

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте на тему:

Разработка мобильного приложения для планирования совместного досуга «GatherRound»

Выполнил студент:

группы #БПМИ2210, 3 курса Захаров Иван Александрович

Захаров Иван Александрович

Принял руководитель проекта:

Иванов Андрей Александрович

Доцент, Базовая кафедра Т-Банка,

Факультет компьютерных наук НИУ ВШЭ

Москва 2025

Содержание

1 Основные термины и определения	4
2 Введение	7
2.1 Цель	7
2.2 Задачи	7
2.3 Актуальность работы	8
2.3.1 Особенности отрасли	8
2.3.2 Ценностные предложения	9
2.3.3 Рисковые гипотезы	9
2.4 План работы	10
3 Обзор литературы	12
3.1 Разработка Android-приложения	12
3.2 Создание современного пользовательского интерфейса	13
3.3 Алгоритмы на графах	13
4 Алгоритм поиска оптимальной станции для встречи	14
4.1 Постановка задачи	14
4.2 Алгоритм решения	14
4.3 Оценка времени работы	16
4.4 Применение	16
4.5 Реализация	17
5 Пользовательский интерфейс	20
5.1 Текстовый ввод	20
5.2 Интерактивная карта метро	21
5.3 Источник SVG	21
5.4 Загрузка и отображение SVG-карты метро	21

5.5	Обработка кликов. Настройка контроллера взаимодействия с JavaScript	22
5.6	Отображение пеших маршрутов	22
6	Заключение	25
6.1	Результаты	25
6.2	Развитие работы	26
Список литературы		28

1 Основные термины и определения

1. **Android** — мобильная операционная система, основанная на модифицированной версии ядра Linux. Наиболее широко используемая версия Android разрабатывается компанией Google. Является наиболее широко используемой операционной системой в мире [1].
2. **Dokka** — инструмент документирования API для Kotlin. Распознает комментарии KDoc в Kotlin и комментарии Javadoc в Java. Dokka может генерировать документацию в нескольких форматах, включая собственный современный формат HTML, различные варианты Markdown и Javadoc HTML в Java [2].
3. **Espresso** — фреймворк, позволяющий создавать автоматизированные сценарии для тестирования пользовательского интерфейса [3].
4. **Glide** — библиотека для загрузки и отображения мультимедиа в Android-приложениях [4].
5. **Jetpack Compose** — рекомендуемый Android современный фреймворк для разработки пользовательского интерфейса. Разрабатывается Google, имеет открытый исходный код [8].
6. **JUnit** — фреймворк для модульного тестирования программного обеспечения на языке Java [9].
7. **Kotlin** — кроссплатформенный, статически типизированный, объектно-ориентированный язык программирования. Работает поверх Java Virtual Machine и разрабатывается компанией JetBrains. Отмечен Google как приоритетный язык в разработке под Android [10].

8. **LiveData** — класс, представляющий собой хранилище данных, работающее по принципу паттерна Observer (наблюдатель). С ним можно выполнять два основных действия: помещать в него объекты и подписываться на него, чтобы получать информацию об объектах, которые помещают в хранилище. LiveData умеет определять, активен подписчик или нет, и отправлять данные только активным подписчикам [20].
9. **Mockito** — фреймворк для тестирования программного обеспечения на языке Java. Позволяет эффективно создавать «заглушки» - экземпляры классов с заданными свойствами, которые используются тестируемым кодом. Часто они не должны быть полнофункциональными — от них требуется конкретное предсказуемое поведение для упрощения процесса тестирования [21].
10. **Model-View-ViewModel (MVVM)** — архитектурный шаблон в разработке ПО, который облегчает разделение разработки графического интерфейса пользователя (GUI) от разработки бизнес-логики или внутренней логики (модели) таким образом, чтобы представление не зависело от какой-либо конкретной платформы модели [12].
11. **Moshi** — библиотека для парсинга JSON в Java или Kotlin-классы и обратного преобразования классов в JSON [13].
12. **Retrofit** — библиотека, которая упрощает работу с сетевыми запросами в приложениях на Android. Поддерживает асинхронные запросы, обрабатывает JSON [19].
13. **Room** — библиотека, предоставляющая удобную обертку для работы с базой данных SQLite [22].
14. **SVG** (Scalable Vector Graphics — «масштабируемая векторная гра-

фика») — язык разметки для создания изображений и текстов в масштабируемой векторной графике [14].

