ОЛИМПИАДА ШКОЛЬНИКОВ «ШАГ В БУДУЩЕЕ»

НАУЧНО-ОБРАЗОВАТЕЛЬНОЕ СОРЕВНОВАНИЕ «ШАГ В БУДУЩЕЕ, МОСКВА»

397

регистрационный номер

_____Информатика и системы управления_____

название факультета

_____Компьютерные системы и сети_____

название кафедры

Программное обеспечение, автоматизирующее построение туристических маршрутов

название работы

Автор: *_Волынец _Александра_Андреевна*

фамилия, имя, отчество

ГБОУ_школа_№_368_ «Лосиный остров»,

_____11 класс «А» _____

наименование учебного заведения, класс

Научный руководитель: *_Шайхутдинов_Артур_Анисович*

фамилия, имя, отчество

_____Кафедра ИУ6_____

место работы

_____Ассистент_____

звание, должность

подпись научного руководителя

Москва – 2019

Аннотация

Данный проект помогает упростить процесс составления прогулочного пешеходного маршрута в независимости от местоположения, с учетом личных предпочтений и предполагаемого времени в пути.

Цель работы – разработка системы построения прогулочных туристических маршрутов, которая будет предоставлять пользователю актуальную информацию о достопримечательностях, культурных и социальных объектах в непосредственной близости к его местоположению и строить наиболее рациональные и подходящие по времени маршруты.

В процессе создания системы был разработан алгоритм поиска выбранных, согласно предпочтениям пользователя, мест и построения оптимальных путей перемещения между ними. Для реализации данного проекта было решено создать мобильное приложение на ОС Android на языке java с использованием возможностей Google Maps API для получения актуальных данных о находящихся поблизости объектах и их взаимном расположении.

В программном коде приложения использованы библиотеки Retrofit 2 и SDK Maps картографического сервиса Google.

В итоге была разработана программа, составляющая на основании предпочтений пользователя несколько маршрутов с указанием примерного времени движения, а также информацию о каждом из предлагаемых к посещению мест.

Термины и сокращения

API (Application Programming Interface) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

SDK (Software Development Kit) — набор средств разработки, который позволяет специалистам по программному обеспечению создавать приложения для определённого пакета программ, программного обеспечения базовых средств разработки, аппаратной платформы, компьютерной системы, игровых консолей, операционных систем и прочих платформ.

GET-запрос — это вид HTTP-запроса, который используется для запроса содержимого указанного ресурса.

ООП (Объектно-ориентированное программирование) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Содержание

Введение

Поездки в неизвестные города и страны — это неотъемлемая часть жизни каждого современного человека. Когда люди отправляются в путешествие или в командировку, они сталкиваются с проблемой выбора подходящего путеводителя. В наше время большинство путешественников предпочитают электронные путеводители и карты их бумажным аналогам. Они не занимают место в багаже и всегда находятся под рукой.

В современных магазинах приложений пользователям предоставляется выбор из огромного количества различных туристических приложений. Наибольшей популярностью пользуются два типа таких приложений: карты с отмеченными на них достопримечательностями, и путеводители с маршрутами, составленными разработчиками. К первому типу можно отнести популярные приложения CityMaps2Go и LonelyPlanet. Они предоставляют пользователю возможность работать с картой местности, а также получать некоторую информацию об отмеченных на ней местах отдыха и развлечения. Но существенный минус этих приложений — невозможность составления в них каких-либо маршрутов. А ведь именно выбор интересующих мест и составление оптимального пути перемещения между ними отнимает огромное количество времени у путешественников. Если же рассматривать второй тип туристических приложений, к которому можно отнести, например, izi.Travel, то здесь основным недостатком является ограниченность выбора маршрутов и невозможность их изменения. Таким образом, даже при таком большом количестве различных приложений, выбор оптимального варианта, совмещающего в себе предоставление актуальной информации как о социальных и культурных объектах, интересующих пользователя, так и о

наиболее рациональном и подходящем по времени пути перемещения между ними, оказывается сложной задачей.

