СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) Інформаційних технологій та електроніки

(повне найменування інституту, факультету)

Кафедра Програмування та математики

(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр

(бакалавр, магістр)

Спеціальність ІПЗ-17д Інженерія програмного забезпечення

(шифр і назва спеціальності)

спеціалізація _____

(назва спеціалізації)

на тему Розробка мобільного застосунка для пошуку спортивних майданчиків та спортивних залів

_____

Виконав: студент групи ІПЗ-17д _____ А.С. Богомолов

(підпис) (ініціали і прізвище)

Керівник _____ В.О. Лифар

(підпис) (ініціали і прізвище)

Завідувач кафедри _____ В.О. Лифар

(підпис) (ініціали і прізвище)

Рецензент _____ О.І. Захожай

(підпис) (ініціали і прізвище)

Сєвєродонецьк - 2021

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ дипломної роботи студента гр. ІПЗ-17д Богомолов А.С.

Науковий керівник

проф., д.т.н. _____ доц. Лифар В.О.

Оцінка наукового керівника: _____

Рецездент Захожай О.І. каф. ПМ (м. Сєвєродонецьк), проф. д.т.н. доцент

ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту: _____

_____

Голова ЕК,

проф. Кафедри ПМ

к.т.н., доцент _____Захожай О.І.

підпис

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет інформаційних технологій та електроніки

Кафедра програмування та математики

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 121 „Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМ,
д.т.н., доцент
_____ Лифар В.О.
«___» _____ 2021 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТА
Богомолова Андрія Сергійовича

1. Тема роботи Розробка мобільного застосунку для пошуку спортивних майданчиків та залів.

керівник роботи д.т.н., доц. Лифар Володимир Олексійович

затверджені наказом вищого навчального закладу від «12» квітня 2021 року № 68/15.16

2. Строк подання студентом роботи 11 червня 2021 р.

3. Вихідні дані до роботи

Об'єктом досліджень є розробка мобільного застосунку.

3.1 Літературні джерела

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

4.1 Вступ

4.2 Аналіз предметної галузі (огляд літератури), з висвітленням наступних питань:

Значення мобільних застосунків у сучасному житті.

Аналіз існуючих платформ.

Особливості застосунків для пошуку майданчивів.

4.3 Основна частина, в якої висвітлити:

Особливості технологій використаних при розробці

Моделювання взаємодії користувача із застосунком

Розробка застосунку.

4.4 Висновки

4.5 Перелік використаних джерел

5. Перелік графічного матеріалу немає

6. Дата видачі завдання 12 квітня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Одержання завдання на виконання роботи	05.03.2021	
2	Укладання і погодження з керівником плану і етапів виконання роботи	15.03.2021	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз використаних ресурсів»	01.04.2021	
4	Аналіз шляхів виконання завдання.Вибір і погодження з керівником оптимального шляху	12.04.2021	
3	Розробка фронтенд частини Застосунку	05.05.2021	
5	Розробка бек-енду застосунку	20.05.2021	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	02.06.2021	
7	Здача готової пояснювальної записки на кафедру	11.06.2021	
8	Укладання доповіді і презентації	15.06.2021	

Студент _____ Богомолов А.С.
(підпис)

Керівник роботи _____ Лифар В.О.
(підпис)

РЕФЕРАТ

Робота містить: 56 сторінки основного тексту, 16 сторінок додатків, 74 рисунків, 1 таблицю, 15 використаних джерел.

Об'єктом дослідження є можливість розробки і створення Андроїд застосунку.

Метою дипломної роботи є розробка застувунку «SportSearch»

Дипломная работа складається з трьох основних частин.

У першій частині розглядаються сучасні технології, що використовуються при розробці Android застосунків та порівнюються з аналогами. Порівнюються процес розробки застосунків під різні платформи. Порівнюються аналоги розробляємого застосунку.

У цій статті розглядається технології, які буде потрібно використовувати для розробки застосунку.

Друга частина показує модель проекту та структурузастосунку. А також описує механізми роботи використовуємих у розробці застосунку.

У третій частині статті описується практична частина створення застосунку, а саме створення самрсисних та використання встроєних віджетів для відображення інформації, створення функіональної частини застосунку.

Система впроваджена відповідно до всього технічного персоналу.

Результатом цієї роботи єрозроблений застосунок для пошуку спортивних майданчиків та залів «SportSearch».

Зміст

ВСТУП

Актуальність дослідження

У теплі періоди більшість спортсменів тренуються на свіжому повітрі. Для заняття баскетболом волейболом та іншими видами активного спорту, спортсменам необхідно знати про доступні майданчики, заходи проводяться на них, кількість осіб, що буде тренуватися. Навпаки, в холодні періоди спортсмени бажають продовжувати тренування, але через холод проводити тренування на вулиці не завжди можливо. Тому вони шукають доступні зали, в яких можна тренуватися. Як і в теплі періоди їм потрібно знати, які зали доступні, заходи проводяться в них і кількість осіб, яке буде тренуватися.

Об'єкт дослідження Об'єктом данної бакалаврської кваліфікаційної роботи є Android застосунок.

Предмет дослідження Дослідження спрямованні на розробку Android застосунку для пошуку спортивних майданчиків та залів.

Мета дослідження Метою даної роботи є створення Android застосунку для пошуку вуличних майданчиків та залів учбових закладів для занять спортом.

Завдання дослідження Потрібно з'ясувати, які потреби можуть бути у кінцевого користувача. Проаналізувати необхідність у впровадженні даних потреб користувача. Визначити які технології будуть задовільними при розробці застосунку. Розробити застосунок відповідно до отриманих результатів дослідження. Оцінити результат виконаної роботи та сформувати висновок.

Методи дослідження

У даній роботі використовують такі методи дослідження, як:

1. Аналіз
2. Аналогія
3. Моделювання
4. Прогнозування

Наукова новизна дослідження Наукова новизна результатів дослідження полягає у тому, що серед конкурентів у даній сфері Android застосунків, не існує

аналогів, які допомагають вирішити проблеми з пошуком місця для занять активними видами спорту як у теплі так і холодні пори року.

Практичне значення отриманих результатів Практичне значення отриманих результатів полягає у розробці мобільного застосунку «SportSearch», який допоможе спортсменам знаходити місця для тренувань у будь-яку пору року.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Значення мобільних застосунків у сучасному житті

У сучасному житті немає нікого хто не може обійтися без мобільного телефону. Та справа вже не в дзвінках і смс. Наше життя стало неможливим без мобільних застосунків. Мобільні застосунки – одна з найпопулярніших праць для програмістів. Кожен день програмісти створюють нові проекти, які допомагають нам у житті.

Всі мобільні застосунки створені таким чином, щоб спростити життя. Кожен може помітити, що з появою месенджерів, звичайні смс відійшли в минуле. Конкуренція таких продуктів дозволяє вибирати тільки найякісніші застосунки, які завоювали увагу всього світу.

1.2 Аналіз існуючийх платформ

Застосунок або сайт?

Смартфони – це перспективна, ще не заповнена конкурентами ніша. Вони дають гарний шанс зайняти своє місце на ринку і заявити про себе на весь світ.

Сайтами користуються усі, ними не здивуєш ні користувачів, а ні колег. В той же час, навіть технічно прості застосунки привертають увагу і мають величезний потенціал для розвитку. Мобільні застосунки – це передусім, річ статусна, що говорить про те, що ви серйозно і надовго вліняетесь у розробку і не відстаєте від часу.

Переваги мобільних застосунків перед веб-застосунками

- застосунок завантажується швидше, ніж тяжкий, наповнений інформацією сайт;
- застосунок працює, навіть якщо смартфон не в мережі;
- застосунок працює краще, ніж старий сайт;
- зручно читати та перегортати сторінки;

- корисна функція пуш-повідомлень, не дозволить користувачам забути про вас.

Недоліки мобільних застосунків

- Сумісність. Забезпечення належного функціонування нативного застосунки залежить від вимог конкретної операційної системи. Це значить, що для кожної платформи (iOS, Android) потрібна окрема робоча версія програми.
- Підтримка, обслуговування. Коли застосунок розробляється для кількох різних платформ, його підтримка вимагає більше часу і грошей. Необхідно регулярно надавати оновлення, виправляти проблеми сумісності з кожним типом пристроїв. Крім того, потрібно завжди нагадувати користувачам про необхідність установки нових оновлень.

Порівняння Android та iOS платформ

Мобільні застосунки реалізуються здебільшого на платформах, як: Android та iOS.

Частка iOS на ринку мобільних операційних систем становить 24,99%, а Android - 74,43%. При цьому протягом перших трьох кварталів 2020 року видатки користувачів iOS в Apple App Store вдвічі перевищують витрати користувачів Android в Google Play Store.

Рисунок 1.1 – Порівняння використання Android та iOS у 2017 та 2021 роках

Особливості Android платформи

Існує кілька причин, чому розробка застосунків на Android зручна для новачка:

Різноманітність пристроїв. На Android зараз працюють не тільки смартфони та планшети, але й телевізори, холодильники, автомобільні аудіосистеми, причому від різних брендів. Відповідно, і спектр застосунків може бути широкий.

Для роботи підійде будь-який комп'ютер. Неважливо, на основі Windows, Linux або Mac. Android Studio і програми з SDK доступні для всіх платформ.

Ком'юніті. За моїми відчуттями, співтовариство розробників під Android трохи ширше, ніж у iOS. Для новачка це вагомий плюс: простіше знайти допомогу на форумах, отримати інформацію з перших вуст, поспілкуватися в чатах.

Можливість адаптувати платформу під спеціалізовану задачу без втрати підтримки. Якщо розробляти застосування під якусь спеціалізовану задачу, то Android підходить для цього краще. Наприклад, недавно з введенням ОФД з'явилася задача адаптації касових апаратів - терміналів для проведення оплати. Компанія «Евотор» для цього взяла платформу Android і створила застосування

для роботи. Для цього вони відмовилися від підтримки Google, прибрали Market. Тобто повністю кастомизировать платформу. І навіть якщо Google піде з Росії, то ці касові апарати працюватимуть. З iOS такого б не вийшло.

Відмінне середовище розробки. Середовище розробки Android Studio заснована на кращій в світі середовищі - Intellij Idea. У цьому середовищі є розумний аналіз коду, зручна навігація, автоматичне форматування коду і багато іншого.

Android - це OpenSource-система. Ви прямо в редакторі коду можете подивитися, як зроблені стандартні елементи, наприклад, кнопки / списки / перемикачі, і писати свій код на основі існуючого. В iOS є великі готові нерозбірні блоки (готову шафу), і якщо раптом вам такий блок не підійшов (хочете інший колір, інший розмір) доведеться викинути його і будувати свій (йти в ліс, рубати дерево, стругати дошки, фарбувати ...).

Але у всього є зворотна сторона медалі. На жаль, деякі плюси Android-розробки є її ж мінусами:

Різноманіття Android-девайсів заважає створити ідеально працюючий продукт. Система коштує на смартфонах, годинах, магнітолах, холодильниках і планшетах абсолютно різних брендів, кожен з яких в умовах конкуренції намагається внести свою «родзинку». В результаті пристрою не працюють однаково, а розробники чисто фізично не можуть знати всі параметри кожного гаджета. Якщо ви робите застосунок-відеореєстратор, будьте готові, що не всі спрацює за планом: система в одній машині може не побачити камеру, в іншій - перевернути зображення, а в третій будуть проблеми з відеозаписом.

Не всі матеріали можуть бути актуальні. Операційна система Android існує вже дуже давно, і багато нюансів самої розробки встигли кілька разів помінятися.

Особливості iOS платформи

Перейдемо до розробки під iOS - це цікава платформа, яка не стоїть на місці. Її основні плюси:

Визначеність у девайсах. Розробник завжди знає, які пристрої працюють, які версії оновлюються, а які скоро вийдуть з експлуатації. Наприклад, зараз на вітрині iPhone 7-8-10-11, а iPhone 6 вже не застарів. Параметри екранів і систем відомі заздалегідь і перевірити, як виглядає застосування на всіх девайсах, цілком реально.

Різноманітність підходів до розробки. Кожен пристрій лінійки Apple - iPhone, iPad, iWatch, Apple TV, Mac OS - вимагає свого підходу, в зв'язку з чим професійний розвиток розробника стає дуже цікавим.

Прозорість оновлень. Щорічно Apple вдосконалює гайдлайни по розробці застосунків під iOS, але не залишає розробників в невіданні: пояснює, як замінити поточні елементи коду на більш продуктивні. Крім того, ком'юніті iOS, хоча поки і нечисленне, але дуже активне і завжди готове допомогти.

Якщо говорити про мінуси, то найістотніший - високий поріг входу в професію. Розробнику обов'язково потрібно мати техніку Apple, а це мінімум 100 тисяч рублів з гаманця при покупці нового пристрою. Звичайно, можна заощадити в два-три рази, купивши моделі минулих серій, але тоді ваша техніка повинна бути не старше трьох років.

Крім того, щоб публікувати застосунки в AppStore, необхідно щорічно продовжувати доступ до аккаунту розробника за $ 99.

1.3 Flutter

Рисунок 1.2 – Лого Flutter

Flutter - безкоштовний і відкритий фреймвор розробки мобільних користувальницьких інтерфейсыв, створений компанією Google і випущений в травні 2017 року.

Flutter - юнна, але все дуже перспективна платформа, яка привернула до себе увагу великих компаній, які створили свої застосунки. Цікавою цю платформу робить простота порівнянно з розробкою сайтів, і швидкістю роботи на рівні з нативними додатками.

За допомогою Flutter можливо створити власний мобільний застосунок з одним набором коду. Це означає, що для створення застосунків, як для IOS так і для Android, можна використовувати одну мову.

Flutter націлений на дві важливі речі:

SDK (Software Development Kit): набір інструментів, який допомагає у розробці застосунків. Також він містить можливості для компіляції коду в нативному машинному коді.

Framework: Колекція функціональних елементів призначеного для користувача інтерфейсу, які можна персоналізувати під особисті цілі.

Для розробки з Flutter використовується мова програмування під назвою Dart. Це також мова Google, створений в жовтні 2011 року, але значно поліпшена в останні роки.

Особливості фреймворка Flutter

На відміну від багатьох відомих мобільних платформ, Flutter не використовує JavaScript. У якості мови програмування для Flutter вибрали Dart,

який компілюється в бінарний код, за рахунок чого досягається швидкість виконання операцій порівнянна з нативними мовами програмування.

Flutter не використовує нативні компоненти, так що не потрібно писати прошарки для комунікації з ними. Замість цього, він отрісовиває увесь інтерфейс сам. Кнопки, текст, фон - все це програмується у середині графічного движка в самому Flutter.

Для побудови UI під Flutter використовується декларативний підхід, натхненний веб-фреймворком ReactJS, на основі віджетів. Для ще більшого приросту у швидкості роботи інтерфейсу віджети перемальовуваються за необхідностю - тільки коли в них щось змінилося.

До всього, в фреймворк вбудований Hot-reload, який полегшує швидкість розробки.

Порівняння з аналогами

Рисунок 1.3 – Порівняння з аналогами

Мовний стек

Давайте подивимося, який фреймворк використовує які мови програмування, і запропонуємо які переваги:

React Native: Він використовує JavaScript, який на сьогодні є одним із найпопулярніших, динамічних та високорівневих мов програмування. Він з'єднує в собі переваги JavaScript і React.JS, і фінансується Facebook.

Істотною стороною React Native, яка робить його найкращим серед інших трьох фреймворків з точки зору PL, є те, що він дозволяє писати кілька компонентів у Swift, Objective-C або Java, у той час, коли розробники цього вимагають. Використовуючи власні модулі та бібліотеки у програмах React Native, ви зможете керувати обчислювально важкими операціями, такими як обробка відео та редагування зображень.

Xamarin: Він використовує C# із середовищем .NET для розробки застосунків для Android, iOS та Mac. Все, що можна досягти за допомогою рідних мов, розробник може зробити на C#, використовуючи Xamarin. Однак розробники не можуть використовувати власні бібліотеки, доступні для iOS та Android із Xamarin, існує безліч доступних бібліотек .NET, які задовольняють бажану потребу.

IONIC: Він використовує HTML5, CSS та JS для розробки та запуску застосунків, і вимагає обгортки Cordova для доступу до власних контролерів платформи. Використовуючи IONIC, також можно використовувати TypeScript, що покращує якість коду.

Flutter: Він використовує Dart для розробки високоякісних програм для Android, iOS та Інтернету. Dart - це дивовижна мова програмування, яка пропонує безліч переваг і базується на C / C++ та Java. Незважаючи на те, що мова нова, незабаром очікується, що ця мова захопить галузь. Dart - одна із причин, чому сьогодні розробники застосунків віддають перевагу розробці Flutter App.