15. **User Interface (UI)** — интерфейс, обеспечивающий передачу информации между пользователем и программными компонентами приложения [16].
16. **ViewModel** — класс, позволяющий активностям и фрагментам сохранять необходимые им объекты при изменении конфигурации (например, повороте экрана) и не пересоздавать их заново [23].

2 Введение

«GatherRound» — приложение, созданное для поиска мероприятий и мест для посещения, а также упрощения организации встреч компаний друзей на культурных мероприятиях.

Идея создания приложения возникла из наблюдения за сложностями, с которыми сталкиваются компании людей при выборе события и места встречи, учитывая разбросанность по городу. Часто процесс организации затягивается, в том числе из-за того, что некоторые из участников тратят значительно больше времени на дорогу, чем другие.

Приложение предлагает возможность поиска мероприятия или места для посещения по заданным станциям метро нахождения одного человека или нескольких людей — это помогает принять решение, что посетить на досуге.

2.1 Цель

Цель разработки «GatherRound» — предоставить пользователям инструмент, который сделает поиск мероприятий быстрым и удобным, упростит процесс организации совместных мероприятий и повысит посещаемость культурных событий.

2.2 Задачи

Задачи проекта разделяются на две части: исследовательскую и программную, а именно:

- Исследовательская часть
 - Изучить основы языка Kotlin.
 - Изучить основы Jetpack Compose.

- Изучить алгоритмы поиска кратчайших путей в гарфах.
 - Разработать оптимальный алгоритм поиска вершины в графе, равноудаленной от заданного множества вершин.
- Программная часть
 - Реализовать функцию построения оптимального маршрута между станциями метро.
 - Реализовать получение данных о мероприятиях и местах их проведения с использованием API KudaGo.ru.
 - Реализовать алгоритм нахождения мероприятия, время в дороге до которого у всех участников примерно одинаково.
 - Реализовать пользовательский интерфейс приложения.
 - Реализовать и внедрить в приложение интерактивную карту метро.
 - Внедрить в приложение построение пеших маршрутов с использованием Yandex MapKit SDK.

2.3 Актуальность работы

2.3.1 Особенности отрасли

- Широкое распространение мобильных устройств и приложений для планирования досуга:
 - Исследование Департамента информационных технологий Москвы 2023 [17] показывает, что почти 90% москвичей не представляют свою жизнь без смартфона. Жители столицы активно используют цифровые устройства для решения повседневных задач, включая планирование досуга, обучение и работу.

- Исследование Rambler&Co 2024 [18] показывает, что более трети россиян хотели бы иметь виртуального помощника по поиску интересных событий и мероприятий. Для поиска информации о досуге 53% опрошенных читают анонсы и дайджесты в онлайн-медиа и на сервисах по продаже билетов, а 20% подписаны на тематические каналы в социальных сетях и мессенджерах.
- Возможные сложности в организации встречи компаний знакомых вследствие длительной изоляции во время Covid-19,
- Рост сложности поиска мероприятий и мест для встречи из-за постоянного увеличения их количества.

2.3.2 Ценностные предложения

- Равное время в пути для всех участников.
- Экономия времени, затрачиваемого на поиск места для встречи.
- Интеграция с API KudaGo для получения актуальной информации о событиях.

2.3.3 Рисковые гипотезы

- **Низкий уровень доверия к новым приложениям**
 - Пользователи могут опасаться предоставлять данные о своем местоположении.
- **Технические ограничения**
 - Возможны сложности с интеграцией API Яндекс Карт.

Таким образом, можно считать, что актуальность работы продемонстрирована.

2.4 План работы

[Ссылка на матрицу-план Хосин-Канри](#)

[Ссылка на диаграмму Ганта](#)

Задача	Дата
Настроить инструмент для измерения покрытия кода на Kotlin тестами.	28/10/2024
Подать заявку и получить API-ключ портала открытых данных правительства Москвы	28/10/2024
Реализовать класс для представления взвешенного графа.	30/10/2024
Реализовать и покрыть тестами алгоритм Дейкстры поиска кратчайших путей в взвешенном графе.	05/11/2024
Реализовать классы для хранения информации о станциях метро, переходах и перегонах между станциями, линиях метро.	13/11/2024
Найти необходимые данные о станциях московского метро (внутри кольцевой линии) и времени в пути между ними. Распарсить и сохранить данные для использования программой.	13/11/2024
Реализовать и протестировать функцию построения оптимального маршрута между станциями метро.	16/11/2024
Реализовать класс для хранения информации о мероприятиях.	30/11/2024
Настроить получение данных о местах проведения мероприятий с сайта KudaGo.ru	30/11/2024
Реализовать алгоритм нахождения станции метро, время в дороге до которой у всех участников примерно одинаково.	14/12/2024
Реализовать функцию для нахождения мероприятия, время в дороге до которого у всех участников примерно одинаково.	15/12/2024
Реализовать и покрыть тестами класс текстового взаимодействия с пользователем.	06/01/2025
Реализовать первую версию приложения и внедрить взаимодействие с пользователем через текстовый интерфейс	11/01/2025
Подготовить первую версию приложения к публикации в RuStore: - создать иконку приложения; - создать описание предназначения и функций приложения; - подготовить снимки экрана из приложения; - заполнить и подать заявку на публикацию	12/01/2025
Пройти модерацию и опубликовать релиз в RuStore	13/01/2025