Моя работа посвящена решению данной проблемы. В процессе исследования различных источников информации и непосредственной разработки программного обеспечения были поставлены следующее задачи:

1) Изучить возможности поиска различных категорий мест и получения актуальных сведений о них.

2) Разработать алгоритм для нахождения достопримечательностей и мест для развлечений и отдыха в непосредственной близости к местоположению пользователя и построения маршрута между ними, в соответствии с отведенным временем.

3) Реализовать алгоритм на языке Java в интерактивной среде разработки Android Studio.

4) Создать пользовательский интерфейс

1 Сравнительный анализ возможностей Google Maps и Яндекс. Карт

Информацию о культурных и социальных объектах и их местоположении хранят картографические платформы. С их помощью и предполагается получать необходимые сведения в рамках данной работы. Два самых популярных ресурса, позволяющих это сделать — это Google Maps и Яндекс. Карты. Они также позволяют разработчикам использовать свои данные и технологии с помощью различных API. Помимо справочной информации о разных местах, нам необходима возможность построения пешеходного маршрута между ними.

Таблица 1 – Анализ возможностей Google Maps и Яндекс. Карт

Критерий	**Яндекс. Карты**	**Google Maps**
Покрытие	Лучшее покрытие России, уступает Google в покрытии мира	Лучшее покрытие всего мира
Справочная информация	Подробная информация об организациях	Неполная информация о российских организациях
Возможность оставить и прочитать отзывы и оценить организацию	Только оценка	Отзыв и оценка
Построение маршрута	Автомобиль, общественный транспорт, пешеходный маршрут. Строит с учетом пробок. Требует подключение к сети интернет для	Автомобиль, общественный транспорт, пешеходный маршрут. Возможность выбрать только один из видов транспорта или вариант пешком. Строит

	построения.	с учетом пробок и расписания общественного транспорта. Требует подключение к сети интернет для построения.

В результате сравнительного анализа (см. табл. 1) данных картографических сервисов был сделан вывод о том, что сервис Google Maps больше подходит для решения поставленной задачи.

Платформа Google Maps для разработчиков предлагает три продукта:

1) Maps

Включает в себя SDK для Android и iOS, а также различные API для построения индивидуально измененных интерактивных карт на различных платформах.

2) Routes

Предоставляет пользователю возможность построить оптимальный путь перемещения между двумя или более точками с учетом пробок на данный момент. Включает в себя Directions API, который позволяет строить пешеходные маршруты, и Distance Matrix API, с помощью которого можно узнать их длительность. Эти два API будут использованы для нахождения походящего по времени пути между объектами, интересующими пользователя.

3) Places

Помогает пользователю получить информацию о 100 миллионах мест по всему миру. Позволяет найти определенные организации по телефону, адресу или категории. Включает в себя SDK для Android и iOS, а также Places API для получения актуальных сведений о миллионах различных объектов с помощью

HTTP-запросов. Для решения задачи поиска мест по категориям и получения информации о них будет использован этот API.

2 Разработка алгоритма

2.1 Работа с API

Работа с выбранными API будет осуществляться по средствам отправки и получения информации через GET-запросы. Ответы на них приходят в формате JSON.

JSON (JavaScript Object Notation) — это текстовый формат обмена данными, основанный на JavaScript. Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON, поэтому вдаваться с подробности работы с этим форматом в рамках данного проекта мы не будем.

2.2 Основа алгоритма программы

В ходе работы программа получает сведенья о местоположении пользователя, о его предпочтениях в выборе категорий интересующих мест, а также о количестве времени, которое он предполагает потратить на маршрут. Используя информацию о времени, алгоритм рассчитывает примерный максимальный радиус перемещения, а также наибольшую длину маршрута, с учетом средней скорости пешехода. Далее с помощью Places API посылаются запросы, с целью узнать список мест, походящих под каждую указанную пользователем их категорию в рассчитанном радиусе. Получив эти данные, необходимо сделать правильную выборку найденных мест для того чтобы построить оптимальный пешеходный маршрут. Данную цель можно достичь, только если при добавлении каждого отдельного места в маршрут проверять, превышает ли при этом весь предшествующий путь вместе с перемещением до

данного места и от него до конца маршрута максимально возможную длину, рассчитанную ранее. Каждый раз алгоритм посылает GET-запрос в Directions API и проверяет это условие.

