреймворк, що побудований на базі JavaScript. Фреймворк об’єднує в собі переваги від JavaScript та React.JS та має можливість створювати модулі за допомогою мов Swift, Objective-C або Java, що придає нативний вигляд застосунку.

– Flutter - це набір для розробки програмного забезпечення, який допомагає швидко створювати додатки для Android та iOS. Крім того, це ключовий метод для розробки додатків Google Fuschia.

– Xamarin фреймворк, який можна використовувати для розробки програм для платформ Android, Windows і iOS, використовуючи C#, .Net, а не JS та HTML-бібліотеки, що надає можливість розробникам створювати програми на дев’яносто відсотків коду для трьох платформ.

Порівняємо два фреймворки з відкритими вихідними кодами від компанії Google фреймворк Flutter й від Microsoft фреймворк Xamarin й виберемо один з них для розробки застосунку. Якщо порівнювати ці фреймворки за популярності серед розробників програмного забезпечення, то фреймворк Flutter виграє свого конкурента. Незважаючи на те, що Xamarin був розроблений в 2011, а Flutter в 2016 - це не завадило їм обігнати досвідченого конкурента.

В плані створення інтерфейсу користувача переможцем буде Flutter через його багатий набір віджетів та більшої гнучкості, розробник може мати контроль над кожним пікселем на екрані. Проте якщо цілю є створення більш рідного нативного дизайну застосунку, то краще використовувати Xamarin. Також перевогою у Flutter буде функція 'Hot Reload', вона дозволяє бачити зміни інтерфейсу в реальному часі.

Продуктивність залежить також від завдання застосунку, якщо має бути близьким до нативного, то тут перемагає Xamarin. В інших випадках Flutter буде швидше реалізовувати сценарії з складними операціями, візуальними ефектами та важкими обчисленнями.

Flutter та Xamarin мають чудові результати в плані безпеки, вони вважаються високобезпечними технологіями, проте слід ознайомитися з їх методами безпеки,

оскільки вони відрізняються і можуть не підійти під ваші конкретні потреби у безпеці.

Також потрібно визначитися як будуть зберігатися дані нашого застосунку, наприклад за допомогою хмарних баз даних. Хмарна база даних це організоване, структуроване зібрання даних, що знаходяться на хмарних платформах, за допомогою яких можна легко отримати доступ та керувати даними.

Отже, розглянемо платформи, що надають можливості використовувати хмарні бази даних:

– AWS (Amazon Web Services) – є наймасштабовішою та найрозширенішою хмарною платформою, що надає великий спектр послуг, у тому числі хмарні БД. Платформу можна використовувати для широкого спектру завдань, від простих веб-сайтів до складних корпоративних систем. Хмарні БД, які доступні для використання: реляційні - Amazon RDS, NoSQL - Amazon DynamoDB та сховища об’єктів - Amazon S3. Загалом платформа надійна та безпечна, але для початківців може бути складною для використання.

– Azure від Microsoft – це друга за величиною хмарна платформа, що надає широкий спектр послуг. Можна використовувати, як для простих веб-сайтів так і для складних корпоративних систем. Хмарні БД, що включає в собі платформа: реляційні - Azure SQL Database, NoSQL - Azure Cosmos DB і сховища об’єктів - Azure Blob Storage. Azure тісно пов’язаний з іншими продуктами Microsoft. Також для початківців може бути складно використовувати дану платформу.

– Firebase від Google – платформа, що надає хмарну БД і багато інших послуг для розробки мобільних і веб-застосунків. Firebase простий у використанні та налаштуванні, ідеально підходить для початківців. У нього є функції синхронізації даних у режимі реального часу та авторизації користувачів , що оптимізовано для мобільних і веб-застосунків. Платформа пропонує безкоштовний план, тому її можна використовувати для невеликих проектів і тому може не бути настільки масштабованим, як AWS або Azure.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Технічне завдання

Розробка кросплатформенного застосунку путівника для легшого подорожування Японією для мобільних пристроїв. Завдяки використання таких технологій, як Flutter для створення UI елементів, Dart. Для бекенду використовування платформи Firebase,а саме: для роботи з базою даних використання Firestore Database, а для створення аккаунтів для мандрівників використання Firebase Authentication. Для інтеграції карти та відображення місць на ній використання Google Maps API. Використання плагіну Google Maps Flutter для роботи з картами в Flutter та плагіну Flutter Rating Bar для створення та реалізації рейтингової системи місць.

Для розробки використовувати Visual Studio Code з встановленим Flutter SDK та для створення віртуального мобільного емулятора використовувати Android Studio.

2.2 Засоби розробки

Мова програмування Dart була створена 10 жовтня 2011 року, а її перша версія під назвою Dart 1.0 була випущена 14 листопада 2013 року. Розробкою займається компанія Google. Сама мова Dart є динамічною об’єктно-орієнтованою мовою програмування та синтаксично схожа на JavaScript, Java та C.

Flutter це безкоштовний інструмент з відкритим вихідним кодом для створення інтерфейсу користувача від Google. Flutter дозволяє створювати красиві, нативно скомпільовані застосунки для мобільних, веб та настільних пристроїв.

Flutter — це розширююча багатошарова система. Кожна з його окремих бібліотек залежить від базового рівня, але жоден рівень не має привілейованого

доступу до нижнього рівня, і кожна частина рівня структури розроблена як необов'язкова та замінна (див. Рис. 2.1).

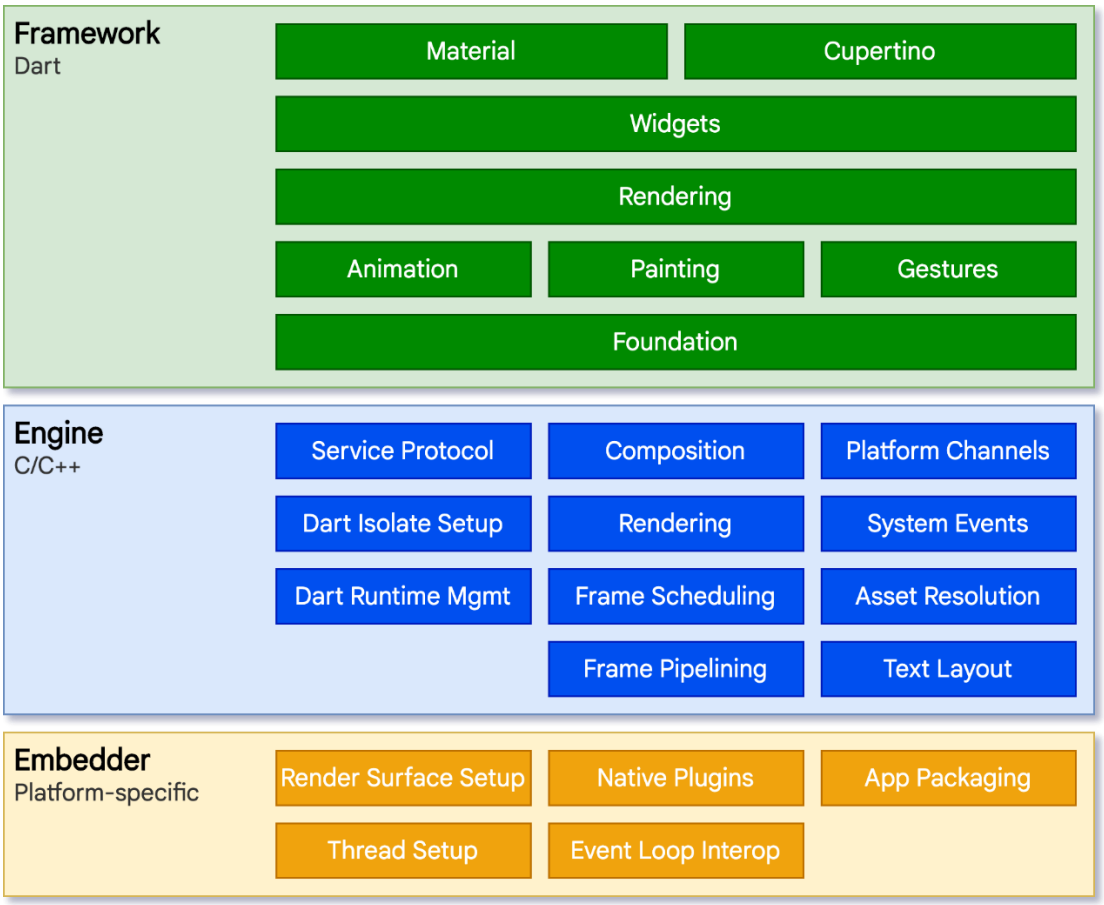


Рис. 2.1 Діаграма архітектурних шарів Flutter

Базові класи й сервіси, такі як анімація, малювання та жести, пропонують загальновживані абстракції над базовою основою.

Шар рендерингу надає абстракцію, щоб працювати з макетом. Цей шар дозволяє побудувати дерево візуальних об'єктів. Ми можемо маніпулювати цими об'єктами в будь-який час, а дерево автоматично змінює макет відповідно до наших змін.

Шар віджетів є прикладом абстракції композиції. Кожен об'єкт візуалізації в шарі рендерингу належить до відповідного класу в шарі віджетів. Крім того, шар віджетів дозволяє ідентифікувати комбінації класів, які можна використовувати

для подальшого використання. На цьому етапі використовується модель реактивного програмування.

Для впровадження мов проектування Material або iOS використовуються бібліотеки Material та Cupertino, що пропонують розширені набори елементів керування, які використовують примітиви композиції шару віджетів.

Flutter Rating Bar – це потужний компонент для визначення рівня задоволеності користувача за допомогою рейтингу зірок. Особливістю цього компоненту є те, що його можна налаштувати за власним бажанням. Можемо встановити оцінку і в п’ять зірок так і в шість чи більше, також пропонує налаштувати колір та розмір зірок, інтервал між ними або взагалі обрати як ми хочемо відобразити, вертикально чи горизонтально. Якщо оцінка три з половиною, то компонент тоді відобразить три зірки з половиною, а також якщо захочемо, то можемо замінити зірки на інше відображення, наприклад сердець чи смайликів з різними емоціями.

Google Maps Flutter – це плагін, що надає можливість додавати карти на основі даних карт від Google у наш застосунок. Використовується за допомогою віджета GoogleMap, в якому можемо настроїти позицію камери для місця з якого хочемо почати , масштабування з яким дивитися, створити мітку на карті та опис для неї. Також за допомогою плагіну можемо обрати тип для карти: звичайний, супутниковий, рельєфний та гібридний.

Firebase це платформа, що надає багато можливостей для створення кросплатформених застосунків, а саме використовується для розроблення мобільних та веб застосунків. Розвивається та підтримується платформа за допомогою компанії Google, що придбала її у 2014 році. Firebase рахується, як один з кращих варіантів для розробки кросплатформенних застосунків, так як дуже на відмінно взаємодіє з фреймворком Flutter, що також був створений від компанії Google.

Платформа надає такий можливий функціонал розробникам, як надання інструментів та сервісів для створення та подальшої підтримки бекенду застосунків, тобто можемо назвати її платформою для Backend-as-a-Service (BaaS).

Firebase Realtime – це база даних у реальному часі, за допомогою NoSQL вона дає змогу зберігати та синхронізувати дані між користувачами в реальному часі. Також надає такі можливості, як:

– Користування можливе з будь-якого пристрою

– Створення безсерверних програм

– Оптимізований для використання в автономному режимі

– Сильна безпека даних

Firebase Storage – це один з сервісів, що за підтримки від Google Cloud Storage дає можливість зберігати та обслуговувати через хмару матеріал від користувачів, а саме від відео й фото до музики.

Firestore Database – це хмарна NoSQL для зберігання та синхронізації даних. Підтримує синхронізацію даних між клієнтськими застосунками через прослуховувачі в реальному часі та пропонує підтримку в автономному режимі, для того щоб мали можливість створювати адаптивні застосунки, що працюють незалежно від затримки мережі чи під'єднання до інтернету.

Firebase Authentication – це безкоштована функція авторизації користувачів, що надається від Firebase як її серверні служби. Ця функція легко інтегрується у застосунок та надає змогу інтегрувати авторизацію через логін та пароль або авторизуватися через використання платформ: Google, Twitter (X), Facebook та GitHub.

Visual Studio Code – є безкоштовним та легким, але в той час і потужним редактором вихідного коду, що працює на всіх операційних системах. Надає великий вибір для завантаження розширень та вибору мов для програмування.

2.3 Функціональні вимоги

Після порівняння функціоналу застосунків аналогів для подорожування, визначили основні функції, що мають бути реалізовані у застосунку путівнику для легшого подорожування Японією. Отже наведені такі функціональні вимоги для застосунку:

Авторизація. Для мандрівнику необхідно мати аккаунт для безпеки даних та можливістю додавати місця у вподобані та для оцінки місць.

Категорії. Мандрівнику необхідно мати можливість знаходити місця за обраною категорією. У нього повинен відображатися список категорій з можливістю перейти на сторінку обраної категорії для перегляду місць.

Вподобане. Користувач повинен мати можливість обрати вподобану йому пам’ятку, щоб переглянути її на сторінці вподобаних місць. Також мати можливість видалити пам’ятку з вподобаних.

Популярні місця. Мандрівник повинен мати можливість переглядати популярні місця та мати можливість оцінити місце. Відображення популярних місць залежить від оцінки користувачів, якщо місце має оцінку вище чотирьох зірок, то воно вважається популярним та відображається користувачу.

Інтерактивна карта. Користувач може переглядати на карті місця та при виборі на карті місця натиснути на нього та отримати детальну інформацію про вибране місце. Також користувач може на сторінці місця натиснути кнопку та побачити де знаходиться місце на карті.

2.4 Нефункціональні вимоги

Портативність. Застосунок повинен працювати без будь-яких змін у продуктивності як для мобільних пристроях під ОС Android, так і для пристроях з IOS.

Сумісність. Застосунок повинен підтримувати пристрої, що працюють на Android з версії 9.0 та для пристроях з IOS з версії 15.0.

Безпека. Необхідно використовувати безпечну авторизацію мандрівника, що забезпечує надійність та запобігає доступу до персональних даних мандрівника, наприклад через Firebase Authentication.

Локалізація. Застосунок повинен мати автентичний для місцевості інтерфейс та інші культурні аспекти.

Масштабованість. Для підвищення продуктивності застосунка необхідно використовувати хмарні сховища даних, як Firetore Database

2.5 Діаграма класів

На даній діаграмі (див. Рис. 2.2) продемонстровано класи, що в результаті допомагають створити та відобразити місця. Короткий опис яку функцію виконують класи:

- User містить інформації про користувача.
- Category містить назву категорії та її іконку.
- Sights містить інформацію про місце.
- CityTab відображає список пам’яток у вигляді плитки та вибір категорії у вигляді горизонтального списку у вкладці міста.
- CityTile відображає окрему пам’ятку у вигляді плитки та відображає кнопку вподобання.
- SightPage відображає детальну інформацію про певну пам’ятку.
- RaitingBar відображає шкалу у вигляді зірок та має функцію оцінки місця від користувача.
- LikeButton кнопка, що виконує функцію додавання чи видалення з вподобання.
- FirebaseService клас, що зберігає та отримує інформацію з Firebase та робить запити для відображення місць за айді місця, категорією та містом, також має логіку отримання оціненого місця від користувачів.

Також можемо побачити на діаграмі, що деякі класи наслідують інтерфейси StatelessWidget та StatefulWidget, що є частино Flutter.