Ось як ви можете оцінити React Native проти Xamarin проти Ionic проти Flutter на основі переваг, які пропонують їх мови програмування(*Рисунок 0-4*):

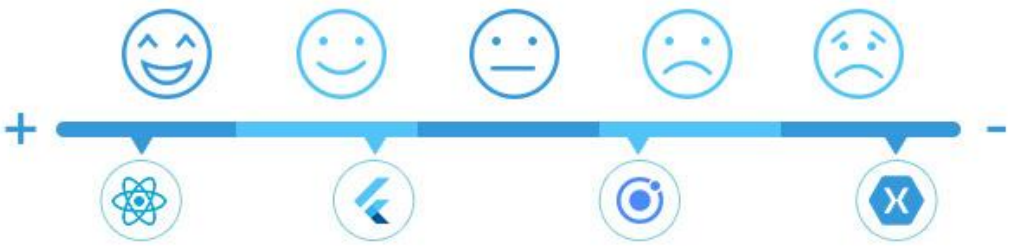


Рисунок 1.4 – Вподобання за мовним стеком

Продуктивність

Цей атрибут є найважливішим і говорить сам за себе - як працюють програми, створені в різних рамках, розглядаючи продуктивність?

React Native: Продуктивність, яку вона забезпечує, дуже схожа на власні програми, оскільки вона відображає елементи коду спеціально до власних API. React додатково дозволяє розробникам використовувати власні модулі, написані рідною мовою, для написання коду для складних операцій. Однак їх не можна використовувати повторно на двох платформах; їх головне призначення - забезпечити більш високі показники.

Xamarin: Ефективність Xamarin також вважається закритою для власних. Xamarin має два способи створення мобільних застосунків, а саме Xamarin-Android / Xamarin-iOS та Xamarin-Forms.

Застосунки Xamarin-Android / iOS працюють як власні, оскільки їх крос-платформні можливості зосереджені на спільному обґрунтуванні роботи, а не на кодовій базі. Це допомагає досягти власної продуктивності, яка неможлива завдяки рішенням, які інтерпретують код під час виконання.

Однак підхід Xamarin-Forms орієнтований на широкий обмін кодами з менш специфічною поведінкою платформи. Це знижує продуктивність коду у більшості операцій, порівняно з різними платформами.

Ionic: Що стосується продуктивності, Ionic програє гру. Його продуктивність не така схожа на рідну, як пропозиції Xamarin, React Native або Flutter, оскільки вона використовує веб-технології для візуалізації застосунку. Такий підхід зменшує швидкість. Крім того, розробка іонних застосунків не використовує власних компонентів, а намагається створити природний вигляд та відчуття за допомогою веб-технологій.

Перевагою Ionic є процес швидкого тестування, який миттєво запускається у браузері, який впорядковує процес розробки.

Flutter: якщо порівнювати з урахуванням продуктивності застосунків, саме Флаттер переймає корону над своїми конкурентами. Оскільки він має переваги Dart і не існує моста JavaScript для запуску взаємодії з рідними компонентами пристрою, швидкість, яку він пропонує, вражає.

Ось результат суперечок React Native проти Xamarin проти Ionic проти Flutter, заснованих на продуктивності, яку вони пропонують:

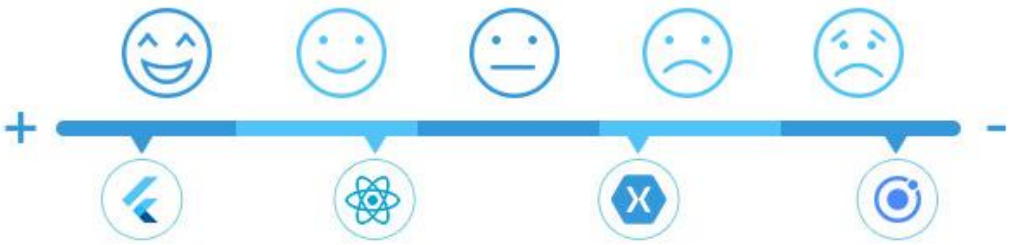


Рисунок 1.5 – Вподобання за продуктивністю

Графічний інтерфейс користувача

Користувачі оцінюють застосунки протягом перших кількох секунд використання, і тому графічний інтерфейс програми повинен бути залученим, проте бути простим - давайте подивимося, що пропонує ця структура:

React Native: Модулі React Native асоціюються з рідними контролерами інтерфейсу користувача, що забезпечує дивовижну взаємодію з користувачем, яка дуже близька до власних програм. Він додатково використовує бібліотеку ReactJS з великими елементами інтерфейсу та спрощує розробку інтерфейсу.

Xamarin: дозволяє створювати інтерфейс двома різними способами: за допомогою Xamarin-Android / iOS або Форм Xamarin. Перший займає багато часу, але гарантує природний вигляд та відчуття в плані UX.

За допомогою Форм Xamarin можливо значно пришвидшити процес розробки та заощадити багато ресурсів, але ціною місцевого зовнішнього вигляду. Це може бути хорошим рішенням для внутрішніх та корпоративних підприємств, де частина інтерфейсу не є настільки важливою, як у публічних застосунках.

Ionic: Ionic UI взагалі не використовує природні елементи і відображає все в HTML та CSS. Тоді це застосовує Кордову, щоб надати мобільний мобільний досвід. Кутові компоненти, що супроводжують фреймворк, також дозволяють застосункам Ionic виглядати як власні.

Flutter: Flutter забезпечує найкращі користувальницькі інтерфейси. Без сумніву, Ionic та Xamarin надають нам застосунки для різних платформ, але їх ефективність та продуктивність не можуть перевершити Flutter та React Native.

Вони застрягли і не реагують, якщо застосунок важкий і використовуються більше власних компонентів інтерфейсу.

Ось результат дискусії React Native проти Xamarin проти Ionic vs Flutter залежно від інтерфейсу, який вони пропонують:

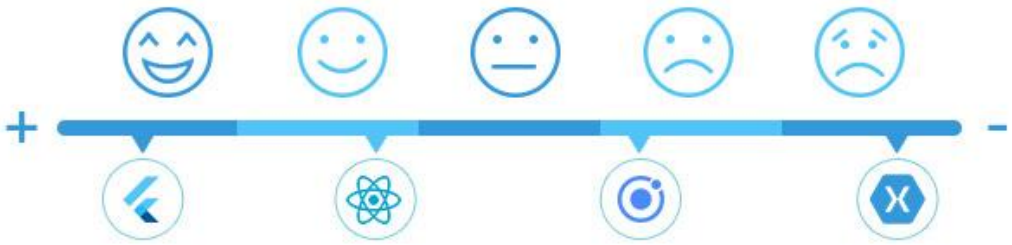


Рисунок 1.6 – Вподобання за UI

Ринок та громада

Наявність динамічної екосистеми - це те, за що варто подякувати - але важливо, наскільки популярний варіант? Подивимось:

React Native: React Native - король, коли справа доходить до визнання ринку та громади. Мережа його розробників швидко розростається, і на сьогоднішній день має багато досвідчених React Native Engineers. Це дозволяє легко розпочати проект React. Він використовує дивовижно популярну бібліотеку (React) та найважливішу мову веб-розробки (JavaScript) та забезпечує справжні рідні програми. Ці якості роблять його міцною платформою і є причиною її слави.

Ionic: це другий за популярністю фреймворк після React. Це надає розробникам можливість створювати рідні мобільні застосунки якомога швидше.

Xamarin: Xamarin - це також тихий популярний фреймворк. І все-таки Microsoft докладає багато зусиль для розширення спільноти Xamarin. Розробники, які працюють в екосистемі Microsoft, можуть без особливих зусиль почати працювати з інновацією завдяки її активній підтримці.

Flutter: Flutter - це новий фреймворк для спільноти зараз і не дуже популярний. Але Google інтенсивно рекламує його, що показує, що вони хочуть

зробити це важливою справою у світі мобільних пристроїв. Незважаючи на те, що у нього все ще є деякі точки поколювання, використовувати його цікаво, і ви можете швидко переходити від ідеї до прототипу до програми.

Ось результат дискусії React Native проти Xamarin проти Ionic vs Flutter, заснованої на визнанні та надійності галузі:

Рисунок 1.7 – Вподобання за ринком

Підтримувані платформи

React Native: Android 4.1+, iOS 8+

Xamarin: Android 4.0.3+, iOS 8+, Window 10

Ionic: Android 4.4+, iOS 8+, Windows 10

Flutter: Android 4.1+, iOS 8+, Windows7+

Повторне використання коду

За допомогою цього атрибуту ви отримаєте уявлення про те, яку частину написаного вами коду можливо використовувати повторно для обох платформ, що є основною метою цих фреймворків.

React Native: використовує власні компоненти, написані на Objective-C, Swift або Java для підвищення продуктивності програми. Але ці компоненти не можливо використовувати повторно на інших платформах. Отже, розробникам потрібно трохи попрацювати над зміною цієї конкретної кодової бази. Однак прийміть ці власні компоненти, що залишиться від кодової бази, можливо використати повторно.

Xamarin: Він не вимагає перемикання між середовищами розробки: усі програми Xamarin розроблені у Visual Studio. Зазвичай до 96% вихідного коду

можливо повторно використовувати з Формами Xamarin, що прискорює процес розробки.

Ionic: Суттєвою частиною програм, розроблених в Ionic, є їх універсальність. Незалежно від даної ОС, вони будуть працювати однаково добре на кожній з них. Але деякі компоненти інтерфейсу користувача повинні бути змінені на правила, продиктовані певною платформою, що вимагатиме додаткових зусиль.

Flutter: У React Native ми маємо готові до використання функції, що прискорює швидкість розвитку. Однак у Flutter потрібно додати спеціальні файли як для платформи Android, так і для iOS залежно від їхніх правил.

Ось результат дискусії React Native проти Xamarin проти Ionic vs. Flutter на основі їх повторного використання коду:

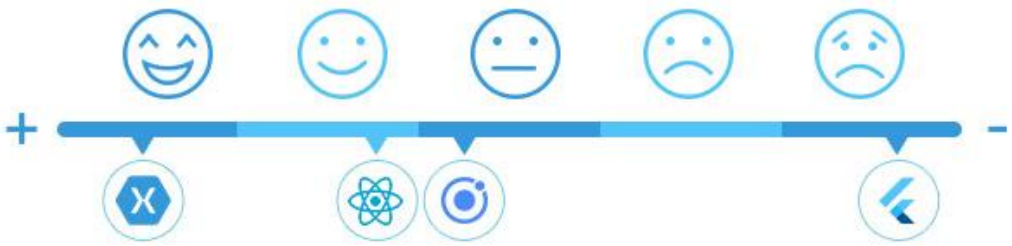


Рисунок 1.8 – Вподобання за повторним використанням коду

Популярні програми

React Native: Facebook, Instagram, Airbnb, UberEats

Xamarin: Olo, the World Bank, Storyo.

Ionic: JustWatch, Nationwide та багато інших.

Flutter: HamilTon

Ціноутворення

React Native і *Flutter*: обидва цілком відкриті фреймворки. Інженери можуть безкоштовно використовувати цю систему та їхні бібліотеки.

Xamarin: Xamarin вимагає від розробників встановити IDE, який надається за передплатою. Однак він також пропонує безкоштовну версію для студентів та непідприємницьких підприємств, що мають до 5 користувачів.

Ionic: Це безкоштовний фреймворк з доступним кодом для розробки крос-платформних застосунків. Але організація пропонує свою версію Pro, яка є платною. Компанія гарантує, що Ionic Pro пришвидшує процес розробки.

1.4 Dart

Рисунок 1.9 – Лого Dart

Dart - це швидка, об'єктно-орієнтована мова програмування, заснована на парадигмі, яка використовується для розробки кроссплатформенних застосунків. Ця мова програмування, створена Google, позиціонується в якості заміни / альтернативи JavaScript. Один з розробників мови Марк Міллер написав, що JavaScript «має фундаментальні вади», які неможливо виправити.

Ви можете використовувати Dart для написання простих фрагментів коду і повнофункціональних застосунків. У Dart є рішення практично для всього - від розробки застосунків для настільних комп'ютерів до веб-застосунків, мобільних застосунків, сценаріїв командного рядка і серверних сценаріїв.

Dart поставляється з гнучкою технологією компілятора, що дозволяє запускати код і створювати його відповідно до цільового вимогою. Dart надає середовище розробки мобільних застосунків під назвою Flutter.

1.5 Розлянуті бази даних для Flutter

Оскільки зараз на ринку доступно багато баз даних, вибір найкращої може бути складним завданням. Тут ми відфільтрували п’ять кращих баз даних, які можно використовувати для програмування на Flutter. Ось трохи деталей про кожного з них.

Back4App

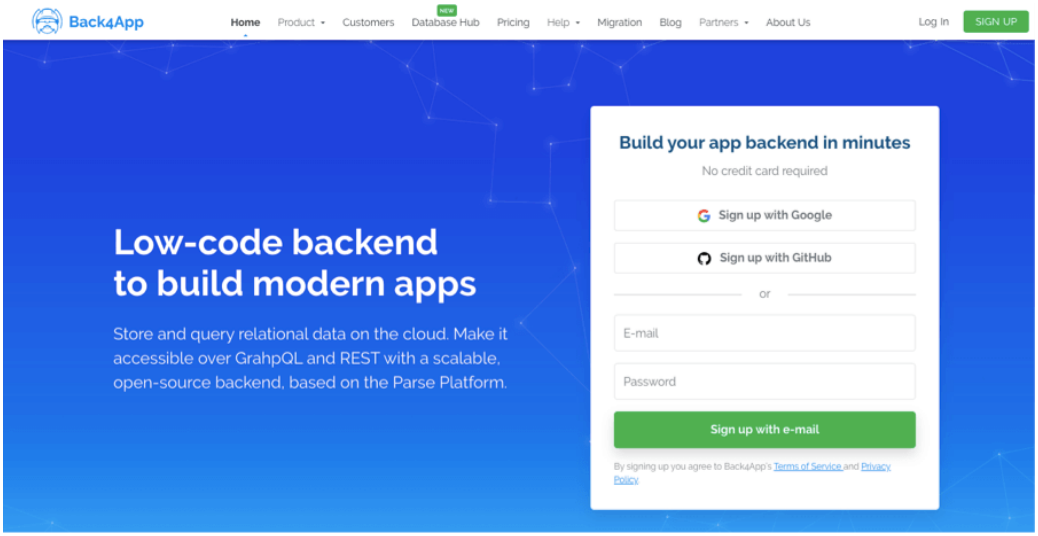


Рисунок 1.10 – Back4App

Back4app - це реляційна база даних для ваших програм Flutter, яка постачається з багатьма функціями, які готові до впровадження завдяки зручній інформаційній панелі. Для з'єднань даних можно використовувати REST API і GraphQL API, щоб додати багато функцій до програми.

Особливості

- Бази даних SQL та NoSQL
- API Graphql та REST
- Повідомлення
- Аутентифікація
- Зберігання

Ціноутворення

- Безкоштовний рівень
- Спільний хостинг від 5 доларів на місяць
- Виділений хостинг від 250 доларів на місяць

Firebase

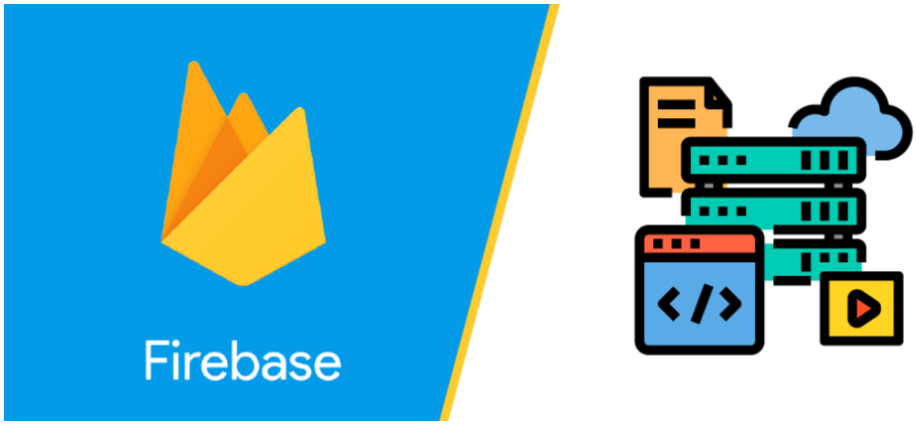


Рисунок 1.11 – Firebase

Firebase постачає базу даних NoSQL для застосунків Flutter, де протокол JSON може бути використаний для управління всіма операціями пошуку та зберігання даних. Це одна з найбільш підходящих опцій Flutter Apps завдяки своїй функції синхронізації даних та швидкому часу завантаження.

Особливості

- NoSQL DB
- API (лише REST)
- Аутентифікація
- Аналітика
- Зберігання

Ціноутворення

- Безкоштовний рівень
- Оплачуйте по ходу моделі для платних застосунків

SQ Lite

Рисунок 1.12 – SQLite

SQ Lite - це дуже надійна платформа баз даних для ваших програм Flutter завдяки своїй базі даних SQL. Підтримка SQL полегшує кожному розробнику

здійснення транзакцій даних. Однак тут схема не застосовується для розробників, що полегшує процес.