Для того чтобы пользователь мог выбрать наиболее подходящий ему маршрут, составляется четыре разных варианта выборки культурных и социальных объектов. После этого для каждого из них составляется оптимальный маршрут (см. прил. А), по которому, с помощью Distance Matrix API, проверяется соответствие времени, затрачиваемого на составленный путь, указанному пользователем времени. В итоге выбирается два наиболее походящих варианта.

Про каждую предполагаемую остановку этих маршрутов, с помощью Places API, узнается вся доступная сервису Google информация: рейтинг, фотографии, отзывы, часы работы.

3 Реализация программы

3.1 Объектно-ориентированное программирование

В результате детального анализа различных вариантов реализации программы мы пришли к заключению, что наиболее подходящим языком программирования для решения поставленных задач является Java. Одним из главных его свойств является объектно-ориентированная методология[1]. По этой причине все программы, написанные с помощью Java, в той или иной степени являются объектно-ориентированными. Соответственно, подробное изучение ООП, а именно его структуры (см. рис. 1) и основных принципов и методов – это одна из основных задач, которые необходимо разрешить при разработке программы.

Главное понятие ООП – это объект. В классе определяются его тип и данные, которые будут служить для дальнейшего построения объектов в ходе программы.

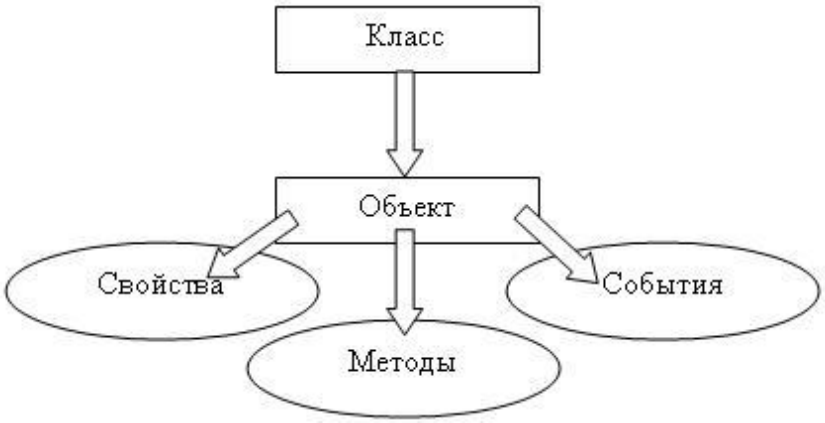


Рисунок 1 – Структура ООП

Суть объектно-ориентированного программирования заключается в трех основных принципах:

1. Инкапсуляция

Этот механизм программирования, объединяющий код и данные, который имеет возможность упаковывать различную информацию и функции в один компонент. Данные внутри такого объекта могут иметь различные модификации доступа внутри программы.

2. Полиморфизм

Это свойство, позволяющее обращаться к какому-либо общему классу действий с помощью одного интерфейса. Таким образом, существует возможность создать универсальный интерфейс для группы взаимосвязанных действий.

3. Наследование

Это процесс, в ходе которого один объект наследует свойства другого. Именно с помощью этого принципа поддерживается классификация. Каждый компонент подкласса содержит те же свойства, что и компонент надкласса.

Классы служат прочным основанием для объектно-ориентированного программирования на Java. В них определяются данные и код, который может выполнять различные действия над ними. Основным классом в приложении, процесс разработки которого описывается в данном исследовании, является MapsActivity (см. прил. Б). В нем описаны основные методы и функции, а также содержаться главные объекты и переменные, используемые программой. В свою очередь, класс MapsActivity является дочерним классом FragmentActivity. Все классы, использованные в программе, показаны на диаграмме (см. рис. 2).