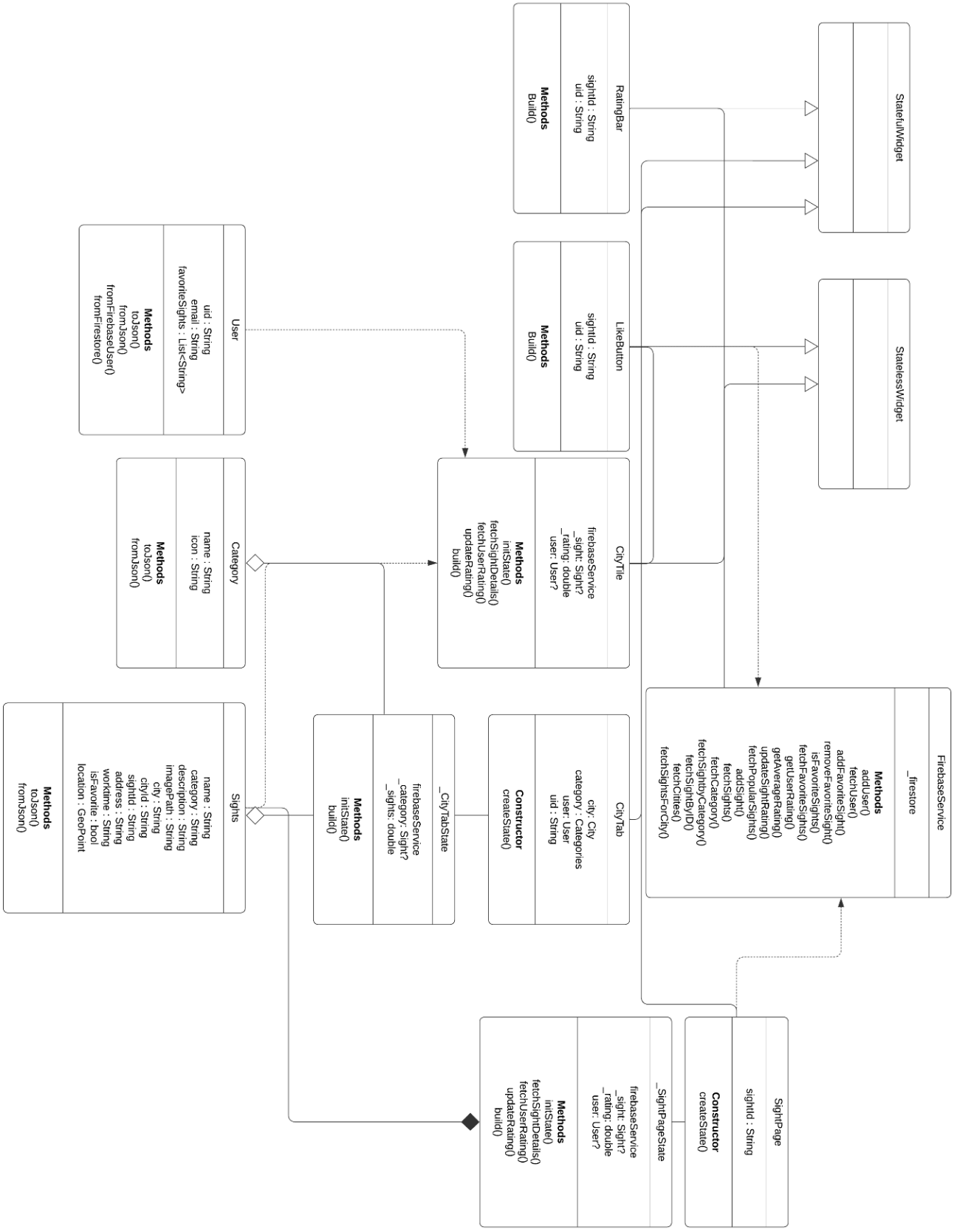


Рис. 2.2 Діаграма класів

2.6 Діаграма варіантів використання

Use case – це діаграма, що відображає певний перелік дій актора, за яким він взаємодіє зі застосунком для виконання певної дії й досягнення конкретної цілі.

Дана діаграма (див. Рис. 2.3) відображає основні варіанти використання застосунку для легшого подорожування Японією. На цій діаграмі варіантів використання у ролі актора є мандрівник, що планує ознайомитися з місцями та система Firebase, що відповідає за авторизацію мандрівника. Мандрівник має такий доступ до функцій, як: реєстрація та вхід до застосунку, вибір необхідного міста, обрати певну категорію для місця, обрати певне місце та переглянути інформацію про нього, також має можливість додати місце до вподобаних чи оцінити його, а ще переглянути місця на карті.

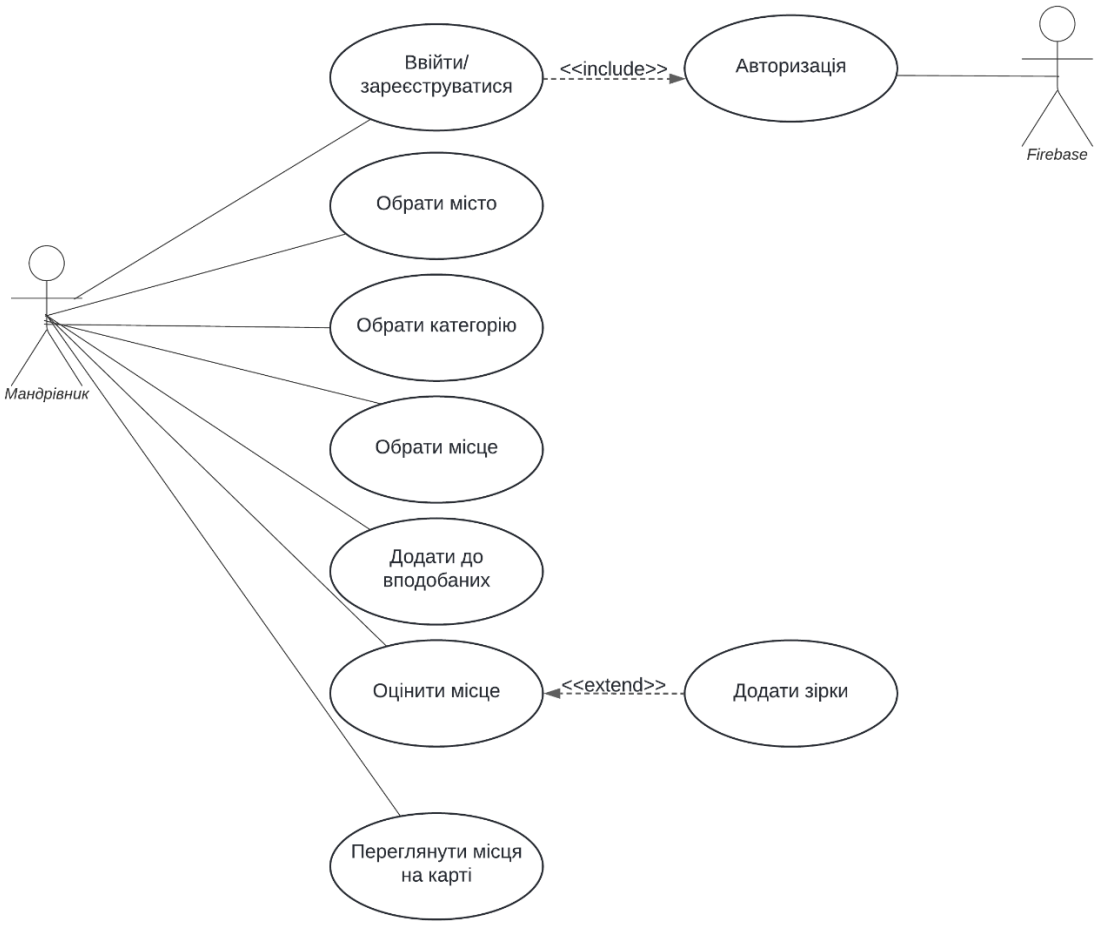


Рис. 2.3 Діаграма варіантів використання

2.7 User Flow діаграма

User Flow діаграма або ж мапа сценарію користувача – це відображення кроків користувача в рамках якогось одного певного сценарію, для вирішення певної задачі. Одна мапа сценарію користувача розв’язує проблему лише однієї певної задачі, для вирішення іншої необхідно створювати іншу мапу сценарію.

Дана мапа сценарію (див. рисунок 2.4) зображує кроки мандрівника для відкриття сторінки з інформацією про певне місце. Короткий опис кроків відображених в діаграмі:

Початкова сторінка. Мандрівник з початкової вітальної сторінки переходить до наступної сторінки з входом чи пропонуванням реєстрації.

Наступний крок. Якщо користувач зареєстрований він здійснює вхід до застосунку та потрапляє на головну сторінку застосунка, але якщо ні то проходить реєстрацію та після потрапляє на головну сторінку.

Головний екран. Якщо користувач обирає категорію на головній сторінці, то він потрапляє до сторінки зі списком місць за обраною категорією. Але якщо користувач не обрав категорію, то він переглядає популярні місця на головній сторінці

Наступний крок. Якщо користувач не обрав необхідне місце він продовжує пошук зі списку популярних чи обраних за категорією місць. Та якщо користувач обрав місце він може додати місце до вподобаних та якщо бажає відкриває сторінку з місцем. Також користувач може не додавати місце до вподобаних, а просто відкрити сторінку з інформацією про дану місце.

Кінець. Після відкриття сторінки місця сценарій користувача закінчується, оскільки задача відкрити сторінку місця була виконана.

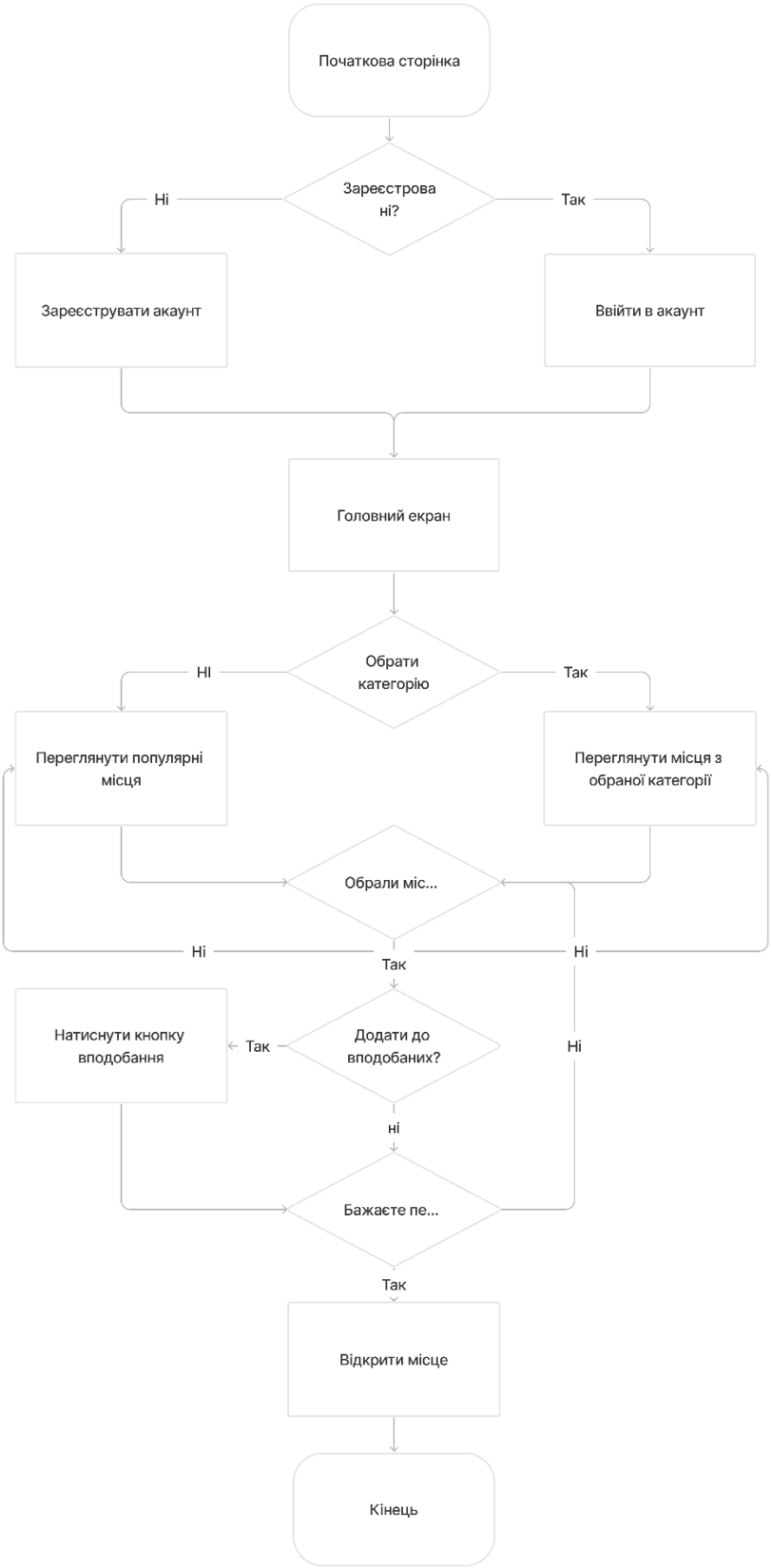


Рис. 2.4 User Flow діаграма

3 ОПИС РОЗРОБКИ ЗАСТОСУНКУ

3.1 Опис сторінок застосунку

Застосунок має сторінки для авторизації та сторінок для взаємодії з місцями, розглянемо їх:

– Початкова сторінка

– Сторінка для входу

– Сторінка для реєстрації

– Головна

– Вподобані

– Категорії

– Сторінка місця

– Карта

– Додаткова інформація

– Налаштування

Початкова сторінка – є вітальним екраном для користувачів, вона служить початковим місцем, з якого користувачі зможуть перейти до наступних етапів для запуску застосунку.

Сторінка для входу – це екран де мандрівник вводить свої дані для входу в особистий акаунт, також слугує переходом на сторінку реєстрації, якщо мандрівник не має акаунту.

Сторінка для реєстрації – слугує для реєстрації нових користувачів у застосунку за допомогою пошти та паролю. На цій сторінці окрім поля для введення пошти та паролю, ще є поле для того , щоб користувач підтвердив свій пароль.

Головна – це сторінка на якій знаходиться TabBar, тобто це елемент інтерфейсу, що допомагає з навігацією у застосунку. На цьому знаходиться список з міст, де користувач обирає певне місто в якому хотів би побачити місця. Також

на головній відображається список з категоріями, наприклад такі, як : парки, гори, храми та інші. Обравши категорію користувач переходить на сторінку певної категорії. І останній список, що відображається на головній сторінці це список з популярних місць. Популярність місць визначається через оцінки користувачів і на головній відображаються лише ті місця, що мають оцінку вище чотирьох зірок. Також користувач може додати до вподобаних популярні місця або відкрити та прочитати інформацію про конкретне місце.

Вподобані – це сторінка на якій відображаються всі вподобані місця користувача, користувач може прибрати їх з вподобаних або відкрити та дізнатися більше про дані місця.

Категорії – це сторінка на якій відображається список місць, що належать до певної категорії, на цій сторінці мандрівник має можливість додати місця, що сподобалися до вподобаних, щоб не шукати потім або відразу відкриту сторінку з інформацією про місце.

Сторінка місця – слугує для представлення мандрівнику інформації про місце, на даній сторінці відображається фотографія місця, опис, мандрівник може оцінити дане місце за допомогою шкали оцінювання зірками, а також має можливість натиснути на кнопку, що відобразить місцезнаходження пам’ятки на карті.

Карта – це сторінка на якій відображені позначки з місця, мандрівник може натиснути на позначку та дізнатися, що за місце знаходиться на карті та натиснувши ще раз на опис позначки, то відкриється сторінка з детальною інформацією про дане місце.

Додаткова інформація – це сторінка на якій знаходиться додаткова інформація незалежна від місць, наприклад користувач може дізнатися більше інформації про країну.

Налаштування – це місце де користувач може змінити фон та вийти з свого акаунту.

3.2 Створення та управління базою даних

Для створення та підключення бази даних нам необхідно виконати декілька кроків:

1. Зареєструватися на сервісі Firebase
2. Створити проект на Firebase
3. Через командну строку встановити Firebase CLI
4. Встановити плагін firebase_core та firebase_auth
5. Підключити базу даних
6. Створити класи, що будуть приймати та відправляти дані
7. Створити універсальний клас, що буде відповідати за управління різними даними з бази даних
8. Заповнення даних

Отже для першого кроку нам потрібно перейти на сервіс Firebase та створити новий акаунт або увійти за допомогою облікового запису Google.

У наступному кроці вже ввійшовши в акаунт потрібно створити новий проект (див. Рис. 3.1), потрібно ввести назву для свого проекту та якщо потрібно підключити Google Analytics

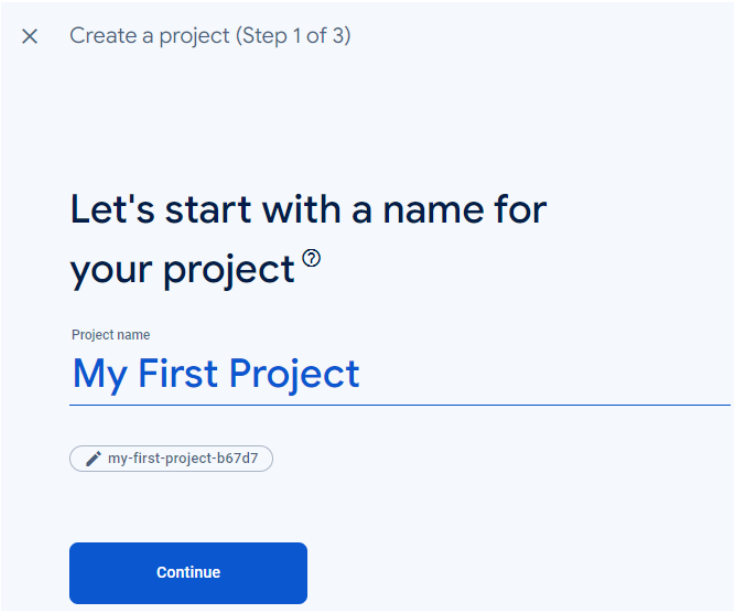


Рис. 3.1 Створення проекту на Firebase

Третім кроком нам потрібно через командну строку встановити Firebase CLI це використовується для тестування, керування та розгортання нашого створеного проекту на сервісі Firebase із командного рядка (див. Рис. 3.2).

```
PS D:\Flutter Training\Arktravel\arktravel> npm install -g firebase-tools

changed 672 packages in 25s

70 packages are looking for funding
  run `npm fund` for details
```

Рис. 3.2 Встановлення Firebase CLI

У четвертому кроці ми встановлюємо плагін firebase_core, який відповідає за підключення програми Flutter до проекту Firebase (див. Рис. 3.3) та для роботи з авторизацією користувача встановити firebase_auth.

```
PS D:\Flutter Training\Arktravel\arktravel> flutter pub add firebase_core
Resolving dependencies...
  _flutterfire_internals 1.3.16 (1.3.33 available)
  cached_network_image_web 1.1.1 (1.2.0 available)
  cloud_firestore 4.14.0 (4.17.3 available)
  cloud_firestore_platform_interface 6.1.0 (6.2.3 available)
  cloud_firestore_web 3.9.0 (3.12.3 available)
  collection 1.17.2 (1.18.0 available)
  cupertino_icons 1.0.6 (1.0.8 available)
  ffi 2.1.0 (2.1.2 available)
```

Рис. 3.3 Встановлення плагіну firebase_core

Для перевірки що все було встановлено, потрібно побачити, що в файлі pubspec.yaml з’явилися наші завантажені плагіни (див. Рис. 3.4).

```
30 dependencies:
31   flutter:
32     sdk: flutter
33
34   image: ^3.2.0
35
36   firebase_core: ^2.24.2
37   firebase_auth: ^4.16.0
38   cloud_firestore: ^4.14.0
```

Рис. 3.4 Вигляд файлу pubspec.yaml після встановлень плагінів

Наступним кроком після встановлення всього необхідно підключити базу через командну строку. Для цього нам потрібно в терміналі ввести firebase login та ввести свою пошту через яку реєструвалися на Firebase. Потім вводимо команду flutterfire_cli та вказуємо шлях для експорту. Обираємо наш проект та далі зі списку обираємо платформи, що будуть підтримувати нашу конфігурацію. Після виконання цих дій ми зможемо користуватися у нашому Firebase проекті Firestore Database та Firebase Authentication.