Особливості

- Безсерверний
- Нульова конфігурація
- Відкрите джерело
- Компактний
- Один файл БД

Ціноутворення

- Безкоштовно завантажити

Moor БД

Рисунок 1.13 – Moor

Якщо вам потрібна велика безпека даних ваших програм Flutter, тоді Moor може бути найкращим варіантом для вас. Це не тільки безпечно, але це база даних, яка також має безліч функцій. Однією з найкращих можливостей є простий код бази даних для всіх функцій бази даних.

Особливості

- Декларативні таблиці
- Вільні запити
- Гнучкий
- Легко вчитися
- Безкоштовний шаблон

Ціноутворення

- Безкоштовно завантажити

Таблица 0-1 – Ціноутворення баз даних

Постачальник послуг	Категорія	Ціни Управління хостингом
Back4App	Реляційні	Безкоштовний рівень або 5 доларів на місяць
Firebase	Нереляційні	Безкоштовний рівень або платіть, коли ви виставляєте рахунки
SQ Lite	Реляційні	$ 1,99 / міс
Moor DB	Реляційні	$ 5 / міс

1.6 Особливості застосунку для пошуку майданчиків

Спортивними майданчиками можуть бути як майданчики відкритого типу, на вулиці, так і зали учбових установ або спортивних комплексів. Майданчик може бути безкоштовним та відкритим для усіх охочих, або закритого типу на платній основі.

1.7 Особливості конкурентного середовища серед застосунків подібного типу

Застосунки для пошуку спортивних майданчиків, повинні давати можливість шукати майданчики на карті з можливістю добовлять собстенних. Користувачі цього застосунка можуть переглянути інформацію про дані місцях і обсусіть їх або постової оцінку. Для обговорення в застосунку повинен распологатся чат. Чат може бути як глобальний так і локальний. Користувачі повинні мати можливість зарігістріватся різними способами. Одна з особонно даних застосунків є можливість переглядати користувачами, своїх поточних тренувань.

Аналіз існуючих застосунків для пошуку майданчиків

На ринку застосунків вже існують застосунки для пошуку спортивних майданчиків. Кожне з них мають свої особливості і напрямки і в цьому пункті будуть розглянуті вони, а також чим відрізнятиметься застосунок розроблене мною.

Playseek

Рисунок 1.14 – Playseek

Застосунок для пошуку і створення спортивних подій. Розроблений нашими співвітчизниками. Дозволяє знаходити різні події починаючи з футболу закінчуючи плаванням і гімностікой. У програмі є можливість відстежувати поточні тренування, набирати людей з друзів і випадкових користувачів, отримувати повідомлення про майбутні події, листуватися з учасниками групи.

Площадка

Рисунок 1.15 – Площадка

Застосунок для пошуку спортивних майданчиків. У застосунку можно: знайти відповідний спортзал, корт або поле по заданих параметрах, майданчики видаються списком або відображаються на карті, забронювати і сплатити оренду на деяких спортивних об'єктах. Найбільшою проблемою є те, що застосунок працює лише на території Росії.

CourtFinder

Рисунок 1.16 – CourtFinder

Застосунок CourtFinder дозволяє знайти на карті найближчі до вас корти або місця в будь-якій точці світу. Додавати назву, адресу кожного баскетбольного майданчика, а також опис того, як туди дістатися. Додайти власні фотографії до кортів. Продивлятися погодні умови для кожного корту, щоб передбачити, чи варто планувати гру на день.

Проблеммою є направленність застосунку тільки на пошук баскетбольних кортів.

2 МОДЕЛЬ ПРОЕКТУ ТА СТРУКТУРА

2.1 Механізм роботи Flutter

Архітектурні шари

Флаттер розроблений як розширювана багатошарова система. Він існує як серія незалежних бібліотек, кожна з яких залежить від базового рівня. Жоден шар не має привілейованого доступу до шару нижче, і кожна частина рівня фреймворку спроектована як необов'язкова та замінна.

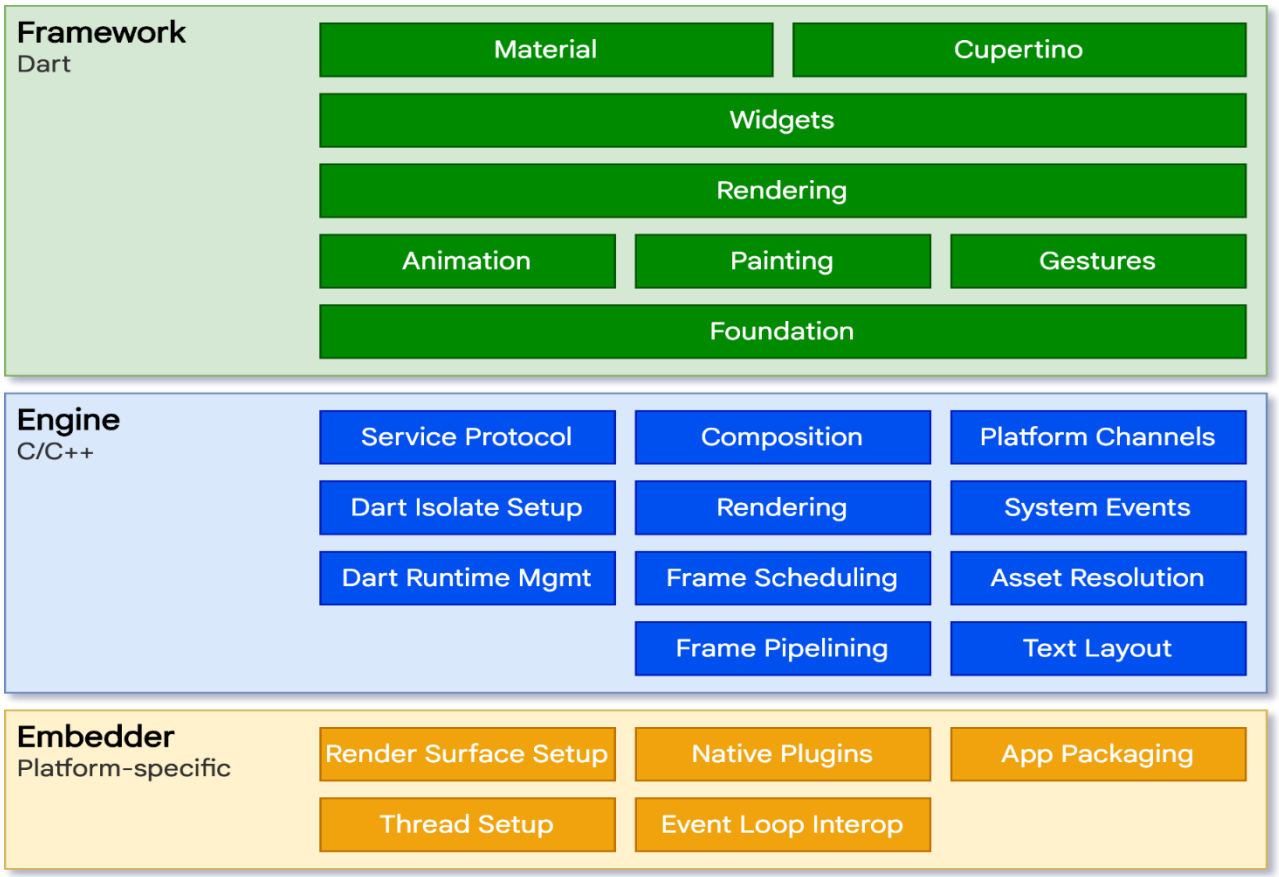


Рисунок 2.1 – Архітектурні шари

Для основної операційної системи програми Flutter упаковуються так само, як і будь-яка інша рідна програма. Специфічний для платформи вбудовувач забезпечує точку входу; узгоджується з базовою операційною системою для доступу до таких послуг, як рендеринг поверхонь, доступність та введення; та керує циклом подій повідомлення. Embedder написаний мовою, яка підходить для платформи: в даний час Java та C ++ для Android, Objective-C / Objective-C++ для iOS та macOS та C++ для Windows та Linux. Використовуючи

embedder, код Flutter може бути інтегрований до існуючої програми як модуль, або код може представляти собою весь вміст програми. Flutter включає ряд вбудованих програм для загальних цільових платформ, але існують і інші вбудовувальні програми.

В основі Flutter лежить механізм Flutter, який здебільшого написаний на C++ і підтримує примітиви, необхідні для підтримки всіх програм Flutter. Двигун відповідає за растеризацію складених сцен, коли потрібно намалювати новий кадр. Він забезпечує низькорівневу реалізацію основного API Flutter, включаючи графіку (через Skia), макет тексту, файлові та мережеві вводи-виводи, підтримку доступності, архітектуру плагінів, а також середовище виконання та компіляції Dart.

Механізм піддається дії Flutter через dart: ui, який обгортає базовий код C++ у класах Dart. Ця бібліотека надає примітиви найнижчого рівня, такі як класи керування підсистемами введення, графіки та тексту.

Як правило, розробники взаємодіють з Flutter через фреймворк, який забезпечує сучасну реактивну структуру, написану мовою Dart. Він включає багатий набір бібліотек платформи, верстки та основоположних бібліотек, що складається з низки шарів. Працюючи знизу вгору, маємо:

- Основні основоположні класи та такі основні послуги, як анімація, малювання та жести, які пропонують часто використовувані абстракції над основним фундаментом.
- Шар рендерингу забезпечує абстракцію для роботи з макетом. За допомогою цього шару ви можете побудувати дерево об’єктів, що відображаються. Ви можете маніпулювати цими об’єктами динамічно, дерево автоматично оновлює макет, щоб відобразити ваші зміни.
- Шар віджетів є абстракцією композиції. Кожен об'єкт візуалізації на рівні візуалізації має відповідний клас на рівні віджетів. Крім того, рівень віджетів дозволяє визначити комбінації класів, які можливо повторно використовувати. Це рівень, на якому представлена модель реактивного програмування.

- Бібліотеки Material і Cupertino пропонують вичерпні набори елементів керування, які використовують примітиви складу віджета для реалізації мов дизайну Material або iOS.

Каркас Флаттера порівняно невеликий; багато функцій вищого рівня, які можуть використовувати розробники, реалізовані у вигляді пакетів, включаючи плагіни платформи, такі як камера та веб-перегляд, а також функції агностики платформи, такі як символи, http та анімація, що базуються на основних бібліотеках Dart та Flutter. Деякі з цих пакетів походять із ширшої екосистеми, що охоплює такі послуги, як оплата в додатках, аутентифікація Apple та анімація.

Решта цього огляду широко спрямована вниз по шарах, починаючи з реактивної парадигми розвитку інтерфейсу користувача. Потім ми описуємо, як віджети складаються разом і перетворюються на об'єкти, які можуть бути відтворені як частина програми. Ми описуємо, як Flutter взаємодіє з іншим кодом на рівні платформи, перш ніж дати короткий підсумок того, як веб-підтримка Flutter відрізняється від інших цілей [20].

Віджети

Flutter використовує віджети як одиниці композиції. Віджети є будівельними елементами користувальницького інтерфейсу Flutter, і кожен віджет є незмінною частиною інтерфейсу.

Віджети утворюють ієрархію на основі композиції. Кожен віджет знаходиться всередині батьківського елемента і може отримувати їх контекст. Ця структура існує починаючи від кореневого віджета (контейнера, в якому розміщена застосунок Flutter, як правило, MaterialApp або CupertinoApp), як показано на рис. 2-2.

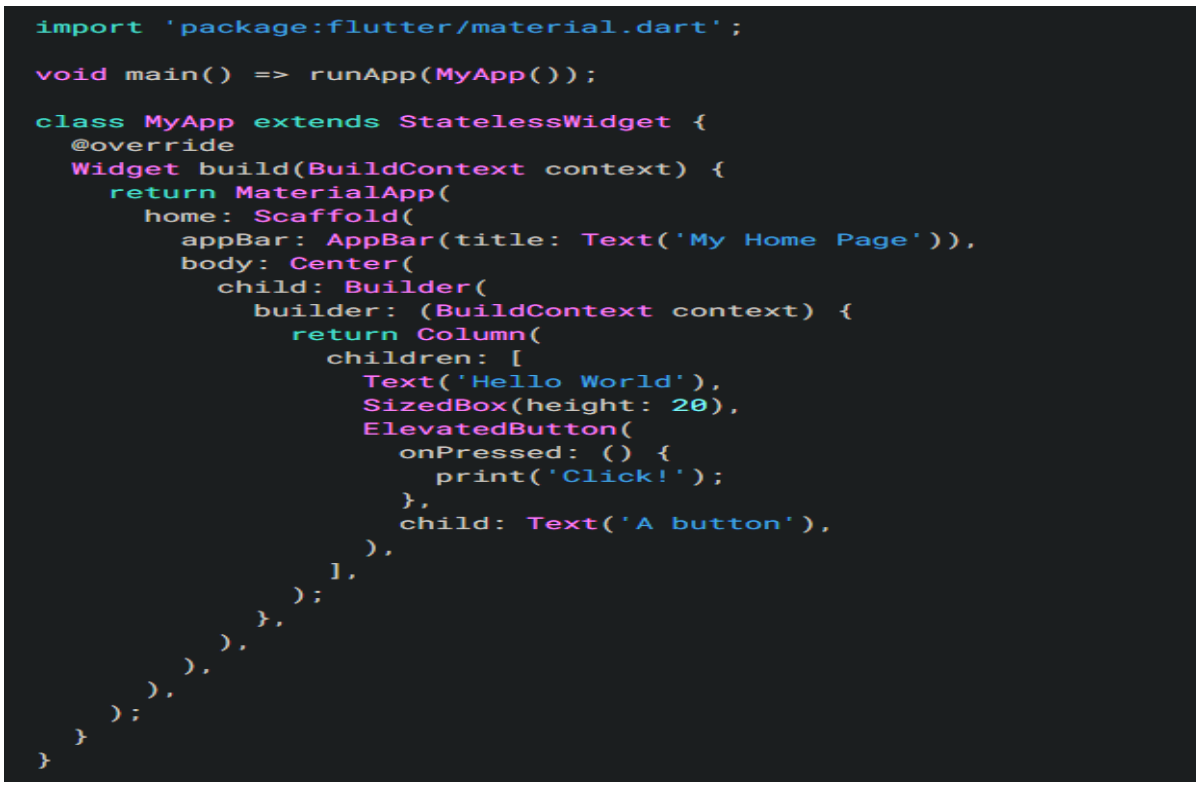


Рисунок 2.2 –Структура віджетів

У попередньому коді всі екземпляри класів - віджети.

Застосунок оновлює свій користувальницький інтерфейс у відповідь на події, повідомляючи фреймворку перемалювати віджет в ієрархії іншим віджетом. Потім фреймворк порівнює нові та старі віджети та оновлює інтерфейс користувача.

У Flutter є власні реалізації кожного елемента керування інтерфейсом, і не використовуєті, що надаються системою, наприклад: існує чиста реалізація Dart як для елемента управління iOS Switch, так і для Android.

Цей підхід забезпечує кілька переваг:

- Забезпечує необмежену розтяжність. Розробник, який хоче використовувати управління Switch, може створити його будь-яким способом, і не обмежується точками розширення, наданими ОС.
- Уникає складних місць продуктивності, дозволяючи Flutter складати всю сцену відразу, не переходячи між кодом Flutter та кодом платформи.

Відмежовує програмну поведінку від залежностей операційної системи. Застосунок виглядає однаково у всіх версіях ОС, навіть якщо ОС змінила реалізацію своїх елементів управління [21].

Стан віджета

Фреймворк вводить два основних класи віджетів: віджети з формуванням стану та без них.

У багатьох віджетах немає змінних станів: вони не мають властивостей, які змінюються з часом (наприклад, піктограма чи мітка). Ці віджети підкласу StatelessWidget.

Однак, якщо унікальні характеристики віджета потрібно змінити на основі взаємодії користувача чи інших факторів, цей віджет має статус стану. Наприклад, якщо у віджеті є лічильник, який збільшується щоразу, коли користувач натискає кнопку, тоді значення лічильника є станом для цього віджета. Коли це значення змінюється, віджет потрібно відновити, щоб оновити його частину інтерфейсу. Ці віджети мають підклас StatefulWidget, і (оскільки сам віджет є незмінним) вони зберігають змінний стан в окремому класі, що має підкласи State. StatefulWidgets не мають методу побудови; натомість їхній користувальницький інтерфейс будується через їхній об'єкт State.

Кожного разу, коли ви мутуєте об'єкт стану (наприклад, збільшуючи лічильник), ви повинні викликати setState (), щоб сигналізувати про фреймворк для оновлення користувальницького інтерфейсу, знову викликаючи метод побудови стану.

