КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра квантової радіофізики та наноелектроніки

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Ганна КАРЛАШ
« 20 » травня 2023 р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА
на тему:

«РОЗРОБКА МОБІЛЬНОГО НАВІГАЦІЙНОГО ЗАСТОСУНКУ ДЛЯ ВІДВІДУВАЧІВ УНІВЕРСИТЕТУ»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 105 Прикладна фізика
ОНП «Прикладна фізика та наноматеріали»
Сироєжин Валерій Валерійович _____

Науковий керівник:
к.ф.-м. н., доцент
Нечипорук Олексій Юрійович _____

Рецензент:
к. фіз.-м. н., доцент, НТУУ КПІ
Донець Андрій Георгійович _____

Засвідчую, що у цій магістерській роботі
немає запозичень з праць інших авторів без
відповідних посилань,
студент _____

Робота допущена до захисту в ЕК рішенням кафедри квантової радіофізики та наноелектроніки від «19» травня 2023 р., протокол № 19.

Завідувач кафедри _____,
канд. фіз.мат. наук, доцент
Карлаш Ганна Юріївна

Київ 2023

РЕФЕРАТ

Дипломна робота: 50 ст., 7 рис., 10 джерел, 3 додатки.

В роботі описано дослідження реалізації необхідних компонентів для функціонування бажаного мобільного програмного забезпечення, ціль якого полягає у отриманні навігаційної інформації всередині приміщень, використовуючи будь який функціонал мобільного пристрою, такий як: сенсори руху, супутникову систему навігації, чи камеру. Застосунок повинен надавати можливість визначати своє розташування всередині будівлі та побудувати маршрут до потрібного місця призначення, а також відображати додаткову інформацію про місцевість, наприклад, місця відпочинку, їдальні, wc і т.д. Застосунок має простий та зручний інтерфейс для використання користувачами всіх рівнів підготовки та сумісний із популярними мобільними платформами.

ЗАСТОСУНОК ДЛЯ ANDROID ТА IOS ПРИСТРОЇВ, НАПИСАНИЙ НА МОВІ DART З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ FLUTTER, В IDE ANDROID STUDIO, ТА ВКЛЮЧАЄ В СЕБЕ РОБОТУ З GPS, ГІРОСКОПОМ, АКСЕЛЕРОМЕТРОМ ТА МАГНІТОМЕТРОМ.

ЗМІСТ

ВСТУП

Сучасний світ швидко розвивається, тому застосунки стають все більш точними і функціональними. Фреймворк Flutter, який розроблений корпорацією Google, є однією з найбільш перспективних платформ для створення мобільних застосунків, веб сайтів, а також настільних застосунків під Windows, MacOS та Linux, котрий використовує мову програмування Dart [1].

Використаний фреймворк Flutter має вбудовану систему рендерингу, яка дозволяє відображати компоненти інтерфейсу користувача набагато швидше, ніж інші платформи (такі як React Native та Xamarin). Flutter використовує швидку та ефективну двомірну графіку для відображення компонентів інтерфейсу користувача та дозволяє створювати вражаючі ефекти та анімацію. Крім того, він має багатий набір вбудованих віджетів та можливості кастомізації, що дозволяє створювати застосунки з привабливим та професійним дизайном. Слід зазначити, що Flutter дозволяє створювати нативні застосунки, які запускаються на пристроях безпосередньо, що забезпечує кращу продуктивність та швидкість. Він має простий та легкий в освоєнні синтаксис, що дозволяє швидко розробляти застосунки з мінімальними зусиллями [2].

Синтаксис Dart має схожість із мовами C, Python, та JavaScript. Однією з головних переваг мови Dart є те, що вона є компільованою мовою зі сміттєзбіркою. Це означає, що Dart може бути виконаним швидше, ніж інтерпретовані мови програмування, а також що розробникам не потрібно докладати зусиль для управління пам'яттю, так як сміттєзбірник забезпечує автоматичне управління пам'яттю. Dart також має ряд сучасних функцій, таких як асинхронність та стрілкові функції, що полегшують розробку. Flutter дозволяє використовувати ці функції в зв'язці з віджетами, що дозволяє легко та швидко розробляти інтерактивні та анімаційні застосунки [3].

Цією роботою я маю на меті створити інтерактивне програмне забезпечення для мобільних пристроїв (Android та iOS), що використовує гіроскоп, акселерометр, магнітометр та GPS. Мета програми полягає у забезпеченні зручної та ефективної навігації у навчальному закладі, надаючи оптимальний маршрут до потрібного приміщення всередині корпусу за допомогою функціоналу застосунку. Для досягнення цієї мети програма має такі компоненти:

• Для відображення мапи найближчого корпусу університету використовується GPS, який допоможе визначити відстань до його місцезнаходження.

• Для визначення кутової швидкості обертання пристрою необхідний гіроскоп.

• Для визначення прискорення та визначення кількості кроків, щоб встановити нове місцезнаходження користувача, знадобиться акселерометр.

• Для визначення напрямку магнітного поля використовується магнітометр.

Завдяки використанню можливостей Flutter та навчального матеріалу, вказаного в джерелах, я націлений реалізувати функціональний та легкий у користуванні застосунок для навігації в межах університету, який буде актуальним у зв’язку з використанням сучасного фреймворку та сучасної мови програмування. Застосунок складається з багатьох компонентів, кожен з яких продовжують вдосконалювати як на технічному, так і на програмному рівні. Наш університет відвідують нові студенти та іноземці, котрі будуть зацікавлені у зручній навігації під час перших відвідувань, поки не адаптуються.

Не зважаючи на те, що ні цій мові, ні, тим паче, розробці програмного забезпечення для мобільних пристроїв на мові Dart мене не вчили, мене зацікавила реалізація даного проекту, так як сфера моєї теперішньої роботи також стосується сенсорів для навігації зовнішніх незалежних пристроїв. Тому приділю увагу не тільки опису роботи окремих компонентів програми, котрі у подальшому повинні будуть взаємодіяти в одному проекті, але і фізиці кожного елемента мобільного пристрою, для досягнення цілісності розуміння проекту.

Огляд фізичної частини

Global Positioning System, простіше кажучи, GPS, чесно кажучи, неймовірна технологія, яка дозволяє нам визначати наше місцезнаходження на Землі з точністю до 10 метрів. Так, звісно, що такої точності нам не достатньо для роботи програми від одного лише GPS, але про це пізніше.

Така точність GPS можлива завдяки супутникам, що обертаються навколо Землі та передають радіосигнали на частоті 1575,42 МГц. Кожен супутник має точний атомний годинник, що дозволяє передавати інформацію про місцезнаходження, час та інші параметри. Приймач GPS на Землі отримує ці сигнали від різних супутників та використовує їх для визначення свого місцезнаходження.

Технологія GPS ґрунтується на супутниковій геодезії та теорії відносності. Приймач аналізує час, який потрібен для того, щоб сигнал досягнув його від супутника, та розраховує відстань до кожного супутника. Приймач також аналізує точний час, який передається супутником, та використовує його для синхронізації свого власного годинника з годинником супутника. Знаючи відстань до кількох супутників та їх місцезнаходження на орбіті, приймач GPS може визначити своє точне місцезнаходження на поверхні Землі. Технологія GPS вже не є розкішшю, а стала важливою складовою сучасного світу. Вона широко використовується в автомобільній, морській та авіаційній промисловості, де вона допомагає водіям та пілотам орієнтуватись в просторі. Крім того, GPS є надзвичайно важливим інструментом у сфері військової безпеки [4].

Розуміючи те, що GPS використовує сигнали, що передаються супутниками, для визначення місцезнаходження та часу, можемо зробити висновок, що за рахунок кількості супутників, які видно для приймача GPS, та кількості сигналів, які він може отримувати та обробити, спостерігається вплив на точність позиціонування при використані навігаційного пристрою.

Для підвищення точності, можна об'єднати GPS з інерційною системою навігації. Інерційна навігація працює на використанні гіроскопу та акселерометру для вимірювання прискорення та кутового руху навігаційного пристрою. Ці вимірювання використовуються для визначення позиції та швидкості без використання зовнішніх сигналів, таких як GPS. Інерційні системи навігації можуть втрачати точність через помилки вимірювання та дрейфу гіроскопів. Коригування помилок та дрейфу може здійснюватися за допомогою калібрування гіроскопів та акселерометрів, а також з використанням фільтрів Калмана.

Магнітометр використовується для вимірювання магнітного поля Землі і можуть бути використані як додаткове джерело інформації для визначення орієнтації. Окрім переліченого, для отримання додаткової інформації про місцезнаходження та навколишнє середовище використовують камери [5].

Зауважу, що калібрування мобільних сенсорів є важливим для збереження точності. Зокрема, необхідно проводити регулярне калібрування акселерометру та гіроскопу для корекції помилок та зменшення дрейфу. На сьогоднішній день, за рахунок комбінування даних від кількох сенсорів покращують точність та надійність позиціонування мобільного пристрою. У зв’язку з відсутністю необхідної точності GPS для навігації у будівлі, була поставлена задача створити макет застосунку для наглядної оцінки функціоналу мобільних пристроїв, та розробити рішення для досягнення поставленої цілі по навігації у межах будівлі.

Схожі існуючі рішення

Зазвичай люди не знають деталей технології навігації, але я спробую пояснити це простими словами. Карта, яка дає можливість користувачам знаходити місцезнаходження, дивитися зображення з висоти пташиного польоту, чи планувати маршрути – використовує комп'ютери, які перевіряють інтернет на наявність даних про місця, збирають їх та зберігають у великих базах даних. На сьогоднішній день, користувач може надати GPS дані, щоб побудувати маршрут до бажаного місця, яке ввів у поле пошуку, після чого, інтерактивна карта запустить пошук на серверах та поверне результати. Результати показують на карті у вигляді маркерів або іншими символами. Якщо користувач натисне на маркер, то він побачить додаткову інформацію про це місце, таку як: адресу, телефон, відгуки та інші деталі. Крім того, сучасні карти можуть допомогти планувати маршрути, відображати дорожні затори та навіть показувати відео з маршруту. Для цього використовуються дані з GPS-навігації, яка допомагає визначити місцезнаходження користувача та швидкість його руху. Загалом, це дуже корисний інструмент, який допомагає людям легко знаходити інформацію про різні місця та планувати свої маршрути.

Якщо вас цікавить спосіб навігації у приміщенні за допомогою інтерактивної карти, то вам буде корисна ця ідея реалізації. Окрім представленого програмного коду, я додам кілька ідей для вдосконалення цієї цікавої задачі, яка буде особливо корисною для великих будівель, як Червоний корпус КНУ ім. Тараса Шевченка.