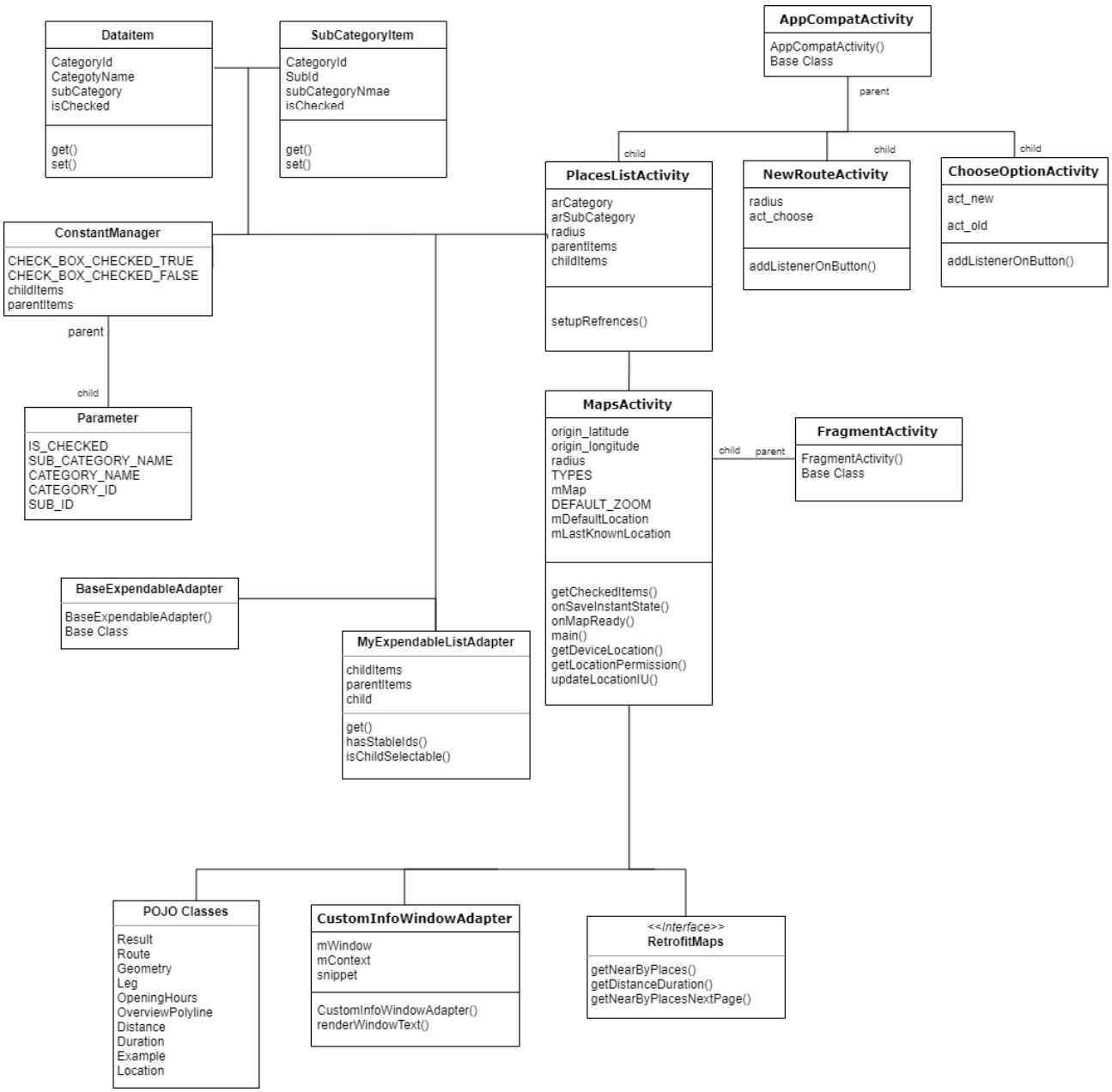


Рисунок 2 – Диаграмма классов

3.2 Работа с ОС Android

Для реализации приложения была выбрана интерактивная среда разработки Android Studio[1].

[1] Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, — официальное средство разработки Android приложений.