Наявність окремих об’єктів стану та віджетів дозволяє іншим віджетам обробляти віджети як без громадянства, так і з відмітними станами точно так само, не турбуючись про втрату стану. Замість необхідності триматися за дитину, щоб зберегти її стан, батько може створити новий екземпляр дитини в будь-який час, не втрачаючи стійкий стан дитини. Структура виконує всю роботу з пошуку та повторного використання існуючих об'єктів стану, коли це доречно [22].

Управління станом

Отже, якщо багато віджетів можуть містити стан, як керувати станом і передавати його по системі?

Як і в будь-якому іншому класі, ви можете використовувати конструктор у віджеті для ініціалізації його даних, тому метод build () може гарантувати, що будь-який дочірній віджет створюється за допомогою екземпляра даних, які йому потрібні:

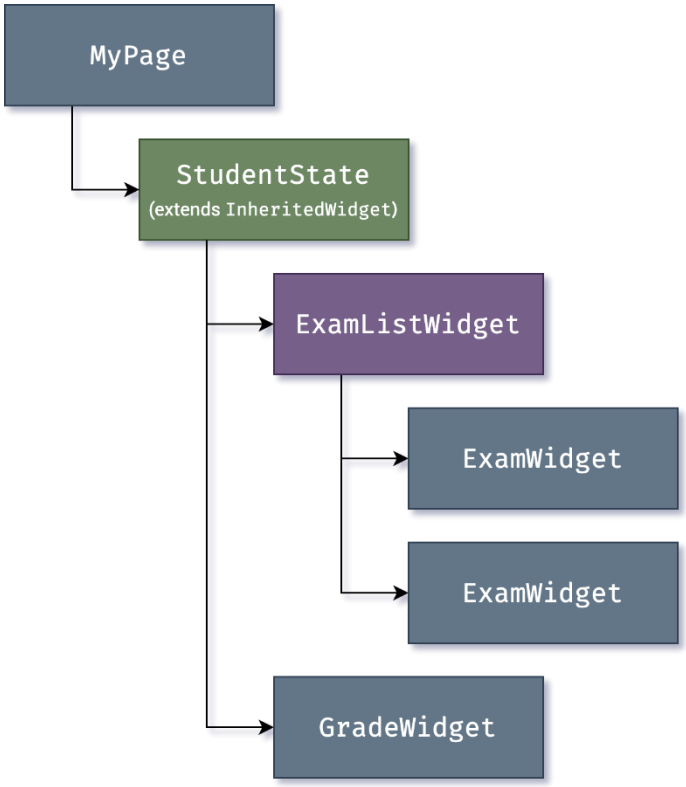


Рисунок 2.3 –Вкладенність віджетів

Всякий раз, коли одному з об'єктів ExamWidget або GradeWidget потрібні дані від StudentState, він тепер може отримати до нього доступ за допомогою такої команди, як:

```
final studentState = StudentState.of(context);
```

Рисунок 2.4 –Отримання контескту віджету

Виклик of (context) приймає контекст побудови (дескриптор поточного розташування віджета) і повертає найближчого предка у дереві, який відповідає типу StudentState. InheritedWidgets також пропонують метод updateShouldNotify (), який Flutter викликає, щоб визначити, чи має зміна стану викликати відновлення дочірніх віджетів, які його використовують.

Сам Flutter широко використовує InheritedWidget як частину рамки для спільного стану, наприклад візуальну тему програми, яка включає такі

властивості, як стилі кольорів і типів, які поширені у всій програмі. Метод MaterialApp build () вставляє тему в дерево під час побудови, а потім глибше в ієрархії віджет може використовувати метод .of () для пошуку відповідних даних теми, наприклад:

```
Container(
  color: Theme.of(context).secondaryHeaderColor,
  child: Text(
    'Text with a background color',
    style: Theme.of(context).textTheme.headline6,
  ),
);
```

Рисунок 2.5 – Отримання тем з контексту

Цей підхід також використовується для Навігатора, який забезпечує маршрутизацію сторінок; та MediaQuery, що забезпечує доступ до таких показників екрану, як орієнтація, розміри та яскравість.

У міру зростання додатків більш досконалі підходи до управління державою, які зменшують церемонію створення та використання віджетів, що містять статус, стають більш привабливими. Багато додатків Flutter використовують пакети службових програм, такі як провайдер, який забезпечує обгортку навколо InheritedWidget. Пошарова архітектура Flutter також дає можливість альтернативних підходів до реалізації перетворення стану в інтерфейс, наприклад пакет flutter_hooks [23].

Канали платформи

Для мобільних і настільних додатків Flutter дозволяє вводити користувацький код через канал платформи, який є простим механізмом для зв'язку між кодом Dart та кодом певної платформи вашого хост-додатку. Створюючи загальний канал (інкапсулюючи ім'я та кодек), ви можете надсилати та отримувати повідомлення між Dart та компонентом платформи, написаними такою мовою, як Kotlin або Swift. Дані серіалізуються з типу Dart, як Map, у стандартний формат, а потім десеріалізуються в еквівалентне подання в Kotlin (наприклад, HashMap) або Swift (наприклад, Dictionary).

Далі наведено простий приклад каналу платформи дзвінка Dart до одержувача обробки подій у Kotlin (Android) або Swift (iOS) [24]:

```dart
// Dart side
const channel = MethodChannel('foo');
final String greeting = await channel.invokeMethod('bar', 'world');
print(greeting);
```

Рисунок 2.6 – Dart нативний код

```kotlin
// Android (Kotlin)
val channel = MethodChannel(flutterView, "foo")
channel.setMethodCallHandler { call, result ->
  when (call.method) {
    "bar" -> result.success("Hello, ${call.arguments}")
    else -> result.notImplemented()
  }
}
```

Рисунок 2.7 – Код для Android

```swift
// iOS (Swift)
let channel = FlutterMethodChannel(name: "foo", binaryMessenger: flutterView)
channel.setMethodCallHandler {
  (call: FlutterMethodCall, result: FlutterResult) -> Void in
  switch (call.method) {
    case "bar": result("Hello, \(call.arguments as! String)")
    default: result(FlutterMethodNotImplemented)
  }
}
```

Рисунок 2.8 – *Код для iOS*

2.2 Опис роботи Firebase

Служби Firebase

Служби Firebase можна поділити на декілька груп:

Розробка і тестування застосунку:

- База даних у реальному часі
- Ауторизація
- Тест лабораторія
- Аналітика помилок
- Хмарні методи
- Сховище
- Хмарне зберігання
- Моніторинг ефективності
- Попередження про непередбачуване завершення роботи
- Хостинг

Приріст та збільшення аудиторії:

- Firebase Analytics
- Запрошеня
- Хмарні повідомлення
- Прогнози
- AdMob
- Динамічні посилання
- Adwords
- Віддалене налаштування
- Індексація додатків

База даних у реальному часі

База даних Firebase у реальному часі - це хмарна база даних NoSQL, яка дозволяє зберігати та синхронізувати інформацію між користувачами в режимі реального часу.

База даних - це насправді лише один великий об’єкт JSON, яким розробники можуть керувати в режимі реального часу.

Рисунок 2.9 –Структура бази даних

Лише за допомогою єдиного API база даних Firebase надає застосунку як поточне значення даних, так і будь-які оновлення цих даних.

Синхронізація в режимі реального часу полегшує користувачам доступ до даних із будь-якого пристрою. База даних також допомагає користувачам співпрацювати між собою.

Ще одна дивовижна перевага такої бази даних полягає в тому, що вона постачається з мобільними та веб-пакетами SDK, що дозволяє створювати застосунки без потреби в Інтернет з’єднані.

Коли користувачі переходять у автономний режим, пакети SDK використовують локальний кеш на пристрої для обслуговування та зберігання змін. Коли пристрій підключається до Інтернету, локальні дані автоматично синхронізуються.

База даних також можливо інтегрувати з автентифікацією Firebase, щоб забезпечити простий процес автентифікації.

Аутентифікація

Рисунок 2.10 –Firebase

Аунтифікація Firebase надає серверні сервіси, прості у використанні SDK та готові бібліотеки інтерфейсу для автентифікації користувачів у застосунку.

Як правило, на створення власної системи автентифікації знадобляться місяці. Після цього потрібно буде мати спеціальну команду для підтримки цієї системи. Але якщо використовувати Firebase, можна налаштувати всю систему менш ніж за 10 рядків коду, який буде обробляти все для вас, включаючи складні операції.

Автентифікувати користувачів можна такими способами:

- За електронною поштою та паролем
- За номером телефону
- Google
- Facebook
- Twitter

Використання автентифікації Firebase спрощує створення захищених систем автентифікації, а також покращує вхід та вбудовування для кінцевих користувачів.

Firestore

Рисунок 2.11 –Firestore

Хмарне сховище Firestore - це база даних NoSQL, яка дозволяє легко зберігати, синхронізувати та отримувати дані у глобальному масштабі.

Хоча це може здатися чимось схожим на базу даних, хмарне сховище Firestore привносить на платформу багато нового, що відрізняє її = від бази даних.

2.3 Взаємодія користувача із застосунком

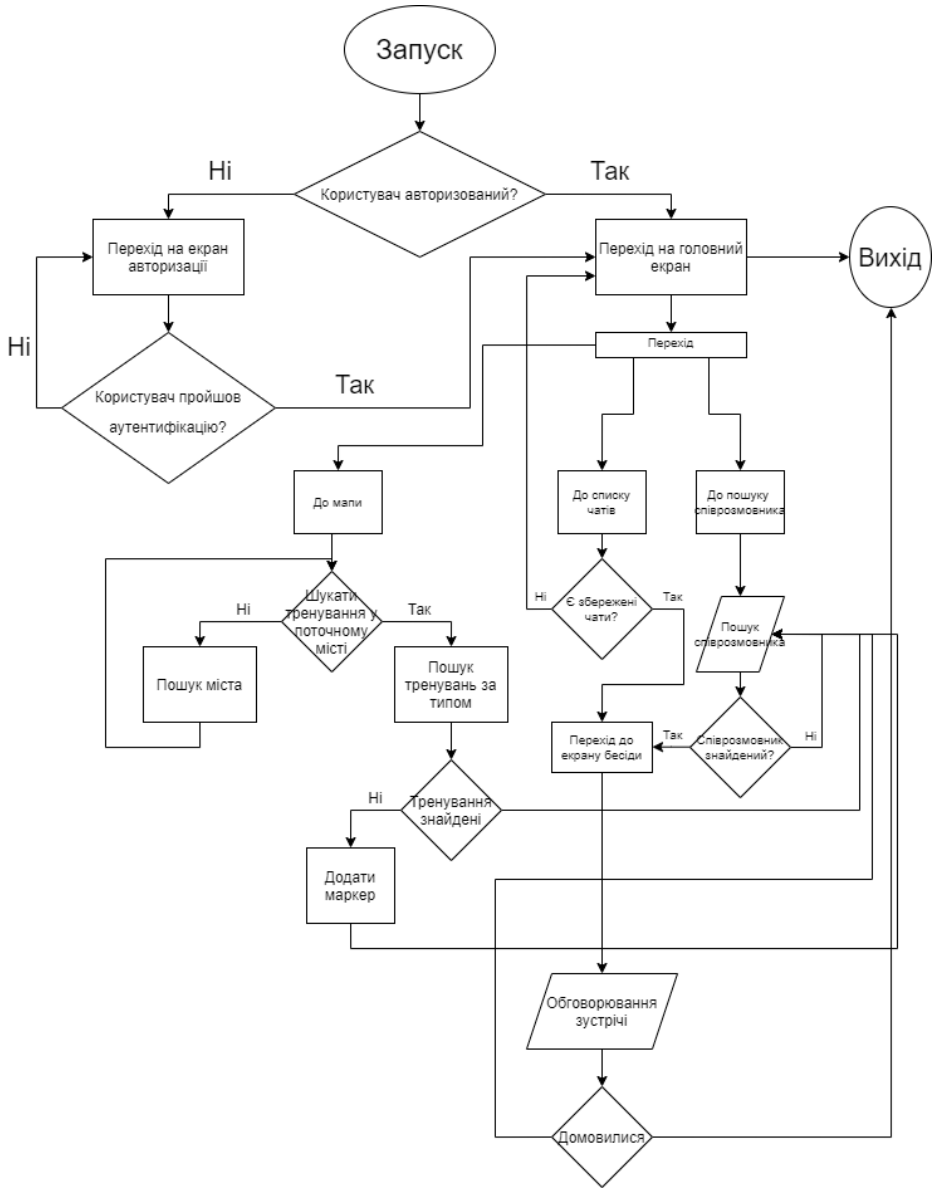


Рисунок 2.12 – взаємодія користувача із застосунком

Рис. 2.12 представляє взаємодію користувача із застосунком. У першу чергу користувач потрапляє на екран авторизації, за умови якщо користувач не був раніше авторизован. Після авторизації, потрапляє до головного екрану, у якому може вибрати перехід до мапи чи пошуку співрозмовників або вже збережених чатів. На екрані мапи користувач може шукати тренуваня у поточному місті або знайденому користувачем. Якщо користувач не може знайти місце тренування у місті, він може додати своє. Після знаходження тренуваня користувач може шукати однодумців, з якими він буде тренуватися. У кінці користувач домовляется про зустріч та вимикає застосунок.

2.4 Графічний опис процесів застосунку

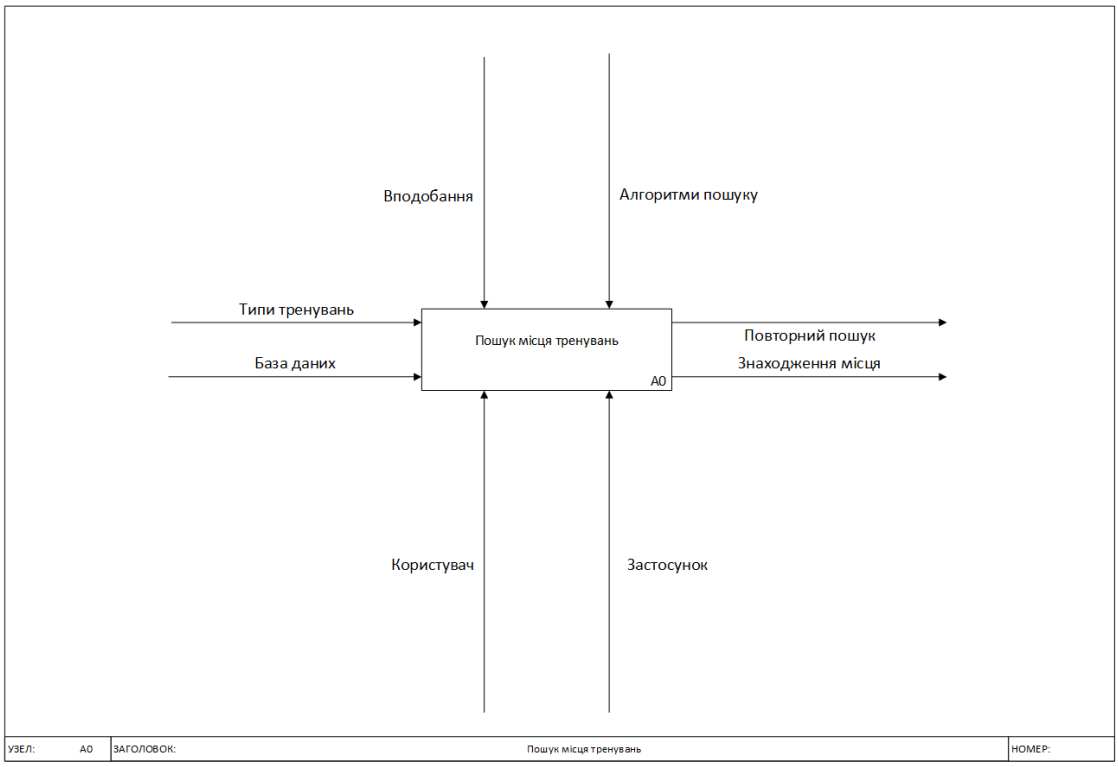


Рисунок 2.13 – IDEF0 модель пошуку місця тренувань

На рис. 2.13 зображано загальний вид пошуку місця тренувань. Вхідні дані: База даних та типи тренувань. Керуючими є користувач та застосунок. На виході отримуємо знаходжені місця тренувань.