У шостому кроці ми створюємо окремі файли під кожен клас, наприклад: клас для користувача, місць, міст та категорії (див. Рис. 3.5).

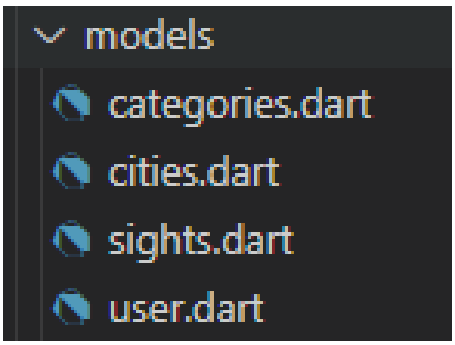
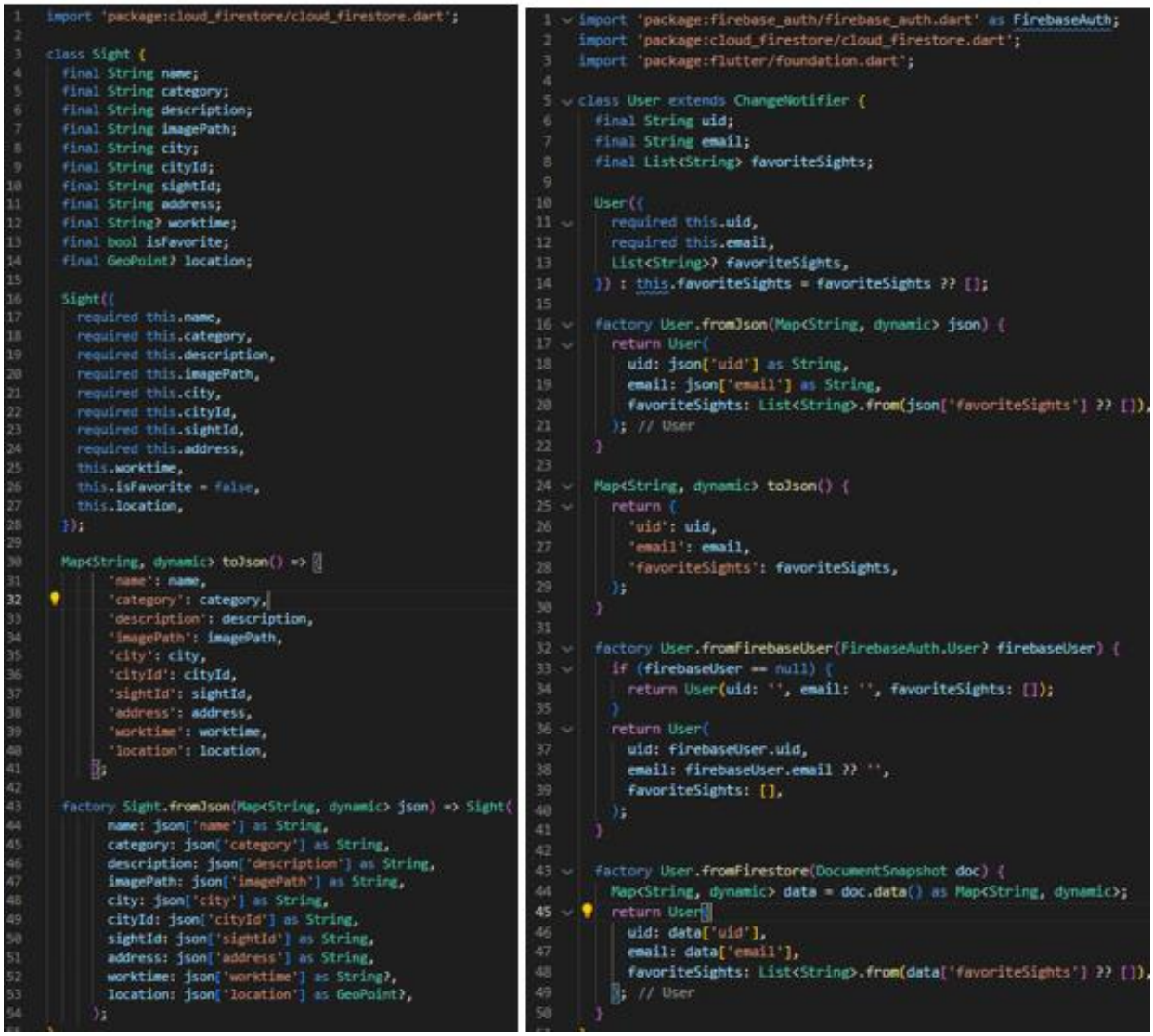


Рис. 3.5 Демонстрація файлів для класів

У файлах ми створюємо класи з атрибутами, що повинні міститися в них та методи, що перетворюють їх у формат JSON та назад. Якщо ми не вказуємо у атрибуті в кінці знак «?» після вказання типа (String?) , то заповнення цих даних має бути обов'язковим і в конструкторі ми вказуємо required, а якщо атрибут може бути null, то ставиться «?». Вказані поля з використанням required будуть обов’язково передані при створенні об’єкта, а необов’язкові будуть пропущеними або вже мати значення, як наприклад атрибут isFavorite з типом bool що має значення False. Таку структуру мають всі файли, що мають використовуватися для створення об’єктів у майбутньому. Окрім класу User, він ще включає методи, що використовуються для додавання даних про вподобані користувачем місцях (див. Рис. 3.6).



```dart
import 'package:cloud_firestore/cloud_firestore.dart';

class Sight {
  final String name;
  final String category;
  final String description;
  final String imagePath;
  final String city;
  final String cityId;
  final String sightId;
  final String address;
  final String? worktime;
  final bool isFavorite;
  final GeoPoint? location;

  Sight({
    required this.name,
    required this.category,
    required this.description,
    required this.imagePath,
    required this.city,
    required this.cityId,
    required this.sightId,
    required this.address,
    this.worktime,
    this.isFavorite = false,
    this.location,
  });

  Map<String, dynamic> toJson() => {
        'name': name,
        'category': category,
        'description': description,
        'imagePath': imagePath,
        'city': city,
        'cityId': cityId,
        'sightId': sightId,
        'address': address,
        'worktime': worktime,
        'location': location,
      };

  factory Sight.fromJson(Map<String, dynamic> json) => Sight(
        name: json['name'] as String,
        category: json['category'] as String,
        description: json['description'] as String,
        imagePath: json['imagePath'] as String,
        city: json['city'] as String,
        cityId: json['cityId'] as String,
        sightId: json['sightId'] as String,
        address: json['address'] as String,
        worktime: json['worktime'] as String?,
        location: json['location'] as GeoPoint?,
      );
```

```dart
import 'package:firebase_auth/firebase_auth.dart' as FirebaseAuth;
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/foundation.dart';

class User extends ChangeNotifier {
  final String uid;
  final String email;
  final List<String> favoriteSights;

  User({
    required this.uid,
    required this.email,
    List<String>? favoriteSights,
  }) : this.favoriteSights = favoriteSights ?? [];

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      uid: json['uid'] as String,
      email: json['email'] as String,
      favoriteSights: List<String>.from(json['favoriteSights'] ?? []),
    ); // User
  }

  Map<String, dynamic> toJson() {
    return {
      'uid': uid,
      'email': email,
      'favoriteSights': favoriteSights,
    };
  }

  factory User.fromFirebaseUser(FirebaseAuth.User? firebaseUser) {
    if (firebaseUser == null) {
      return User(uid: '', email: '', favoriteSights: []);
    }
    return User(
      uid: firebaseUser.uid,
      email: firebaseUser.email ?? '',
      favoriteSights: [],
    );
  }

  factory User.fromFirestore(DocumentSnapshot doc) {
    Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
    return User(
      uid: data['uid'],
      email: data['email'],
      favoriteSights: List<String>.from(data['favoriteSights'] ?? []),
    ); // User
  }
```

Рис. 3.6 Демонстрація структури класів

У сьомому кроці ми створюємо окремий файл в якому будемо робити різні запити до нашої бази даних. Для цього нам необхідно імпортувати пакет, щоб працювати з Firestore (див. Рис. 3.7).

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
```

Рис. 3.7 Демонстрація імпорту пакета

Після того як ми імпортували пакет, ми зможемо використовувати функціонал Firestore. Для того щоб постійно не вказувати FirebaseFirestore.instance та функцію що нам потрібна, то ми присвоїмо значення таким чином :

final FirebaseFirestore _firestore = FirebaseFirestore.instance. Тепер ми можемо вказувати _firestore та необхідну функцію, наприклад _firestore.collection замість

FirebaseFirestore.instance.collection. Отже, розглянемо приклад функції запитів до нашої бази даних (див. Рис. 3.8):

```
163   Future<List<Sight>> fetchSights() async {
164     try {
165       final sightsCollection = _firestore.collection('Sights');
166       final querySnapshot = await sightsCollection.get();
167       final sights = querySnapshot.docs.map((doc) => Sight.fromJson(doc.data())).toList();
168       return sights;
169     } catch (error) {
170       print('Error fetching sights: $error');
171       return [];
172     }
173   }
174
175   Future<List<Categories>> fetchCategory() async {
176     try {
177       final categoryCollection = _firestore.collection('Category');
178       final querySnapshot = await categoryCollection.get();
179       final category = querySnapshot.docs.map((doc) => Categories.fromJson(doc.data())).toList();
180       return category;
181     } catch (error) {
182       print('Error fetching category: $error');
183       return [];
184     }
185   }
```

Рис. 3.8 Демонстрація запитів до бази даних

На даному рисунку ми бачимо два запита до бази даних для знаходження інформації про місця та категорій. Ми в кожній звертаємось до необхідної для колекції у Firestore та отримуємо дані з них, тоді в sights/category ми повертаємо дані зі списку документів та через функцію map() перетворюємо кожен документ на об’єкт Sight/Category за допомогою метода fromJson та результати map() ми перетворюємо у список завдяки toList(). Якщо у нас не виходить то ми отримуємо помилку, що не змогли знайти.

У восьмому кроці ми заповнемо дані через Firebase. Отже, відкриваємо Firestore Database та створюємо колекції, заповнюємо їх необхідними документами та в документах заповнюємо необхідну нам інформацію (див. Рис. 3.9).

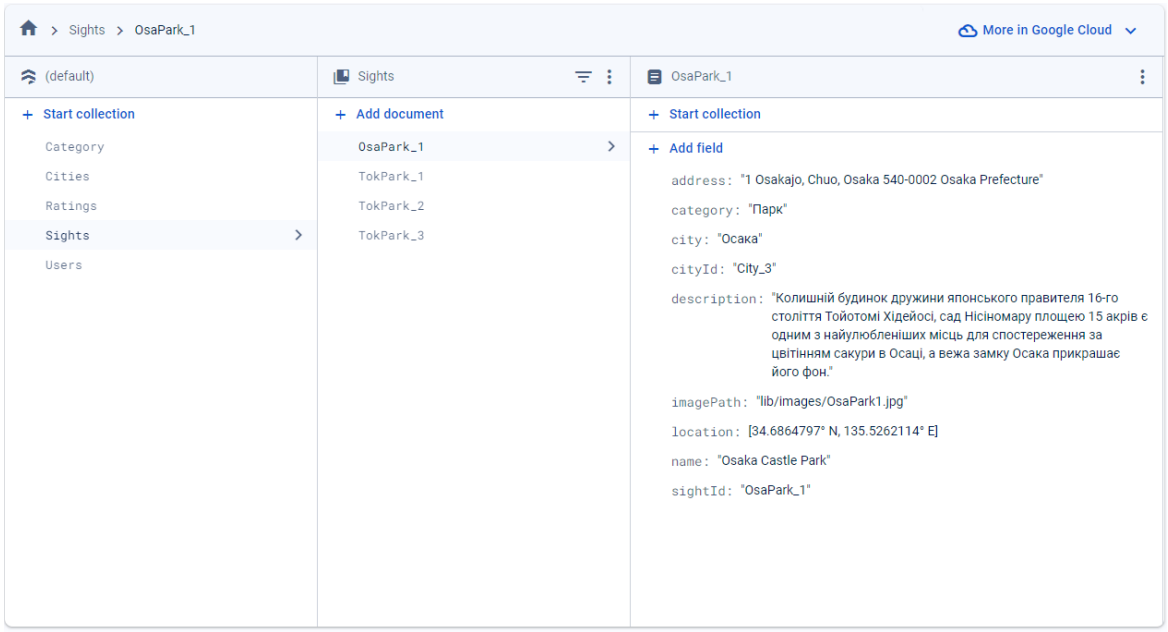


Рис. 3.9 Демонстрація бази даних

Також при заповненні даних документа потрібно перевіряти, що вони вірно підписані, так само як і в наших класах та мали однаковий зазначений тип, інакше будемо отримувати помилку, так як маємо використовувати обов’язкові атрибути які були зазначені у класі.

3.3 Створення компонентів

Flutter пропонує великий вибір компонетів для використання, проте ми будемо використовувати не тільки компоненти від Flutter, а й створювати свої універсальні компоненти. Отже створимо такі компоненти:

– Кнопка
– Кнопка вподобання
– Рейтинг бар
– Текстове поле
– Перемикач (drawer)
– Категорії

Кнопка. Отже, створимо нашу кнопку в ній ми будемо використовувати текст, іконку та функцію onTap (див. Рис. 3.10).

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

class MyButton extends StatelessWidget{
  final String text;
  final void Function()? onTap;

  const MyButton({
    super.key,
    required this.text,
    required this.onTap
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        decoration: BoxDecoration(color: Color.fromRGBO(203,64,66,1.000),borderRadius: BorderRadius.circular(10)),
        margin:  const EdgeInsets.symmetric(horizontal: 25.0),
        padding: const EdgeInsets.symmetric(vertical: 15.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              text,
              style: const TextStyle(
                fontWeight: FontWeight.w800,
                fontSize: 18,
                color: Colors.white
                ), // TextStyle
                ), // Text

            const SizedBox(width: 10),

            const Icon(Icons.arrow_forward_rounded,color: Colors.white)
              ],
          ), // Row
      ), // Container
    ); // GestureDetector
  }
}
```

Рис. 3.10 Демонстрація коду кнопки

Оскільки, наша кнопка буде використовуватися у різних місцях, ми задали їй два атрибута для тексту та функції, ці атрибути є обов’язковими тому стоїть напис required, тобто коли ми будемо додавати цю кнопку в іншому коді, то нам необхідно заповнити ці дані, інакше у нас буде помилка в коді і застосунок не скомпілюється. Отже далі ми використовуємо віджет GestureDetector, він використовується для відслідковування натиску та додаємо в нього нашу функцію onTap, ми додали цю функцію, тому що будемо потім використовувати її у різних місцях та з різним налаштуванням, тому ми тут не пишемо логіки, що буде відбуватися при натисканні. Також в ньому знаходиться контейнер, що має заокруглення та червоний колір, під тематику Японії. В контейнері знаходиться рядок в якому знаходиться текст, він так само як і onTap буде використовуватися при виклику в іншому коді, тому немає конкретного напису та знаходиться іконка стрілки.