Огляд програмної частини

Для написання цієї програми використовується IDE Android Studio [6]. А для роботи з фреймворком Flutter та мовою Dart завантажувався плагін, котрий був приєднаний у налаштуваннях при першому включені IDE [7]. За допомогою цих інструментів можна легко створювати та редагувати свій проект.

Під час створення проекту можна вибрати шаблон застосунку, який відповідатиме потребам, але під час роботи вони використовуватись не будуть.

У проекті Flutter є декілька важливих файлів, які необхідно змінювати:

- *main.dart* – це основний файл, який запускає застосунок. В ньому можна створювати та редагувати віджети, які відображаються у застосунку.
- *pubspec.yaml* – це файл метаданих застосунку, в якому написаний список залежностей та інші налаштування мовою YAML. Його необхідно доповнювати новими бібліотеками та файлами, що використовуються у проекті. Його необхідно заповнити використаними метаданими для запуску застосунку. Дивіться детальніше у Додатку А “Метадані застосунку”.
- *assets* – це папка, в якій зберігаються зображення, шрифти та інші ресурси.
- *lib* – це папка, в якій зберігається основний код. Також у цій папці можуть знаходитись додаткові файли для розбиття задач на окремі аспекти, такі як: робота з алгоритмами обчислення, правила обробки даних, обробка помилок, процеси взаємодії з користувачем та інше.

Щоб зробити застосунок, який буде виглядати чудово на різних пристроях та забезпечуватиме зручну навігацію для користувачів, необхідно розуміти принципи мобільної розробки, такі як: responsive design та navigation. У Flutter є багато готових віджетів та компонентів, які можна використовувати. Освоєння Dart та Flutter є цікавим та захоплюючим, тому час, витрачений на експериментування та розробку проекту, може бути значним через різноманітність функціоналу.

Розробка інструментів для роботи з датчиками мобільного пристрою

Розробка інструментів для взаємодії з компонентами мобільного пристрою є важливою складовою розробки сучасних застосунків. GPS, гіроскоп, акселерометр та магнітометр дозволяють отримувати різноманітну інформацію про стан пристрою та його розташування в просторі, що може бути використано для створення необхідних функцій у застосунку.

GPS дозволяє отримувати інформацію про місцезнаходження користувача, що надає можливість позбавити користувача виконувати ручний пошук карти будівлі у застосунку. Гіроскоп та акселерометр дозволяють визначити орієнтацію пристрою та виміряти його рух. Магнітометр дозволяє визначити магнітне поле навколо пристрою, що дає можливість удосконалити навігацію.

Також акселерометр дозволяє вимірювати прискорення руху пристрою в трьох вимірах, що може бути використано для відображення швидкість руху пристрою, напрямок його руху або виміряти силу впливу зовнішніх факторів на пристрій. Таким чином, використання акселерометра може допомогти розширити можливості програми та зробити її більш корисною для користувача.

Отже, всі етапи розробки необхідних інструментів для взаємодії з мобільними компонентами є важливими для досягнення потенційного мобільного навігаційного застосунку, який дозволить оцінити реалістичність роботи такої програми. Було прийняте рішення не використовувати мобільну камеру у цьому проекті через незручності під час його використання. Незручність полягає в тому, що для використання мобільної камери в застосунку потрібно тримати пристрій перед обличчям, що може демотивувати користувача використовувати його.

Етап 1. Отримання GPS даних

Написаний мною програмний код та результат його виконання подано у Додатку Б “Ескізи компонентів для застосунку”, у Розділі 1 “Отримання GPS даних”. Перейдемо до опису цього ескізу.

Після запуску застосунку, функція runApp() приймає в себе віджет, який будується кореневим елементом застосунку MaterialApp, який є вбудованим в Flutter і забезпечує базовий матеріальний дизайн застосунку. Функція main() запускає MyApp() в MaterialApp для побудови головного інтерфейсу користувача. Клас MyApp є становим віджетом, який наслідується від StatefulWidget [8]. Цей віджет містить три змінні стану _latitude, _longitude і _error. Значення змінних _latitude і _longitude використовуються для збереження даних про місцезнаходження, тоді як _error зберігає помилку, яка виникає під час спроби отримати місцезнаходження.

У методі initState() здійснюється ініціалізація програми та перевірка на дозвіл користувача на отримання даних про місцезнаходження. Якщо дозвіл не надано, метод requestPermission() викликається для запиту дозволу, після чого результат перевіряється і зберігається в змінну permission. Якщо дозвіл було надано, метод _getCurrentLocation() викликається для отримання даних про місцезнаходження.

Метод _getCurrentLocation() використовує бібліотеку geolocator для отримання місцезнаходження користувача [9]. Якщо місцезнаходження успішно отримане, значення _latitude і _longitude оновлюються, а змінна _isLoading встановлюється в false. Якщо сталася помилка, значення _error оновлюється з текстом помилки.

Метод build() є частиною фреймворку Flutter і викликається щоразу, коли необхідно перебудувати інтерфейс користувача. Залежно від значення змінної _isLoading відображає завантаження або координати користувача. Якщо при отриманні координат сталася помилка, або їх не було надано, змінна _error містить текст помилки, який буде відображений на екрані. В кращому випадку, на екрані буде відображатись одержані значення координат.

Етап 2. Визначення найближчого корпусу до користувача

Написаний мною програмний код та результат його виконання подано у Додатку Б “Ескізи компонентів для застосунку”, у Розділі 2 “Визначення найближчого корпусу до користувача”. Перейдемо до опису цього ескізу.

Ідея цього компоненту полягає в тому, аби визначити, який корпус університету знаходиться найближче до користувача, щоб автоматично увімкнути карту корпусу, у якому знаходиться користувач. Програма використовує бібліотеку geolocator, яка дозволяє отримати доступ до геоданих пристрою. Програма запитує дозвіл на використання геоданих при першому включенні. Якщо користувач не дає дозволу, або якщо програма не може отримати дані про місцезнаходження, програма буде працювати в ручному режимі без визначення найближчого корпусу університету. Якщо користувач дозволяє доступ до геоданих, програма отримує поточні координати розташування пристрою та визначить найближчий корпус університету з наданих їй даних у програмі та запустить карту корпусу у застосунку.

При запуску програми викликається метод main(), який створює екземпляр MaterialApp, в якому встановлюється екран застосунку, реалізований класом MyApp. Клас MyApp є нащадком класу StatefulWidget і містить в собі два обов'язкових методи: createState() та build(). Метод createState() створює екземпляр класу _MyAppState, який містить логіку застосунку.

В конструкторі класу _MyAppState створюються три змінні: _latitude, _longitude та _error, які відповідають за зберігання координат користувача, повідомлення про помилку та флаг, що вказує, чи завантажується застосунок в даний момент. Значення змінних _latitude та _longitude встановлюються після успішного отримання координат користувача, а змінна _error заповнюється в разі помилки.

Для визначення координат користувача програма використовує бібліотеку geolocator, яка дозволяє взаємодіяти з GPS-модулем пристрою. Метод _requestPermission() перевіряє наявність дозволу на використання GPS і в разі відмови відправляє запит на його отримання. Після успішного отримання дозволу викликається метод _getCurrentLocation(), який отримує поточні координати користувача та знаходить найближчий об'єкт на мапі, обчислюючи відстань між координатами користувача та кожним об'єктом на мапі.

Відстань обчислюється за допомогою методу distanceBetween() бібліотеки geolocator, а найближчий об'єкт визначається як об'єкт з найменшою відстанню до поточної геопозиції користувача. Для знаходження найближчого об'єкту використовується цикл for, який проходить по списку координат об'єктів і обчислює відстань між поточною геопозицією користувача і кожним об'єктом. Якщо відстань до поточного об'єкту менша за відстань до попереднього найближчого об'єкту, то поточний об'єкт стає найближчим. У методі _getCurrentLocation() спочатку отримується поточна геопозиція користувача за допомогою методу getCurrentPosition() бібліотеки geolocator. Після отримання геопозиції користувача обчислюється найближчий об'єкт за допомогою циклу for, який проходить по списку координат об'єктів objectCoordinates та порівнює відстань до кожного об'єкту. Для кожного елементу масиву виконується функція Geolocator.distanceBetween(), яка повертає відстань між поточним місцезнаходженням користувача та цим об'єктом. Якщо відстань до поточного об'єкту менша, ніж поточна відстань closestDistance, то цей об'єкт стає новим об'єктом і його індекс зберігається у змінній _closestObjectIndex. Після завершення циклу, програма використовує значення _closestObjectIndex, щоб визначити назву та координати найближчого об'єкта для відображення на екрані. У методі build() відображається геопозиція користувача, зображення найближчого об'єкту та координати цього об'єкту. Якщо при обчисленні геопозиції виникає помилка, то змінній _error присвоюється текст помилки, а змінній _isLoading присвоюється значення false.

Етап 3. Використанням сенсорів для обертання зображення

Написаний мною програмний код та результат його виконання подано у Додатку Б “Ескізи компонентів для застосунку”, у Розділі 3 “Використанням сенсорів для обертання зображення”. Перейдемо до опису цього ескізу.

Хочу зауважити, що у остаточній версії застосунку буде використовуватись зображення схеми мапи приміщення. Вона повинна містити встановлений кут нахилу відносно північного напрямку. Оскільки мені не відомі кути нахилу схем мап відносно північного полюсу для досягнення відповідності з реальністю, я залишив це питання для подальшого вдосконалення наступним розробникам.

Ціль цієї програми – реалізувати роботу компаса для зображень мап корпусів, використовуючи гіроскоп, акселерометр та магнітометр мобільного пристрою. Програма використовує бібліотеку sensors_plus для доступу до даних датчиків [10]. Код налаштовує обробники подій датчиків та визначає градуси нахилу та повороту пристрою відносно початкового положення. Використовуючи отримані дані, програма обчислює кут повороту, який використовується для повороту зображення відповідно до орієнтації пристрою.

Основна логіка програми полягає в наступному: дані з акселерометра використовуються для визначення нахилу пристрою, а потім ці дані комбінуються з даними магнітометра для визначення напрямку. Далі, цей напрямок використовується для повороту картинки на екрані.

В програмі створюється клас MyApp, який наслідується від StatefulWidget. Клас містить два поля типу double, які відповідають за нахил і орієнтацію мобільного пристрою.

У функції initState() додається прослуховувач подій акселерометра та магнітометра, щоб програма могла відстежувати зміни в значеннях сенсорів та змінювати положення зображення на екрані.