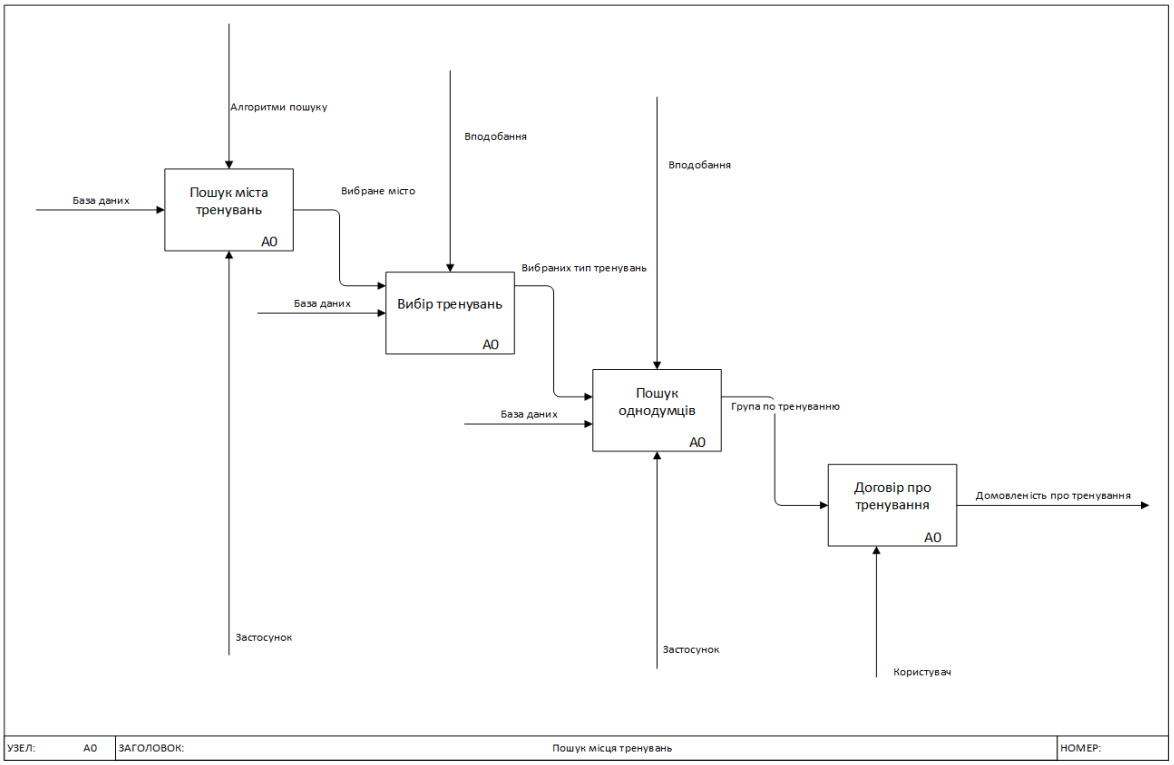


Рисунок 2.14 – розширений вигляд моделі IDEF0 для пошуку місць тренування

На рис.2.14 показаний повний вигляд пошуку місць тренування. Модель розділилася на: Пошук міста тренувань, Вибір тренувань, Пошук однодумців та Доровір ро тренування. Далі буде розглянуто розширену модель Вибір тренувань та Пошук однодумців.

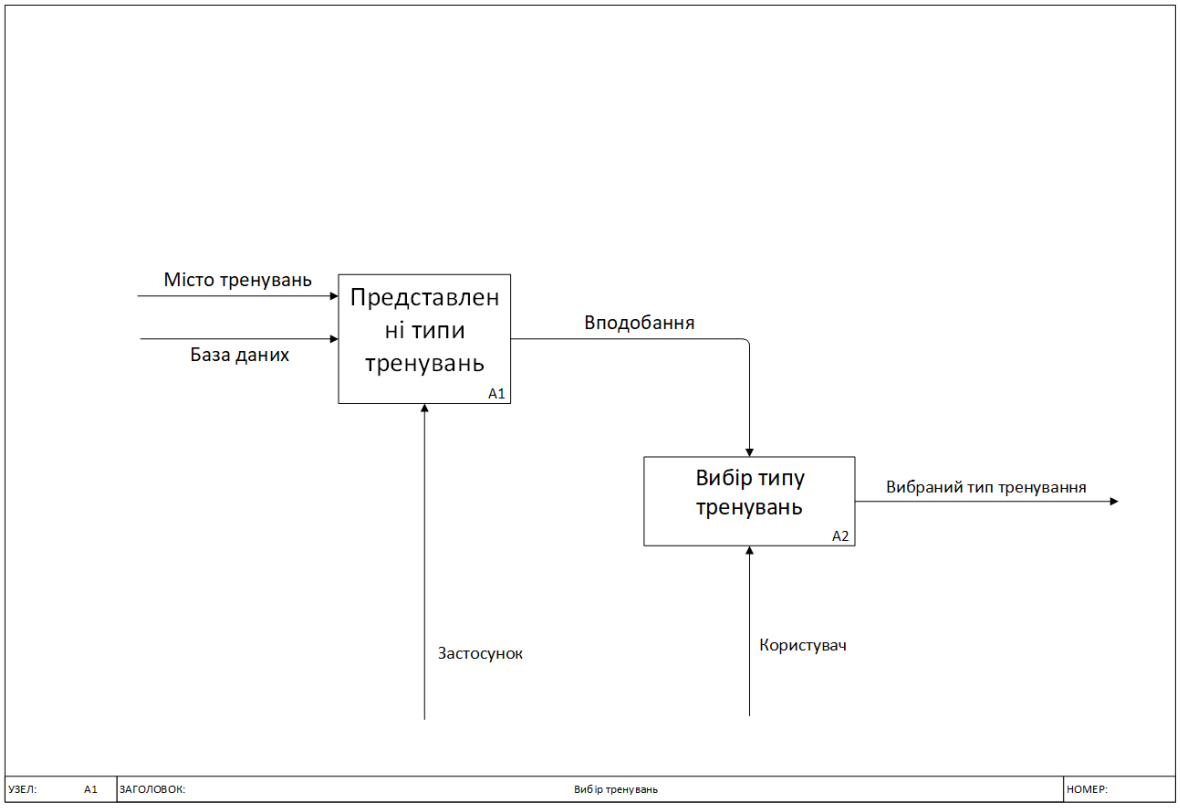


Рисунок 2.15 – модель IDEF0 вибору типу тренувань

На рис.2.15 показаний детальний процес вибору користувачем типу тренувань. Застосунок представляє усі типи тренувань у конкретному місті з бази даних, а користувач вибтрає вподобаний тип тренувань із списку.

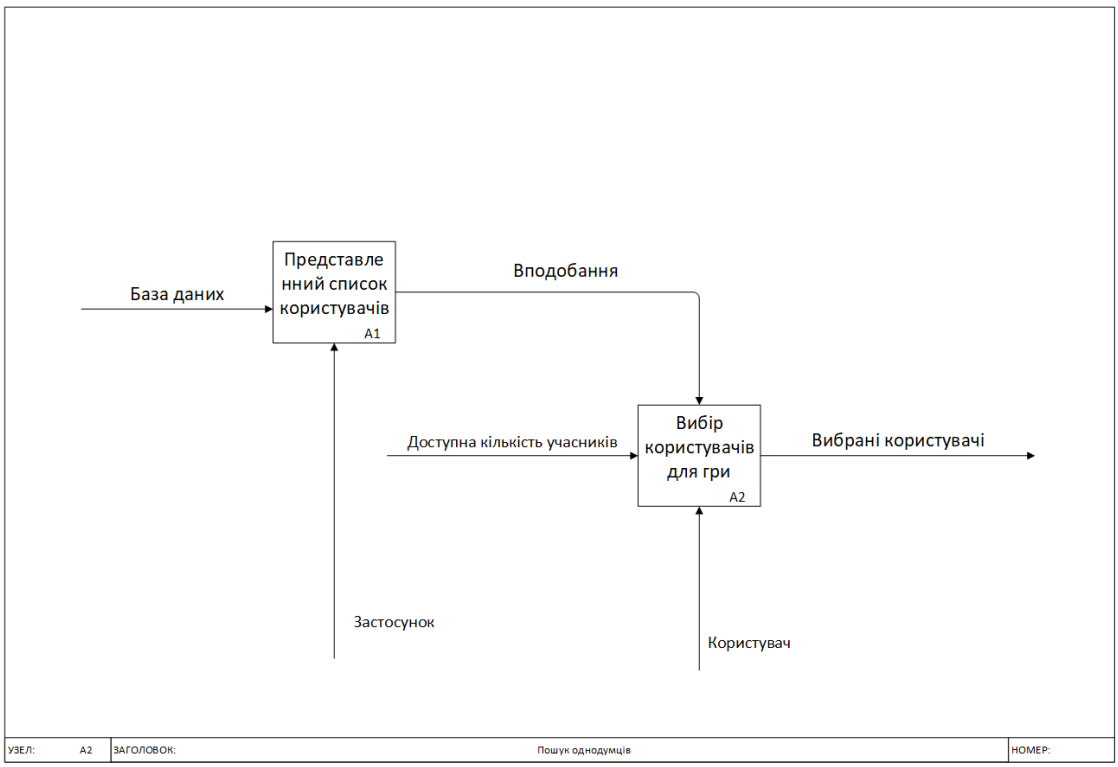


Рисунок 2.16 – модель IDEF0 вибору однодумців для тренувань

На рис. 2.16 показаний детальний процес вибору користувачем однодумців для тренувань. Користувач може вибрати з ким тренуватися із списку бази даниз користувачей. Користувач вибирає певну кількість користувачів в залежності від кількості учасників в команді.

Моделі IDEF0 описують процес пошуку користувачем необхідного тренування у поточному місті. Основною функцією системи є пошук місця тренувань. Таким чином, робота нашої контекстної діаграми - обслуговування користувача системи.

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Опис екранів застосунку та їх функціональної частини

Кнопки авторизації / реєстрації

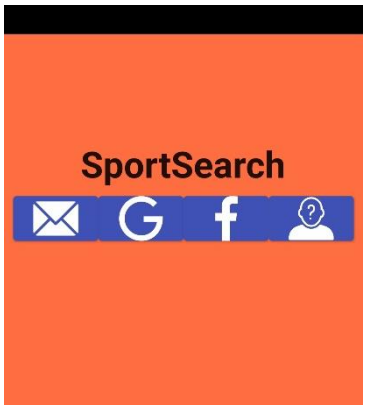


Рисунок 3-1 – Кнопки авторизації

Це екран авторизації / реєстрації. На ньому представлений логотип програми, а також кнопки вибору методу авторизації / регістації. Користувач може ввійти за допомогою пошти, акктаунта гугл, фейсбук або ж анонімно.

Код представленого екрану:

- Віджет кнопок. Залежно від того, у яке значення виставленно параметр iconName, кнопка авторизує відповідним чином.

```
Widget _authButtons(){
  return Padding(
    padding: EdgeInsets.all(10),
    child: Row(
      children: [
        _authButtonSvg(iconName: "email", func: _showAuthToggle),
        _authButtonSvg(iconName: "google", func: _loginButtonAction),
        _authButtonSvg(iconName: "facebook", func: _loginButtonAction),
        _authButtonSvg(iconName: "anonymous", func: _loginButtonAction),
      ],
    ),  // Row
  );  // Padding
}
```

Рисунок 3-2 – Віджет кнопок

- Внутрішній устрій кожної кнопки як віджету.

```dart
Widget _authButtonSvg({@required String iconName, @required Function func}){
  var _btnSize = _width / 4 * .5;
  return Expanded(
    child: ElevatedButton(
      style: ElevatedButton.styleFrom(
        primary: Colors.indigo,
        padding: EdgeInsets.zero
      ),
      onPressed: () => (iconName != "email") ? func(iconName) : func(),
      child:  SvgPicture.asset(
        "assets/icons/$iconName\.svg",
        color: Colors.white,
        width: _btnSize,
        height: _btnSize,
      ),  // SvgPicture.asset
    )  // ElevatedButton
  );  // Expanded
}
```

Рисунок 3-3 – Віджет окремої кнопки

- Функція авторизації, яка в залежності від отриманого типу кнопки, авторизує користувача відповідним способом.

```dart
void _loginButtonAction(String type) async {
  switch(type){
    case "emailForm":
      _email = _emailController.text;
      _password = _passwordController.text;

      _sharedService.saveUserEmailSharedPreferences(_email);
      if(_email.isEmpty || _password.isEmpty) {
        clearText();
        return;
      }
      authBloc.loginEmail(_email.trim(), _password.trim());
      clearText();
      break;
    case "facebook":
      authBloc.loginFacebook();
      break;
    case "google":
      authBloc.loginGoogle();
      break;
    case "anonymous":
      authBloc.loginAnonymously();
      break;
  }
}
```

Рисунок 3-4 – Авторизація поштою

Форма авторизації за допомогою пошти

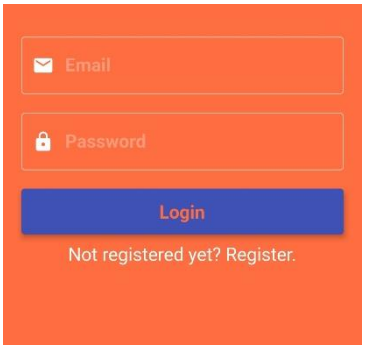


Рисунок 3-5 – Форма арторизації

На цьому екрані присутній два текстових поля для введення пошти та пароля. При натисканні на кнопку авторизації, додаток звіряє введені дані і в разі правильності, перенаправляє на головний екран. Під кнопкою знаходиться перехід на форму реєстрації.

Код представленого екрану:

Вид форми авторизації / реєстрації. Залежно від того, виставлена змінна showLogin в True, користувачеві відображається форма авторизації або ж реєстрації.

```
Widget _form(String buttonText, Function btnF){
  return Container(
    child: Column(
      children: [
        Form(
          key: _formKey,
          child: Column(
            children: [
              (_showLogin)
                  ?
              Container() :
              Padding(
                padding: EdgeInsets.only(top: 50),
                child: _input(Icon(Icons.account_circle), "Username", _usernameController, false),
              ),  // Padding
              Padding(
                padding: EdgeInsets.only(bottom: 20, top: (_showLogin) ? 50 : 20),
                child: _input(Icon(Icons.email), "Email", _emailController, false),
              ),  // Padding
              Padding(
                padding: EdgeInsets.only(bottom: 20),
                child: _input(Icon(Icons.lock), "Password", _passwordController, true),
              ),  // Padding
            ],
          ),  // Column
        ),  // Form
        Padding(
          padding: EdgeInsets.symmetric(horizontal: 20),
          child: Container(
            height: 50,
            width: MediaQuery.of(context).size.width,
            child: _button(buttonText, btnF),
          ),  // Container
        ),  // Padding
      ],
    ),  // Column
  );  // Container
}
```

Рисунок 3-6 – Віджет форми

Метод авторизації в блоці управління аутентифікацією. Приймає в якості параметрів пошту та пароль, після чого зберігає для автоматичного входа у застосунок, після чого авторизирується в Firebase.

```
loginEmail(String email, String password) async {
  try{
    _databaseBloc.getUserByEmail(email).then((val){
      _snapshot = val;
      _sharedService.saveUserLoggedInSharedPreferences(true);
      _sharedService.saveUserEmailSharedPreferences(email);
      _sharedService.saveUsernameSharedPreferences(_snapshot.docs[0].get("username"));
      _sharedService.saveUserPasswordSharedPreferences(password);
    });
    User user = await _authService.singInEmailPass(email, password);
  }catch(e){
    print(e);
    return null;
  }
  notifyListeners();
}
```

Рисунок 3-7 – Метод авторизації

Форма реєстрації за допомогою пошти

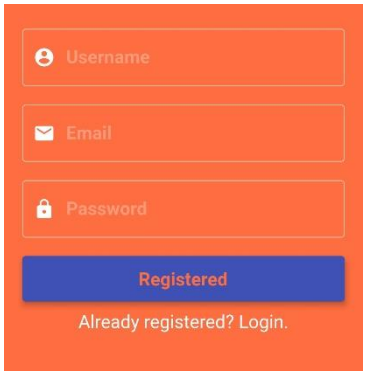


Рисунок 3-8 – Форма реєстрації

Подібно формі авторизації має поля для введення пошти та пароля, але додатково содердітся поле введення імені користувача, яке відображається при пошуку іншими користувачами.

Код представленого екрану:

Кнопка зміни авторизації на реєстрацію та навпаки.

```
Padding(
  padding: EdgeInsets.all(10),
  child: GestureDetector(
    child: Text(
      "Not registered yet? Register.",
      style: TextStyle(
        fontSize: 20,
        color: Colors.white,
      ),  // TextStyle
    ),  // Text
    onTap: (){
      setState(() {
        _showLogin = false;
        clearText();
      });
    },
  ),  // GestureDetector
)  // Padding
```

Рисунок 3-9 – Кнопка зміни форми

Функція реєстрації за допомогою пошти. Дані для реєстрації отримуються з текстових полів за допомогою відповідних контроллерів беручи їх поле text. Потім перевіряється чи заповнені усі, у разі не заповнення функція завершує виконання. Якщо помилок не було виявлено створюється словник із імені користувача та його пошти. У кінці роботи функція реєструє користувача у системі та зберігає дані на носії для автоматичної аунтифікації.

```
void _registerButtonAction() async {
  _username = _usernameController.text;
  _email = _emailController.text;
  _password = _passwordController.text;


  if(_username.isEmpty || _email.isEmpty || _password.isEmpty) return;

  Map<String, String> userInfoMap = {
    "username" : _username,
    "email" : _email,
  };
  User user;
  await _authService.registerEmailPass(_email.trim(), _password.trim()).then(
          (value){
            if(value != null){
              user = value;
              _sharedService.saveUserLoggedInSharedPreferences(true);
              _sharedService.saveUsernameSharedPreferences(_username);
              _sharedService.saveUserEmailSharedPreferences(_email);
              _sharedService.saveUserPasswordSharedPreferences(_password);
              _databaseBloc.uploadUserInfo(userInfoMap);
              clearText();
            }
          }
  );
}
```

Рисунок 3-10 – Функція реєстрації поштою

Головне вікно застосунку

Рисунок 3-11 – Головний екран

З даного екрану здійснюєть перехід до усіх інших екранів застосунку. На ньому знаходяться: Мапа, Профіль користувача, Чати, Групи та Пошук користувачів.