Приклад використання кнопки на початковій сторінці та сторінці входу (див. Рис. 3.11 – 3.14):

```
51          //Кнопка
52          MyButton(
53            text: "Почати",
54            onTap: () {
55              Navigator.push(context, MaterialPageRoute(
56                builder: (context) => AuthGate())); // MaterialPageRoute
57            },
58            ), // MyButton
```

Рис. 3.11 Демонстрація використання кнопки у коді початкової сторінки

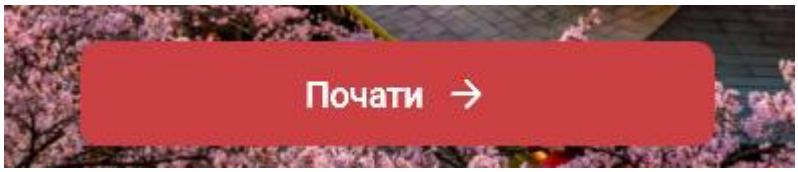


Рис. 3.12 Демонстрація кнопки на початковій сторінці

```
31          //Кнопка
32          MyButton(
33              text: "Увійти",
34              onTap: login,
35              ), // MyButton
```

Рис. 3.13 Демонстрація використання кнопки у коді сторінки входу

Рис. 3.14 Демонстрація кнопки на початковій сторінці

Можемо помітити, що в коді у початкової сторінки в onTap ми відразу пишемо логіку для функції нашої кнопки, а в коді сторінки входу ми виклакаємо іншу функцію, що буде виконуватися після натискання. Таким чином працює наша створена кнопка.

Кнопка вподобання. Має атрибути для визначення айді користувача та місця. Виконує відразу декілька функцій з нашого класу для роботи з базою даних, а саме робить перевірку чи вподобане місце та використовуючи віджет GestureDetector для натискання робимо логіку, також з використанням функцій з нашого класу FirebaseService. А для нашої іконки теж зробили логіку, вона змінює звичайну іконку серця на іконку серця з обведенням та зміною кольорів з червоного на білий (див. Рис. 3.15-3.18).

```dart
import 'package:flutter/material.dart';
import 'package:japan_guide_app/firebase/firebase_service.dart';

class LikeButton extends StatelessWidget {
  final String sightId;
  final String uid;

  const LikeButton({
    required this.sightId,
    required this.uid,
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return FutureBuilder<bool>(
      future: FirebaseService().isFavoriteSight(uid, sightId),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return CircularProgressIndicator();
        } else if (snapshot.hasError) {
          return Icon(Icons.favorite_border);
        } else {
          final isFavorite = snapshot.data!;

          return GestureDetector(
            onTap: () async {
              print('Current user uid: $uid');
              if (isFavorite) {
                await FirebaseService().removeFavoriteSight(uid, sightId);
              } else {
                await FirebaseService().addFavoriteSight(uid, sightId);
              }
            },
            child: Icon(
              isFavorite ? Icons.favorite : Icons.favorite_border,
              color: isFavorite ? Colors.red : Colors.white,
            ), // Icon
          ); // GestureDetector
        }
      },
    ); // FutureBuilder
  }
}

```

Рис. 3.15 Демонстрація коду кнопки вподобання

```dart
Future<bool> isFavoriteSight(String uid, String sightId) async {
  print('Checking if sight $sightId is favorite for uid: $uid');
  try {
    final userDoc = await _firestore.collection('Users').doc(uid).get();
    if (userDoc.exists) {
      final user = CustomUser.User.fromJson(userDoc.data()!);
      return user.favoriteSights.contains(sightId);
    } else {
      return false;
    }
  } catch (error) {
    print('Error checking favorite sight: $error');
    return false;
  }
}
```

Рис. 3.16 Демонстрація коду логіки чи вподобане місце

```dart
Future<void> addFavoriteSight(String uid, String sightId) async {
  print('Adding favorite sight for uid: $uid');
  final userRef = _firestore.collection('Users').doc(uid);
  await userRef.update({
    'favoriteSights': FieldValue.arrayUnion([sightId])
  });
}

Future<void> removeFavoriteSight(String uid, String sightId) async {
  final userRef = _firestore.collection('Users').doc(uid);
  await userRef.update({
    'favoriteSights': FieldValue.arrayRemove([sightId])
  });
}
```

Рис. 3.17 Демонстрація коду логіки додавання до списку вподобаних та видалення

Рис. 3.18 Демонстрація кнопки вподобання

Рейтинг бар. Для оцінювання місць мандрівником, був створений рейтинг бар, що відображає оцінку мандрівника у вигляді шкали з п'яти зірок. Він має схожу

логіку з копкою вподобання тільки виконує різні функції, а саме: отримання оцінки користувача , отримання середньої оцінки та при натисканні, щоб зберігався результат користувача (див. Рис. 3.19).

```dart
class _RatingBarState extends State<RatingBar> {
  double _userRating = 0.0;
  double _averageRating = 0.0;
  bool _isRated = false;

  @override
  void initState() {
    super.initState();
    _fetchUserRating();
    _fetchAverageRating();
  }

  Future<void> _fetchUserRating() async {
    final userRating = await FirebaseService().getUserRating(widget.sightId, widget.uid);
    setState(() {
      _userRating = userRating;
      _isRated = userRating > 0;
    });
  }

  Future<void> _fetchAverageRating() async {
    final averageRating = await FirebaseService().getAverageRating(widget.sightId);
    setState(() {
      _averageRating = averageRating;
    });
  }

  Future<void> _submitRating(double rating) async {
    await FirebaseService().updateSightRating(widget.sightId, widget.uid, rating);
    setState(() {
      _userRating = rating;
      _isRated = true;
    });
    await _fetchAverageRating();
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        _isRated
            ? Text(
                'Your rating: $_userRating',
                style: TextStyle(fontSize: 16),
              ) // Text
            : Text(
                'Rate this sight:',
                style: TextStyle(fontSize: 16),
              ), // Text
        SizedBox(height: 8),
        _isRated
            ? SizedBox.shrink()
            : Slider(
                value: _userRating,
                min: 0,
                max: 5,
                divisions: 5,
                onChanged: (value) {
                  setState(() {
                    _userRating = value;
                  });
                },
                label: '$_userRating',
              ), // Slider
        SizedBox(height: 8),
        ElevatedButton(
          onPressed: () {
            _submitRating(_userRating);
          },
          child: Text(_isRated ? 'Update Rating' : 'Submit Rating'),
        ), // ElevatedButton
        SizedBox(height: 8),
        Text(
          'Average Rating: $_averageRating',
          style: TextStyle(fontSize: 16),
        ), // Text
      ],
    ); // Column
  }
}
```

Рис. 3.19 Демонстрація коду рейтинг бара

Рейтинг місця вираховується через пошук в колекції оцінених місць в якій знаходяться документи з назвою айді парків і тоді в парку є документ з що містить айді користувачів, що проголосували та їх. Після пошуку отримуємо значення оцінок та додаємо їх до списку, а після додаємо їх значення та ділимо на кількість оцінок, а далі результат зберігається в колекції місць для документа нашого оціненого місця (див. Рис. 3.20).

```
121  Future<void> _updateAverageRating(String sightId) async {
122    final ratingsSnapshot = await _firestore
123        .collection('Ratings')
124        .doc(sightId)
125        .collection('UserRatings')
126        .get();
127
128    final ratings = ratingsSnapshot.docs
129        .map((doc) => doc['rating'] as double)
130        .toList();
131
132    final averageRating = ratings.isNotEmpty
133        ? ratings.reduce((a, b) => a + b) / ratings.length
134        : 0;
135
136    final sightRef = _firestore.collection('Sights').doc(sightId);
137    await sightRef.update({'averageRating': averageRating});
138  }
```

Рис. 3.20 Демонстрація коду підрахунку середньої оцінки

Категорія. Компонент категорії схожий за принципом вже зробленої нами кнопки, тільки в цей раз, назва категорії та її іконка будуть братися з класу категорій, що створений для отримання даних з бази даних. Саме тому ми викликаємо Categories та називаємо category, а для того щоб використати конкретні дані вводимо category.name для отримання назви категорії та category.item для отримання коду іконки, що буде використовуватися (див. Рис. 3.21-3.22).

```dart
import 'package:flutter/material.dart';
import '../models/categories.dart';

class MyCategory extends StatelessWidget {
  final Categories category;
  final void Function()? onTap;

  const MyCategory({
    required this.category,
    required this.onTap,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        decoration: BoxDecoration(
          color: Color.fromRGBO(203,64,66,1.000),
          borderRadius: BorderRadius.circular(8),
        ), // BoxDecoration
        width: 100,
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(IconData(
          category.icon,
          fontFamily: "MaterialIcons"
          ),  // IconData
          size: 40,
          color: Colors.white,),  // Icon
          SizedBox(height: 8),
          Text(
            category.name,
            style: TextStyle(
              fontFamily: "JapanFont",
              fontWeight: FontWeight.bold,
              color: Colors.white,
              fontSize: 16,
              ), // TextStyle
            ),  // Text
          ],
        ), // Column
      ), // Container
    ); // GestureDetector
  }
}
```

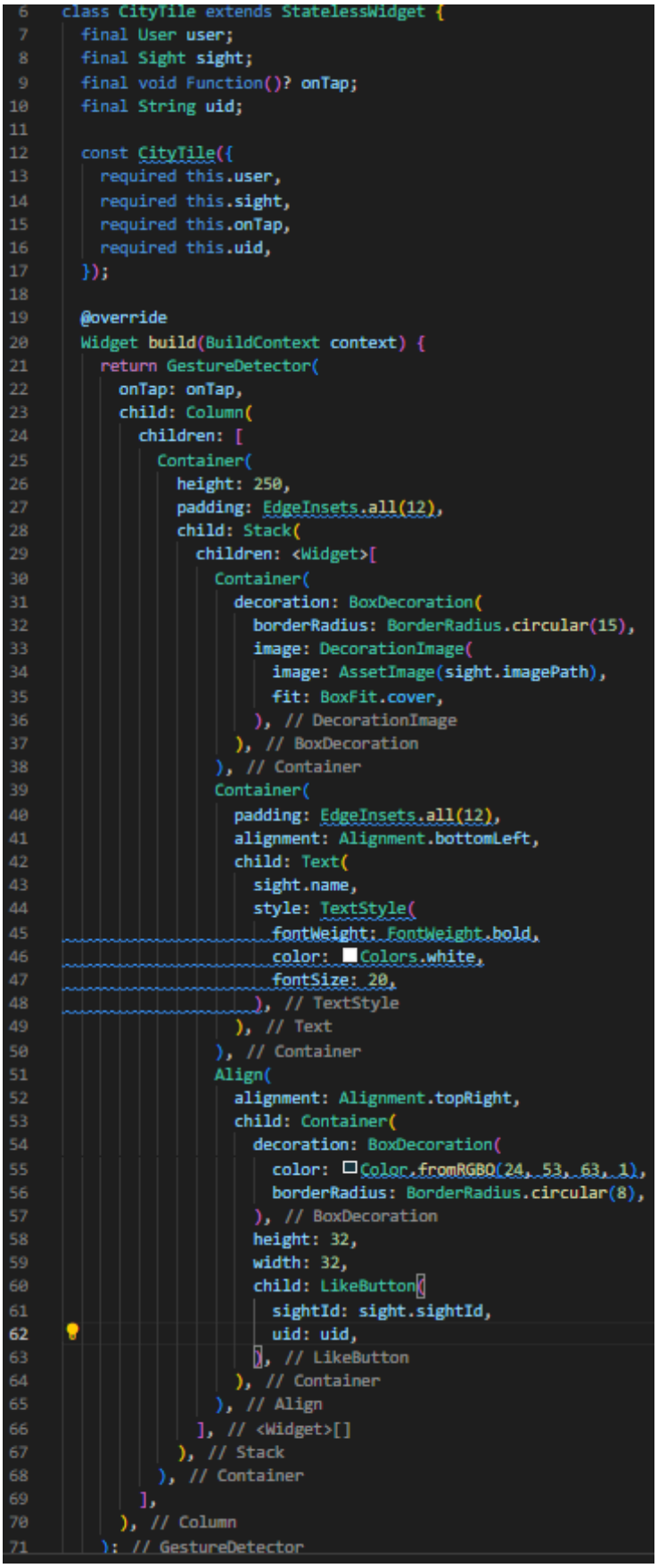
Рис. 3.21 Демонстрація коду категорій

Рис. 3.22 Демонстрація категорій

3.4 Створення карточки місця та сторінки місця

Карточка місця. Для створення карточки місця ми будемо використовувати класи користувача та місця для використання інформації з них. З класу місць для карточки ми будемо брати значення назви місця та фотографії, ми будемо відображати прямокутний контейнер в якому буде знаходитися фото місця та поверх фото назва місця й наш розроблений компонент кнопка вподобання. Для того, щоб розмістити ці віджети один на одного потрібно використати віджет Stack, він кожен наступний віджет накладає на попередній, тому спочатку ми розташовуємо віджет контейнер з фотографією, потім з текстом та використовуємо віджет Align для розміщення нашої кнопки вподобання у правому верхньому вікні. Клас користувача тут використовується для кнопки вподобання, щоб розуміти який користувач додає місце до вподобаних (див. Рис. 3.23-3.24).

Рис. 3.23 Демонстрація карточки місця



```dart
class CityTile extends StatelessWidget {
  final User user;
  final Sight sight;
  final void Function()? onTap;
  final String uid;

  const CityTile({
    required this.user,
    required this.sight,
    required this.onTap,
    required this.uid,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Column(
        children: [
          Container(
            height: 250,
            padding: EdgeInsets.all(12),
            child: Stack(
              children: <Widget>[
                Container(
                  decoration: BoxDecoration(
                    borderRadius: BorderRadius.circular(15),
                    image: DecorationImage(
                      image: AssetImage(sight.imagePath),
                      fit: BoxFit.cover,
                    ), // DecorationImage
                  ), // BoxDecoration
                ), // Container
                Container(
                  padding: EdgeInsets.all(12),
                  alignment: Alignment.bottomLeft,
                  child: Text(
                    sight.name,
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.white,
                      fontSize: 20,
                    ), // TextStyle
                  ), // Text
                ), // Container
                Align(
                  alignment: Alignment.topRight,
                  child: Container(
                    decoration: BoxDecoration(
                      color: Color.fromRGBO(24, 53, 63, 1),
                      borderRadius: BorderRadius.circular(8),
                    ), // BoxDecoration
                    height: 32,
                    width: 32,
                    child: LikeButton(
                      sightId: sight.sightId,
                      uid: uid,
                    ), // LikeButton
                  ), // Container
                ), // Align
              ], // <Widget>[]
            ), // Stack
          ), // Container
        ],
      ), // Column
    ); // GestureDetector
```

Рис. 3.24 Демонстрація коду карточки місця

Сторінка місця. На нашій сторінці має бути детальна інформація про місце, а саме: назва, місто, адреса, час роботи, шкала оцінювання, опис та кнопка, що відкриває карту та показує місце. Ми використовуємо класи місця та користувача, клас місця для заповнення інформацією сторінки, користувача для фіксації його оцінки місця. Також ми створили додатковий компонент для нашої сторінки, що робить можливість показувати більше чи менше інформації про місце. Для відображення інформації ми використовуємо функції з нашого класу для роботи з базою даних, а саме функція для знаходження деталей місця, знаходження рейтингу користувача та оновлення рейтингу (див. Рис. 3.25-3.27).

```
35  Future<void> fetchSightDetails(String sightId) async {
36    try {
37      final sight = await firebaseService.fetchSightById(sightId);
38      setState(() {
39        _sight = sight;
40      });
41    } catch (error) {
42      print('Error fetching sight: $error');
43    }
44  }
45
46  Future<void> fetchUserRating(String sightId, String userId) async {
47    try {
48      final rating = await firebaseService.getUserRating(sightId, userId);
49      setState(() {
50        _rating = rating;
51      });
52    } catch (error) {
53      print('Error fetching user rating: $error');
54    }
55  }
56
57  Future<void> updateRating(double rating) async {
58    if (user != null) {
59      try {
60        await firebaseService.updateSightRating(widget.sightId, user!.uid, rating);
61        setState(() {
62          _rating = rating;
63        });
64      } catch (error) {
65        print('Error updating rating: $error');
66      }
67    } else {
68      print('User not logged in');
69    }
70  }
```

Рис. 3.25 Демонстрація коду використаних функцій

```dart
class _ExpandableTextState extends State<ExpandableText> {
  bool _readMore = true;

  @override
  Widget build(BuildContext context) {
    final textSpan = TextSpan(
      text: widget.text,
      style: widget.style ?? TextStyle(fontSize: 18),
    ); // TextSpan

    final textPainter = TextPainter(
      text: textSpan,
      maxLines: widget.trimLines,
      textDirection: TextDirection.ltr,
    );

    textPainter.layout(maxWidth: MediaQuery.of(context).size.width);

    final textExceeds = textPainter.didExceedMaxLines;

    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Text.rich(
          TextSpan(
            children: [
              if (_readMore && textExceeds)
                TextSpan(
                  text: widget.text.substring(0,
                  textPainter.getPositionForOffset(Offset(
                      textPainter.width, textPainter.height)).offset) + "...",
                ) // TextSpan
              else
                TextSpan(
                  text: widget.text,
                ), // TextSpan
            ],
          ), // TextSpan
          style: widget.style,
        ), // Text.rich
        if (textExceeds)
          InkWell(
            onTap: () {
              setState(() {
                _readMore = !_readMore;
              });
            },
            child: Text(
              _readMore ? 'Дивитися більше' : 'Дивитися менше',
              style: TextStyle(
                fontSize: 14,
                fontWeight: FontWeight.bold,
                color: Color.fromRGBO(203,64,66,1.000),
```

Рис. 3.26 Демонстрація коду компонента для зменшення тесту

Рис. 3.27 Демонстрація сторінки місця

3.5 Головна сторінка

Головна сторінка відображає таб бар для вибору міста, нижнє навігаційне меню та вже розроблену нами сторінку, що відображає в собі віджети категорії та карточок місць. У коді головної сторінки ми вже будемо використовувати наші готові класи шаблони, що були створені для відображення категорій та карточок міст, тепер ми будемо заповняти їх даними. Тобто ми будемо відображати всі місця, міста та категорії, що заповнені у нашій базі даних.