Чтобы использовать в этой среде возможности Google Maps API, необходимо подключить Maps SDK. Для работы с HTTP-запросами была выбрана библиотека Retrofit 2, разработанная компанией “Square”. Этой библиотекой удобно пользоваться для отправки запроса к различным веб-сервисам с командами GET, POST, PUT, DELETE. Для того чтобы использовать возможности библиотеки, необходимо создать интерфейс, указать имя метода, а также добавить необходимые параметры:

для Places API:

- location (обязательный) - широта и долгота местоположения, в окрестностях которого следует искать места. Эта информация должна быть указана как объект google. maps. LatLng.
- radius (обязательный) - расстояние в метрах, в пределах которого должны быть получены результаты для мест. Значение для radius рекомендуется задавать в зависимости от точности сигнала о местонахождении, передаваемого датчиком местоположения. Следует учесть, что при заданном значении radius отдается предпочтение результатам в указанной области, однако строгое ограничение отсутствует, и могут отображаться также результаты за пределами этой области.
- types (необязательный) - ограничивает результаты местами, тип которых соответствует хотя бы одному из указанных.
- language (необязательный) - код языка, указывающий, на каком языке следует по возможности возвращать результаты.
- key (обязательный) - ключ API вашего приложения. Этот ключ служит для идентификации вашего приложения с целью управления квотами. Помимо этого, он позволяет сразу же делать доступными для вашего приложения места, которые были добавлены в нем. Чтобы создать проект API и получить ключ необходимо посетить консоль API.

для Directions API:

- origin (обязательный) – адрес или текстовое значение широты и долготы отправного пункта маршрута. Если адрес передается в виде строки, для вычисления маршрута служба выполняет геокодирование строки и преобразует ее в значения координат широты и долготы. При передаче координат не допускаются пробелы между значениями широты и долготы.
- destination (обязательный) – адрес или текстовое значение широты и долготы отправного пункта маршрута. Если адрес передается в виде строки, для вычисления маршрута служба выполняет геокодирование строки и преобразует ее в значения координат широты и долготы.
- key(обязательный)
- language (необязательный)
- mode (необязательный) – способ перемещения (пешим ходом, на машине, на общественном транспорте или на велосипеде)
- waypoints (необязательный) – список мест, которые необходимо включить в маршрут. Указать их можно с помощью координат, place ID или адреса.

В интерфейсе задаются команды-запросы для сервера (см. рис. 3), которые он объединяет с базовым адресом сайта и перечисленными выше параметрами.

```
public interface RetrofitMaps {

    @GET("api/place/nearbysearch/json?sensor=true&key=AIzaSyAFXeGAfW8NsbhT8yXj0fhsVmvUEnP3AsA&language=ru")
    Call<Example> getNearbyPlaces(@Query("location") String location, @Query("radius") int radius);

    @GET("api/directions/json?key=AIzaSyAFXeGAfW8NsbhT8yXj0fhsVmvUEnP3AsA&units=metric&mode=walking")
    Call<Example2> getDistanceDuration( @Query("origin") String origin, @Query("destination")
            String destination, @Query("waypoints") String waypoints);
```

Рисунок 3 – Интерфейс для отправки запросов в Places API и в Directions API

Для обработки полученных данных нужно проанализировать структуру ответа сайта в виде JSON и создать на его основе Java-классы в виде POJO[2](см. прил. В).

Для того чтобы не нарушать инкапсуляцию класса, при создании POJO классов мы используем модификатор доступа private. После этого необходимо указать аннотации[3], используемые при работе с библиотекой Retrofit 2:

1. GET

Данная аннотация передает компилятору информацию о типе запроса (GET), а также указанный разработчиком параметр.

2. POST

Данная аннотация передает компилятору информацию о типе запроса (POST), также указанный разработчиком параметр.

После указания аннотации мы создаем два метода:

1. Метод чтения (англ. getter)

Он позволяет получить данные, доступ к которым напрямую ограничен. Данный метод помогает реализовать гибкий механизм инкапсуляции.

2. Устанавливающий метод (англ. setter) или модифицирующий метод, мутатор (англ. mutator)

Он используется в объектно-ориентированном программировании для того, чтобы присвоить какое-либо значение инкапсулированному полю, например, обработав при этом недопустимые присваивания.