Код представленого екрану:

Головний екран – це генеруємий список з обраною кількістю елементів, кожний елемент представляє собою блок, який описано у наступному пункті.

```
body:
Builder(
  builder: (context) =>
    Stack(
      children: [
        Container(
          decoration: BoxDecoration(
            color: Colors.white
          ),  // BoxDecoration
          child: Wrap(
            children: List.generate(_size, (index) => bloc(context, screenWidth, index, _size)),
          ),  // Wrap
        ),  // Container
      ]
    ),  // Stack
)  // Builder
```

Рисунок 3-12 – Постоєння блоків головного екрану

Блок списку з попереднього пункту, представляє собою область натискання, ширина та висота якої залежать від ширини пристрою. У якості картинки виступає зображення з ассетів у застосунку.

```
Widget bloc(BuildContext context,double width, int index, int size){
  bool isExpanded = expandableState[index];
  final double _tileWidth = width / 2;
  final double _tileHeight = (MediaQuery.of(context).size.height - Scaffold.of(context).appBarMaxHeight) / 3;
  final double _borderWidth = 2;
  return GestureDetector(
    onTap: () {
      _navigateToScreen(context, index);
    },
    child:
    Container(
      width: _tileWidth,
      height: _tileHeight,
      decoration: BoxDecoration(
          color: Color(0xffff5100),
          border: Border.all(
              color: Color(0xFFFF2600),
              width: _borderWidth,
          ), // Border.all
      ), // BoxDecoration
      child: Padding(
        padding: const EdgeInsets.all(40.0),
        child: tileImage(index),
      ), // Padding
    ) // Container
  ); // GestureDetector
}
```

Рисунок 3-13 – Віджет окремого блоку

Знайдені користувачі із запиту в пошуковому рядку

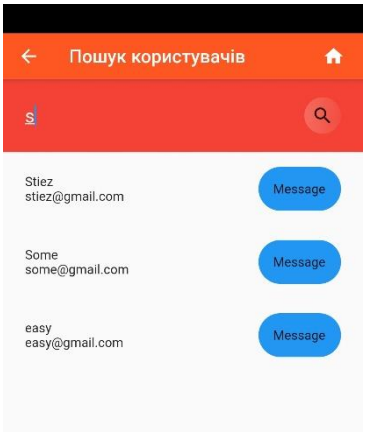


Рисунок 3-14 – Екран пошуку користувачів

За допомогою цього екрани можна шукати користувачів у системі на листуватися з ними. Користувачів шукає в базі даних порівнюючи кожний введений символ у поле пошуку. У результаті отримуємо список з користувача та його пошти, а також кнопкою, щоб перейти на екран розмови.

Код представленого екрану:

Поле пошуку користувачів. За допомогою контроллера з поля можна отримати поточне значення. При зміні значення у полі виконується функція пошуку користувачів.

```
child: Row(
  children: [
  Expanded(
    child: TextField(
      controller: _searchController,
      onChanged: _search(),
      style: TextStyle(
        color: Colors.white,
        fontSize: 20,
      ),  // TextStyle
      decoration: InputDecoration(
        hintText: "Введите имя пользователя",
        hintStyle: TextStyle(
            color: Colors.white54,
            fontSize: 20,
        ),  // TextStyle
        border: InputBorder.none
      ),  // InputDecoration
    ),  // TextField
  ),  // Expanded
  GestureDetector(...)  // GestureDetector
  ],
 ),  // Row
),  // Container
_searchList(),
,
```

Рисунок 3-15 – Поле пошуку користувачів

Функція пошуку користувачів. При виконанні функції, виконується метод отримання списку користувачів із бази даних, після чого отримані результати заносяться в змінну, при зміні якої перемальовується віджет із списким користувачів.

```
_search(){
  _databaseBloc
      .getUsersListByUserName(_searchController.text)
      .then(
          (value){
            setState(() {
              _userDocs = value;
            });
          }
          );
}
```

Рисунок 3-16 – Функція пошуку користувачів

Метод отримання користувачів із бази даних за введеним ім’ям користувача. При пустому імені метод завершую виконяння. Якщо усе добре, то метод отримує усіх користувачів у системі та відсіює несхожі на введене ім’я. У результаті отримуємо список знайденних користувачів.

```
getUsersListByUserName(String username) async{
 _usersDocs = [];
 if(username == "") return null;
 try{
  _snapshot = await _databaseService.getUsersByUsername(username);
  _snapshot.docs.forEach((element) {
   if((element.get("username").contains(username) || element.get("username").toLowerCase().contains(username)) && element.get("username") != Constants.myName){
     _usersDocs.add(element);
   }
  });
 }catch(e) {
  print(e);
 }
 return _usersDocs.toList();
}
```

Рисунок 3-17 – Метод отримання користувачів із бази даних

Метод створення екрану бесіди. У якості парамертів отримує імена користувачів бесіди, та на їх основі створює індифікаційний код кімнати. Потім

формує список користувачів, який додає до словника, який у свою чергу передається до бази даних.

```
createChatRoom(String username, username2){
  String chatRoomId = _getChatRoomId(username, username2);
  List<String> users = [username, username2];
  Map<String, dynamic> chatRoomMap = {
    "users" : users,
    "chatRoomId" : chatRoomId,
  };
  _databaseService.createChatRoom(chatRoomId, chatRoomMap);
  return chatRoomId;
}
```

Рисунок 3-18 – Метод створення екрану бесіди

Екран бесіди з іншим користувачем

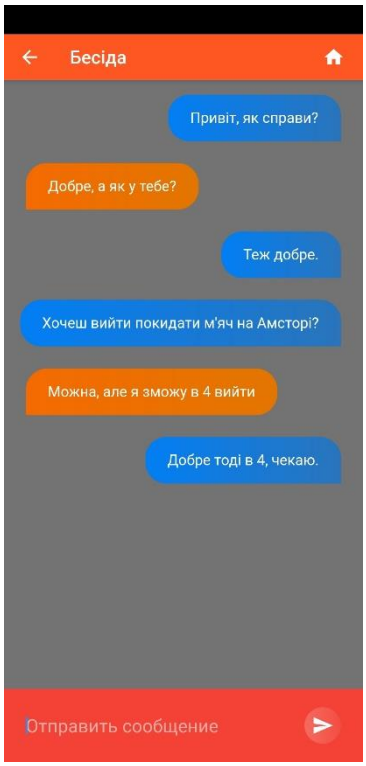


Рисунок 3-19 – Екран бесіди

Екран бесіди у якому користувачі можуть листуватися. Листування користувачів зберігаєть у базі даних та вивантажуеться при переході на даний екран. Дані з бази даних сотуються за часом завантаження.

Код представленого екрану:

Поле вводу повідомлення. При натискані на детектор натискання, виконується функція відправки повідомлення до бази даних та відображення донного повідомлення на екрані.

```
Expanded(
  child: TextField(
    controller: _sendController,
    style: TextStyle(
      color: Colors.white,
      fontSize: 20,
    ), // TextStyle
    decoration: InputDecoration(
        hintText: "Отправить сообщение",
        hintStyle: TextStyle(
          color: Colors.white54,
          fontSize: 20,
        ), // TextStyle
        border: InputBorder.none
    ), // InputDecoration
  ), // TextField
), // Expanded
GestureDetector(
  onTap: () {
    _sendMassageFunc();
  },
```

Рисунок 3-20 – Поле вводу повідомлення

Функція відправки повідомлення до бази даних. Перевіряє чи заповнене поле повідомлення, після чого формує словник із повідомлення, імені відправника та часу відправлення у мілісекундах. У результаті відправляє дані повідомлення у базу даних та очищає поле повідомлення.

```
_sendMassageFunc(){
  if(_sendController.text.isNotEmpty){
    var _sendMassage = _sendController.text;
    Map<String, dynamic> messageMap = {
      'message' : _sendMassage,
      'sendBy'  : Constants.myName,
      'time'    : DateTime.now().microsecondsSinceEpoch,
    };
    _databaseBloc.addChatMassage(widget.chatRoomId, messageMap);
    _scrollController.jumpTo(_scrollController.position.maxScrollExtent);
    _sendController.clear();
  }
}
```

Рисунок 3-21 – Функція відправки повідомлення

Метод додавання даних повідомлення у базу даних. Дані додаються у відповідну колекцію у базі даних в залежності від індифікаційного коду поточної кімнати.

```
addChatMessages(String chatRoomId, messageMap) async{
  await _firebaseFirestore.collection("chatRoom")
      .doc(chatRoomId)
      .collection("messages")
      .add(messageMap).catchError((e){
        print(e.toString());
  });
}
```

Рисунок 3-22 – Метод додавання даних повідомлення у базу даних

Список повідомлень відправлених користувачами. У коді екрану прослуховується потік даних повідомлень, при змінні якого, до списку додаються повідомлення.

```
class ChatRoomTile extends StatelessWidget {
  final String username;
  final String chatRoomId;
  const ChatRoomTile(this.username, this.chatRoomId, {Key key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: (){
        Navigator.pushNamed(context, '/conversation', arguments: chatRoomId);
      },
      child: Container(
        padding: EdgeInsets.symmetric(horizontal: 24, vertical: 16),
        child: Row(
          children: [
            Container(
              height: 40,
              width: 40,
              alignment: Alignment.center,
              decoration: BoxDecoration(
                color: Colors.blue,
                borderRadius: BorderRadius.circular(40),
              ),  // BoxDecoration
              child: Text("${username.substring(0,1).toUpperCase()}"),
            ),  // Container
            SizedBox(width: 8,),
            Text(username),
          ],
        ),  // Row
      ),  // Container
    );  // GestureDetector
  }
```

Рисунок 3-23 – Список повідомлень відправлених користувачами

Мапа

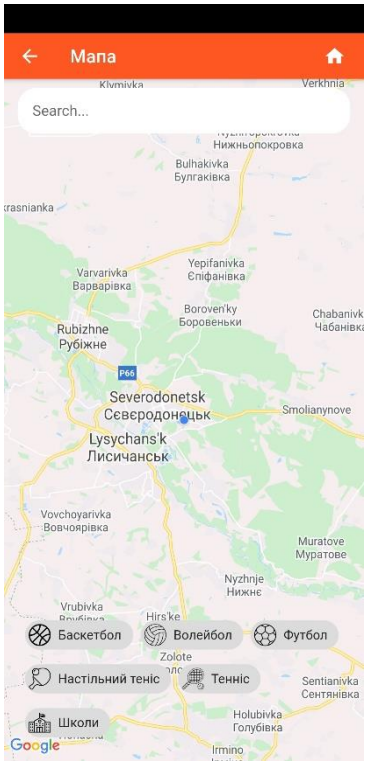


Рисунок 3-24 – Мапа

Екран мапи містить карту, поле пошуку міст та фільтри тренувань. Поле пошуку та фільтри позіціонуються на мапою, що поліпшує огляд мапи.

Код представленого екрану:

Даний віджет дозволяє відображати мапу на екрані. Головними параметрами віджету є:

маркери, дозволяє відображати заркери на мапі;

початковаПозиція, встановлює початкову позицію камери на мапі, у застосунку отримується поточне положення пристрою;

onMapCreated, встановлює контроллер мапи для керевання нею;

onLongPless, виконувана функція при довгому натисканні, у застосунку додавання власного маркеру.

```
GoogleMap(
  mapType: MapType.normal,
  markers: Set<Marker>.of(mapBloc.markers),
  myLocationEnabled: true,
  myLocationButtonEnabled: false,
  zoomControlsEnabled: false,
  zoomGesturesEnabled: true,
  tiltGesturesEnabled: false,
  initialCameraPosition: CameraPosition(
      target: LatLng(mapBloc.getCurrentLocation().latitude, mapBloc.getCurrentLocation().longitude),
      zoom: 10
  ),  // CameraPosition
  scrollGesturesEnabled: true,
  onMapCreated: (GoogleMapController controller){
    _mapController.complete(controller);
  },
  onLongPress: _addMarker,
),  // GoogleMap
```

Рисунок 3-25 – Віджет мапи

Самописний віджет фільтрів. Отримує назву фільтру на тип фільтрування. В залежності в назви фільтру вивантажує картинки з відповідної папки для зображень. При виборі фільтру виконується метод перемикання маркерів на мапі.

```
class _FilterChipItems extends StatelessWidget{
  MapBloc mapBloc;
  String label;
  String type;
  _FilterChipItems(this.mapBloc, {Key key, this.label, this.type}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return FilterChip(
    ├─label: Text(label),
    ├─avatar: SvgPicture.asset(
    (type != "school") ? "assets/markers/sports/$type.svg" : "assets/markers/edu_inst/$type.svg",
          width: 50,
          height: 50,
      ),  // SvgPicture.asset
      onSelected: (val)=>mapBloc.togglePlaceType(type, val),
      selected: mapBloc.placesType.contains(type),
      selectedColor: Colors.blue,
    );  // FilterChip
  }
}
```

Рисунок 3-26 – Самописний віджет фільтрів

Автодоповнення при пошуку населеного пункту

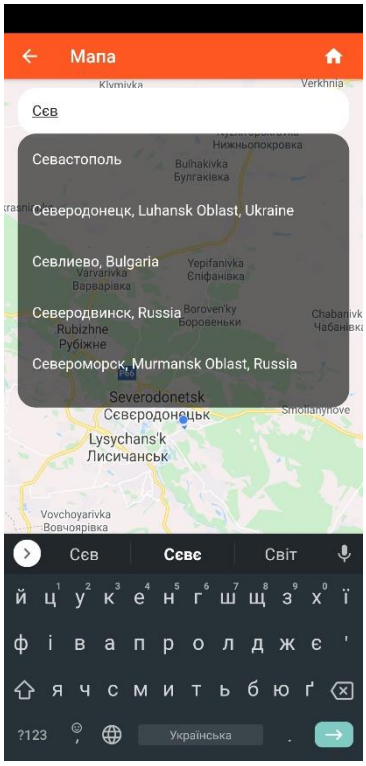


Рисунок 3-27 – Автодоповнення

Екран дозволяє шукати міта у полі для пошуку. Знайдені міста формують список, який відображається над мапою. При натисканні на відповідний елемент камера переміщається на відповідне місто.

Код представленого екрану:

Поле для пошуку міста. При наборі запиту в поле, виконується метод пошуку міст, у результаті чого відображаються знайдені міста у списку.

```
Expanded(
└─ child: TextField(
    focusNode: _focusNode,
    controller: _searchController,
    cursorColor: Colors.black,
    keyboardType: TextInputType.text,
    textInputAction: TextInputAction.go,
    decoration: InputDecoration(
        border: InputBorder.none,
        contentPadding: EdgeInsets.symmetric(horizontal: 15),
        hintText: "Search..."
    ),  // InputDecoration
    onChanged: (value) {
      print(value);
      mapBloc.searchPlaces(value);
      },
  ),  // TextField
),  // Expanded
```

Рисунок 3-28 – Поле для пошуку міста

- У данній частині код виконується відображення списку знайдених міст, який формується з результатів пошуку у полі searchResults, при зміні якого перемальовується список міст.

```
Container(
  height: 300.0,
  child: ListView.builder(
    itemCount: mapBloc.searchResults.length,
    itemBuilder: (context, index){
      return ListTile(
        title: Text(mapBloc.searchResults[index].description,
          style: TextStyle(
            color: Colors.white,
          ),  // TextStyle
        ),  // Text
        onTap: (){
          mapBloc.setSelectedLocation(
              mapBloc.searchResults[index].placeId
          );
        },
      );  // ListTile
    },
  ),  // ListView.builder
)  // Container
```

Рисунок 3-29 –Ввідображення списку знайдених міст

- Метод який використовується у функції пошуку міст. Він відсилає запит на віддалений сервер по API, у якому змінюється запит пошуку. У результаті отримується відповідь з якої потрібнг дістати поле predictions. У результаті повертається список знайдених міст.

```
Future<List<PlaceSearch>> getAutoComplete(String search) async{
  var url = 'https://maps.googleapis.com/maps/api/place/autocomplete/json?input=$search&types=(cities)&key=$_key';
  var response = await http.get(
      Uri.https(
          Uri.parse(url).authority,
          Uri.parse(url).path,
          Uri.parse(url).queryParameters
      )
  );
  var json = convert.jsonDecode(response.body);
  var jsonResult = json['predictions'] as List;
  return jsonResult.map((place) => PlaceSearch.fromJson(place)).toList();
}
```

Рисунок 3-30 – Метод пошуку міст за допомогою API

Додавання маркера спортивного майданчика

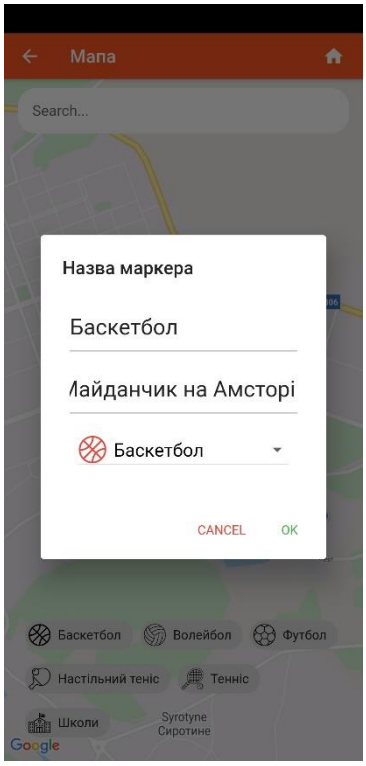


Рисунок 3-31 – *Додавання маркера*

При довгому натискані на мапу, користувачеві відображається діалогове вікно у якому можно встановити назву та опис маркеру, а також тип тренувань по цьому маркеру. При підтвердженні, дані відпрвляються до бази даних, в іншому випадку маркер не додається.