Задача	Дата
Найти необходимые данные о станциях московского метро (всех) и времени в пути между ними. Распарсить и сохранить данные для использования программой.	27/01/2025
Реализовать и внедрить в приложение интерактивную карту метро	15/03/2025
Подготовить вторую версию приложения к публикации в RuStore	17/03/2025
Пройти модерацию и опубликовать вторую версию приложения в RuStore	17/03/2025
Подключить Yandex MapKit SDK	01/04/2025
Реализовать и внедрить в приложение отображение пешего маршрута от местоположения пользователя к выбранной станции метро	01/04/2025
Реализовать и внедрить в приложение отображение пешего маршрута от станции метро к выбранному месту	25/04/2025
Подготовить третью версию приложения к публикации в RuStore	29/04/2025
Пройти модерацию и опубликовать третью версию приложения в RuStore	29/04/2025

Таблица 2.1: План работ на основе [диаграммы Ганта](#).

3 Обзор литературы

При разработке мобильного приложения «GatherRound», предоставляющего информацию о культурных мероприятиях, проходящих в городе, и поддерживающего интеграцию интерактивной карты метро, важно учитывать специфику разработки Android-приложений и известные графовые алгоритмы для оптимальной реализации построения маршрутов. Для этого были выбраны следующие источники, которые описывают основы Android-разработки, современные подходы к созданию пользовательских интерфейсов и алгоритмы на графах.

3.1 Разработка Android-приложения

Книга «How to Build Android Apps with Kotlin» (2021, Alex Forrester и др.) служит современным пособием по разработке Android-приложений с использованием языка Kotlin [5]. Книга описывает ключевые аспекты разработки современного Android-приложения:

- архитектурные подходы (MVVM, LiveData, ViewModel),
- инструменты для локального хранения данных (Room),
- инструменты для работы с внешними API и загрузки данных (Retrofit, Moshi, Glide),
- инструменты для тестирования приложения (JUnit, Mockito, Espresso).

Также в данной книге рассматриваются интеграция Google Maps API, управление пользовательскими разрешениями и выполнение фоновых операций, что необходимо для реализации интерактивной карты метро и уведомлений.

3.2 Создание современного пользовательского интерфейса

Книга «Jetpack Compose 1.2 Essentials» (2022, Neil Smyth) [7] описывает функционал новейшего инструмента разработки интерфейсов на Android — Jetpack Compose. Этот фреймворк значительно упрощает создание интерактивных и динамических элементов UI. Его использование позволяет реализовать интуитивно понятную карту метро с функционалом выбора станций и визуализации маршрутов. Jetpack Compose является оптимальным выбором для разработки UI, поскольку Compose использует декларативный API и позволяет писать меньше кода по сравнению с HTML и JavaScript для реализации аналогичного интерфейса. Данный фреймворк разрабатывается компанией Google и полностью поддерживается средой разработки Android Studio.

3.3 Алгоритмы на графах

Сайт neerc.ifmo.ru/wiki [15] представляет собой обширный сборник алгоритмических решений. Данный сборник полезен для изучения алгоритмов поиска кратчайших путей в графах. Подробные теоретические обоснования и наличие кода делают этот ресурс полезным для решения задач построения и оптимизации маршрутов.