---

[2] POJO (*Plain Old Java Object*) — «старый добрый Java-объект», простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для объектов предметной области.

[3] Java-аннотация — в языке Java специальная форма синтаксических метаданных, которая может быть добавлена в исходный код. Аннотации используются для анализа кода, компиляции или выполнения. Аннотируемы пакеты, классы, методы, переменные и параметры.

Упомянутые выше методы используются при работе программы для организации различных свойств и методов. В результате получаем класс-модель, отражающий данные, которые будут приходить после отправки HTTP-запроса.

Помимо разработанного ранее основного алгоритма программного обеспечения, было необходимо продумать схему работы приложения (см. рис. 4) и его интерфейс. Для этого была создана структурная блок-схема, представленная ниже.

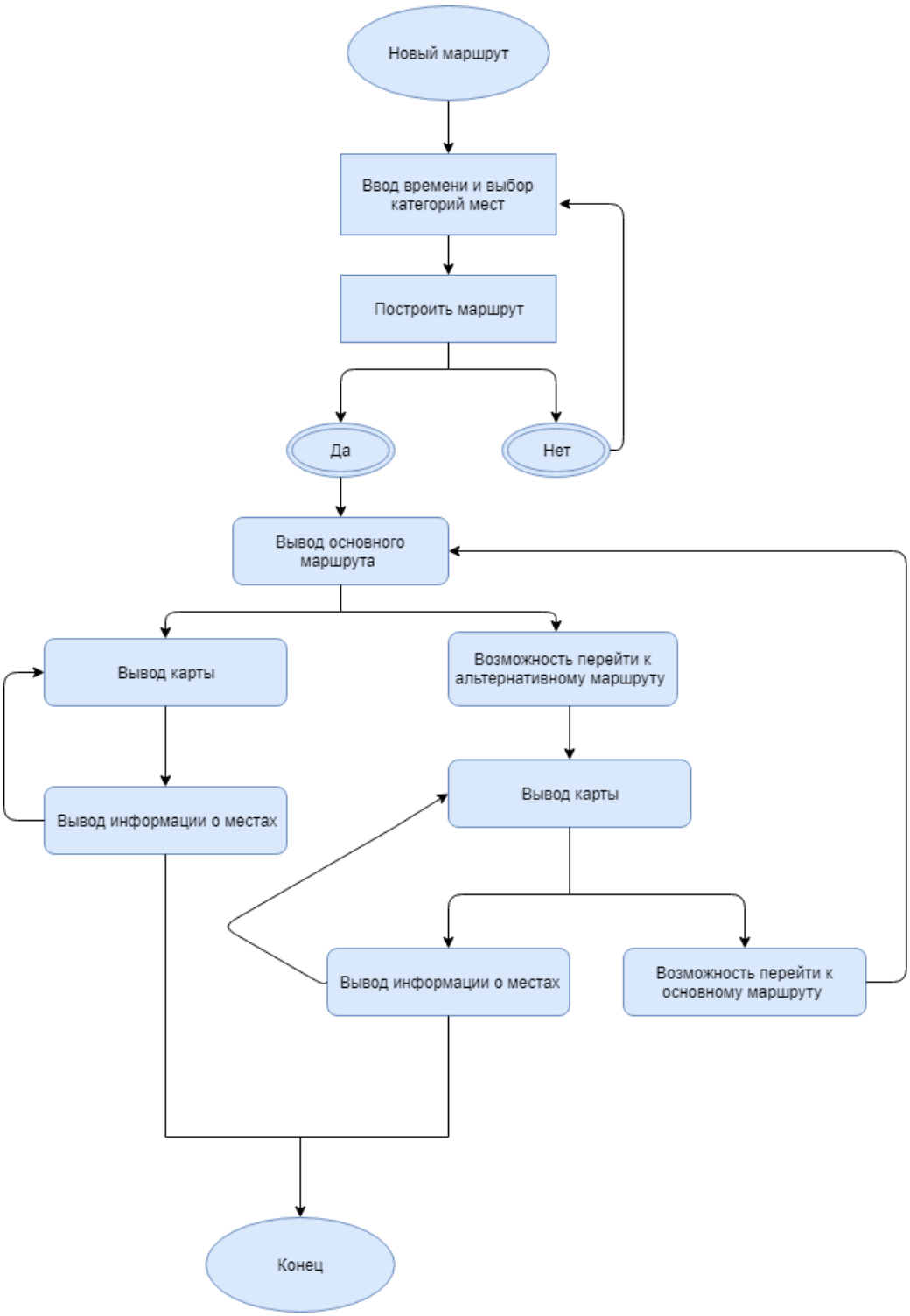


Рисунок 4 – Схема алгоритма

4 Тестирование

Для того чтобы выявить возможные ошибки в работе программы необходимо ее протестировать. Проанализировав различные способы

тестирования [2], мы сделали вывод, что наиболее оптимальный вариант – это проводить проверку в два этапа: сначала протестировать сам алгоритм, а потом интерфейс приложения.

4.1 Тестирование алгоритма программы в автоматическом режиме

Программе передается информация о различных наборах категорий мест и времени, которое предполается потратить на маршрут, а затем с помощью отправки запроса в Distance Matrix API узнается реальная длительность построенного по этим данным маршрута.