Код представленого екрану:

Функція, яка виконується при довгому натисканні. У якості парамерку приймає поточні координати. У першу чергу виконується функція показу діалогового вікна, після отримує результати винанання функції та в залежності від результату відображає про додавання маркеру у базу даних або не додавання.

Рисунок 3-32 – Функція при довгому натисканні

Список типів тренувань. Список пипів та їх назв отримується з словника типів тренувань. З кожного елементу словника формується елемент списку

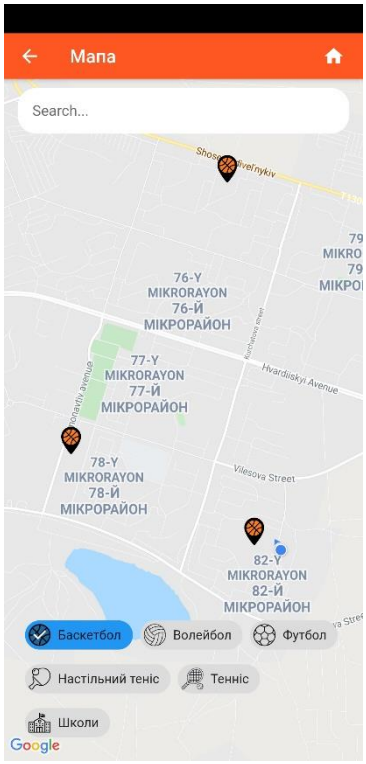
Рисунок 3-33 – Список типів тренувань

- Кнопка підтвердження додавання маркеру до бази даних. При натискані формується словник з назви маркеру, його опису, типу, координат та міста знаходження маркеру. Після формування дані додаються до бази даних, поля вводу чистяться та повертається до мапи.

```
TextButton(
  style: TextButton.styleFrom(
      primary: Colors.green,
      textStyle: TextStyle(
          color: Colors.white
      )  // TextStyle
  ),
  child: Text('OK'),
  onPressed: () {
    _codeDialog = true;
    List<double> latLng = [point.latitude, point.longitude];
    _resultMarkerDialog = {
      'name' : _markerNameController.text,
      'description' : _markerDescController.text,
      'type' : _chosenValueToDB,
      'latLng' : latLng,
      'location' : _currentPlace.name.substring(0, _currentPlace.name.indexOf(',')),
    };
    _databaseBloc.addMarker(_resultMarkerDialog);
    _markerNameController.clear();
    _markerDescController.clear();
    Navigator.pop(context);
  },
),  // TextButton
```

Рисунок 3-34 – Кнопка підтвердження додавання маркеру

Фільтр спортивних майданчиків в поточному населеному пункті



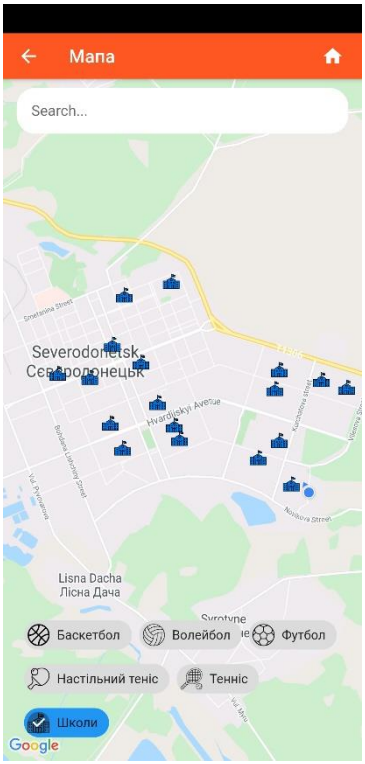


Рисунок 3-35 Маркери майданчиків Рисунок 3-36 Маркера навчальних закладів

Фільтри дозволяють шукати маркери в залежності від типу тренувань. Існують спортивні майданчики та начальні заклади. Спортивні майданчики додають самі користувачі, натомість навчальні заклади шукаються автоматично.

Код представленого екрану:

Прослуховування поля bounds, яке містить пов’язані маркери. При зміні дпного поля буде відмовідно оновлена камера мапи за допомогою контролера.

```
boundsSubscription = mapBloc.bounds.stream.listen((bounds) async {
  final GoogleMapController controller = await _mapController.future;
  controller.animateCamera(
    CameraUpdate.newLatLngBounds(bounds, 0.0),
  );
});
```

Рисунок 3-37 – Прослуховування поля bounds

Метод зміни відображення маркерів на мапі. Отримує тип маркеру та перамерт вибору, чи був вибран відповідний фільтр. Масив типів маркерів не пустий то потрібно пройтися по масиву та знайти відповідні місця у базі даних або через API, якщо це насчальні заклади. Потім потрібно створити маркери до

усіх знайдених місць та додати до масиву маркерів. У завершення потрібно встановити поле bounds для відображення маркерів на мапі.

```
togglePlaceType(String value, bool selected) async{
  if(selected){
    placesType.add(value);
  }else{
    placesType.remove(value);
  }
  if(placesType.isNotEmpty){
    for(int i = 0; i < placesType.length; i++){
        List<Place> tempPlaces = [], places = [];
        for(int i = 0; i < placesType.length; i++){
          tempPlaces = await _placesService.getPlaces(
              selectedLocationStatic.geometry.location.lat,
              selectedLocationStatic.geometry.location.lng,
              placesType[i]
          );
          places = new List.from(places)..addAll(tempPlaces);
        }
        markers = [];

        if(places.length > 0){
          for(int i = 0; i < places.length; i++){
            var newMarker = _markersService.createMarketFromPlace(places[i]);
            markers.add(newMarker);
          }
        }

        var locationMarker = _markersService.createMarketFromPlace(selectedLocationStatic);
        markers.add(locationMarker);

        var _bounds = _markersService.bounds(Set<Marker>.of(markers));

        bounds.add(_bounds);
    }
  }
```

Рисунок 3-38 – Метод зміни відображення маркерів на мапі

Метод створення маркеру відповідно до отриманого місця.

```
Marker createMarketFromPlace(Place place){
  var markerId = place.name;

  return Marker(
    markerId: MarkerId(markerId),
    draggable: false,
    infoWindow: InfoWindow(
      title: place.name,
      snippet: place.vicinity,
    ),  // InfoWindow
    position: LatLng(place.geometry.location.lat, place.geometry.location.lng),
  );  // Marker
}
```

Рисунок 3-39 – Метод створення маркеру

ВИСНОВОКИ З ДИПЛОМНОЇ РОБОТИ

Практичним значенням отриманих результатів є розробка мобільного застосунку «SportSearch», який допоможе спортсменам знаходити місця для тренувань у будь-яку пору року.

У теплі періоди більшість спортсменів тренуються на свіжому повітрі. Для заняття баскетболом волейболом та іншими видами активного спорту, спортсменам необхідно знати про доступні майданчики, заходи проводяться на них, кількість осіб, що буде тренуватися. Навпаки, в холодні періоди спортсмени бажають продовжувати тренування, але через холод проводити тренування на вулиці не завжди можливо. Тому вони шукають доступні зали, в яких можна тренуватися. Як і в теплі періоди їм потрібно знати, які зали доступні, заходи проводяться в них і кількість осіб, яке буде тренуватися.

Наукова новизна результатів дослідження полягає у тому, що серед конкурентів у даній сфері Android застосунків, існує мала кількість аналогів, які допомагають вирішити проблеми з пошуком місця для занять активними видами спорту як у теплі так і холодні пори року.

Було з’ясувано, які потреби можуть бути у кінцевого користувача. Проаналізувано необхідність у впровадженні даних потреб користувача. Визначено які технології будуть задовільними при розробці застосунку. Розроблено застосунок відповідно до отриманих результатів дослідження. Оцінено результати виконаної роботи та сформувано висновок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мобільні додатки, сучасні тенденції розвитку [Електронний ресурс]. -Режим доступу: https://shd.com.ua/mobile_app-or-site - Дата звернення: 18.04.2021 р.
2. Android или iOS: что выбрать для старта карьеры в мобильной разработке [Електронний ресурс]. -Режим доступу: https://rb.ru/opinion/android-ili-ios/ Дата звернення: 20.04.2021 р.
3. [React Native vs. Xamarin vs. Ionic vs. Flutter: Which is best for Cross-Platform Mobile App Development? [Електронний ресурс]. -Режим доступу: https://www.apptunix.com/blog/frameworks-cross-platform-mobile-app-development/ Дата звернення: 23.04.21 р.
4. Flutter. Плюсы и минусы [Електронний ресурс]. -Режим доступу: https://habr.com/ru/company/simbirsoft/blog/441766/ Дата звернення: 25.04.2021 р.
5. Про Flutter, кратко: Основы [Електронний ресурс]. -Режим доступу: https://habr.com/ru/post/430918/ Дата звернення: 30.04.2021 р.
6. Что такое Flutter и почему вы должны изучать его в 2020 году [Електронний ресурс]. -Режим доступу: https://habr.com/ru/post/481326/ Дата звернення: 1.05.2021 р.
7. Types of databases [Електронний ресурс]. -Режим доступу: https://www.tutorialspoint.com/Types-of-databases Дата звернення: 1.05.2021 р.
8. Types of Database Management Systems [Електронний ресурс]. -Режим доступу: https://www.c-sharpcorner.com/UploadFile/65fc13/types-of-database-management-systems/ Дата звернення: 6.05.2021 р.
9. Flutter in Action : підручник / за ред. Eric Windmill. – Manning Publications; 1st edition (December 10, 2019). – 310 с.
10. Top 5 Databases for your Flutter Application [Електронний ресурс]. -Режим доступу: https://blog.back4app.com/flutter-database/ Дата звернення: 10.05.2021 р.

11. How Flutter Works [Електронний ресурс]. -Режим доступу: https://buildflutter.com/how-flutter-works/ Дата звернення: 13.05.2021 р.
12. Architectural layers [Електронний ресурс]. -Режим доступу: https://flutter.dev/docs/resources/architectural-overview#architectural-layers Дата звернення: 14.05.2021 р.
13. Dart in Action : підручник / за ред. Eric Windmill. – Manning Publications; 1 edition (February 1, 2013). – 428 с.
14. Widgets [Електронний ресурс]. -Режим доступу: https://flutter.dev/docs/resources/architectural-overview#widgets Дата звернення: 18.05.2021 р.
15. Widget-state [Електронний ресурс]. -Режим доступу: https://flutter.dev/docs/resources/architectural-overview#widget-state Дата звернення: 19.05.2021 р.
16. State-management [Електронний ресурс]. -Режим доступу: https://flutter.dev/docs/resources/architectural-overview#state-management Дата звернення: 24.05.2021 р.
17. Platform-channels [Електронний ресурс]. -Режим доступу: https://flutter.dev/docs/resources/architectural-overview#platform-channels Дата звернення: 28.05.2021 р.

Додатки Скорочений лістинг застосунку

Додаток А Екран авторизації

```
class LoginScreen extends StatefulWidget {
  LoginScreen({Key key}) : super(key: key);
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();

  TextEditingController _usernameController = new TextEditingController();
  TextEditingController _emailController = new TextEditingController();
  TextEditingController _passwordController = new TextEditingController();

  String _username;
  String _email;
  String _password;
  bool _showEmailAuth = false;
  bool _showLogin = true;
  QuerySnapshot _snapshot;

  AuthService _authService = AuthService();
  Validate _validate = Validate();
  DatabaseBloc _databaseBloc = DatabaseBloc();
  SharedService _sharedService = SharedService();

  StreamSubscription<User> _loginStateSubscription;

  @override
  void initState() {
    final _authBloc = Provider.of<AuthBloc>(context, listen: false);
    _loginStateSubscription = _authBloc.currentUser.listen((fbUser) {
    if(fbUser != null){
      Navigator.of(context).pushNamed('/home');
    }
    });
    super.initState();
  }

  @override
  void dispose() {
    _loginStateSubscription.cancel();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    var authBloc = Provider.of<AuthBloc>(context);
    var _width = MediaQuery.of(context).size.width;
```

```
// Start Functions bloc
void clearText(){
  _emailController.clear();
  _passwordController.clear();
}

//Login function
void _loginButtonAction(String type) async {
  switch(type){
    case "emailForm":
      _email = _emailController.text;
      _password = _passwordController.text;

      _sharedService.saveUserEmailSharedPreferences(_email);
      if(_email.isEmpty || _password.isEmpty) {
        clearText();
        return;
      }
      authBloc.loginEmail(_email.trim(), _password.trim());
      clearText();
      break;
    case "facebook":
      authBloc.loginFacebook();
      break;
    case "google":
      authBloc.loginGoogle();
      break;
    case "anonymous":
      authBloc.loginAnonymously();
      break;
  }
}
/// Registering with Email function
void _registerButtonAction() async {
  _username = _usernameController.text;
  _email = _emailController.text;
  _password = _passwordController.text;


  if(_username.isEmpty || _email.isEmpty || _password.isEmpty) return;

  Map<String, String> userInfoMap = {
    "username" : _username,
    "email" : _email,
  };
  User user;
  await _authService.registerEmailPass(_email.trim(), _password.trim()).then(
        (value){
          if(value != null){
            user = value;
            _sharedService.saveUserLoggedInSharedPreferences(true);
            _sharedService.saveUsernameSharedPreferences(_username);
```

```
                _sharedService.saveUserEmailSharedPreferences(_email);
                _sharedService.saveUserPasswordSharedPreferences(_password);
                _databaseBloc.uploadUserInfo(userInfoMap);
                clearText();
              }
            }
      );
    }

    _showAuthToggle(){
      setState(() {
        _showEmailAuth = !_showEmailAuth;
      });
    }
    // Stop Functions bloc

    // Start Widgets bloc
    Widget _logo(){
      return Padding(
        padding: const EdgeInsets.only(top: 120),
        child: Container(
          child: Align(
            child: Text(
              "SportSearch",
              style: TextStyle(
                fontSize: 40,
                fontWeight: FontWeight.bold
              ),),
          ),
        ),
      );
    }

    Widget _input(Icon icon, String hint, TextEditingController controller, bool obscure,{Function
validateFunc}){
      return Container(
        padding: EdgeInsets.symmetric(horizontal: 20.0),
        child: TextFormField(
          validator: (val) => null,
          controller: controller,
          obscureText: obscure,
          style: TextStyle(
            fontSize: 20,
            color: Colors.white
          ),
          decoration: InputDecoration(
            hintStyle: TextStyle(
              fontWeight: FontWeight.bold,
              fontSize: 20,
              color: Colors.white30,
            ),
            hintText: hint,
```

```
        focusedBorder: OutlineInputBorder(
          borderSide: BorderSide(color: Colors.indigoAccent, width: 3),
        ),
        enabledBorder: OutlineInputBorder(
          borderSide: BorderSide(color: Colors.white54, width: 1),
        ),
        prefixIcon: Padding(
          padding: EdgeInsets.symmetric(horizontal: 10.0),
          child: IconTheme(
            data: IconThemeData(
              color: Colors.white,
            ),
            child: icon,
          ),
        )
      ),
    ),
  );
}

Widget _button(String text, Function func){
  return ElevatedButton(
    onPressed: () {
      print(text);
      if(text == "Login"){
        func("emailForm");
      }else{
        func();
      }
    },
    style: ElevatedButton.styleFrom(
      primary: Colors.indigo,
      onPrimary: Theme.of(context).primaryColor,
      elevation: 5
    ),
    child: Text(
      text,
      style: TextStyle(
        fontWeight: FontWeight.bold,
        color: Theme.of(context).primaryColor,
        fontSize: 20,
      ),
    ),
  );
}

Widget _form(String buttonText, Function btnF){
  return Container(
    child: Column(
      children: [
        Form(
          key: _formKey,
```

```
          child: Column(
            children: [
              (_showLogin)
                  ?
              Container() :
              Padding(
                padding: EdgeInsets.only(top: 50),
                child: _input(Icon(Icons.account_circle), "Username", _usernameController, false),
              ),
              Padding(
                padding: EdgeInsets.only(bottom: 20, top: (_showLogin) ? 50 : 20),
                child: _input(Icon(Icons.email), "Email", _emailController, false),
              ),
              Padding(
                padding: EdgeInsets.only(bottom: 20),
                child: _input(Icon(Icons.lock), "Password", _passwordController, true),
              ),
            ],
          ),
        ),
        Padding(
          padding: EdgeInsets.symmetric(horizontal: 20),
          child: Container(
            height: 50,
            width: MediaQuery.of(context).size.width,
            child: _button(buttonText, btnF),
          ),
        ),
      ],
    ),
  );
}

Widget _authButtonSvg({@required String iconName, @required Function func}){
  var _btnSize = _width / 4 * .5;
  return Expanded(
    child: ElevatedButton(
      style: ElevatedButton.styleFrom(
        primary: Colors.indigo,
        padding: EdgeInsets.zero
      ),
      onPressed: () => (iconName != "email") ? func(iconName) : func(),
      child:  SvgPicture.asset(
        "assets/icons/$iconName\.svg",
        color: Colors.white,
        width: _btnSize,
        height: _btnSize,
      ),
    )
  );
}
```

```
Widget _authButtons(){
  return Padding(
    padding: EdgeInsets.all(10),
    child: Row(
      children: [
        _authButtonSvg(iconName: "email", func: _showAuthToggle),
        _authButtonSvg(iconName: "google", func: _loginButtonAction),
        _authButtonSvg(iconName: "facebook", func: _loginButtonAction),
        _authButtonSvg(iconName: "anonymous", func: _loginButtonAction),
      ],
    ),
  );
}
// Stop Widgets bloc

return Scaffold(
  backgroundColor: Theme.of(context).primaryColor,
  body: SingleChildScrollView(
    child: Column(
      children: [
        _logo(),
        _authButtons(),
        (!_showEmailAuth)
            ?
            Container()
            :
        (_showLogin)
              ?
            Column(
              children: [
                _form("Login", _loginButtonAction),
                Padding(
                  padding: EdgeInsets.all(10),
                  child: GestureDetector(
                    child: Text(
                      "Not registered yet? Register.",
                      style: TextStyle(
                        fontSize: 20,
                        color: Colors.white,
                      ),
                    ),
                    onTap: (){
                      setState(() {
                        _showLogin = false;
                        clearText();
                      });
                    },
                  ),
                )
              ],
            )
              :
```

```
            Column(
              children: [
                _form("Registered", _registerButtonAction),
                Padding(
                  padding: EdgeInsets.all(10),
                  child: GestureDetector(
                    child: Text(
                      "Already registered? Login.",
                      style: TextStyle(
                        fontSize: 20,
                        color: Colors.white,
                      ),
                    ),
                    onTap: (){
                      setState(() {
                        _showLogin = true;
                        clearText();
                      });
                    },
                  ),
                )
              ],
            )
          ],
        ),
      ),
    );
  }
}
```