4 Алгоритм поиска оптимальной станции для встречи

4.1 Постановка задачи

Реализовать алгоритм поиска такой станции метро, чтобы максимальное расстояние от неё до множества выбранных станций было минимальным. Рассмотрим множество выбранных вершин

$$V_{chosen} = v_1, v_2, \dots, v_k.$$

Требуется найти такую вершину $v_{meeting}$, чтобы минимизировать величину

$$t(v_{meeting}) = \max_{i=1}^k \text{dist}(v_i, v_{meeting}).$$

4.2 Алгоритм решения

1. Выполним следующий предварительный подсчет. Для каждой вершины v_i из множества V_{chosen} при помощи алгоритма Дейкстры¹. найдем расстояния до всех остальных вершин графа. Таким образом, мы получим список хэш-таблиц, где $dists[v_i][j]$ — расстояние от вершины v_i до вершины j .
2. Рассмотрим функцию $f(x)$, $\text{dom } f(x) = \mathbb{N}_0$. $f(x) = 1$, если существует такая вершина $v_{meeting}$, что $t(v_{meeting}) \leq x$, иначе $f(x) = 0$. Заметим, что функция f монотонна: для любого конечного непустого множества V_{chosen} и любой вершины $v_{meeting}$ существует некоторое такое число $X \in \mathbb{N}_0$, что $\forall x < X f(x) = 0$ и $\forall x \geq X f(x) = 1$. Следовательно,

¹Описание алгоритма Дейкстры см. по [ссылке](#) [15].

можно применить двоичный поиск² по значениям функции f , чтобы найти X .

Пусть на очередной итерации двоичного поиска мы зафиксировали расстояние t . Теперь надо эффективно проверить, существует ли такая вершина $v_{meeting}$, что $t(v_{meeting}) < t$. Для этого можно использовать подсчитанный заранее массив $dists$. Задача сводится к следующей: проверить, что

$$I = \bigcap_{i=1}^k \bigcup_{j=1}^{dists[v_i][j] \leq t} \{j\} \neq \emptyset.$$

Иными словами, нужно проверить, что существует вершина, которая содержится во всех хэш-таблицах $dists[v_1], dists[v_2], \dots, dists[v_k]$ и при этом соответствующее ей расстояние в каждой таблице не превышает t .

3. Итак, нужно проверить, что

$$I = \bigcap_{i=1}^k \bigcup_{j=1}^{dists[v_i][j] \leq t} \{j\} \neq \emptyset.$$

Это можно сделать следующим образом.

Создадим множество $intersection$, в котором будут храниться вершины, составляющие «пересечение» уже рассмотренных на данный момент хэш-таблиц. Изначально в $intersection$ лежат все вершины графа. Повторяя следующий алгоритм для каждой из k таблиц $dists[v_1], dists[v_2], \dots, dists[v_k]$: проходимся по очередной таблице $dists[v_i]$. Для вершины j проверяем, что $dists[v_i][j] \leq t$. Если это не так, рассматриваемая вершина не соответствует ограничению, и

²Описание алгоритма двоичного поиска см. по [ссылке \[15\]](#).

мы переходим к следующей вершине в таблице. Если вершина соответствует ограничению и она содержится в множестве $intersection$, мы добавляем её в новое множество-пересечение $new_intersection$. После того как мы обработали всю таблицу $dists[v_i]$, присвоим $intersection$ значение $new_intersection$.

4.3 Оценка времени работы

Пусть k - размер множества выбранных вершин V_{chosen} , n – общее число вершин в графе. Тогда предподсчет работает за $\mathcal{O}(kn \log n)$ (т.к. для каждой из k вершин вызывается алгоритм Дейкстры за $\mathcal{O}(n \log n)$).

Поиск общих вершин, удовлетворяющих ограничению, в двух массивах работает за $\mathcal{O}(n)$, поскольку происходит $\mathcal{O}(n)$ операций с hashmap. Проверка в двоичном поиске работает за $\mathcal{O}(kn)$, поскольку функция для поиска общих вершин в двух массивах вызывается k раз.

Таким образом, двоичный поиск работает за $\mathcal{O}(kn \log C)$, где C – максимальное расстояние между вершинами в графе.

Итоговая сложность алгоритма: $\mathcal{O}(kn \log n + kn \log C)$

4.4 Применение

API сайта KudaGo.ru предоставляет данные о ближайших к местам проведения мероприятий станциях метро.

С использованием этих данных и описанного выше алгоритма реализована функция поиска событий, максимальное время в пути до которых от выбранных станций минимально.