У змінній indent використовується ширина екрану пристрою для розрахунку проміжку між зображенням та лівим і правим краєм екрану. Також у програмі є функція _onAccelerometerEvent(), яка обчислює кут нахилу мобільного пристрою в радіанах, і функція _onMagnetometerEvent(), яка спочатку обчислює початковий кут орієнтації пристрою відносно північного полюсу, а потім коригує його в залежності від діапазону від 90 до 270 градусів та виправляє кут орієнтації відносно початкового кута орієнтації.

В функції build() створюється Scaffold, який містить Stack з зображенням, яке повертається з функції Image.asset(). Зображення обертається на кут, який обчислюється у функції _onAccelerometerEvent(), за допомогою параметра Transform.rotate().

Етап 4. Масштабування та межі переміщення мапи

Написаний мною програмний код та результат його виконання подано у Додатку Б “Ескізи компонентів для застосунку”, у Розділі 4 “Масштабування та межі переміщення мапи”. Перейдемо до опису цього ескізу.

У функції main() створюється початковий екран за допомогою класу MyApp, який є нащадком класу StatefulWidget і може змінювати свій вміст залежно від взаємодії з користувачем. Стан застосунку, такий як положення, нахил і масштаб, визначається в класі _MyAppState. Метод initState() відслідковує зміни на акселерометрі та магнітометрі та викликає відповідні методи для оновлення стану. Клас _MyAppState містить методи для обробки подій, пов'язаних зі збільшенням чи зменшенням масштабу, а також для зміни положення зображення.

Для отримання даних з акселерометра та магнітометра використовується бібліотека sensors_plus, а для побудови інтерфейсу користувача – "flutter/material.dart" та "flutter/services.dart". При отриманні даних від акселерометра, програма рахує кут нахилу пристрою. При отриманні даних від магнітометра, програма рахує кут повороту пристрою відносно північного напрямку. За допомогою отриманих даних кута нахилу та кута повороту, повертає зображення на екрані.

Окрім використання датчиків, таких як акселерометр і магнітометр, щоб відстежувати нахил та поворот пристрою, ця програма містить інтерактивні функції, що дозволяють користувачу пересувати зображення на екрані та масштабувати його.

Функція _onPanUpdate викликається кожен раз, коли відбувається оновлення переміщення пальця користувача на екрані. Відбувається зчитування величини зміщення по вісі x та y. Обчислюється нове положення зображення на площині відносно його поточного положення, з урахуванням зчитаного зміщення. Величина зміщення обмежується значеннями indent за допомогою методу clamp. Нове положення зображення зберігається в змінну _position, яка потім

використовується в Transform.translate для переміщення зображення на екрані. Всі дії відбуваються в методі setState, щоб спричинити оновлення екрану з новим значенням _position та зображенням в новому положенні.

Дві функції _increaseScale та _decreaseScale необхідні для збільшення та зменшення масштабу зображення. Вони можуть дозволити користувачеві зручно налаштувати розмір зображення, щоб воно відповідало його потребам та вподобанням. До того ж, це може допомогти покращити користувацький досвід, зокрема для людей зі зниженим зором, які можуть збільшити зображення для зручного використання застосунку. Усі зміни стану здійснюються в методі setState(), який дозволяє оновити програму.

Клас SystemChrome забезпечує контроль над системними налаштуваннями, такими як орієнтація екрану, на Android та iOS пристроях, в якому використовується функція DeviceOrientation зі значенням portraitUp, яка означає, що програма повинна відображатися тільки в портретній орієнтації з вертикальним розташуванням екрану. Це дозволяє не перезавантажувати інтерфейс програми при зміні орієнтації, але залишає можливість повернути екран у горизонтальний стан для зручного використання інтерактивної карти.

Етап 5. Розробка інтерфейсу для побудови маршруту

Написаний мною програмний код та результат його виконання подано у Додатку Б “Ескізи компонентів для застосунку”, у Розділі 5 “Розробка інтерфейсу для побудови маршруту”. Дана програма виконує наступні дії:

- відображає на мапі два кола, що вказують на початок та кінцеву маршруту;
- при зміні нахилу телефону, зображення повертається в напрямку протилежному до нахилу разом з мітками, що відповідає роботі компасу;
- змінює масштаб зображення та міток за допомогою кнопок керування;
- переміщує зображення та його мітки за допомогою жестів перетягування;

Цей проект складається з двох класів: MyApp та _MyAppState:

Клас MyApp є нащадком класу StatefulWidget, який використовується для створення вікна застосунку та відображення зображення з можливістю його зміни. Цей клас містить метод createState(), який повертає об'єкт класу _MyAppState.

Клас _MyAppState є нащадком класу State<MyApp>, який містить логіку взаємодії з користувачем. У цьому класі відбувається обробка жестів, що дозволяє користувачеві змінювати масштаб та переміщувати зображення. Також в цьому класі використовуються дані, отримані з акселерометра та магнітометра, для повернення зображення у напрямку, протилежному до нахилу телефону. Клас містить також інші змінні та константи, необхідні для правильної роботи програми.

Функція build() складається з кількох вкладених віджетів, що розміщені один на одному в графічному інтерфейсі. Зображення розміщене по центру і зменшується/збільшується за допомогою кнопок zoom in/out. Є дві панелі вибору початку та кінця маршруту, які змінюють положення двох кругів на зображенні. Мітки переміщуються при перетягуванні, обертанні та масштабуванні зображення, так як закріплені на його поверхні.

Етап 6. Реалізація крокоміру за допомогою акселерометра

Написаний мною програмний код та результат його виконання подано у Додатку Б “Ескізи компонентів для застосунку”, у Розділі 6 “Реалізація крокоміру за допомогою акселерометра”. Перейдемо до опису цього ескізу.

Ця програма рахує періодичну зміну показників отриманих від акселерометра при пересуванні користувача з пристроєм. Коли користувач зробить кілька кроків, пристрій видає сигнали, які сприймає акселерометр, а програма, в свою чергу, використовуючи пакет sensors_plus для доступу до цих сигналів, спробує визначити кількість змін значення прискорення. Коли користувач робить крок, зміна значення прискорення збільшується, і якщо зміна значення перевищує задане порогове значення _threshold, програма вважатиме, що був зроблений крок. Кожен раз, коли програма фіксує крок, вона збільшує кількість кроків _stepCount на одиницю. Кількість кроків відображається на екрані у віджеті тексту. Код програми був написаний таким чином, щоб кожен раз, коли змінюється кількість кроків, інформація на екрані також оновлювалась.

У цьому ескізі створений клас MyApp, який успадковує клас StatefulWidget. Усередині класу MyApp створюється клас _MyAppState, який успадковує клас State. У класі _MyAppState визначена змінна _stepCount, яка відстежуватиме кількість кроків користувача, і метод initState(), який викликається під час створення віджета та ініціалізує підписку на події акселерометра.

Коли відбувається подія на акселерометрі – оновлюється значення _stepCount за допомогою методу setState(). Також перевіряється, чи відбувся крок, і якщо так, збільшується лічильник кроків. У методі build() створюємо віджет програми, який складається з рядка, що відображає кількість кроків, і кнопки, яка викликає метод _incrementStepCount() для збільшення лічильника кроків.

У подальшому, ідея цього компоненту полягатиме в тому, щоб за рахунок підрахунку кроків користувача визначати виконання задачі, оновлюючи побудований маршрут на схемі приміщення.

Розробка фінального застосунку

Написаний мною програмний код та результат його виконання подано у Додатку В “Фінальна версія застосунку”. Перейдемо до опису цього ескізу.

Новачку ця програма може здатися складною, проте, знання основ програмування та документації Flutter допоможуть зрозуміти, як все працює. Ця програма, що працює з декількома сенсорами мобільного телефону, надає можливості для створення функціонального мобільного застосунку, який може обробляти дані у реальному часі та взаємодіяти з користувачем. Вона відображає графічні елементи, такі як кнопки, текстові поля та картинки, для цього використовуються різні віджети, які можуть бути змінені за допомогою різноманітних параметрів.

Програма має три кнопки на головній сторінці застосунку. Перша кнопка дозволяє автоматизувати процес пошуку найближчого корпусу за допомогою GPS, щоб уникнути використання ручного режиму пошуку, який викликається після натискання третьої кнопки. Друга кнопка відкриває панель обрання місцезнаходження та цілі напрямку, для оновлення міток на карті. Також під кнопками виводяться дані про зроблені кроки для відлагодження програми.

ВИСНОВКИ

Ця програма є ознайомчим проектом з можливостями реалізації певного функціоналу на мові програмування Dart з використанням фреймворку Flutter. Вона демонструє, як використовувати певні бібліотеки для роботи з мобільними датчиками, як побудувати інтерфейс користувача та взаємодіяти з елементами управління. Програма має такі функції, як відстеження місцезнаходження для визначення найближчого корпусу, відображення двох міток на інтерактивній карті, які були обрані у панелі побудови маршруту, що потребує обрати найближчий кабінет до місцезнаходження користувача та ціль його напрямку. Ця програма може бути використана як основа для розробки мобільного застосунку, пов'язаного з навігацією у межах будівлі.

Що стосується моїх пропозицій для удосконалення цього проекту:

Я раджу використати API Google Maps, щоб помістити зображення схем будинку на карті. Це надасть можливість використовувати функціонал існуючого сервісу, такий як сенсори для управління картою, GPS-трекер, та можливість малювати маршрути на вулицях міста. Про те як працювати з Google Maps, я писав у своїй бакалаврській роботі, де є програмний код на мові JS, на якій було реалізовано побудова ліній на інтерактивній карті, жести та інформаційні вікна. Проаналізувавши її, буде не складно реалізувати аналогічним чином взаємодію з API від Google, так як мова Dart є альтернативою мови JS. Недоліком цієї пропозиції для користувача є необхідність використовувати Інтернет, оскільки цього потребує сервіс Google.

Після того, як програма буде працювати за допомогою інтернет-карти, на ній необхідно вимкнути зайві вікна інтерфейсу, маркери та обов’язково вимикати GPS-трекер під час перегляду схеми будівлі, що встановлена на відповідному їй місці. Відслідковуючи напрямок та підраховуючи кількість зроблених кроків, можна реалізувати переміщення нової мітки місцезнаходження користувача у межах будівлі, доки не буде пройдений побудований їм маршрут.