Для початку створимо апп бар для нашої сторінки та нижнє навігаційне меню, яке буде складатися з назви та іконки певного меню. Всього користувач зможе перейти через навігаційне меню на чотири сторінки, а саме на головну, сторінку вподобаних місць, сторінки з додатковою інформацією та мапою (див. Рис. 3.28).

```
52  @override
53  Widget build(BuildContext context) {
54    return Scaffold(
55      appBar: AppBar(
56        title: Text(
57          "Я П О Н І Я",
58          style: TextStyle(fontFamily: "JapanFont"),
59        ), // Text
60      ), // AppBar
61      drawer: const MyDrawer(),
62      body: _getBody(_selectedIndex),
63      bottomNavigationBar: BottomNavigationBar(
64        items: const <BottomNavigationBarItem>[
65          BottomNavigationBarItem(
66            backgroundColor: Color.fromRGBO(203,64,66,1.000),
67            icon: Icon(Icons.home),
68            label: 'Головна',
69          ), // BottomNavigationBarItem
70          BottomNavigationBarItem(
71            icon: Icon(Icons.favorite),
72            label: 'Вподобані',
73          ), // BottomNavigationBarItem
74          BottomNavigationBarItem(
75            icon: Icon(Icons.info_outlined),
76            label: 'Додатково',
77          ), // BottomNavigationBarItem
78          BottomNavigationBarItem(
79            icon: Icon(Icons.map),
80            label: 'Мапа',
81          ), // BottomNavigationBarItem
82        ], // <BottomNavigationBarItem>[]
83        currentIndex: _selectedIndex,
84        onTap: _onItemTapped,
85      ), // BottomNavigationBar
86    ); // Scaffold
87  }
```

Рис. 3.28 Демонстрація коду для створення апп бару та нижнього навігаційного меню

Далі робимо логіку для нашого нижнього навігаційного меню, яке буде відкривати для нас інші сторінки застосунку (див. Рис. 3.29).

```
90   Widget _getBody(int index) {
91     switch (index) {
92       case 0:
93         return _buildHome();
94       case 1:
95         return FutureBuilder<LocalUser.User?>(
96           future: _user,
97           builder: (context, userSnapshot) {
98             if (userSnapshot.connectionState == ConnectionState.waiting) {
99               return Center(child: CircularProgressIndicator());
100            } else if (userSnapshot.hasError) {
101              return Center(child: Text('Error fetching user data'));
102            } else if (!userSnapshot.hasData) {
103              return Center(child: Text('Please log in'));
104            } else {
105              final user = userSnapshot.data!;
106              return FavoritesPage(user: user);
107            }
108          },
109        ); // FutureBuilder
110      case 2:
111        return const AdditionPage();
112      case 3:
113        return const MapPage();
114      default:
115        return Container();
116    }
117  }
```

Рис. 3.29 – Демонстрація коду логіки нижнього навігаційного меню

Після робимо перевірки, що знайшли дані для користувача, міст та місць та заповнюємо їх через наші клас шаблон, що вже розробили. Отже наш TabBar заповнюється списком міст, а TabBarView, відображає наповнення для кожного міста. Кількість табів, що створиться вираховується кількістю міст створених у базі даних (див. Рис. 3.30).

```
return DefaultTabController(
  length: cities.length,
  child: Column(
    children: [
      const SizedBox(height: 10,),
      TabBar(
        isScrollable: true,
        tabs: cities.map((city) => MyTab(text: city.name)).toList(),
      ), // TabBar
      Expanded(
        child: TabBarView(
          children: cities.map((city) {
            if (uid != null) {
              return CityTab(city: city, category: category[0], user: user, uid: uid!);
            } else {
              return Center(child: Text('User ID not available'));
            }
          }).toList(),
        ), // TabBarView
      ), // Expanded
    ],
  ), // Column
); // DefaultTabController
```

Рис. 3.30 Демонстрація коду заповнення TabBar та TabBarView

І в результаті ми отримуємо список табів міст, що можна гортати для пошуку необхідного міста, список з вибором категорії та список популярних місць серед користувачів, що відображає місця які мають оцінку вище чотирьох зірок (див. Рис. 3.31).

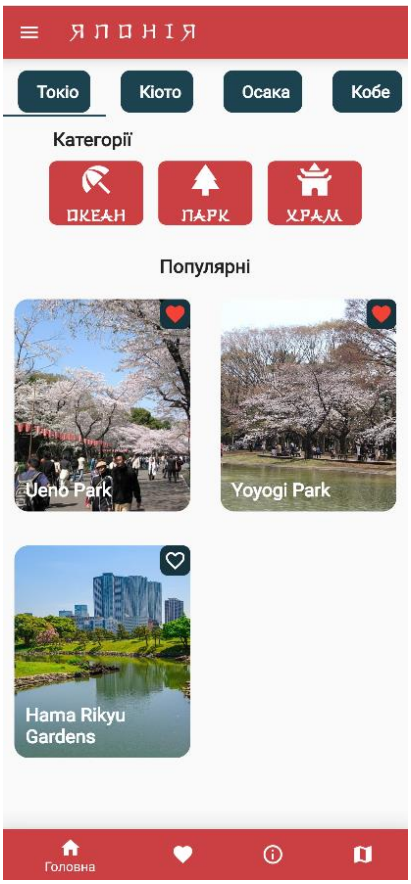


Рис. 3.31 Демонстрація головної сторінки

3.6 Створення мапи

Для того щоб створити мапу нам необхідно встановити плагін Google Maps Flutter. Для інсталяції в терміналі вводимо команду: flutter pub add google_maps_flutter. Після цього реєструємося на сервісі Google Maps Platform та отримуємо звідти API ключ і додаємо його в файли наших пристроїв, щоб могли відображати карту.

Тепер створюємо файл, що буде відображати мапу та наші мітки з розташуванням місць. Отже, імпортуємо в наш файл пакет:

import 'package:google_maps_flutter/google_maps_flutter.dart';

Даний пакет надає нам можливість використовувати віджет GoogleMap, в якому ми можемо налаштовати з якого місця буде відображатися наша мапа, додавати маркери та з яким приближенням буде відображатися місце. У цьому віджеті є ще багато інших функцій для застосування, проте вони нам не потрібні. Отже, налаштовуємо наш віджет таким чином, що приближення місць у нас буде 15, якщо ми відкриваємо карту через нижнє навігаційне меню, то буде відображатися за координатами наша вказана центральна точка, а якщо відкривається через кнопку, що знаходиться на сторінці місця, то буде відображати на карті мітку з тим місцем. Оскільки, місць в нас багато, то будемо використовувати список міток (див. Рис. 3.32).

```
72    @override
73    Widget build(BuildContext context) {
74      return Scaffold(
75        body: GoogleMap(
76          onMapCreated: _onMapCreated,
77          initialCameraPosition: CameraPosition(
78            target: widget.targetLocation ?? _center,
79            zoom: 15.0,
80          ), // CameraPosition
81          markers: _markers,
82        ), // GoogleMap
83      ); // Scaffold
84    }
85  }
```

Рис. 3.32 Демонстрація коду віджета GoogleMap

Після цього налаштуємо віджет Marker, який відповідає за налаштування мітки. В ній ми вказуємо айді для міток та їх координати, а також додаємо інформаційне меню, що буде відображати коротку інформацію про вибрану мітку. У інформаційному меню відобразимо назву місця, функцію натискання та текст, що буде пропонувати натиснути ще раз на інформаційне меню, щоб відкрити інформацію про обране місце (див. Рис. 3.33).

```dart
Future<void> _loadSights() async {
  try {
    List<Sight> sights = await _firebaseService.fetchSights();
    setState(() {
      _markers = sights.where((sight) => sight.location != null).map((sight) {
        return Marker(
          markerId: MarkerId(sight.sightId),
          position: LatLng(sight.location!.latitude, sight.location!.longitude),
          infoWindow: InfoWindow(
            title: sight.name,
            snippet: "Натисніть щоб дізнатися більше про місце",
            onTap: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => SightPage(
                          sightId: sight.sightId,
                        ), // SightPage
                      ), // MaterialPageRoute
                    );
                  }
          ), // InfoWindow
        ); // Marker
      }).toSet();
    });
  } catch (e) {
    print('Error fetching sights: $e');
  }
}
```

Рис. 3.33 Демонстрація коду для мітки

Також можемо побачити, що використовуємо конструкцію try, для того щоб зробити запит до бази та знайти наше місце та перевірити, якщо вказані координати то повернути наш віджет Marker з вказаними координатами місця, а якщо не знаходимо то отримуємо повідомлення, що не знайшли. Координати отримуємо через тип даних GeoPoint, який зберігає в собі два значення double широти (Latitude) та довготи (Longitude) . В результаті отримуємо карту зображену на рисунку 3.34.

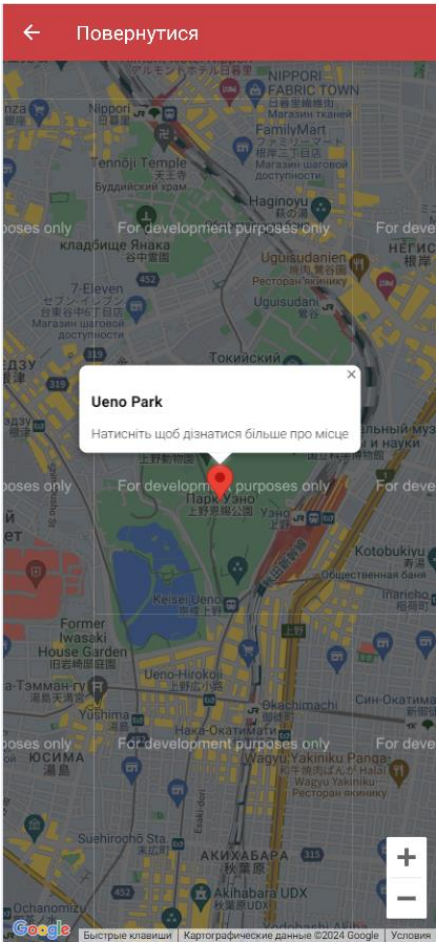


Рис. 3.34 Демонстрація мапи та мітки з описом

3.7 Тестування програмного забезпечення

Модульне тестування або Unit test – це тестування для перевірки різних методів, класів та функцій, що є ізольованими від інших систем для перевірки справності коду та отриманні очікуваних результатів. Тестування таким методом допомагає виявити помилки в коді на початковому етапі розроблення, що економить час та запобігає появі нових помилок.

Проведемо Unit test для нашої моделі Sight, що отримує дані з JSON, який ми створили та перевіряємо його на перетворення назад у свій формат. Зробимо помилки у файлі та побачимо результат (див. Рис. 3.35-3.37).

```dart
test('Sight model should convert from JSON correctly', () {
  final json = {
    'name': 'Тестова місце',
    'category': 'храм',
    'description': 'Кравсивий стародавній храм.',
    'imagePath': 'images/test.jpg',
    'city': 'Токіо',
    'cityId': 'Ток_1',
    'sightId': 'Храм_1',
    'address': '123 вулиця',
    'worktime': 8,
    'location': GeoPoint(35.6895, 139.6917),
  };

  final sight = Sight.fromJson(json);

  expect(sight.name, 'Храм Кітсуне');
  expect(sight.category, 'храм');
  expect(sight.description, 'Кравсивий стародавній храм.');
  expect(sight.imagePath, 'images/test.jpg');
  expect(sight.city, 'Токіо');
  expect(sight.cityId, 'Ток_1');
  expect(sight.sightId, 'Храм_1');
  expect(sight.address, '123 вулиця');
  expect(sight.worktime, '9:00 - 17:00');
  expect(sight.location!.latitude, 35.6895);
  expect(sight.location!.longitude, 139.6917);
});
```

Рис. 3.35 Демонстрація коду Unit тестування

```
Expected: 'Храм Кітсуне' Sight model should convert from JSON correctly
  Expected: 'Храм Кітсуне'
    Actual: 'Тестова місце'
     Which: is different.
                Expected: Храм Кітсу ...
                  Actual: Тестова мі ...
                               ^
                Differ at offset 0

package:matcher                                    expect
package:flutter_test/src/widget_tester.dart 454:18 expect
test\sight_test.dart 23:7                          main.<fn>.<fn>
```

Рис. 3.36 Демонстрація першої отриманої помилки

```
type 'int' is not a subtype of type 'String?' in type cast Sight model should convert from JSON correctly
  type 'int' is not a subtype of type 'String?' in type cast
  package:japan_guide_app/models/sights.dart 52:36  new Sight.fromJson
  test\sight_test.dart 21:27                        main.<fn>.<fn>
```

Рис. 3.37 Демонстрація другої отриманої помилки

Ми отримали помилку, що очікуваний результат та актуальний відрізняється, а саме можемо побачити, що відрізняється текстова частина. Також отримали іншу помилку, вона повідомляє про те, що тип int не є підтипом String?, це через вказання

нами числа, а не тексту як зазначається в класі Sight. Після введення коректних даних отримуємо позитивний результат тесту.

Ручне тестування – це коли тестувальник або розробник виконує перевірку застосунку у якості користувача та наприклад перевіряє функціонал застосунку, інтерфейс користувача. При перевірці функціоналу тестувальник оцінює чи вірно виконується певна функція застосунку, якщо функція виконує свою задачу вірно, то тест пройдено. При перевірці інтерфейсу користувача звертають увагу на кнопки, меню, поля чи вони знаходяться на вірному місці, чи правильний відступ між елементами, чи всі необхідні елементи натискаються.

Отже тестування застосунку є невід’ємною частиною його розробки та необхідно приділяти увагу тестуванню для запобіганню виникненню помилок у програмному забезпеченні.

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено мобільний застосунок путівник для легшого подорожування Японією. Її актуальність полягає у тому, що наразі процес планування та проведення подорожей Японією досить складна задача для мандрівників, особливо тих, хто не володіє японською мовою та не знайомий з місцевими особливостями.

1. Проаналізовано предметну область дипломної роботи та виявлено п’ять застосунків, що мають подібний функціоанал, а саме: Вся Україна, Redigo, Путівник - World Explorer, Sygic Travel Maps Trip Planner, Токіо путівник. Було визначено переваги та недоліки, а також створена порівняльна таблиця для цих застосунків.

2. Розроблено технічне завдання, а також визначили функціональні та нефункціональні вимоги.

3. Дослідивши найкращі засоби для створення кросплаформенного застосунку, було обрано такі інструменти, як: мова програмування Dart, фреймворк Flutter, платформа Firebase, середовище розробки Visual Studio Code та Google Maps API.

4. Розроблено застосунок мобільний кросплатформенний застосунок путівник для легшого подорожування Японією з використанням найкращих засобів для розробки кросплатформенних застосунків, а саме: Flutter та Firebase.

5. Протестовано застосунок за допомогою ручного та Unit тестування.

6. Робота пройшла абробацію на науково-технічній конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» за темами «Аналіз і порівняння фрейморків Flutter та Xamarin для розробки кросплатформеного застосунку» та «Використання платформи Firebase для створення кросплатформених застосунків» Київ, ДУІКТ, 24 квітня 2024 року.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ragin A. 10 key statistics that show the importance of mobile solutions in the travel industry. *Zoftify*. URL: https://zoftify.com/blog/mobile-solutions-for-the-travel-industry (date of access: 24.05.2024).
2. Travel app trends and insights for 2023 and beyond | Adjust. *Accelerate your app's growth with Adjust | Adjust*. URL: https://www.adjust.com/blog/travel-apps-2023/ (date of access: 24.05.2024).
3. Travel websites: global traffic by device 2023 | Statista. *Statista*. URL: https://www.statista.com/statistics/1323824/travel-tourism-websites-traffic-by-device-worldwide/ (date of access: 24.05.2024).
4. Douglas A. Mobile business travel application usage. *Journal of hospitality and tourism technology*. 2019. Vol. 10, no. 3. P. 269–285. URL: https://doi.org/10.1108/jhtt-01-2018-0002 (date of access: 24.05.2024).
5. Evgeniy A. Вся україна. Версія 1.0.
6. Toozla LLC. Redigo. Version 1.8. URL: https://redigo.ru.aptoide.com/app.
7. Tasmanic Editions. World explorer - путівник. Версія 4.1.1. URL: https://play.google.com/store/apps/details?id=com.audioguidia.worldexplorer&hl=uk.
8. Tripomatic s.r.o. Sygic travel maps trip planner. Version 5.18.1. URL: https://play.google.com/store/apps/details?id=com.tripomatic&hl=uk.
9. ETIPS INC. Токіо путівник. Версія 1.0.29. URL: https://play.google.com/store/apps/details?id=com.etips.tokyo.travel.guide&hl=uk.
10. Top 10 best cross platform app development frameworks. *Appinventiv*. URL: https://appinventiv.com/blog/cross-platform-app-frameworks/ (date of access: 24.05.2024).
11. SATTAR, Arif Md, et al. Accelerating Cross-platform Development with Flutter Framework. 2023.