4.5 Реализация

```
def find_optimal_vertex(Set[Vertex] chosen_vertexes) ->
    Pair[Vertex, int]:
    """
    Находит такую вершину графа, что максимальное расстояние
    от нее до вершины из множества выбранных вершин минимально.
    Возвращает найденную вершину и максимальное расстояние
    от нее до вершины из множества выбранных вершин.
    """
    if not chosen_vertexes:
        return null, null

    if len(chosen_vertexes) == 1:
        return min(chosen_vertexes), 0

    dists = precalc_dists(chosen_vertexes)
    # функция
    # precalc_dists(Set[Vertex]) -> Dict[Vertex, Dict[Vertex,
    # → int]]
    # запускает алгоритм Дейкстры от каждой вершины из
    # переданного множества и возвращает хэш-таблицу,
    # состоящую из хэш-таблиц пар (вершина, расстояние до
    # рассматриваемой вершины из множества выбранных)

    left = 0
    right = MAX_DIST
```

```

while (left + 1 < right):
    mid = (left + right) // 2

    intersection = k_intersection(dicts = dists,
                                   bound = mid)

if intersection:
    optimal_vertex = next(iter(intersection))
    right = mid
else:
    left = mid

val optimal_dist = right
return optimal_vertex, optimal_dist

def k_intersection(dicts: Dict[Dict[Vertex, int]], 
                    bound: Int) -> Set[Vertex]:
    """
    Находит общие ключи нескольких хэш-таблиц,
    (вершина, расстояние), при этом значения,
    соответствующие ключам,
    не преексходят заданное число bound.
    """

if not dicts:
    return []

intersection = []
for vertex, dist in next(iter(dicts.values())):

```

```

    if dist <= bound:
        intersection.add(vertex)

    for vertex, dict in dicts.items():
        intersection = intersection(cur_intersection =
            ↳ intersection,
            new_dict = dict,
            bound = bound)

    return intersection

```

```

def intersection(cur_intersection: Set[Vertex],
                 new_dict: Dict[Vertex, int],
                 bound: Int) -> Set[Vertex]:
    """

```

Находит ключи словаря new_dict, такие что соответствующие им значения не превосходят заданное число bound и содержатся в множестве cur_intersection.

```

    """
    result = []
    for vertex, dist in new_dict.items():
        if dist <= bound and vertex in cur_intersection:
            result.add(vertex)

    return result

```

5 Пользовательский интерфейс

Реализован текстовый ввод станций и интерактивная карта метрополитена с использованием Jetpack Compose.

Для того, чтобы элементы интерфейса выглядели лаконично и однородно, используются стандартные элементы библиотек `androidx.compose.material` и `androidx.compose.material3` Android-фронтенда:

- `OutlinedTextField` для полей текстового ввода,
- `icons` для отображения иконок кнопок добавления и удаления станций,
- `DropdownMenuItem` для выпадающего меню,
- `Card` для отображения подобранных мест для встречи.

5.1 Текстовый ввод

В нижней части экрана, как показано на рисунке 5.1а, расположены поля для текстового ввода названий станций. Удаление и добавление полей производится при нажатии на соответствующие кнопки рядом с полем для ввода. Слева от каждого поля расположена кнопка для его удаления, справа от нижнего поля расположена кнопка для добавления пустого поля.

Для удобства реализовано выпадающее меню (см. рисунок 5.1б) с использованием компонента `DropdownMenuItem`. В меню отображаются те станции, названия которых совпадают с введённым текстом. При нажатии на элемент меню название выбранной станции отображается целиком в поле для ввода, и станция добавляется в множество выбранных.

Блок текстового ввода занимает 40% высоты экрана, реализована прокрутка полей для ввода с помощью модификатора `Modifier.verticalScroll`.

5.2 Интерактивная карта метро

Интерактивная карта позволяет выбирать станции, нажимая на них (т.е. на название станции или на соответствующий кружок). Реализовано следующее взаимодействие между картой и текстовым вводом. Возможность выбрать станцию на интерактивной карте становится доступной, если есть пустое поле для ввода. Название выбранной на карте станции отображается в первом сверху пустом поле для ввода. При вводе станции вручную через текстовое поле, выбранная станция выделяется на карте.

Названия выбранных станций выделяются на карте жирным шрифтом, а соответствующие им кружки окрашиваются в чёрный цвет. При отмене выбора (т.е. нажатии на уже выбранную станцию) стиль элементов возвращается к исходному состоянию.

Карта масштабируется жестом «pinch-to-zoom», т.е. если коснуться экрана в области карты двумя пальцами и свести их, масштаб уменьшится, если развести — увеличится.

5.3 Источник SVG

Код SVG карты взят с [официального сайта Московского метрополитена](#).

5.4 Загрузка и отображение SVG-карты метро

Реализована загрузка SVG-файла по заданной веб-ссылке с помощью библиотеки `Retrofit` и класса `LiveData` в компоненте `Repository.kt`.

SVG-карта конвертируется в HTML, загружается в компонент `WebView` и становится доступной для интерактивного взаимодействия.