В моїй програмі, так і в запропонованій реалізації, залишається проблемою визначення поверху. Цю функцію хотілося б реалізувати, використовуючи барометр, але, на жаль, наявність такого сенсору у мобільних пристроях є рідкістю. Саме тому я пропоную вирішити це питання, використовуючи магнітометр та акселерометр, аби визначати на яку висоту був піднятий чи опущений пристрій відносно землі та місця включення застосунку.

Враховуючи, що програмі відоме місце розташування користувача та його цілі, обрані ним при побудові маршруту, буде цікаво реалізований перехід між схемами приміщення за рахунок порівняння розрахованого показника та відносними, що свідчать про перехід на інший поверх, для перемикання інформації про нове місце розташування користувача у застосунку.

Через великий обсяг роботи та обмежений час для реалізації цієї задачі самотужки, а також відсутність команди програмістів, готових приєднатися до проекту, мені вдалося розглянути можливі способи написання мобільного застосунку та проаналізувати задачу, щоб визначити найкращий спосіб для її виконання. На даний час я маю робочу модель програми, що потребує вдосконалення для безпосереднього використання гостями університету.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Mainkar, Prajyot. Google Flutter Mobile Development Quick Start Guide. Packt Publishing Ltd., 2019. – 146 с.

[2] Windmill, Eric. Flutter in Action. Manning Publications Co., 2020. – 368 с.

[3] Sinha, Sanjib. Beginning Flutter 3.0 with Dart: A Beginner to Pro. Learn how to build Advanced Flutter 3.0 Apps. Leanpub, 2022. – 412 с.

[4] Noureldin, A., Karamat, T. B., & Georgy, J. Fundamentals of Inertial Navigation, Satellite-Based Positioning and their Integration. Springer Science & Business Media, 2013. – 323 с.

[5] Manish J. Gajjar. Mobile Sensors and Context-Aware Computing, Elsevier, 2017. – 340 с.

[6] IDE Android Studio (2014). Android Mobile App Developer Tools. [Електронний ресурс]. Режим доступу: https://developer.android.com/

[7] SDK Flutter (2014). Flutter documentation. [Електронний ресурс]. Режим доступу: https://docs.flutter.dev/

[8] Віджети Material Components. Офіційна документація. [Електронний ресурс]. Режим доступу: https://docs.flutter.dev/ui/widgets/material

[9] Geolocator. Pub.dev, Baseflow, 2023, [Електронний ресурс]. Режим доступу: https://pub.dev/packages/geolocator/versions/9.0.2

[10] Sensors_plus. Pub.dev, Flutter Community, 2023, [Електронний ресурс]. Режим доступу: https://pub.dev/packages/sensors_plus/versions/2.0.3

Додаток А. Метадані застосунку

Основні компоненти файлу pubspec.yaml:

1. *name* - назва застосунку ;
2. *description* - опис застосунку ;
3. *version* - поточна версія застосунку ;
4. *environment* - середовище, для якого розробляється застосунок ;
5. *dependencies* - перелік залежностей, необхідних для роботи застосунку ;
6. *dev_dependencies* - перелік залежностей, необхідних тільки для розробки ;

```
name: pathfinder
description: The project was created to implement an indoor navigation
system for mobile devices.
publish_to: 'none'
version: 2023.04

environment:
 sdk: '>=2.12.0 <3.0.0'

dependencies:
 flutter:
   sdk: flutter
 geolocator: ^9.0.2
 sensors_plus: ^2.0.3
 weight_slider: ^1.3.0-nullsafety+1
 location: ^4.4.0
 geodesy: ^0.4.0-nullsafety.0
 cupertino_icons: ^1.0.2

dev_dependencies:
 flutter_test:
   sdk: flutter
 flutter_lints: ^2.0.0

flutter:
 uses-material-design: true
 assets:
   - assets/images/One.jpg
   - assets/images/Two.jpg
   - assets/images/Three.jpg
   - assets/images/Four.jpg
   - assets/images/compass.png
```

Додаток Б. Ескізи компонентів для застосунку

Розділ 1. Отримання GPS даних

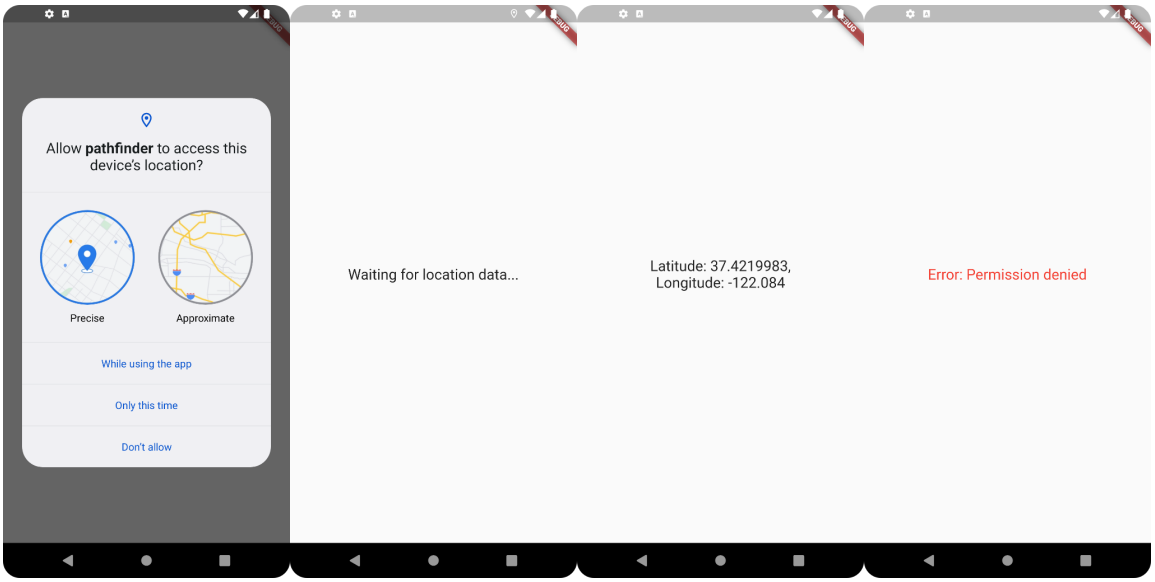

Рис. 1 Стадії роботи: надання дозволу; отримання GPS даних; їх виведення; або помилка.

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

void main() { runApp( MaterialApp( home: MyApp(), ), ); }

class MyApp extends StatefulWidget {
 @override
 _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
 late String _latitude;
 late String _longitude;
 String _error = '';
 bool _isLoading = true;
 @override
 void initState() {
   super.initState();
   _requestPermission();
 }

 Future<void> _requestPermission() async {
   LocationPermission permission = await Geolocator.checkPermission();
   if (permission == LocationPermission.denied) {
     permission = await Geolocator.requestPermission();
     if (permission != LocationPermission.whileInUse &&
         permission != LocationPermission.always) {
       setState(() {
         _isLoading = false;
         _error = 'Permission denied';
       });
       return;
```

```
      }
    }
    _getCurrentLocation();
  }

  Future<void> _getCurrentLocation() async {
    try {
      Position position = await
Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
      setState(() {
        _latitude = position.latitude.toString();
        _longitude = position.longitude.toString();
        _isLoading = false;
        _error = '';
      });
    } catch (e) {
      setState(() {
        _error = e.toString();
        _isLoading = false;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            if (_isLoading)
              const Text(
                'Waiting for location data...',
                style: TextStyle(fontSize: 20),
              )
            else if (_error.isEmpty)
              Text(
                'Latitude: $_latitude,\nLongitude: $_longitude',
                style: const TextStyle(fontSize: 20),
                textAlign: TextAlign.center,
              )
            else
              Text(
                'Error: $_error',
                style: const TextStyle(fontSize: 20, color: Colors.red),
              ),
          ],
        ),
      ),
    );
  }
}
```

Розділ 2. Визначення найближчого корпусу до користувача

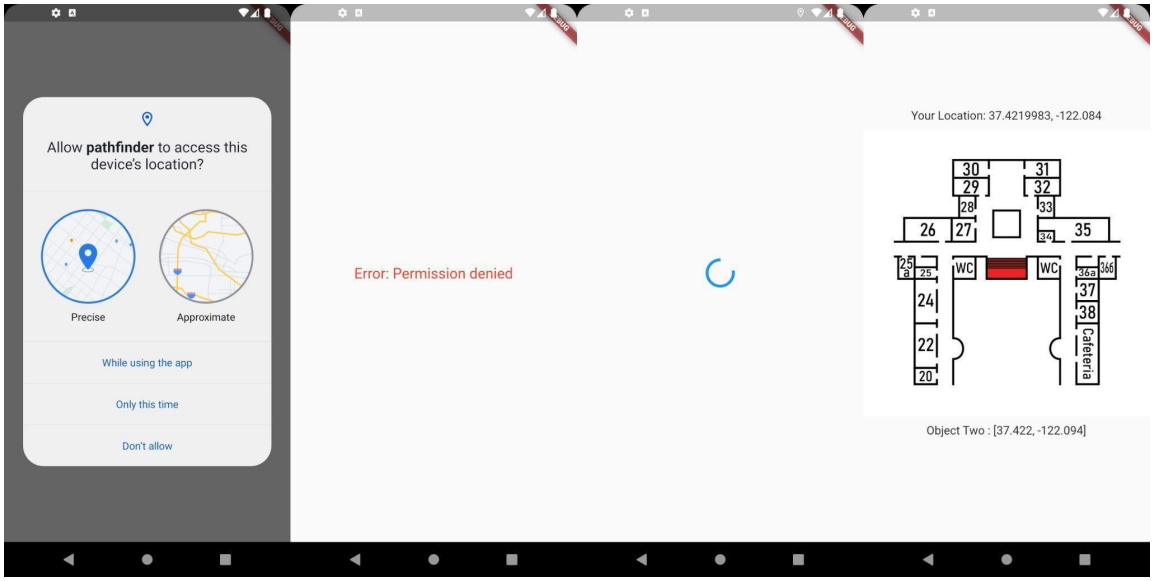

Рис. 2 Стадії роботи: надання дозволу; помилка; або завантаження; та виведення результату.

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

void main() { runApp( MaterialApp( home: MyApp(), ), ); }

class MyApp extends StatefulWidget {
 @override
 _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
 late String _latitude;
 late String _longitude;
 String _error = '';
 bool _isLoading = true;
 int _closestObjectIndex = 0;
 final List<List<double>> objectCoordinates = [
   [37.432, -122.084],
   [37.422, -122.094],
   [37.422, -122.074],
   [37.412, -122.084]
 ];
 final List<String> objectImageNames = ['One', 'Two', 'Three', 'Four'];
 @override
 void initState() {
   super.initState();
   _requestPermission();
 }
```

```
Future<void> _requestPermission() async {
  LocationPermission permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied) {
    permission = await Geolocator.requestPermission();
    if (permission != LocationPermission.whileInUse &&
        permission != LocationPermission.always) {
      setState(() {
        _isLoading = false;
        _error = 'Permission denied';
      });
      return;
    }
  }
  _getCurrentLocation();
}

Future<void> _getCurrentLocation() async {
  try {
    Position position = await
Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
    setState(() {
      _latitude = position.latitude.toString();
      _longitude = position.longitude.toString();
      _isLoading = false;
      _error = '';
    });

    double closestDistance = double.infinity;
    for (int i = 0; i < objectCoordinates.length; i++) {
      List<double> coordinatesGPS = objectCoordinates[i];
      double distance = await Geolocator.distanceBetween(
        position.latitude,
        position.longitude,
        coordinatesGPS[0],
        coordinatesGPS[1],
      );
      if (distance < closestDistance) {
        closestDistance = distance;
        _closestObjectIndex = i;
      }
    }
  } catch (e) {
    setState(() {
      _error = e.toString();
      _isLoading = false;
    });
  }
}
```

```
 @override
 Widget build(BuildContext context) {
   String objectImageName = objectImageNames[_closestObjectIndex];
   return Scaffold(
     body: _isLoading
         ? Center(
       child: CircularProgressIndicator(),
     )
         : _error.isNotEmpty
         ? Center(
       child: Text(
           'Error: $_error',
           style: const TextStyle(fontSize: 20, color: Colors.red),
       ),
     )
         : Center(
       child: Column(
         mainAxisAlignment: MainAxisAlignment.center,
         children: [
           Text(
             'Your Location: $_latitude, $_longitude',
             style: TextStyle(fontSize: 16),
           ),
           SizedBox(height: 10),
           Image.asset('assets/images/$objectImageName.jpg'),
           SizedBox(height: 10),
           Text(
             'Object ${objectImageNames[_closestObjectIndex]} :
${objectCoordinates[_closestObjectIndex]}',
             style: TextStyle(fontSize: 16),
           ),
         ],
       ),
     ),
   );
 }
}
```