12. Flutter vs xamarin: what's better for app development? - existek blog. *Offshore Software Development Company | Custom Software Development Services*. URL: https://existek.com/blog/flutter-vs-xamarin-what-is-better/ (date of access: 24.05.2024).
13. Prudnikov V. Flutter vs xamarin: what to choose for mobile app?. *WTF Blog*. URL: https://blog.flutter.wtf/flutter-vs-xamarin/ (date of access: 24.05.2024).
14. What is xamarin.forms? - xamarin. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/previous-versions/xamarin/get-started/what-is-xamarin-forms (date of access: 24.05.2024).
15. Faq. *Docs | Flutter*. URL: https://docs.flutter.dev/resources/faq#what-kinds-of-apps-can-i-build-with-flutter (date of access: 24.05.2024).
16. Firebase vs azure: top differences - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/firebase-vs-azure/ (date of access: 24.05.2024).
17. Flutter architectural overview. *Docs | Flutter*. URL: https://docs.flutter.dev/resources/architecturaloverview#:~:text=Flutter%20is%20a%20cross-platform,directly%20with%20underlying%20platform%20services. (date of access: 24.05.2024).
18. Kaur I. Mobile cloud computing: using firebase auth. *International journal of computer science and mobile computing*. 2022. Vol. 11, no. 4. P. 61–68. URL: https://doi.org/10.47760/ijcsmc.2022.v11i04.008 (date of access: 24.05.2024).
19. Before you begin | google maps for flutter | google for developers. *Google for Developers*. URL: https://developers.google.com/maps/flutter-package/overview (date of access: 24.05.2024).
20. An introduction to unit testing. *Docs | Flutter*. URL: https://docs.flutter.dev/cookbook/testing/unit/introduction (date of access: 24.05.2024).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку путівника для легшого подорожування Японією мовою Dart з використанням фреймворку Flutter

Виконав студент 4 курсу
групи ПД-41
Степанченко Артем Олегович
Керівник роботи
доктор філософії (PhD), доцент кафедри ІПЗ Дібрівний Олесь Андрійович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи -** спрощення процесу подорожування Японією, знаходження інформації про місця та планування подорожі
- **Об’єкт дослідження -** процес подорожування Японією, знаходження інформації про місця та планування подорожі
- **Предмет дослідження -** мобільний застосунок путівник для підтримки процесу легшого подорожування Японією

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати сферу туризму та визначити доцільність розробки мобільного застосунку для подорожування Японією.
2. Проаналізувати існуючі програмні забезпечення,що мають схожий функціонал та виділити їх переваги й недоліки.
3. Порівняти та обрати технології, інструменти і програмні засоби для реалізації мобільного застосунку путівника для легшого подорожування Японією.
4. Реалізувати базовий функціонал застосунку.
5. Провести тестування застосунку.
6. Пройти апробацію на науково-технічних конференціях

3

АНАЛІЗ АНАЛОГІВ

Застосунок	Вся Україна	Redigo	Путівник - World Explorer	Sygic Travel Maps Trip Planner	Токіо путівник	**ArkTravel**
Кросплатформенність	-	-	-	+	+	**+**
Інтерфейс враховує особливості Японії	-	-	-	-	-	**+**
Орієнтованість путівника	Вся країна	Весь світ	Весь світ	Весь світ	Місто	**Вся країна**
Карти місцевості	+	+	+	+	+	**+**
Рейтинг місць від користувачів	-	-	-	-	-	**+**
Є додаткова інформація для користувача	-	-	-	-	+	**+**
Має унікальний додатковий функціонал пов’язаний з країною	-	-	-	-	-	**+**

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні:

1. Пошук та додавання інформації про місця, пам'ятки;
2. Розподілення місць за відповідними категоріями, такими як парк, храми та інші, щоб зробити зручним вибір необхідних об'єктів
3. Можливість додавати місця у вподобані, щоб було зручніше знайти зацікавлені місця;
4. Відображати список популярних місць, щоб користувач міг ознайомитися з думкою інших користувачів;
5. Інтерактивна карта, яка містить позначки для швикого пошуку та навігації;

Не функціональні:

1. Застосунок повинен підтримувати пристрої, що працюють на Android з версії 9.0 та для пристроях з IOS з версії 15.0.
2. Застосунок повинен працювати без будь-яких змін у продуктивності для всіх мобільних ОС.
3. Необхідно використовувати безпечну авторизацію мандрівника
4. Застосунок повинен мати автентичний для місцевості інтерфейс та інші культурні аспекти.

5

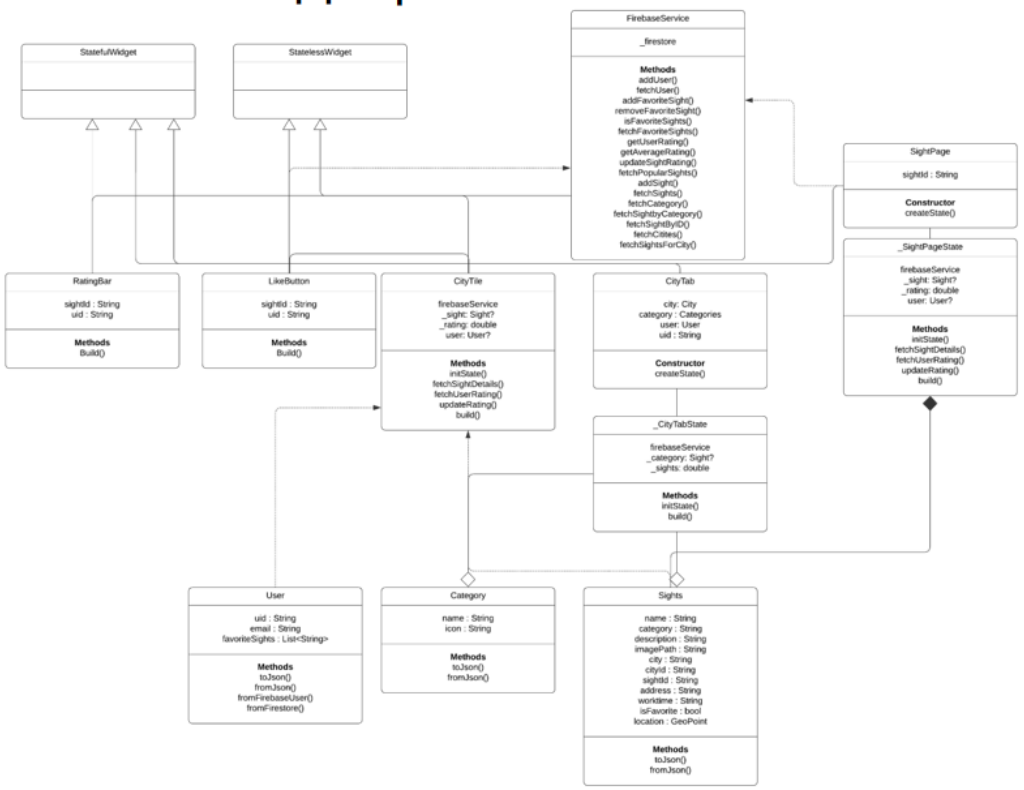
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

Діаграма варіантів використання

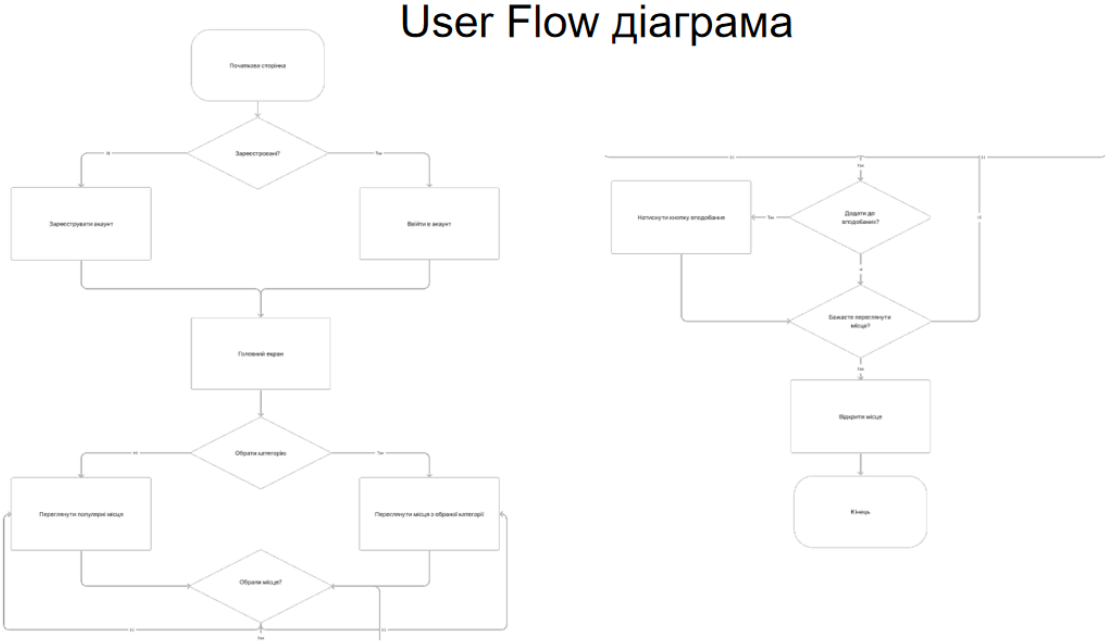
7

Діаграма класів



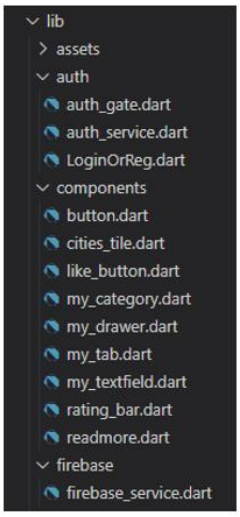
8

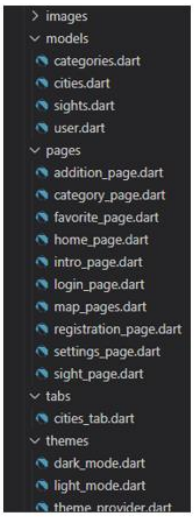
User Flow діаграма



9

Файлова структура проєкту

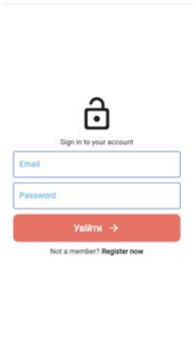




10

ЕКРАННІ ФОРМИ

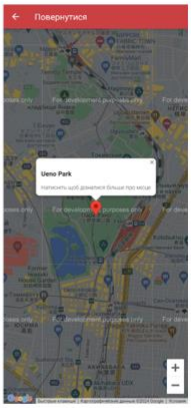
Стартова сторінка



Вхід

Головна сторінка

Опис місця



Мапа

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Степанченко А.О., Дібрівний О.А. Аналіз і порівняння фрейморків Flutter та Xamarin для розробки кросплатформеного застосунку. Всеукраїнська наукова-технічна конференція “Застосування програмного забезпечення в ІКТ”, 24 квітня 2023р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 69-70.
2. Степанченко А.О., Дібрівний О.А. Використання платформи Firebase для створення кросплатформених застосунків. Всеукраїнська наукова-технічна конференція “Застосування програмного забезпечення в ІКТ”, 24 квітня 2023р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 157-158.

12

ВИСНОВКИ

1. Проведено аналіз сфери туризму та визначено, що розробка мобільного застосунку є доцільною через попит мандрівників.
2. Проаналізовано предметну область дипломної роботи та виявлено п’ять мобільних застосунків з подібним функціоналом, а саме: “Вся Україна”, “Redigo”, “Путівник - World Explorer”, “Sygic Travel Maps Trip Planner”, “Токіо путівник”.
3. Проведено порівняння та обрано такі засоби реалізації як: мова програмування Dart, фреймворк Flutter, база даних Firebase та середа розробки Visual Studio Code.
4. Розроблено мобільний застосунок для легшого подорожування Японією, дотримуючись вимог,а саме такий функціонал: перегляд за категоріями, додавання до вподобаних, використання карти та інші.
5. Проведено тестування функціоналу мобільного застосунку за допомогою Unit та ручного тестування.

13

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Код для кнопки

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

class MyButton extends StatelessWidget{
  final String text;
  final void Function()? onTap;

  const MyButton({
    super.key,
    required this.text,
    required this.onTap
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        decoration: BoxDecoration(color: Color.fromRGBO(203,64,66,1.000),borderRadius: BorderRadius.circular(10)),
        margin:  const EdgeInsets.symmetric(horizontal: 25.0),
        padding: const EdgeInsets.symmetric(vertical: 15.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              text,
              style: const TextStyle(
                fontWeight: FontWeight.w800,
                fontSize: 18,
                color: Colors.white
                ),
                ),

            const SizedBox(width: 10),

            const Icon(Icons.arrow_forward_rounded,color: Colors.white)
              ],
          ),
      ),
    );
  }
}
```

Код для карточки місця

```
import 'package:flutter/material.dart';
import 'package:japan_guide_app/components/like_button.dart';
import 'package:japan_guide_app/models/user.dart';
import '../models/sights.dart';

class CityTile extends StatelessWidget {
  final User user;
  final Sight sight;
  final void Function()? onTap;
  final String uid;

  const CityTile({
    required this.user,
    required this.sight,
    required this.onTap,
    required this.uid,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Column(
        children: [
          Container(
            height: 250,
            padding: EdgeInsets.all(12),
            child: Stack(
              children: <Widget>[
                Container(
                  decoration: BoxDecoration(
                    borderRadius: BorderRadius.circular(15),
                    image: DecorationImage(
                      image: AssetImage(sight.imagePath),
                      fit: BoxFit.cover,
                    ),
                  ),
                ),
                Container(
                  padding: EdgeInsets.all(12),
                  alignment: Alignment.bottomLeft,
                  child: Text(
                    sight.name,
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.white,
                      fontSize: 20,
                    ),
```

```
              ),
            ),
            Align(
              alignment: Alignment.topRight,
              child: Container(
                decoration: BoxDecoration(
                  color: Color.fromRGBO(24, 53, 63, 1),
                  borderRadius: BorderRadius.circular(8),
                ),
                height: 32,
                width: 32,
                child: LikeButton(
                  sightId: sight.sightId,
                  uid: uid,
                ),
              ),
            ),
          ],
        ),
      ),
    ],
  ),
 );
 }
}
```

Код для кнопки вподобання

```
import 'package:flutter/material.dart';
import 'package:japan_guide_app/firebase/firebase_service.dart';

class LikeButton extends StatefulWidget {
 final String sightId;
 final String uid;

 const LikeButton({
  required this.sightId,
  required this.uid,
  Key? key,
 }) : super(key: key);

 @override
 _LikeButtonState createState() => _LikeButtonState();
}

class _LikeButtonState extends State<LikeButton> {
 late Future<bool> _isFavoriteFuture;

 @override
 void initState() {
  super.initState();
```

```
    _isFavoriteFuture = FirebaseService().isFavoriteSight(widget.uid, widget.sightId);
  }

  @override
  Widget build(BuildContext context) {
    return FutureBuilder<bool>(
      future: _isFavoriteFuture,
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return CircularProgressIndicator();
        } else if (snapshot.hasError) {
          return Icon(Icons.favorite_border);
        } else {
          final isFavorite = snapshot.data!;

          return GestureDetector(
            onTap: () async {
              if (isFavorite) {
                await FirebaseService().removeFavoriteSight(widget.uid, widget.sightId);
              } else {
                await FirebaseService().addFavoriteSight(widget.uid, widget.sightId);
              }
              setState(() {
                _isFavoriteFuture = FirebaseService().isFavoriteSight(widget.uid, widget.sightId);
              });
            },
            child: Icon(
              isFavorite ? Icons.favorite : Icons.favorite_border,
              color: isFavorite ? Colors.red : Colors.white,
            ),
          );
        }
      },
    );
  }
}
```

Код для категорій

```
import 'package:flutter/material.dart';
import '../models/categories.dart';

class MyCategory extends StatelessWidget {
  final Categories category;
  final void Function()? onTap;

  const MyCategory({
    required this.category,
    required this.onTap,
  });
```