5.5 Обработка кликов. Настройка контроллера взаимодействия с JavaScript

Интерактивность карты обеспечивается JavaScript-кодом, встроенным в HTML-страницу, загруженную в WebView [6] [11].

В JS-коде реализована следующая логика. Отслеживаются клики по элементам SVG (кружкам и названиям станций). При клике определяется, выбрана ли уже станция, на которую кликнули, и вызываются соответствующие функции:

- `selectStation()` — визуально выделяет выбранную станцию;
- `deselectStation()` — снимает выделение.

JavaScript сообщает об изменениях состояния в Android-приложении через интерфейс `MapInterface`.

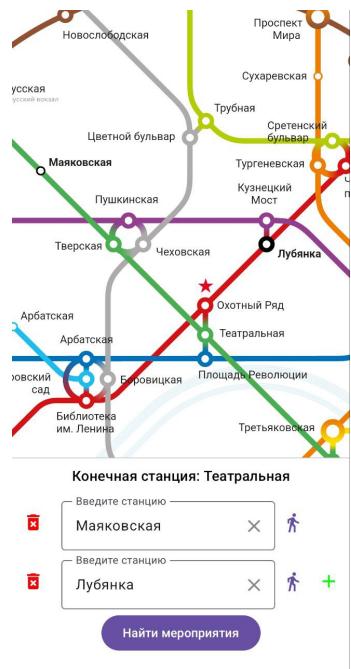
Вызовы функций `androidObj.onStationClick()` и `androidObj.onStationClickedWithResult()` передают информацию о выбранной станции в Kotlin-коде.

JavaScript ожидает ответа от Kotlin-кода, чтобы подтвердить возможность выбора (например, есть ли пустое окно для ввода, чтобы отобразить в нём название выбранной станции).

5.6 Отображение пеших маршрутов

После того как пользователь ввел все станции и нажал на кнопку «Найти места для встречи», подобранные места отмечаются на карте маркерами при помощи Yandex MapKit SDK [24] на основании примеров из официального репозитория данного SDK [25] (рисунок 5.1c). При нажатии на маркер отображается название выбранного места, его адрес и кнопка, позволяющая построить пеший маршрут от конечной станции метро до него (рисунок 5.1d).

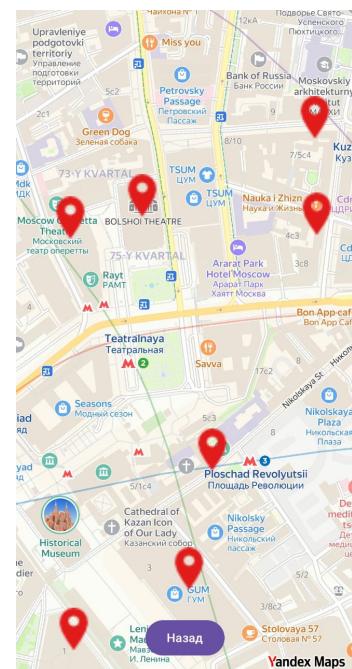
Реализовано перенаправление в приложение (при наличии) или на сайт Яндекс.Карт для отображения пешего маршрута от местоположения пользователя до выбранной им станции метро (рисунок [5.1e](#)) и от конечной станции до выбранного пользователем места (рисунок [5.1f](#)).



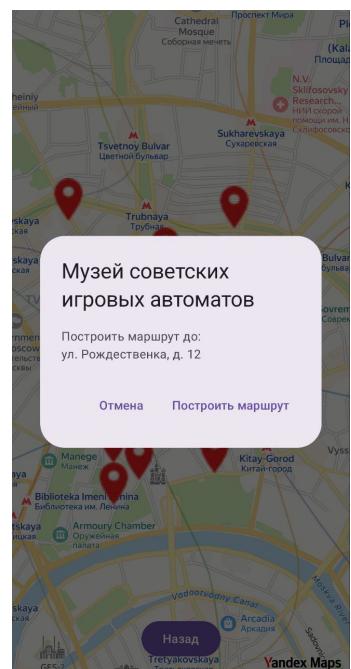
(а) Рисунок 5.1а - Экран ввода станций



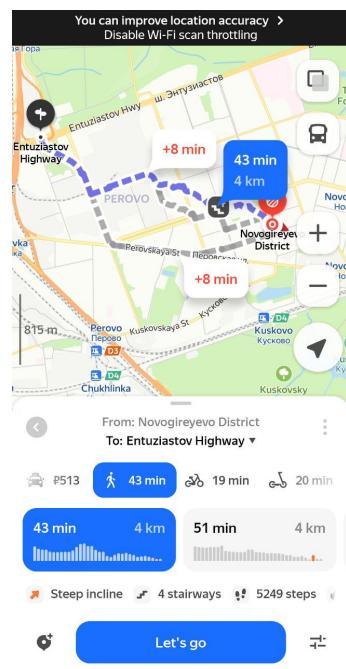
(б) Рисунок 5.1б - Выпадающее меню, появляющееся при вводе названия станции вручную