Розділ 3. Використанням сенсорів для обертання зображення

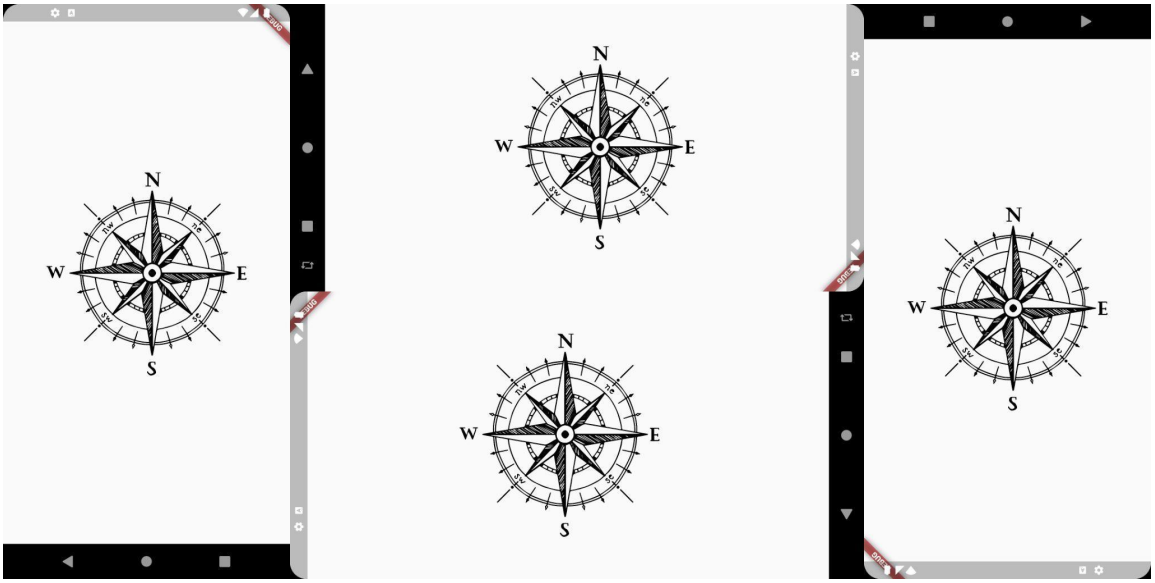

Рис. 3 Демонстрація реалізації повороту зображення при обертанні телефону.

```
import 'package:sensors_plus/sensors_plus.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'dart:math';
void main() {
 WidgetsFlutterBinding.ensureInitialized();
 SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp])
     .then((_) { runApp(MaterialApp( home: MyApp(), )); });
}
class MyApp extends StatefulWidget {
 @override _MyAppState createState() => _MyAppState();
}
class _MyAppState extends State<MyApp> {
 @override
 void initState() {
   super.initState();
   accelerometerEvents.listen(_onAccelerometerEvent);
   magnetometerEvents.listen(_onMagnetometerEvent);
 }
 double get indent => MediaQuery.of(context).size.width *(2-sqrt(2))/4;
 double _tilt = 0, _heading = 0;
 void _onAccelerometerEvent(AccelerometerEvent event) {
   final double x = event.x;
   final double y = event.y;
   final double radians = atan2(y, x) - (pi / 2.0);
   setState(() { _tilt = radians; });
 }
 void _onMagnetometerEvent(MagnetometerEvent event) {
   final double x = event.x;
   final double y = event.y;
   final double z = event.z;
```

```
    final double initialRadians = atan2(y, x);
    final double initialDegrees = initialRadians * 180.0 / pi;
    final double currentRadians = atan2(y * cos(_tilt) - z * sin(_tilt),
x);
    double currentDegrees = currentRadians * 180.0 / pi;
    if (currentDegrees > 90 && currentDegrees < 270) {
      currentDegrees += 180;
    }
    double headingCorrection = initialDegrees - currentDegrees;
    if (headingCorrection < -180) { headingCorrection += 360; }
    else if (headingCorrection > 180) { headingCorrection -= 360; }
    setState(() { _heading = currentDegrees + headingCorrection; });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: GestureDetector(
        child: Stack(
          children: [
            Positioned(
              top: 0,
              bottom: 0,
              left: indent,
              right: indent,
              child: Transform.rotate(
                angle: -_tilt,
                child: Image.asset(
                  'assets/images/compass.png',
                  fit: BoxFit.contain,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Розділ 4. Масштабування та межі переміщення мапи

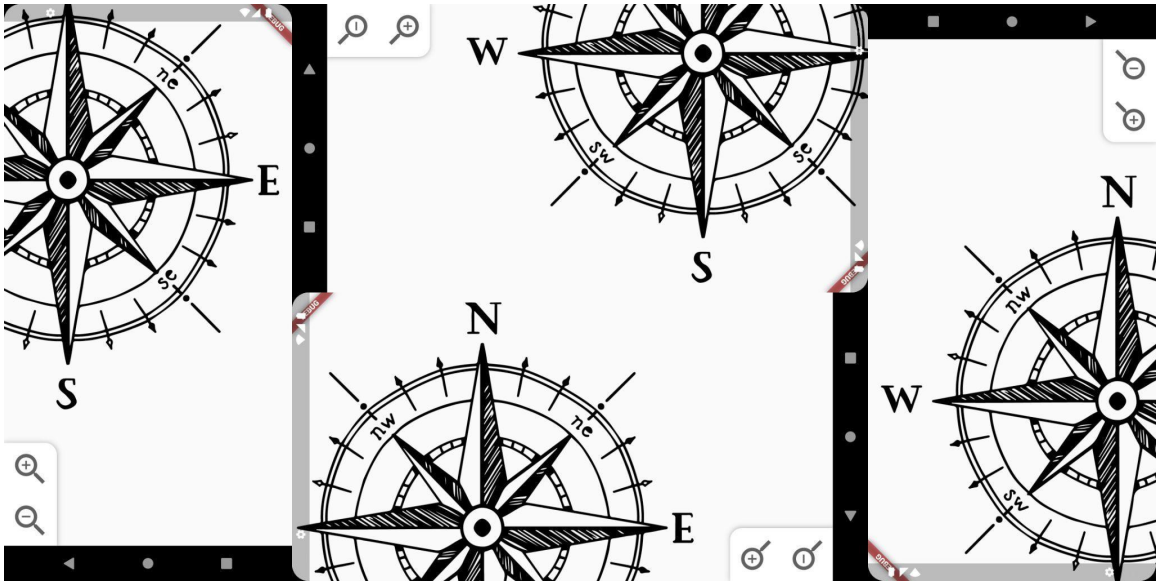

Рис. 4 Демонстрація функціоналу: обертання, пересування та масштабування зображення.

```
import 'package:sensors_plus/sensors_plus.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'dart:math';
void main() {
 WidgetsFlutterBinding.ensureInitialized();
 SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp])
 .then((_) { runApp(MaterialApp( home: MyApp(), )); });
}

class MyApp extends StatefulWidget {
 @override
 _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {

 Offset _position = Offset.zero;
 double get indent => MediaQuery.of(context).size.width * (2-sqrt(2)) /
4;
 double _tilt = 0, _heading = 0, _scale = 1;

 final double minScale = 0.8, maxScale = 2.5, scaleStep = 0.1,
buttonSize = 56, panelHeight = 140;

  @override
 void initState() {
   super.initState();
   accelerometerEvents.listen(_onAccelerometerEvent);
   magnetometerEvents.listen(_onMagnetometerEvent);
 }
```