Додаток Б Екран мапи

```
class MapScreen extends StatefulWidget {
  @override
  _MapScreenState createState() => _MapScreenState();
}

class _MapScreenState extends State<MapScreen> {
  Completer<GoogleMapController> _mapController = Completer();
  StreamSubscription locationSubscription;
  StreamSubscription boundsSubscription;
  DatabaseBloc _databaseBloc = DatabaseBloc();
  TextEditingController _searchController = new TextEditingController();
  TextEditingController _markerNameController =  new TextEditingController();
  TextEditingController _markerDescController =  new TextEditingController();

  FocusNode _focusNode;
  Place _currentPlace;
  Map<String, dynamic> _resultMarkerDialog;
  bool _codeDialog;

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

```
  final mapBloc = Provider.of<MapBloc>(context, listen: false);
  _focusNode = FocusNode();

  locationSubscription = mapBloc.selectedLocation.stream.listen((place) {
    if(place != null){
      setState(() {
        _currentPlace = place;
      });
      _goToPlace(place);
    }
  });

  boundsSubscription = mapBloc.bounds.stream.listen((bounds) async {
    final GoogleMapController controller = await _mapController.future;
    controller.animateCamera(
      CameraUpdate.newLatLngBounds(bounds, 0.0),
    );
  });

  _focusNode.addListener(() {
    if(!_focusNode.hasFocus) _searchController.clear();
  });
}

@override
void dispose() {
  locationSubscription.cancel();
  boundsSubscription.cancel();
  super.dispose();
}


static const Map<String, String> _sportTypes = {
  "Баскетбол"         : "basketball",
  "Волейбол"          : "volleyball",
  "Футбол"            : "football",
  "Настільний теніс"  : "tableTennis",
  "Теннiс"            : "tennis",
};

Future<void> _displayTextInputDialog(BuildContext context, LatLng point) async {
  String _markerName = "";
  String _markerDesc = "";
  String _chosenValue;
  String _chosenValueToDB;

  return showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: Text('Назва маркера'),
          content: SingleChildScrollView(
```

```
child: Column(
  children: [
    Padding(
      padding: const EdgeInsets.all(8.0),
      child: TextField(

        controller: _markerNameController,
        style: TextStyle(
          fontSize: 24,
        ),
        decoration: InputDecoration(hintText: "Введіть назву маркера"),
      ),
    ),
    Padding(
      padding: const EdgeInsets.all(8.0),
      child: TextField(
        controller: _markerDescController,
        style: TextStyle(
          fontSize: 24,
        ),
        decoration: InputDecoration(hintText: "Введіть опис маркера"),
      ),
    ),
    Padding(
      padding: const EdgeInsets.all(8.0),
      child: StatefulBuilder(
        builder: (BuildContext context, StateSetter dropDownState){
          return DropdownButton<String>(
            value: _chosenValue,
            elevation: 5,
            style: TextStyle(color: Colors.black),

            items: _sportTypes.map((String text, image) {
              return MapEntry(
                  text,
                  DropdownMenuItem<String>(
                    value: text,
                    child: _DropdownMenuItem(title: text, image: image,),
                  ));
            }).values.toList(),
            hint: Text(
              "Виберіть тип майданчика",
              style: TextStyle(
                  color: Colors.black,
                  fontSize: 16,
                  fontWeight: FontWeight.w600),
            ),
            onChanged: (String value) {
              dropDownState(() {
                _chosenValue = value;
                _chosenValueToDB = _sportTypes[value];
              });
```

```
                    },
                  );
                }
              ),
            )
          ],
        ),
      ),
      actions: <Widget>[
        TextButton(
          style: TextButton.styleFrom(
              primary: Colors.red,
              textStyle: TextStyle(
                  color: Colors.white
              )
          ),
          child: Text('CANCEL'),
          onPressed: () {
            _codeDialog = false;
            _markerNameController.clear();
            _markerDescController.clear();
            Navigator.pop(context);
          },
        ),
        TextButton(
          style: TextButton.styleFrom(
              primary: Colors.green,
              textStyle: TextStyle(
                  color: Colors.white
              )
          ),
          child: Text('OK'),
          onPressed: () {
            _codeDialog = true;
            List<double> latLng = [point.latitude, point.longitude];
            _resultMarkerDialog = {
              'name' : _markerNameController.text,
              'description' : _markerDescController.text,
              'type' : _chosenValueToDB,
              'latLng' : latLng,
              'location' : _currentPlace.name.substring(0, _currentPlace.name.indexOf(',')),
            };
            _databaseBloc.addMarker(_resultMarkerDialog);
            _markerNameController.clear();
            _markerDescController.clear();
            Navigator.pop(context);
          },
        ),
      ],
    );
  });
}
```

```
@override
Widget build(BuildContext context) {
  final mapBloc = Provider.of<MapBloc>(context);
  final _scaffold = ScaffoldMessenger.of(context);

  List<_FilterChipItems> _filterChipItems(){
    /*_filterChipItems().add(_FilterChipItems(mapBloc, label: "Школи", type: "school",));*/
     var result = _sportTypes.entries.map(
           (e) {
         return _FilterChipItems(mapBloc, label: e.key, type: e.value,);
       }).toList();
     result.add(_FilterChipItems(mapBloc, label: "Школи", type: "school",));
     return result;
  }
  _addMarker(LatLng point){
    _displayTextInputDialog(context, point).then(
           (value){
             if(_codeDialog){
               _scaffold.showSnackBar(
                   SnackBar(
                       content: const Text("Маркер був доданий"),
                       action: SnackBarAction(
                         textColor: Colors.green,
                         label: "ОК",
                         onPressed: _scaffold.hideCurrentSnackBar,
                       )
                   )
               );
             }
             else{
               _scaffold.showSnackBar(
                   SnackBar(
                       content: const Text("Маркер не додано"),
                       action: SnackBarAction(
                         textColor: Colors.green,
                         label: "ОК",
                         onPressed: _scaffold.hideCurrentSnackBar,
                       )
                   )
               );
             }
           }
    );
  }
  return Scaffold(
    resizeToAvoidBottomInset: false,
    appBar: AppBarWidget(title: "Мапа"),
    body: (mapBloc.getCurrentLocation() == null)
        ? Center(
            child: CircularProgressIndicator(),
          )
```

```
        : Builder(
        builder: (context) =>
           Stack(
             children: [
              GoogleMap(
                mapType: MapType.normal,
                markers: Set<Marker>.of(mapBloc.markers),
                myLocationEnabled: true,
                myLocationButtonEnabled: false,
                zoomControlsEnabled: false,
                zoomGesturesEnabled: true,
                tiltGesturesEnabled: false,
                initialCameraPosition: CameraPosition(
                   target: LatLng(mapBloc.getCurrentLocation().latitude,
mapBloc.getCurrentLocation().longitude),
                   zoom: 10
                ),
                scrollGesturesEnabled: true,
                onMapCreated: (GoogleMapController controller){
                  _mapController.complete(controller);
                },
                onLongPress: _addMarker,
              ),
              /// Search TextField
              Positioned(
                top: 10,
                right: 15,
                left: 15,
                child:
                   Stack(
                  children: [
                    Container(
                      decoration: BoxDecoration(
                        color: Colors.white,
                        border: Border.all(
                           color: Colors.transparent
                        ),
                        borderRadius: BorderRadius.all(Radius.circular(20)),
                      ),
                      child: Row(
                        children: [
                          Expanded(
                            child: TextField(
                              focusNode: _focusNode,
                              controller: _searchController,
                              cursorColor: Colors.black,
                              keyboardType: TextInputType.text,
                              textInputAction: TextInputAction.go,
                              decoration: InputDecoration(
                                 border: InputBorder.none,
                                 contentPadding: EdgeInsets.symmetric(horizontal: 15),
                                 hintText: "Search..."
```

```
          ),
          onChanged: (value) {
            print(value);
            mapBloc.searchPlaces(value);
            },
        ),
      ),
    ],
  ),
),
if (mapBloc.searchResults != null
    &&
    mapBloc.searchResults.length != 0)
  Padding(
    padding: const EdgeInsets.only(top: 50),
    child: AnimatedContainer(
      duration: Duration(milliseconds: 50),
      child:
      Stack(
          children: [
        Container(
          height: 300.0,
          width: double.infinity,
          decoration: BoxDecoration(
            color: Colors.black.withOpacity(.6),
            backgroundBlendMode: BlendMode.darken,
            border: Border.all(
                color: Colors.transparent
            ),
            borderRadius: BorderRadius.all(Radius.circular(20)),
          ),
        ),
        Container(
          height: 300.0,
          child: ListView.builder(
            itemCount: mapBloc.searchResults.length,
            itemBuilder: (context, index){
              return ListTile(
                title: Text(mapBloc.searchResults[index].description,
                  style: TextStyle(
                    color: Colors.white,
                  ),
                ),
                onTap: (){
                  mapBloc.setSelectedLocation(
                      mapBloc.searchResults[index].placeId
                  );
                },
              );
            },
          ),
        )
```

```
                        ]
                      ),
                    ),
                  ),
                ],
              )
            ),
            /// Filter bar
            if(_keyboardIsVisible())
            Positioned(
              bottom: 10,
              right: 15,
              left: 15,
              child: Stack(
                children: [
                  Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Wrap(
                      spacing: 8.0,
                      children:
                      _filterChipItems(),
                    ),
                  ),
                ],
              ),
            ),
          ],
        )
    ),
  );
 }

 double getMaxHeight(BuildContext context){
  return (MediaQuery.of(context).size.height - Scaffold.of(context).appBarMaxHeight);
 }

 bool _keyboardIsVisible() {
  return (MediaQuery.of(context).viewInsets.bottom == 0.0);
 }

 Future<void> _goToPlace(Place place) async{
  final GoogleMapController controller = await _mapController.future;
  controller.animateCamera(
    CameraUpdate.newCameraPosition(
      CameraPosition(
        target: LatLng(place.geometry.location.lat, place.geometry.location.lng),
        zoom: 12
      )
    )
  );
 }
}
```

```
class _SystemPadding extends StatelessWidget {
  final Widget child;

  _SystemPadding({Key key, this.child}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    var mediaQuery = MediaQuery.of(context);
    return new AnimatedContainer(
        padding: mediaQuery.viewInsets,
        duration: const Duration(milliseconds: 300),
        child: child);
  }
}

class _DropdownMenuItem extends StatelessWidget {
  final String title;
  final String image;
  _DropdownMenuItem({Key key, @required this.title, this.image}) : super(key:key);

  @override
  Widget build(BuildContext context) {
    return Container(
      child: Row(
        children: [
          SvgPicture.asset("assets/markers/sports/$image.svg", color: Colors.red, height: 30,
width:30,),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Text(
              title,
              style: TextStyle(
                fontSize: 20,
              ),
            ),
          )
        ],
      ),
    );
  }
}

class _FilterChipItems extends StatelessWidget{
  MapBloc mapBloc;
  String label;
  String type;
  _FilterChipItems(this.mapBloc, {Key key, this.label, this.type}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return FilterChip(
      label: Text(label),
```

```
      avatar: SvgPicture.asset(
    (type != "school") ? "assets/markers/sports/$type.svg" : "assets/markers/edu_inst/$type.svg",
          width: 50,
          height: 50,
      ),
      onSelected: (val)=>mapBloc.togglePlaceType(type, val),
      selected: mapBloc.placesType.contains(type),
      selectedColor: Colors.blue,
    );
  }
}
```

Сервіс для взаємодії з Google Places API

```
class PlacesService{
  final _key = 'AIzaSyDz8PAA_xYZB84doK4e4gl31NC9dMlEbNM';

  Future<List<PlaceSearch>> getAutoComplete(String search) async{
    var url =
'https://maps.googleapis.com/maps/api/place/autocomplete/json?input=$search&types=(cities)&key
=$_key';
    var response = await http.get(
        Uri.https(
            Uri.parse(url).authority,
            Uri.parse(url).path,
            Uri.parse(url).queryParameters
        )
    );
    var json = convert.jsonDecode(response.body);
    var jsonResult = json['predictions'] as List;
    return jsonResult.map((place) => PlaceSearch.fromJson(place)).toList();
  }

  Future<Place> getPlace(String placeId) async{
    var url =
'https://maps.googleapis.com/maps/api/place/details/json?place_id=$placeId&key=$_key';
    var response = await http.get(
        Uri.https(
            Uri.parse(url).authority,
            Uri.parse(url).path,
            Uri.parse(url).queryParameters
        )
    );
    var json = convert.jsonDecode(response.body);
    var jsonResult = json['result'] as Map<String, dynamic>;
    return Place.fromJson(jsonResult);
  }

  Future<List<Place>> getPlaces(double lat, double lng, String placeType) async{
    var url =
'https://maps.googleapis.com/maps/api/place/textsearch/json?type=$placeType&location=$lat,$lng
&rankby=distance&key=$_key';
    var jsonResults = [], json;
    while(true){
```

```
      var response = await http.get(
          Uri.https(
              Uri.parse(url).authority,
              Uri.parse(url).path,
              Uri.parse(url).queryParameters
          )
      );
      json = convert.jsonDecode(response.body);
      jsonResults = new List.from(jsonResults)..addAll(json['results']);
      if(json['next_page_token'] != null){
        url =
"https://maps.googleapis.com/maps/api/place/textsearch/json?pagetoken=${json['next_page_token']
}&key=$_key";
      }else{
        break;
      }
    }
    return jsonResults.map((place)=>Place.fromJson(place, placeType: placeType)).toList();
  }
}
```