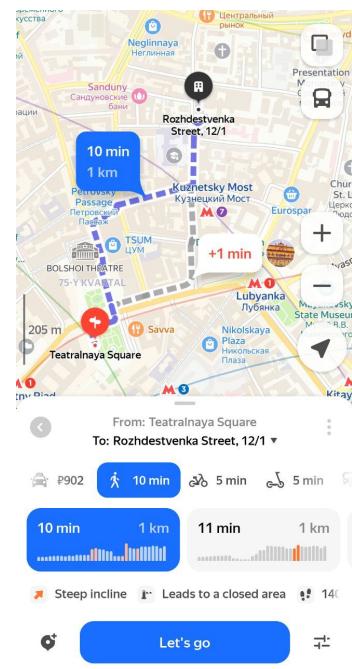
(с) Рисунок 5.1с - Метки на карте, показывающие подобранные места



(д) Рисунок 5.1д - Краткая информация о выбранном пользователе месте для встречи



(е) Рисунок 5.1е - Отображение пешего маршрута от местоположения пользователя до выбранной станции метро

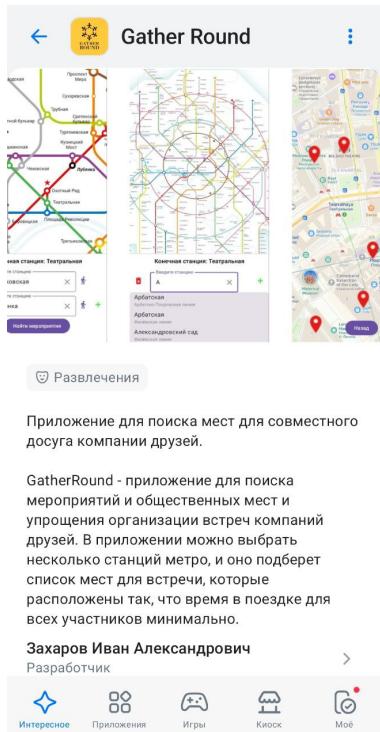


(ж) Рисунок 5.1ж - Отображение пешего маршрута от конечной станции до выбранного места

6 Заключение

В рамках проделанной автором работы программный код опубликован в репозитории: [GitHub-репозиторий с кодом проекта](#).

Приложение опубликовано в RuStore (рисунок 6.1а): [ссылка для скачивания](#).



(а) Рисунок 6.1а - Страница приложения в RuStore

6.1 Результаты

Во время написания работы автором были решены следующие задачи:

- Реализован алгоритм поиска оптимальной станции для встречи на основании алгоритма Дейкстры.
- Реализована интеграция с API KudaGo.ru и функция для получения мест для посещения, расположенных рядом с заданной станцией метро.

- Реализован пользовательский интерфейс, позволяющий вводить станции в текстовые поля для ввода и выбирать их на интерактивной карте метро.
- Реализовано отображение мест рядом с станцией сбора с использованием Yandex MapKit SDK.
- Реализовано перенаправление в приложение (при наличии) или на сайт Яндекс.Карт для отображения пешего маршрута от местоположения пользователя до выбранной стартовой станции и конечной станции до выбранного места.

В процессе разработки приобретены и улучшены навыки:

- Разработки пользовательского интерфейса с использованием Jetpack Compose,
- Работы с REST API,
- Разработки на JavaScript,
- Интеграции JavaScript-кода с Kotlin-приложением,
- Работы с Yandex MapKit SDK.

6.2 Развитие работы

Планируются следующие усовершенствования приложения:

- Добавить другие крупные города, имеющие метрополитен, и для которых существует агрегатор событий и мест для встречи (KudaGo.ru или аналогичный). К таким городам относятся:
 - Санкт-Петербург,
 - Нижний Новгород,

- Екатеринбург,
 - Казань.
- Реализовать фильтрацию мест для посещения по различным параметрам;
 - Реализовать опцию для отображения маршрута в метро от выбранной станции до конечной.