```
 void _onAccelerometerEvent(AccelerometerEvent event) {
   final double x = event.x, y = event.y;
   final double radians = atan2(y, x) - (pi / 2.0);
   setState(() { _tilt = radians; });
 }

 void _onMagnetometerEvent(MagnetometerEvent event) {
   final double x = event.x, y = event.y, z = event.z;
   final double initialRadians = atan2(y, x);
   final double initialDegrees = initialRadians * 180.0 / pi;
   final double currentRadians = atan2(y * cos(_tilt) - z * sin(_tilt),
x);
   double currentDegrees = currentRadians * 180.0 / pi;
   if (currentDegrees > 90 && currentDegrees < 270) {
     currentDegrees += 180;
   }
   double headingCorrection = initialDegrees - currentDegrees;
   if (headingCorrection < -180) { headingCorrection += 360; }
   else if (headingCorrection > 180) { headingCorrection -= 360; }
   setState(() { _heading = currentDegrees + headingCorrection; });
 }

 void _changeScale(bool increase) {
   setState(() {
     _scale = increase
     ? (_scale >= maxScale ? maxScale : _scale + scaleStep)
     : (_scale <= minScale ? minScale : _scale - scaleStep);
   });
 }

 void _onPanUpdate(DragUpdateDetails details) {
   setState(() {
     double dx = details.delta.dx, dy = details.delta.dy;
     double newX = (_position.dx + dx).clamp( -indent, indent, );
     double newY = (_position.dy + dy).clamp( -indent, indent, );
     _position = Offset(newX, newY);
   });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    body: GestureDetector(
      onPanUpdate: _onPanUpdate,
      child: Stack(
        children: [
          Positioned(
            top: 0, bottom: 0, left: indent, right: indent,
            child: Transform.scale(
              scale: _scale,
              child: Transform.translate(
                offset: _position,
                child: Transform.rotate(
                  angle: -_tilt,
                  child: Image.asset(
                    'assets/images/compass.png', fit: BoxFit.contain,
                  ),
```

```
                ),
              ),
            ),
          ),
          Positioned(
            bottom: 0.0, left: 0.0,
            child: Container(
              height: panelHeight,
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.only(
                  topRight: Radius.circular(16.0),
                ),
                boxShadow: [
                  BoxShadow(
                    color: Colors.grey.withOpacity(0.8), blurRadius: 5.0,
                    spreadRadius: 1.0, offset: Offset(0, 1),
                  ),
                ],
              ),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                children: [
                  Flexible(
                    child: IconButton(
                      icon: Icon(Icons.zoom_in), iconSize: buttonSize,
                      onPressed: () => _changeScale(true),
                      color: Colors.grey[700],
                    ),
                  ),
                  Flexible(
                    child: IconButton(
                      icon: Icon(Icons.zoom_out), iconSize: buttonSize,
                      onPressed: () => _changeScale(false),
                      color: Colors.grey[700],
                    ),
                  ),
                ],
              ),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

Розділ 5. Розробка інтерфейсу для побудови маршруту

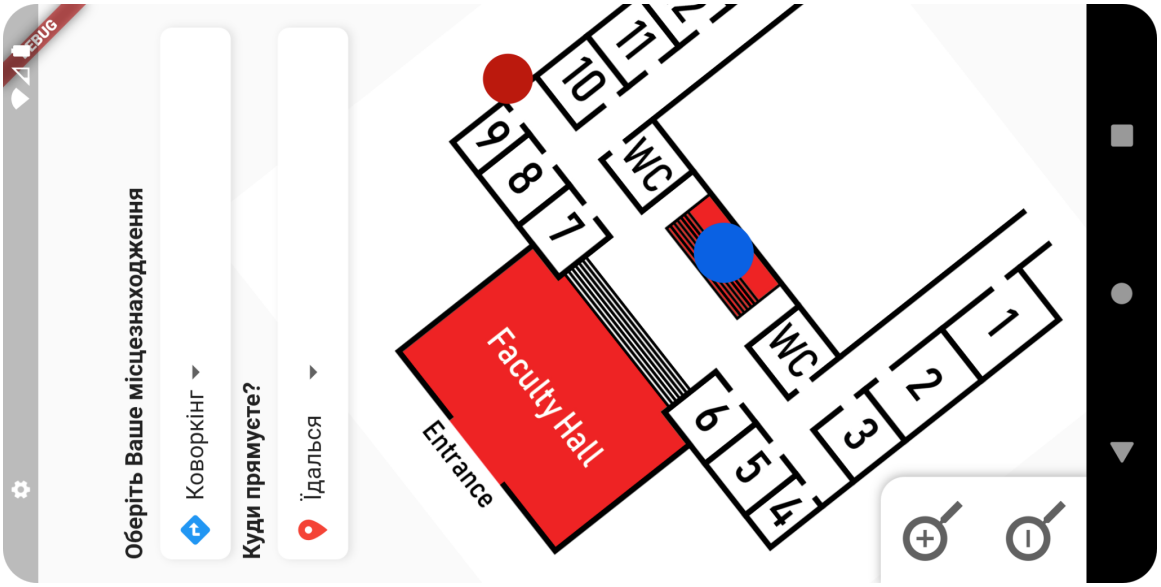

Рис. 5 Демонстрація ручного інтерфейсу для побудови маршруту.

```
import 'package:sensors_plus/sensors_plus.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'dart:math';
void main() {
 WidgetsFlutterBinding.ensureInitialized();
 SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp])
 .then((_) { runApp(MaterialApp( home: MyApp(), )); });
}

class MyApp extends StatefulWidget {
 @override
 _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
 Offset _position = Offset.zero;
 double get indent => MediaQuery.of(context).size.width * (2-sqrt(2))/4;
 double _tilt = 0, _heading = 0, _scale = 1;
 final double minScale = 0.8, maxScale = 2.5, scaleStep = 0.1,
 buttonSize = 56, panelHeight = 140, _circleSize = 20, sizeIndex = 4;
 int _selectedOptionRouteFrom = 0, _selectedOptionRouteTo = 0;

 List<Map<String, dynamic>> _route = [
  {"id": 1, "title": "Їдальня", "top": 365.0, "left": 20.0},
  {"id": 2, "title": "Коворкінг", "top": 340.0, "left": 128.0},
  {"id": 3, "title": "WC", "top": 342.0, "left": 82.0},
 ];
```

```
 @override
 void initState() {
   super.initState();
   accelerometerEvents.listen(_onAccelerometerEvent);
   magnetometerEvents.listen(_onMagnetometerEvent);
 }

 void _onAccelerometerEvent(AccelerometerEvent event) {
   final double x = event.x, y = event.y;
   final double radians = atan2(y, x) - (pi / 2.0);
   setState(() { _tilt = radians; });
 }

 void _onMagnetometerEvent(MagnetometerEvent event) {
   final double x = event.x, y = event.y, z = event.z;
   final double initialRadians = atan2(y, x);
   final double initialDegrees = initialRadians * 180.0 / pi;
   final double currentRadians = atan2(y * cos(_tilt) - z * sin(_tilt),
x);
   double currentDegrees = currentRadians * 180.0 / pi;
   if (currentDegrees > 90 && currentDegrees < 270) {
     currentDegrees += 180;
   }
   double headingCorrection = initialDegrees - currentDegrees;
   if (headingCorrection < -180) { headingCorrection += 360; }
   else if (headingCorrection > 180) { headingCorrection -= 360; }
   setState(() { _heading = currentDegrees + headingCorrection; });
 }

 void _changeScale(bool increase) {
   setState(() {
     _scale = increase
     ? (_scale >= maxScale ? maxScale : _scale + scaleStep)
     : (_scale <= minScale ? minScale : _scale - scaleStep);
   });
 }

 void _onPanUpdate(DragUpdateDetails details) {
   setState(() {
     double dx = details.delta.dx, dy = details.delta.dy;
     double newX = (_position.dx + dx).clamp( -indent, indent, );
     double newY = (_position.dy + dy).clamp( -indent, indent, );
     _position = Offset(newX, newY);
   });
 }
 @override
Widget build(BuildContext context) {
   return Scaffold(
     body: GestureDetector(
       onPanUpdate: _onPanUpdate,
       child: Stack(
         children: [
           Positioned(
             top: 0, bottom: 0, left: indent, right: indent,
             child: Transform.scale(
               scale: _scale,
```

```
                child: Transform.translate(
                  offset: _position,
                  child: Transform.rotate(
                    angle: -_tilt,
                    child: Stack(
                      children: [
                        Center(
                          child: Image.asset(
                            'assets/images/One.jpg', fit: BoxFit.contain,
                          ),
                        ),
                        Positioned(
                     top: _route[_selectedOptionRouteFrom]['top']-sizeIndex/2,
                     left:
_route[_selectedOptionRouteFrom]['left']-sizeIndex/2,
                          child: Container(
                            width: _circleSize+sizeIndex,
                            height: _circleSize+sizeIndex,
                            decoration: BoxDecoration(
                              shape: BoxShape.circle,
                              color: Color(0xFF0A61E3),
                            ),
                          ),
                        ),
                        Positioned(
                          top: _route[_selectedOptionRouteTo]['top'],
                          left: _route[_selectedOptionRouteTo]['left'],
                          child: Container(
                            width: _circleSize, height: _circleSize,
                            decoration: BoxDecoration(
                              shape: BoxShape.circle,
                              color: Color(0xFFBB0D0D),
                            ),
                          ),
                        ),
                      ],
                    ),
                  ),
                ),
              ),
              Positioned(
                bottom: 0.0, left: 0.0,
                child: Container(
                  height: panelHeight,
                  decoration: BoxDecoration(
                    color: Colors.white,
                    borderRadius: BorderRadius.only(
                      topRight: Radius.circular(16.0),
                    ),
                    boxShadow: [
```

```
                    BoxShadow(
                      color: Colors.grey.withOpacity(0.8), blurRadius:
5.0,
                      spreadRadius: 1.0, offset: Offset(0, 1),
                    ),
                  ],
                ),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                  children: [
                    Flexible(
                      child: IconButton(
                        icon: Icon(Icons.zoom_in),
                        onPressed: () => _changeScale(true),
                        iconSize: buttonSize,
                        color: Colors.grey[700],
                      ),
                    ),
                    Flexible(
                      child: IconButton(
                        icon: Icon(Icons.zoom_out),
                        onPressed: () => _changeScale(false),
                        iconSize: buttonSize,
                        color: Colors.grey[700],
                      ),
                    ),
                  ],
                ),
              ),
            ),
            Positioned(
              top: 80.0, left: 16.0, right: 16.0,
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'Оберіть Ваше місцезнаходження',
                    style: TextStyle(
                      fontWeight: FontWeight.bold, fontSize: 16.0,
                    ),
                  ),
                  SizedBox(height: 8.0),
                  Container(
                    decoration: BoxDecoration(
                      color: Colors.white,
                      borderRadius: BorderRadius.circular(8.0),
                      boxShadow: [
                        BoxShadow(
                          color: Colors.black.withOpacity(0.1),
                          blurRadius: 4.0, offset: Offset(0, 2),
                        ),
                      ],
```

```
                    ),
                    child: Row(
                      children: [
                        Padding(
                          padding: const EdgeInsets.all(8.0),
                          child: Icon(
                            Icons.directions, color: Colors.blue,
                          ),
                        ),
                        Expanded(
                          child: DropdownButton<int>(
                            value: _selectedOptionRouteFrom,
                            underline: SizedBox.shrink(),
                            items: _route.map((option) =>
DropdownMenuItem(
                              child: Text(
                                option['title'],
                                textAlign: TextAlign.center,
                              ),
                              value: _route.indexOf(option),
                            ),
                            )
                                .toList(),
                            onChanged: (newValue) {
                              setState(() {
                                _selectedOptionRouteFrom = newValue!;
                              });
                            },
                          ),
                        ),
                      ],
                    ),
                  ),
                ],
              ),
            ),
            Positioned(
              top: 160.0, left: 16.0, right: 16.0,
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'Куди прямуєте?',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: 16.0,
                    ),
                  ),
                  SizedBox(height: 8.0),
                  Container(
                    decoration: BoxDecoration(
                      color: Colors.white,
```

```
                    borderRadius: BorderRadius.circular(8.0),
                    boxShadow: [
                      BoxShadow(
                        color: Colors.black.withOpacity(0.1),
                        blurRadius: 4.0,
                        offset: Offset(0, 2),
                      ),
                    ],
                  ),
                  child: Row(
                    children: [
                      Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: Icon(
                          Icons.location_on,
                          color: Colors.red,
                        ),
                      ),
                      Expanded(
                        child: DropdownButton<int>(
                          value: _selectedOptionRouteTo,
                          underline: SizedBox.shrink(),
                          items: _route.map((option) =>
DropdownMenuItem(
                            child: Text(
                              option['title'],
                              textAlign: TextAlign.center,
                            ),
                            value: _route.indexOf(option),
                          ),
                          )
                              .toList(),
                          onChanged: (newValue) {
                            setState(() {
                              _selectedOptionRouteTo = newValue!;
                            });
                          },
                        ),
                      ),
                    ],
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Розділ 6. Реалізація крокоміру за допомогою акселерометра

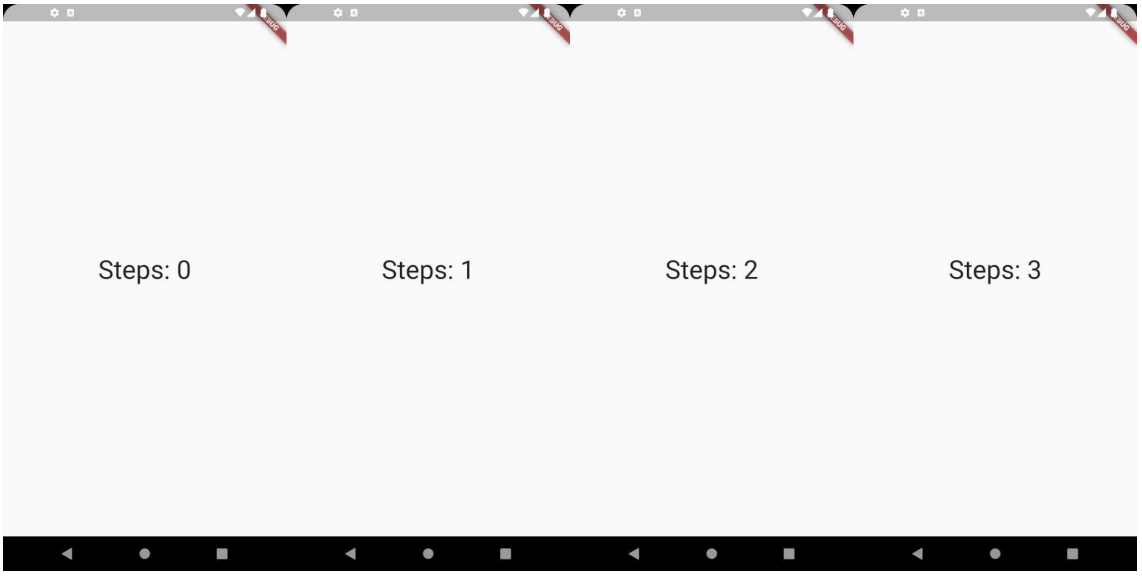

Рис. 6 Демонстрація роботи застосунку на реагування різкі періодичні рухи.