```
 @override
 Widget build(BuildContext context) {
  return GestureDetector(
    onTap: onTap,
    child: Container(
      decoration: BoxDecoration(
        color: Color.fromRGBO(203,64,66,1.000),
        borderRadius: BorderRadius.circular(8),
      ),
       width: 100,
      child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(IconData(
        category.icon,
        fontFamily: "MaterialIcons"
        ),
        size: 40,
        color: Colors.white,),
        SizedBox(height: 8),
        Text(
          category.name,
          style: TextStyle(
            fontFamily: "JapanFont",
            fontWeight: FontWeight.bold,
            color: Colors.white,
            fontSize: 16,
            ),
          ),
        ],
      ),
    ),
   );
  }
}
```

Код для перемикача

```
import 'package:flutter/material.dart';
import 'package:japan_guide_app/auth/auth_service.dart';
import '../pages/settings_page.dart';

class MyDrawer extends StatelessWidget{
  const MyDrawer({super.key});

  void logout() {
    final authService = AuthService();
    authService.signOut();
  }
```

```
@override
Widget build(BuildContext context){
  return Drawer(
    backgroundColor: Theme.of(context).colorScheme.background,
    child: Column(
      children: [
        // logo
        DrawerHeader(
          child: Center(
            child: Icon(
              Icons.place,
              size: 40,
              color: Theme.of(context).colorScheme.inversePrimary,
              ),
            ),
          ),

        // home title
        Padding(
          padding: const EdgeInsets.only(left: 25.0, top: 25.0),
          child: ListTile(
            title: Text("Г О Л О В Н А"),
            leading: const Icon(Icons.home),
            onTap: () => Navigator.pop(context),
          ),
        ),

        // settings title
        Padding(
          padding: const EdgeInsets.only(left: 25.0, top: 25.0),
          child: ListTile(
            title: Text("Н А Л А Ш Т У В А Н Н Я"),
            leading: const Icon(Icons.settings),
            onTap: () {
              // pop drawer
              Navigator.pop(context);

              // navigate to settings page
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => SettingsPage(),
                  ),
                );
            },
          ),
        ),
        Padding(
            padding: const EdgeInsets.only(left: 25.0, top: 25.0),
            child: ListTile(
```

```
              title: Text("В И Й Т И"),
              leading: const Icon(Icons.home),
              onTap: logout,
            ),
          ),
        ]),
    );
  }
}
```

Код для табів міст

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

class MyTab extends StatelessWidget {
  final String text;
  final IconData? icon;

  const MyTab({super.key, required this.text, this.icon});

  @override
  Widget build(BuildContext context) {
    return Tab(
      height: 50,
      child: Container(
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: Color.fromRGBO(30, 67, 80, 1),
          borderRadius: BorderRadius.circular(10),
        ),
        child: Row(
          children: [
            if (icon != null)
              Icon(icon, color: Colors.black),
            const SizedBox(width: 8),
            Text(
              text,
              style: const TextStyle(
                fontWeight: FontWeight.w800,
                fontSize: 18,
                color: Colors.white,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Код для текстових полів

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

class MyTextField extends StatelessWidget {
  final TextEditingController controller;
  final String hintText;
  final bool obscureText;

  const MyTextField({
    super.key,
    required this.controller,
    required this.hintText,
    required this.obscureText
    });

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 25.0),
      child: TextField(
        controller: controller,
        obscureText: obscureText,
        decoration: InputDecoration(
          enabledBorder: OutlineInputBorder(
            borderSide: BorderSide(color: Theme.of(context).colorScheme.tertiary),
            ),
          focusedBorder: OutlineInputBorder(
            borderSide: BorderSide(color: Theme.of(context).colorScheme.primary),
            ),
          hintText: hintText,
          hintStyle: TextStyle(color: Theme.of(context).colorScheme.primary)
        ),
      ),
    );
  }
}
```

Код для рейтинг бару

```dart
import 'package:flutter/material.dart';
import 'package:japan_guide_app/firebase/firebase_service.dart';

class RatingBar extends StatefulWidget {
  final String sightId;
  final String uid;

  const RatingBar({
    required this.sightId,
    required this.uid,
    Key? key,
  }) : super(key: key);
```

```
 @override
  _RatingBarState createState() => _RatingBarState();
}

class _RatingBarState extends State<RatingBar> {
  double _userRating = 0.0;
  double _averageRating = 0.0;
  bool _isRated = false;

  @override
  void initState() {
    super.initState();
    _fetchUserRating();
    _fetchAverageRating();
  }

  Future<void> _fetchUserRating() async {
    final userRating = await FirebaseService().getUserRating(widget.sightId, widget.uid);
    setState(() {
      _userRating = userRating;
      _isRated = userRating > 0;
    });
  }

  Future<void> _fetchAverageRating() async {
    final averageRating = await FirebaseService().getAverageRating(widget.sightId);
    setState(() {
      _averageRating = averageRating;
    });
  }

  Future<void> _submitRating(double rating) async {
    await FirebaseService().updateSightRating(widget.sightId, widget.uid, rating);
    setState(() {
      _userRating = rating;
      _isRated = true;
    });
    await _fetchAverageRating();
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        _isRated
            ? Text(
                'Your rating: $_userRating',
                style: TextStyle(fontSize: 16),
              )
            : Text(
```

```
            'Rate this sight:',
            style: TextStyle(fontSize: 16),
          ),
      SizedBox(height: 8),
      _isRated
          ? SizedBox.shrink()
          : Slider(
              value: _userRating,
              min: 0,
              max: 5,
              divisions: 5,
              onChanged: (value) {
                setState(() {
                  _userRating = value;
                });
              },
              label: '$_userRating',
            ),
      SizedBox(height: 8),
      ElevatedButton(
        onPressed: () {
          _submitRating(_userRating);
        },
        child: Text(_isRated ? 'Update Rating' : 'Submit Rating'),
      ),
      SizedBox(height: 8),
      Text(
        'Average Rating: $_averageRating',
        style: TextStyle(fontSize: 16),
      ),
    ],
  );
 }
}
```

Код для розгортання тексту

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

class ExpandableText extends StatefulWidget {
 final String text;
 final int trimLines;
 final TextStyle? style;

 const ExpandableText({
  required this.text,
  this.trimLines = 4,
  this.style,
 });

 @override
```

```
  _ExpandableTextState createState() => _ExpandableTextState();
}

class _ExpandableTextState extends State<ExpandableText> {
  bool _readMore = true;

  @override
  Widget build(BuildContext context) {
    final textSpan = TextSpan(
      text: widget.text,
      style: widget.style ?? TextStyle(fontSize: 18),
    );

    final textPainter = TextPainter(
      text: textSpan,
      maxLines: widget.trimLines,
      textDirection: TextDirection.ltr,
    );

    textPainter.layout(maxWidth: MediaQuery.of(context).size.width);

    final textExceeds = textPainter.didExceedMaxLines;

    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Text.rich(
          TextSpan(
            children: [
              if (_readMore && textExceeds)
                TextSpan(
                  text: widget.text.substring(0,
                  textPainter.getPositionForOffset(Offset(
                    textPainter.width, textPainter.height)).offset) + "...",
                )
              else
                TextSpan(
                  text: widget.text,
                ),
            ],
          ),
          style: widget.style,
        ),
        if (textExceeds)
          InkWell(
            onTap: () {
              setState(() {
                _readMore = !_readMore;
              });
            },
```

```
          child: Text(
            _readMore ? 'Дивитися більше' : 'Дивитися менше',
            style: TextStyle(
              fontSize: 14,
              fontWeight: FontWeight.bold,
              color: Color.fromRGBO(203,64,66,1.000),
            ),
          ),
        ),
      ],
    );
  }
}
```

Код для роботи з базою даних

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:japan_guide_app/models/user.dart' as CustomUser;
import '../models/categories.dart';
import '../models/cities.dart';
import '../models/sights.dart';

class FirebaseService {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<void> addUser(CustomUser.User user) async {
    final usersCollection = _firestore.collection('Users');
    await usersCollection.doc(user.uid).set(user.toJson());
  }

  Future<CustomUser.User?> fetchUser(String uid) async {
    print('Fetching user with uid: $uid');
    try {
      final userDoc = await _firestore.collection('Users').doc(uid).get();
      if (userDoc.exists) {
        return CustomUser.User.fromJson(userDoc.data()!);
      } else {
        return null;
      }
    } catch (error) {
      print('Error fetching user by ID: $error');
      return null;
    }
  }

  Future<void> addFavoriteSight(String uid, String sightId) async {
    print('Adding favorite sight for uid: $uid');
    final userRef = _firestore.collection('Users').doc(uid);
    await userRef.update({
      'favoriteSights': FieldValue.arrayUnion([sightId])
    });
```

```
}

Future<void> removeFavoriteSight(String uid, String sightId) async {
  final userRef = _firestore.collection('Users').doc(uid);
  await userRef.update({
    'favoriteSights': FieldValue.arrayRemove([sightId])
  });
}

Future<bool> isFavoriteSight(String uid, String sightId) async {
  print('Checking if sight $sightId is favorite for uid: $uid');
  try {
    final userDoc = await _firestore.collection('Users').doc(uid).get();
    if (userDoc.exists) {
      final user = CustomUser.User.fromJson(userDoc.data()!);
      return user.favoriteSights.contains(sightId);
    } else {
      return false;
    }
  } catch (error) {
    print('Error checking favorite sight: $error');
    return false;
  }
}

Future<List<Sight>> fetchFavoriteSights(String uid) async {
  try {
    final userDoc = _firestore.collection('Users').doc(uid);
    final userSnapshot = await userDoc.get();
    if (!userSnapshot.exists) {
      return [];
    }
    final userData = userSnapshot.data()!;
    final favoriteSightIds = List<String>.from(userData['favoriteSights'] ?? []);

    if (favoriteSightIds.isEmpty) {
      return [];
    }

    final sightsCollection = _firestore.collection('Sights');
    final querySnapshot = await sightsCollection.where(FieldPath.documentId, whereIn: favoriteSightIds).get();
    final sights = querySnapshot.docs.map((doc) => Sight.fromJson(doc.data())).toList();
    return sights;
  } catch (error) {
    print('Error fetching favorite sights: $error');
    return [];
  }
}

Future<double> getUserRating(String sightId, String userId) async {
```

```
  final ratingRef = _firestore
      .collection('Ratings')
      .doc(sightId)
      .collection('UserRatings')
      .doc(userId);

  final ratingDoc = await ratingRef.get();
  if (ratingDoc.exists) {
    return ratingDoc.data()!['rating'] as double;
  } else {
    return 0.0;
  }
}

Future<double> getAverageRating(String sightId) async {
  final sightRef = _firestore.collection('Sights').doc(sightId);
  final sightDoc = await sightRef.get();
  if (sightDoc.exists) {
    return sightDoc.data()!['averageRating'] as double;
  } else {
    return 0.0;
  }
}

Future<void> updateSightRating(String sightId, String userId, double rating) async {
  final ratingRef = _firestore
      .collection('Ratings')
      .doc(sightId)
      .collection('UserRatings')
      .doc(userId);

  await ratingRef.set({'rating': rating});
  await _updateAverageRating(sightId);
}

Future<void> _updateAverageRating(String sightId) async {
  final ratingsSnapshot = await _firestore
      .collection('Ratings')
      .doc(sightId)
      .collection('UserRatings')
      .get();

  final ratings = ratingsSnapshot.docs
      .map((doc) => doc['rating'] as double)
      .toList();

  final averageRating = ratings.isNotEmpty
      ? ratings.reduce((a, b) => a + b) / ratings.length
      : 0;
```

```
  final sightRef = _firestore.collection('Sights').doc(sightId);
  await sightRef.update({'averageRating': averageRating});
 }

 Future<List<Sight>> fetchPopularSights() async {
  try {
   final popularSightsSnapshot = await _firestore
       .collection('Sights')
       // .where('cityId', isEqualTo: cityId)
       .where('averageRating', isGreaterThan: 4.0)
       .get();

   final popularSights = popularSightsSnapshot.docs
       .map((doc) => Sight.fromJson(doc.data()))
       .toList();

   return popularSights;
  } catch (error) {
   print('Error fetching popular sights: $error');
   return [];
  }
 }

 Future<void> addSight(Sight sight) async {
  final sightsCollection = _firestore.collection('Sights');
  final docRef = sightsCollection.doc();
  await docRef.set(sight.toJson());
 }

 Future<List<Sight>> fetchSights() async {
  try {
   final sightsCollection = _firestore.collection('Sights');
   final querySnapshot = await sightsCollection.get();
   final sights = querySnapshot.docs.map((doc) => Sight.fromJson(doc.data())).toList();
   return sights;
  } catch (error) {
   print('Error fetching sights: $error');
   return [];
  }
 }

 Future<List<Categories>> fetchCategory() async {
  try {
   final categoryCollection = _firestore.collection('Category');
   final querySnapshot = await categoryCollection.get();
   final category = querySnapshot.docs.map((doc) => Categories.fromJson(doc.data())).toList();
   return category;
  } catch (error) {
   print('Error fetching category: $error');
   return [];
```

```
  }
 }

  Future<List<Sight>> fetchSightbyCategory(String category, cityId) async {
  final sightsRef = _firestore.collection('Sights').where('category', isEqualTo: category).where('cityId', isEqualTo:
cityId);
 final querySnapshot = await sightsRef.get();
 final sights = <Sight>[];
 for (final doc in querySnapshot.docs) {
  final sightData = doc.data();
  final name = sightData['name'] as String;
  final category = sightData['category'] as String;
  final cityId = sightData['cityId'] as String;
  final description = sightData['description'] as String;
  final imagePath = sightData['imagePath'] as String;
  final city = sightData['city'] as String;
  final sightId = sightData['sightId'] as String;
  final address = sightData['address'] as String;
  sights.add(Sight(name: name, category: category, cityId: cityId, description: description, imagePath: imagePath, city:
city, sightId: sightId, address: address));
 }
 return sights;
  }

 Future<Sight?> fetchSightById(String sightId) async {
  print('Fetching sight with ID: $sightId');
    try {
      final sightsCollection = _firestore.collection('Sights').doc(sightId);
      final docSnapshot = await sightsCollection.get();
      if (docSnapshot.exists) {
        return Sight.fromJson(docSnapshot.data()!);
      } else {
        return null;
      }
    } catch (error) {
      print('Error fetching sight by ID: $error');
      return null;
    }
  }

 Future<List<City>> fetchCities() async {
  try {
    final citiesCollection = _firestore.collection('Cities');
    final querySnapshot = await citiesCollection.get();
    final cities = querySnapshot.docs.map((doc) => City.fromJson(doc.data())).toList();
    return cities;
  } catch (error) {
    print('Error fetching cities: $error');
    return [];
  }
```

```
  }

  Future<List<Sight>> fetchSightsForCity(String cityId) async {
    final sightsRef = _firestore.collection('Sights').where('cityId', isEqualTo: cityId);
    final querySnapshot = await sightsRef.get();
    final sights = <Sight>[];
    for (final doc in querySnapshot.docs) {
    final sightData = doc.data();
    final name = sightData['name'] as String;
    final category = sightData['category'] as String;
    final cityId = sightData['cityId'] as String;
    final description = sightData['description'] as String;
    final imagePath = sightData['imagePath'] as String;
    final city = sightData['city'] as String;
    final sightId = sightData['sightId'] as String;
    final address = sightData['address'] as String;
   sights.add(Sight(name: name, category: category, cityId: cityId, description: description, imagePath: imagePath, city:
city, sightId: sightId, address: address));
  }
    return sights;
  }

}
```

Код для відображення сторінки категорій

```
import 'package:flutter/material.dart';
import 'package:japan_guide_app/firebase/firebase_service.dart';
import 'package:japan_guide_app/models/categories.dart';
import 'package:japan_guide_app/models/sights.dart';
import 'package:japan_guide_app/models/user.dart';
import '../components/cities_tile.dart';
import '../models/cities.dart';
import 'sight_page.dart';

class CategoryPage extends StatefulWidget {
  final Categories category;
  final City city;
  final User user;

  CategoryPage({
    required this.category,
    required this.city,
    required this.user,
  });

  @override
  _CategoryPageState createState() => _CategoryPageState();
}
```

```
class _CategoryPageState extends State<CategoryPage> {
 late Future<List<Sight>> _sights;

 @override
 void initState() {
  super.initState();
  _sights = FirebaseService().fetchSightbyCategory(widget.category.name, widget.city.cityId);
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text(
     widget.category.name,
     style: TextStyle(fontFamily: "JapanFont"),
    ),
   ),
   body: FutureBuilder<List<Sight>>(
    future: _sights,
    builder: (context, snapshot) {
     if (snapshot.connectionState == ConnectionState.waiting) {
      return Center(child: CircularProgressIndicator());
     } else if (snapshot.hasError) {
      return Center(child: Text('Error: ${snapshot.error}'));
     } else {
      final sights = snapshot.data!;
      return GridView.builder(
       gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 2,
        crossAxisSpacing: 8.0,
        mainAxisSpacing: 8.0,
        childAspectRatio: 0.75,
       ),
       shrinkWrap: true,
       physics: NeverScrollableScrollPhysics(),
       itemCount: sights.length,
       itemBuilder: (context, index) {
        final sight = sights[index];
        return CityTile(
         sight: sight,
         user: widget.user,
         uid: widget.user.uid,
         onTap: () {
          Navigator.push(
           context,
           MaterialPageRoute(
            builder: (context) => SightPage(
             sightId: sight.sightId,
            ),
```

```
                  ),
                );
              },
            );
          },
        );
      }
    },
  ),
 );
 }
}
```