Список литературы

- [1] *Android (operating system)*. URL: [https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)) (дата обр. 18.04.2025).
- [2] *Dokka*. URL: <https://github.com/Kotlin/dokka> (дата обр. 18.04.2025).
- [3] *Espresso manual*. URL: <https://developer.android.com/training/testing/espresso> (дата обр. 18.04.2025).
- [4] *Glide*. URL: <https://github.com/bumptech/glide> (дата обр. 18.04.2025).
- [5] Alex Forrester, Eran Boudjnah и Alexandru Dumbravan. *How to Build Android Apps with Kotlin*. Packt Publishing, 2021. URL: https://www.dropbox.com/scl/fi/4uqfry30cv09zm4tnw50z/How_to_Build_Android_Apps_with_Kotlin_A_hands_on_guide_to_developing.pdf?rlkey=luxgciqjfub0vxlnq3e3ns13&st=hga22ekn&dl=0.
- [6] *Interactive SVG image in Android app using Kotlin and JavaScript*. URL: <https://medium.com/@scode43/interactive-svg-image-in-android-app-using-kotlin-and-javascript-6715c16397bb> (дата обр. 18.04.2025).
- [7] Neil Smyth. *Jetpack Compose 1.2 Essentials*. Payload Media, Inc., 2022. URL: https://www.dropbox.com/scl/fi/c2hprbxzdaxqhv9qk2b9/Jetpack_Compose_1_2_Essentials_-_Neil_Smyth.pdf?rlkey=88b0kn0atpepdwojtcnqs7iib&st=gbf0x737&dl=0.
- [8] *Jetpack Compose manual*. URL: <https://developer.android.com/compose> (дата обр. 18.04.2025).
- [9] *JUnit*. URL: <https://en.wikipedia.org/wiki/JUnit> (дата обр. 18.04.2025).

- [10] *Kotlin (programming language)*. URL: [https://en.wikipedia.org/wiki/Kotlin_\(programming_language\)](https://en.wikipedia.org/wiki/Kotlin_(programming_language)) (дата обр. 18.04.2025).
- [11] *Making Android interacting with Web App*. URL: <https://medium.com/mobile-app-development-publication/making-android-interacting-with-web-app-921be14f99d8> (дата обр. 18.04.2025).
- [12] *Model-view-viewmodel*. URL: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel> (дата обр. 18.04.2025).
- [13] *Moshi*. URL: <https://github.com/square/moshi> (дата обр. 18.04.2025).
- [14] *SVG*. URL: <https://ru.wikipedia.org/wiki/SVG> (дата обр. 18.04.2025).
- [15] НИУ ИТМО. *Викиконспекты ИТМО*. URL: <https://neerc.ifmo.ru/wiki> (дата обр. 18.04.2025).
- [16] *Интерфейс пользователя*. URL: <https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%BB%D1%82%D0%B5%D0%BB%D1%8F> (дата обр. 18.04.2025).
- [17] *Исследование ДИТ: какие цифровые устройства используют москвичи*. URL: https://www.cnews.ru/news/line/2024-03-22_issledovanie_dit_kakie_tsifrovye?utm_source=chatgpt.com (дата обр. 18.04.2025).
- [18] *Исследование Rambler&Co: 44% россиян проводят досуг вместе с семьёй*. URL: https://www.sostav.ru/blogs/260202/45998?utm_source=chatgpt.com (дата обр. 18.04.2025).
- [19] *Как библиотека Retrofit 2 решает сетевые трудности перевода*. URL: <https://practicum.yandex.ru/blog/retrofit-na-android/> (дата обр. 18.04.2025).

- [20] *LiveData*. URL: <https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/525-urok-2-livedata.html> (дата обр. 18.04.2025).
- [21] *Mockito и как его готовят*. URL: <https://habr.com/ru/articles/444982/> (дата обр. 18.04.2025).
- [22] *Room. Основы*. URL: <https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/529-urok-5-room-osnovy.html> (дата обр. 18.04.2025).
- [23] *ViewModel*. URL: <https://startandroid.ru/ru/uroki/vse-uroki-spiskom/27-course/architecture-components/527-urok-4-viewmodel.html> (дата обр. 18.04.2025).
- [24] *Yandex MapKit и NaviKit SDK*. URL: <https://yandex.maps-api/docs/mapkit/index.html> (дата обр. 18.04.2025).
- [25] *Yandex MapKit and NaviKit Demo Apps*. URL: <https://github.com/yandex/mapkit-android-demo> (дата обр. 18.04.2025).