```
import 'package:flutter/material.dart';
import 'package:sensors_plus/sensors_plus.dart';
void main() { runApp(MyApp()); }
class MyApp extends StatefulWidget {
 @override
 _MyAppState createState() => _MyAppState();
}
class _MyAppState extends State<MyApp> {
 int _stepCount = 0;
 double _previousAccValue = 0;
 double _threshold = 10;
 @override
 void initState() {
   super.initState();
   accelerometerEvents.listen((AccelerometerEvent event) {
     double currentAccValue = event.y;
     if (currentAccValue > _previousAccValue && currentAccValue -
_previousAccValue > _threshold) { setState(() { _stepCount++; }); }
     _previousAccValue = currentAccValue;
   });
 }
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Scaffold(
       body: Center(
         child: Text(
           'Steps: $_stepCount', style: TextStyle(fontSize: 36),
         ),
       ),
     ),
   );
 }
}
```

Додаток В. Фінальна версія застосунку

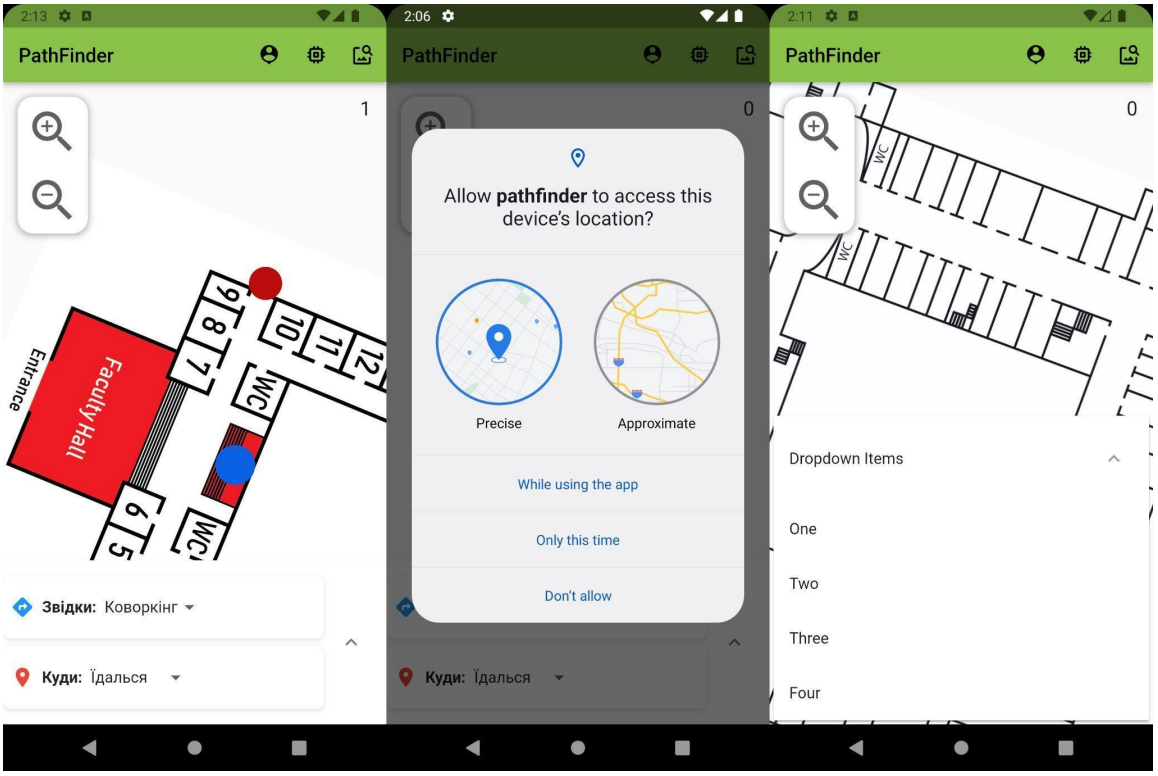

Рис. 7 Демонстрація роботи інтерфейсу користувача.

```
import 'package:sensors_plus/sensors_plus.dart';
import 'package:geolocator/geolocator.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'dart:math';

void main() {
 WidgetsFlutterBinding.ensureInitialized();
 SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp])
     .then((_) { runApp(const MyApp()); });
}

class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'PathFinder',
     debugShowCheckedModeBanner: false,
     theme: ThemeData(
       primarySwatch: Colors.lightGreen,
       visualDensity: VisualDensity.adaptivePlatformDensity,
     ),
     home: const MyHomePage(title: 'PathFinder'),
   );
 }
}
```

```
class MyHomePage extends StatefulWidget {
 const MyHomePage({Key? key, required this.title}) : super(key: key);
 final String title;
 @override State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
 Offset _position = Offset.zero;
 late String _latitude, _longitude;
 bool _isLoading = true,_routeSelection = true,_buildingSelection =
false;
 int _closestObjectIndex = 0, _stepCount = 0;
 int _selectedOptionRouteFrom = 1, _selectedOptionRouteTo = 0;
 double get indent => MediaQuery.of(context).size.width * (2 -sqrt(2))/4;
 double rotationAngle = 0, _tilt = 0, _heading = 0, _scale = 1,
     _previousAccValue = 0, _threshold = 10;
 List<Map<String, dynamic>> _route = [
   { "title": "Їдальня",   "top": 325.0, "left": 20.0  },
   { "title": "Коворкінг", "top": 300.0, "left": 128.0 },
   { "title": "WC",        "top": 302.0, "left": 82.0  },
 ];
 final List<Map<String, List<double>>> objectCoordinates = [
   { 'One':   [37.432, -122.084] }, { 'Two':  [37.412, -122.084] },
   { 'Three': [37.422, -122.074] }, { 'Four': [37.422, -122.094] }
 ];
 final double scaleStep = 0.1, minScale = 0.8, maxScale = 2.5,
     buttonSize = 56, panelHeight = 140, _circleSize = 20, sizeIndex = 4;
 @override
 void initState() {
   super.initState();
   accelerometerEvents.listen(_onAccelerometerEvent);
   magnetometerEvents.listen(_onMagnetometerEvent);
   accelerometerEvents.listen(_pedometer);
 }

 // GPS
 Future<void> _requestPermission() async {
   LocationPermission permission = await Geolocator.checkPermission();
   if (permission == LocationPermission.denied) {
     permission = await Geolocator.requestPermission();
     if (permission != LocationPermission.whileInUse &&
         permission != LocationPermission.always) {
       setState(() {
         _isLoading = false;
         print('Permission denied');
       });
       return;
     }
   }
   _getCurrentLocation();
 }
```