Код для відображення сторінки вподобаних

```
import 'package:flutter/material.dart';
import 'package:japan_guide_app/firebase/firebase_service.dart';
import 'package:japan_guide_app/models/sights.dart';
import 'package:japan_guide_app/models/user.dart';
import '../components/cities_tile.dart';
import 'sight_page.dart';

class FavoritesPage extends StatefulWidget {
 final User user;

 const FavoritesPage({required this.user});

 @override
 _FavoritesPageState createState() => _FavoritesPageState();
}

class _FavoritesPageState extends State<FavoritesPage> {
 late Future<List<Sight>> _favoriteSights;

 @override
 void initState() {
  super.initState();
  _favoriteSights = FirebaseService().fetchFavoriteSights(widget.user.uid);
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Вподобані'),
   ),
   body: FutureBuilder<List<Sight>>(
    future: _favoriteSights,
    builder: (context, snapshot) {
     if (snapshot.connectionState == ConnectionState.waiting) {
      return Center(child: CircularProgressIndicator());
```

```
        } else if (snapshot.hasError) {
          return Center(child: Text('Error: ${snapshot.error}'));
        } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
          return Center(child: Text('Вподобані місця не знайдені'));
        } else {
          final favoriteSights = snapshot.data!;
          return GridView.builder(
            gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
              crossAxisCount: 2,
              crossAxisSpacing: 8.0,
              mainAxisSpacing: 8.0,
              childAspectRatio: 0.75,
            ),
            itemCount: favoriteSights.length,
            itemBuilder: (context, index) {
              final sight = favoriteSights[index];
              return GestureDetector(
                onTap: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => SightPage(
                        sightId: sight.sightId,
                      ),
                    ),
                  );
                },
                child: CityTile(
                      sight: sight,
                      onTap: null,
                      user: widget.user,
                      uid: widget.user.uid,
                )
              );
            },
          );
        }
      },
    ),
  );
 }
}
```

Код для відображення головної сторінки

```
import 'package:firebase_auth/firebase_auth.dart' as FirebaseAuth;
import 'package:flutter/material.dart';
import 'package:japan_guide_app/components/my_drawer.dart';
import 'package:japan_guide_app/firebase/firebase_service.dart';
import 'package:japan_guide_app/models/cities.dart';
import 'package:japan_guide_app/models/user.dart' as LocalUser;
```

```dart
import 'package:japan_guide_app/pages/addition_page.dart';
import 'package:japan_guide_app/pages/favorite_page.dart';
import 'package:japan_guide_app/pages/map_pages.dart';
import '../models/categories.dart';
import '../tabs/cities_tab.dart';
import '../components/my_tab.dart';

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

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  final FirebaseService _firebaseService = FirebaseService();
  late Future<List<City>>? _cities;
  late Future<List<Categories>>? _category;
  late Future<LocalUser.User?> _user;
  String? uid;
  int _selectedIndex = 0;

  @override
  void initState() {
    super.initState();
    _loadData();
  }

  Future<void> _loadData() async {
    _cities = _firebaseService.fetchCities();
    _category = _firebaseService.fetchCategory();
    final currentUser = FirebaseAuth.FirebaseAuth.instance.currentUser;
    if (currentUser != null) {
      setState(() {
        uid = currentUser.uid;
        _user = _firebaseService.fetchUser(uid!);
      });
    }
  }

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
      title: Text(
        "Я П О Н І Я",
        style: TextStyle(fontFamily: "JapanFont"),
      ),
    ),
    drawer: const MyDrawer(),
    body: _getBody(_selectedIndex),
    bottomNavigationBar: BottomNavigationBar(
      items: const <BottomNavigationBarItem>[
        BottomNavigationBarItem(
          backgroundColor: Color.fromRGBO(203,64,66,1.000),
          icon: Icon(Icons.home),
          label: 'Головна',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.favorite),
          label: 'Вподобані',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.info_outlined),
          label: 'Додатково',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.map),
          label: 'Мапа',
        ),
      ],
      currentIndex: _selectedIndex,
      onTap: _onItemTapped,
    ),
  );
}

Widget _getBody(int index) {
  switch (index) {
    case 0:
      return _buildHome();
    case 1:
      return FutureBuilder<LocalUser.User?>(
        future: _user,
        builder: (context, userSnapshot) {
          if (userSnapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (userSnapshot.hasError) {
            return Center(child: Text('Error fetching user data'));
          } else if (!userSnapshot.hasData) {
            return Center(child: Text('Please log in'));
          } else {
            final user = userSnapshot.data!;
            return FavoritesPage(user: user);
```

```
        }
       },
      );
    case 2:
     return const AdditionPage();
    case 3:
     return const MapPage();
    default:
     return Container();
   }
}

Widget _buildHome() {
  return FutureBuilder<LocalUser.User?>(
    future: _user,
    builder: (context, userSnapshot) {
      if (userSnapshot.connectionState == ConnectionState.waiting) {
        return Center(child: CircularProgressIndicator());
      } else if (userSnapshot.hasError) {
        print(userSnapshot.error);
        return Center(child: Text('Error fetching user data'));
      } else if (!userSnapshot.hasData) {
        return Center(child: Text('Please log in'));
      } else {
        final user = userSnapshot.data!;
        return FutureBuilder<List<City>>(
          future: _cities,
          builder: (context, citySnapshot) {
            if (citySnapshot.connectionState == ConnectionState.waiting) {
              return Center(child: CircularProgressIndicator());
            } else if (citySnapshot.hasError) {
              print(citySnapshot.error);
              return Center(child: Text('Error fetching cities'));
            } else {
              final cities = citySnapshot.data!;
              return FutureBuilder<List<Categories>>(
                future: _category,
                builder: (context, categorySnapshot) {
                  if (categorySnapshot.connectionState == ConnectionState.waiting) {
                    return Center(child: CircularProgressIndicator());
                  } else if (categorySnapshot.hasError) {
                    print(categorySnapshot.error);
                    return Center(child: Text('Error fetching categories'));
                  } else {
                    final category = categorySnapshot.data!;
                    return DefaultTabController(
                      length: cities.length,
                      child: Column(
                        children: [
                          const SizedBox(height: 10,),
```

```
                    TabBar(
                      isScrollable: true,
                      tabs: cities.map((city) => MyTab(text: city.name)).toList(),
                    ),
                    Expanded(
                      child: TabBarView(
                        children: cities.map((city) {
                          if (uid != null) {
                            return CityTab(city: city, category: category[0], user: user, uid: uid!);
                          } else {
                            return Center(child: Text('User ID not available'));
                          }
                        }).toList(),
                      ),
                    ),
                  ],
                ),
              );
            }
          },
        );
      }
    },
  );
 }
}
```

Код для відображення сторінки вітання

```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:japan_guide_app/auth/auth_gate.dart';
import 'package:japan_guide_app/components/button.dart';

class IntroPage extends StatelessWidget {
 const IntroPage({super.key});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    body:  Container(
      //Налаштування фотографії для фону
      decoration: const BoxDecoration(
        image: DecorationImage(
          image: AssetImage("lib/images/intro.jpg"),
          fit: BoxFit.cover,
          alignment: Alignment(0.1 , 0.0),
        ),
```

```
      ),
      padding: const EdgeInsets.all(25.0),
      child:  Column(
       //Всі наші об'єкти починаються зліва та знизу
       crossAxisAlignment: CrossAxisAlignment.start,
       mainAxisAlignment: MainAxisAlignment.end,
       children: [
        //Основний текст
        Text(
        "Подорожуй легко!",
        style: TextStyle(
          fontSize: 44,
          color: Colors.white
          )
        ),

        //Відступ
        const SizedBox(height: 20),

        //другорядний текст
        Text(
        "Насолджуйтеся кращими видами Японії розом з нами",
        style: GoogleFonts.raleway(
          fontSize: 18,
          color: Colors.white
        )
        ),

        //Відступ
        const SizedBox(height: 20),

        //Кнопка
        MyButton(
          text: "Почати",
          onTap: () {
            Navigator.push(context, MaterialPageRoute(
              builder: (context) => AuthGate()));
          },
          ),

        const SizedBox(height: 60),

       ],
      ),
     ),
   );
  }
}
```

Код для відображення сторінки логіну

```
import 'package:flutter/material.dart';
import '../auth/auth_service.dart';
import '../components/button.dart';
import '../components/my_textfield.dart';

class LoginPage extends StatefulWidget {
  final void Function()? onTap;

  const LoginPage({
    super.key,
    required this.onTap
    });

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();

  void login() async {

    final _authService = AuthService();

    try{
      await _authService.signIn(emailController.text, passwordController.text);
    }
    catch (error){
      showDialog(
          context: context,
          builder: (context) => AlertDialog(
            title: Text(error.toString()),));
    }

  }

  @override
  Widget build(BuildContext context){
    return Scaffold(
      backgroundColor: Colors.white,
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          //Логотип
          Icon(
            Icons.lock_open_rounded,
            size: 72,
          ),
```

```
const SizedBox(height: 10,),

//Повідомлення , слоган
Text("Sign in to your account",
  style: TextStyle(
    color: Theme.of(context).colorScheme.inversePrimary),
),

const SizedBox(height: 10,),

//Пошта
MyTextField(
  controller: emailController,
  hintText: "Email",
  obscureText: false,
),

const SizedBox(height: 10,),

//Пароль
MyTextField(
  controller: passwordController,
  hintText: "Password",
  obscureText: true,
),

const SizedBox(height: 10,),

//Кнопка
MyButton(
    text: "Увійти",
    onTap: login,
    ),

const SizedBox(height: 10,),

//Регістрація
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text("Not a member?",
    style: TextStyle(
      color: Theme.of(context).colorScheme.inversePrimary),
      ),
    const SizedBox(width: 4,),
    GestureDetector(
      onTap: widget.onTap,
      child: Text("Register now",
      style: TextStyle(
        color: Theme.of(context).colorScheme.inversePrimary,
```

```
                fontWeight: FontWeight.bold
                ),
              ),
            ),
          ],
        )
      ],
    ),
  );
 }
}
```

Код для відображення сторінки мапи

```
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:flutter/material.dart';
import 'package:japan_guide_app/firebase/firebase_service.dart';
import 'package:japan_guide_app/models/sights.dart';
import 'sight_page.dart';

class MapPage extends StatefulWidget {
 final LatLng? targetLocation;

 const MapPage({Key? key, this.targetLocation}) : super(key: key);

 @override
 State<MapPage> createState() => _MapPageState();
}

class _MapPageState extends State<MapPage> {
 late GoogleMapController mapController;
 final LatLng _center = const LatLng(35.6895, 139.6917); // Координати Токіо
 Set<Marker> _markers = {};
 final FirebaseService _firebaseService = FirebaseService();

 @override
 void initState() {
  super.initState();
  _loadSights();
  WidgetsBinding.instance.addPostFrameCallback((_) {
   if (widget.targetLocation != null) {
    _moveToLocation(widget.targetLocation!);
   }
  });
 }

 Future<void> _loadSights() async {
  try {
   List<Sight> sights = await _firebaseService.fetchSights();
   setState(() {
    _markers = sights.where((sight) => sight.location != null).map((sight) {
```

```
        return Marker(
          markerId: MarkerId(sight.sightId),
          position: LatLng(sight.location!.latitude, sight.location!.longitude),
          infoWindow: InfoWindow(
            title: sight.name,
            snippet: "Натисніть щоб дізнатися більше про місце",
            onTap: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => SightPage(
                      sightId: sight.sightId,
                    ),
                  ),
                );
            }
          ),
        );
      }).toSet();
    });
  } catch (e) {
    print('Error fetching sights: $e');
  }
}

void _moveToLocation(LatLng location) {
  mapController.animateCamera(CameraUpdate.newLatLng(location));
}


void _onMapCreated(GoogleMapController controller) {
  mapController = controller;
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: GoogleMap(
      onMapCreated: _onMapCreated,
      initialCameraPosition: CameraPosition(
        target: widget.targetLocation ?? _center,
        zoom: 15.0,
      ),
      markers: _markers,
    ),
  );
 }
}
```

Код для відображення сторінки місця

```dart
import 'package:flutter/material.dart';
import 'package:japan_guide_app/components/button.dart';
import 'package:japan_guide_app/models/sights.dart';
import '../components/readmore.dart';
import '../firebase/firebase_service.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:flutter_rating_bar/flutter_rating_bar.dart';
import 'map_pages.dart';
import 'package:firebase_auth/firebase_auth.dart';

class SightPage extends StatefulWidget {
  final String sightId;

  const SightPage({required this.sightId});

  @override
  State<SightPage> createState() => _SightPageState();
}

class _SightPageState extends State<SightPage> {
  final firebaseService = FirebaseService();
  Sight? _sight;
  double _rating = 0;
  final user = FirebaseAuth.instance.currentUser;

  @override
  void initState() {
    super.initState();
    fetchSightDetails(widget.sightId);
    if (user != null) {
      fetchUserRating(widget.sightId, user!.uid);
    }
  }

  Future<void> fetchSightDetails(String sightId) async {
    try {
      final sight = await firebaseService.fetchSightById(sightId);
      setState(() {
        _sight = sight;
      });
    } catch (error) {
      print('Error fetching sight: $error');
    }
  }

  Future<void> fetchUserRating(String sightId, String userId) async {
    try {
      final rating = await firebaseService.getUserRating(sightId, userId);
      setState(() {
        _rating = rating;
```

```
    });
  } catch (error) {
    print('Error fetching user rating: $error');
  }
}

Future<void> updateRating(double rating) async {
  if (user != null) {
    try {
      await firebaseService.updateSightRating(widget.sightId, user!.uid, rating);
      setState(() {
        _rating = rating;
      });
    } catch (error) {
      print('Error updating rating: $error');
    }
  } else {
    print('User not logged in');
  }
}

@override
Widget build(BuildContext context) {
  if (_sight == null) {
    return const Center(child: CircularProgressIndicator());
  }
  return Scaffold(
    appBar: AppBar(),
    body: SingleChildScrollView(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          ClipRRect(
            borderRadius: BorderRadius.vertical(bottom: Radius.circular(15)),
            child: Image.network(_sight!.imagePath),
          ),
          const SizedBox(height: 10,),
          Container(
            margin: const EdgeInsets.symmetric(horizontal: 10.0),
            child: Text(_sight!.name, style: TextStyle(
              fontWeight: FontWeight.bold,
              fontFamily: "JapanFont",
              fontSize: 28,),
            ),
          ),
          const SizedBox(height: 10,),
          Row(
            children: [
              const SizedBox(width: 10,),
              Icon(Icons.location_city_outlined),
```

```
      Text(_sight!.city, style: const TextStyle(fontSize: 16)),
      const SizedBox(width: 10,),
      Icon(Icons.place_outlined),
      Text(_sight!.address, style: const TextStyle(fontSize: 16)),
      const SizedBox(width: 10,),
      Icon(Icons.access_time),
      Text(_sight!.worktime!, style: const TextStyle(fontSize: 16)),
    ],
  ),
  const SizedBox(height: 10,),
  Container(
    margin: const EdgeInsets.symmetric(horizontal: 10.0),
    child: RatingBar.builder(
      itemSize: 25,
      initialRating: _rating,
      minRating: 0,
      direction: Axis.horizontal,
      allowHalfRating: true,
      itemCount: 5,
      itemPadding: const EdgeInsets.symmetric(horizontal: 2.0),
      itemBuilder: (context, _) => const Icon(
        Icons.star,
        color: Colors.amber,
      ),
      onRatingUpdate: (rating) {
        updateRating(rating);
      },
    ),
  ),

  const SizedBox(height: 10,),
  Container(
    margin: const EdgeInsets.symmetric(horizontal: 10.0),
    child: Text("Опис", style: TextStyle(fontWeight: FontWeight.bold, fontSize: 20,)),
  ),
  const SizedBox(height: 10,),
  Container(
    margin: const EdgeInsets.symmetric(horizontal: 10.0),
    child: ExpandableText(
      text: _sight!.description,
      trimLines: 12,
      style: TextStyle(fontSize: 18),
    ),
  ),
  const SizedBox(height: 20,),
  MyButton(
    text: "Знайти на мапі",
    onTap: () {
      Navigator.push(
        context,
```

```
                MaterialPageRoute(
                  builder: (context) => Scaffold(
                    appBar: AppBar(
                      title: Text('Повернутися'),
                    ),
                    body: MapPage(
                      targetLocation: _sight!.location != null
                          ? LatLng(
                              _sight!.location!.latitude,
                              _sight!.location!.longitude,
                            )
                          : null,
                    ),
                  ),
                ),
              );
            },
          ),
          const SizedBox(height: 20,),
        ],
      ),
    ),
  );
 }
}
```