```
Future<void> _getCurrentLocation() async {
  try {
    Position position = await Geolocator.getCurrentPosition(
        desiredAccuracy: LocationAccuracy.high);
    setState(() {
      _latitude = position.latitude.toString();
      _longitude = position.longitude.toString();
      _isLoading = false;
    });
    double closestDistance = double.infinity;
    List<List<double>> values =
        objectCoordinates.map((e) => e.values.first).toList();
    for (int i = 0; i < values.length; i++) {
      List<double> coordinatesGPS = values[i];
      double distance = await Geolocator.distanceBetween(
        position.latitude, position.longitude,
        coordinatesGPS[0], coordinatesGPS[1],
      );
      if (distance < closestDistance) {
        closestDistance = distance;
        _closestObjectIndex = i;
      }
    }
  } catch (e) {
    setState(() {
      _isLoading = false;
    });
  }
}

// Compass
void _onAccelerometerEvent(AccelerometerEvent event) {
  final double x = event.x, y = event.y;
  final double radians = atan2(y, x) - (pi / 2.0);
  setState(() { _tilt = radians; });
}
void _onMagnetometerEvent(MagnetometerEvent event) {
  final double x = event.x, y = event.y, z = event.z;
  final double initialRadians = atan2(y, x);
  final double initialDegrees = initialRadians * 180.0 / pi;
  final double currentRadians = atan2(y * cos(_tilt) - z * sin(_tilt),
x);
  double currentDegrees = currentRadians * 180.0 / pi;
  if (currentDegrees > 90 && currentDegrees < 270) {
    currentDegrees += 180;
  }
  double headingCorrection = initialDegrees - currentDegrees;
  if (headingCorrection < -180) {
    headingCorrection += 360;
  } else if (headingCorrection > 180) {
    headingCorrection -= 360;
  }
  setState(() {
    _heading = currentDegrees + headingCorrection;
  });
}
```

```
// Pedometer
void _pedometer(AccelerometerEvent event) {
  double currentAccValue = event.y;
  if (currentAccValue > _previousAccValue &&
      currentAccValue - _previousAccValue > _threshold) {
    setState(() { _stepCount++; });
  }
  _previousAccValue = currentAccValue;
}

// Loupe
void _changeScale(bool increase) {
  setState(() {
    _scale = increase
        ? (_scale >= maxScale ? maxScale : _scale + scaleStep)
        : (_scale <= minScale ? minScale : _scale - scaleStep);
  });
}

// Move
void _onPanUpdate(DragUpdateDetails details) {
  setState(() {
    double dx = details.delta.dx, dy = details.delta.dy;
    double newX = (_position.dx + dx).clamp( -indent, indent, );
    double newY = (_position.dy + dy).clamp( -indent, indent, );
    _position = Offset(newX, newY);
  });
}

// Route building button
Widget routeBuilder() {
  return Positioned(
    bottom: 0, left: 0, right: 0,
    child: ExpansionPanelList(
      expansionCallback: (int index, bool isExpanded) {
        setState(() { _routeSelection = !isExpanded; });
      },
      children: [
        ExpansionPanel(
          headerBuilder: (BuildContext context, bool isExpanded) {
            return Column(
              children: [
                Container(
                  decoration: BoxDecoration(
                    color: Colors.white,
                    borderRadius: BorderRadius.circular(8.0),
                    boxShadow: [
                      BoxShadow(
                        color: Colors.black.withOpacity(0.1),
                        blurRadius: 4.0, offset: Offset(0, 2),
                      ),
                    ],
                  ),
                  child: Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Row(
```

```
                children: [
                  Icon(
                    Icons.directions, color: Colors.blue,
                  ),
                  SizedBox(width: 8.0),
                  Text(
                    'Звідки:',
                    style: TextStyle(
                      fontWeight: FontWeight.bold, fontSize: 16.0,
                    ),
                  ),
                  SizedBox(width: 8.0),
                  Expanded(
                    child: DropdownButton<int>(
                      value: _selectedOptionRouteFrom,
                      underline: SizedBox.shrink(), items: _route
                          .map(
                            (option) => DropdownMenuItem(
                              child: Text(
                                option['title'],
                                textAlign: TextAlign.center,
                              ),
                              value: _route.indexOf(option),
                              onTap: () {},
                            ),
                          )
                          .toList(),
                      onChanged: (newValue) {
                        setState(() {
                          _selectedOptionRouteFrom = newValue ??
0;

                        });
                      },
                    ),
                  ),
                ],
              ),
            ),
          ),
          SizedBox(height: 8.0),
          Container(
            decoration: BoxDecoration(
              color: Colors.white,
              borderRadius: BorderRadius.circular(8.0),
              boxShadow: [
                BoxShadow(
                  color: Colors.black.withOpacity(0.1),
                  blurRadius: 4.0, offset: Offset(0, 2),
                ),
              ],
            ),
            child: Padding(
              padding: const EdgeInsets.all(8.0),
              child: Row(
                children: [
                  Icon(
```

```
                            Icons.location_on, color: Colors.red,
                          ),
                          SizedBox(width: 8.0),
                          Text(
                            'Куди:',
                            style: TextStyle(
                              fontWeight: FontWeight.bold, fontSize: 16.0,
                            ),
                          ),
                          SizedBox(width: 8.0),
                          Expanded(
                            child: DropdownButton<int>(
                              value: _selectedOptionRouteTo,
                              underline: SizedBox.shrink(),
                              items: _route
                                  .map(
                                    (option) => DropdownMenuItem(
                                      child: Text(
                                        option['title'],
                                        textAlign: TextAlign.center,
                                      ),
                                      value: _route.indexOf(option),
                                    ),
                                  )
                                  .toList(),
                              onChanged: (newValue) {
                                setState(() {
                                  _selectedOptionRouteTo = newValue ?? 0;
                                });
                              },
                            ),
                          ),
                        ],
                      ),
                    ),
                  ),
                ],
              );
            },
            isExpanded: _routeSelection, body: Container(),
          ),
        ],
      ),
    );
  }

  // Map select button
  Widget manualSelectionMode() {
    List<String> keys = objectCoordinates.map((e) =>
e.keys.first).toList();
    return Positioned(
      bottom: 0, left: 0, right: 0,
      child: Card(
        child: ExpansionPanelList(
          expansionCallback: (int index, bool isExpanded) {
            setState(() { _buildingSelection = !isExpanded; });
```

```
},
```

```
          children: [
            ExpansionPanel(
              canTapOnHeader: true,
              headerBuilder: (BuildContext context, bool isExpanded) {
                return const ListTile( title: Text('Dropdown Items'), );
              },
              body: ListView.builder(
                shrinkWrap: true, itemCount: keys.length,
                itemBuilder: (BuildContext context, int index) {
                  return ListTile(
                    title: Text(keys[index]),
                    onTap: () {
                      setState(() {
                        _closestObjectIndex = index;
                        _buildingSelection = false;
                      });
                    },
                  );
                },
              ),
              isExpanded: _buildingSelection,
            ),
          ],
        ),
      ),
    );
  }

  // Interface
  @override
  Widget build(BuildContext context) {
    List<String> keys = objectCoordinates.map((e) =>
e.keys.first).toList();
    String objectImageName = keys[_closestObjectIndex];
    Widget imageWidget =
Image.asset('assets/images/$objectImageName.jpg');
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
        actions: [
          IconButton(
            icon: const Icon(Icons.location_history),
            onPressed: () { _requestPermission(); },
          ),
          IconButton(
            icon: const Icon(Icons.memory_rounded),
            onPressed: () {
              setState(() {
                _routeSelection = !_routeSelection;
                _buildingSelection = false;
              });
            },
          ),
          IconButton(
            icon: const Icon(Icons.image_search_rounded),
            onPressed: () {
```

```
            setState(() {
              _buildingSelection = !_buildingSelection;
              _routeSelection = false;
            });
          },
        ),
      ],
    ),
    body: GestureDetector(
      onPanUpdate: _onPanUpdate,
      child: Stack(
        children: [
          Positioned(
            top: 0, bottom: 0, left: indent, right: indent,
            child: Transform.scale(
              scale: _scale,
              child: Transform.translate(
                offset: _position,
                child: Transform.rotate(
                  angle: -_tilt,
                  child: Stack(
                    children: [
                      Center(
                        child: imageWidget,
                      ),
                      if (_routeSelection) ...[
                        Positioned(
                          top: _route[_selectedOptionRouteFrom]['top'] -
                              sizeIndex / 2,
                          left: _route[_selectedOptionRouteFrom]['left']
-
                              sizeIndex / 2,
                          child: Container(
                            width: _circleSize + sizeIndex,
                            height: _circleSize + sizeIndex,
                            decoration: BoxDecoration(
                              shape: BoxShape.circle,
                              color: Color(0xFF0A61E3),
                            ),
                          ),
                        ),
                        Positioned(
                          top: _route[_selectedOptionRouteTo]['top'],
                          left: _route[_selectedOptionRouteTo]['left'],
                          child: Container(
                            width: _circleSize, height: _circleSize,
                            decoration: BoxDecoration(
                              shape: BoxShape.circle,
                              color: Color(0xFFBB0D0D),
                            ),
                          ),
                        ),
                      ],
                    ],
                  ),
                ),
              ),
```

```
            ),
          ),
        ),
        Positioned(
          top: 15.0, left: 15.0,
          child: Container(
            height: panelHeight,
            decoration: BoxDecoration(
              color: Colors.white,
              borderRadius: BorderRadius.only(
                topLeft: Radius.circular(16.0),
                topRight: Radius.circular(16.0),
                bottomLeft: Radius.circular(16.0),
                bottomRight: Radius.circular(16.0),
              ),
              boxShadow: [
                BoxShadow(
                  color: Colors.grey.withOpacity(0.8), blurRadius:
5.0,
                  spreadRadius: 1.0, offset: Offset(0, 1),
                ),
              ],
            ),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                Flexible(
                  child: IconButton(
                    icon: Icon(Icons.zoom_in),
                    onPressed: () => _changeScale(true),
                    color: Colors.grey[700],
                    iconSize: buttonSize,
                  ),
                ),
                Flexible(
                  child: IconButton(
                    icon: Icon(Icons.zoom_out),
                    onPressed: () => _changeScale(false),
                    color: Colors.grey[700],
                    iconSize: buttonSize,
                  ),
                ),
              ],
            ),
          ),
        ),
        Positioned(
          top: 15.0, right: 15.0,
          child: Text(
            '$_stepCount', style: TextStyle(fontSize: 20),
          ),
        ),
        if (_buildingSelection) manualSelectionMode(),
        if (_routeSelection) routeBuilder(),
      ],
    ),
```

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