Впоследствии был проведен сравнительный анализ полученной информации, а также были выявлены наиболее оптимальные погрешности измерений, которые необходимо учитывать в расчетах (см. табл. 2).

Таблица 2 – Данные, полученные после тестирование алгоритма

№ варианта сочетания выбранных категорий мест	**Введенное время (в секундах)**	**Получившееся время (в секундах)**	**Количество мест**	**Погрешность, учитываемая при измерениях (в метрах)**
1	3700	2858	5	3000
1	3700	1968	4	1000
1	3700	3516	7	4000
2	**3700**	**3633**	**7**	**3000**
2	3700	2278	4	**1000**
2	3700	4318	8	4000
3	4800	3356	4	4000
4	**4800**	**3813**	**4**	**3000**
4	4800	7870	6	1000
4	4800	10555	7	4000
5	**2000**	**2195**	**7**	**3000**
5	2000	968	4	1000
5	2000	1866	6	2000
5	2000	2293	9	4000
6	7200	3734	5	3000

6	7200	989	4	4000
6	**7200**	**6508**	**8**	**7200**
7	5000	8525	8	3000
7	5000	7838	7	5000
7	5000	7049	7	1000
7	**5000**	**4624**	**6**	**500**
8	**6300**	**5997**	**4**	**3000**
8	6300	2239	2	1000
8	6300	2508	3	500
9	8000	9416	6	3000
9	8000	5176	4	1000
9	8000	3366	3	500
9	8000	7452	4	8000

4.2 Тестирование интерфейса приложения в ручном режиме

При тестировании пользовательского интерфейса нужно убедиться, что функциональность приложения полностью выполняется, а также проверить:

- Внешний вид элементов при разных размерах экранов
- Правильность написания текста
- Правильность расположения различных элементов интерфейса
- Размещение всех сообщений об ошибках

Тестирование было проведено на двух телефонах с операционной системой Android разных марок с разными размерами экранов:

1) Sony Xperia E5

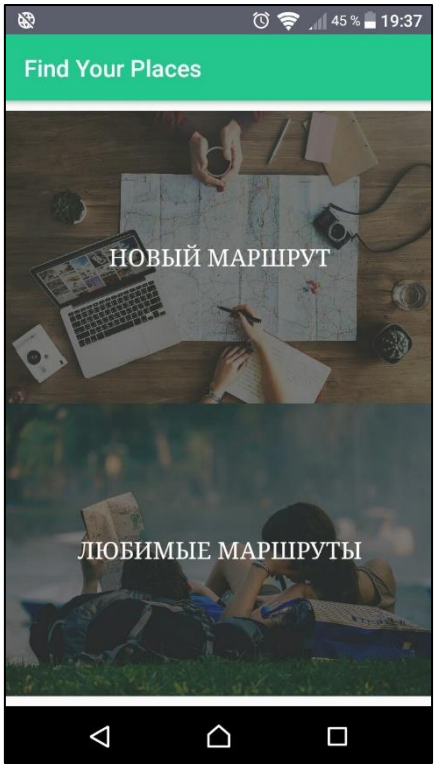
Характеристики:

Диагональ – 5 дюймов

Высота корпуса – 144 мм

Ширина корпуса – 69 мм

Внешний вид интерфейса приложения:



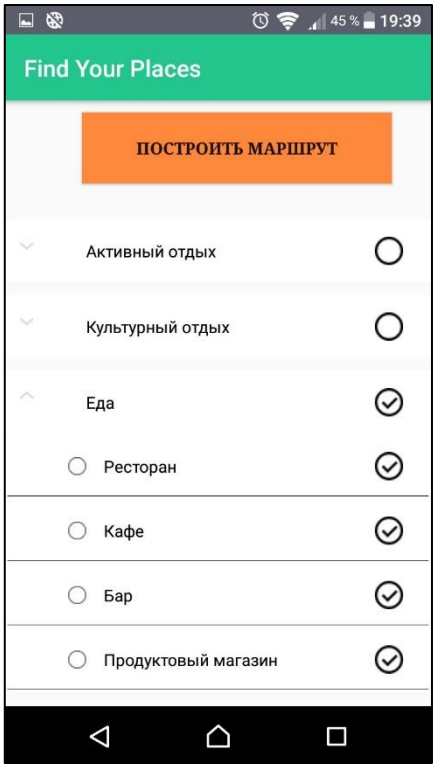


Рисунок 5 – Начальный экран

Рисунок 6 – Экран выбора категорий мест

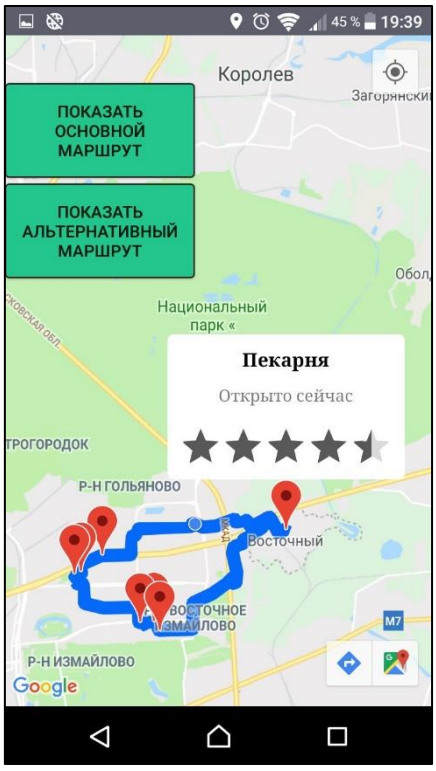


Рисунок 7 – Экран с построенным маршрутом

2) Samsung Galaxy S4 Mini

Характеристики:

Диагональ – 4.3 дюйма

Высота корпуса – 124,6 мм

Ширина корпуса – 61,3 мм

Внешний вид интерфейса приложения:

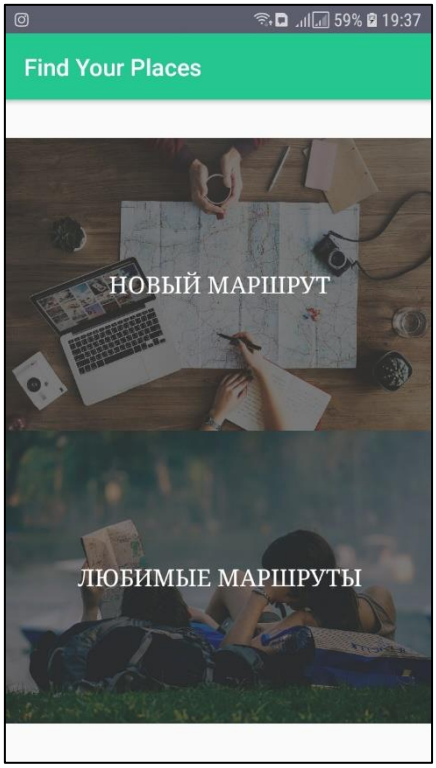


Рисунок 8 – Начальный экран

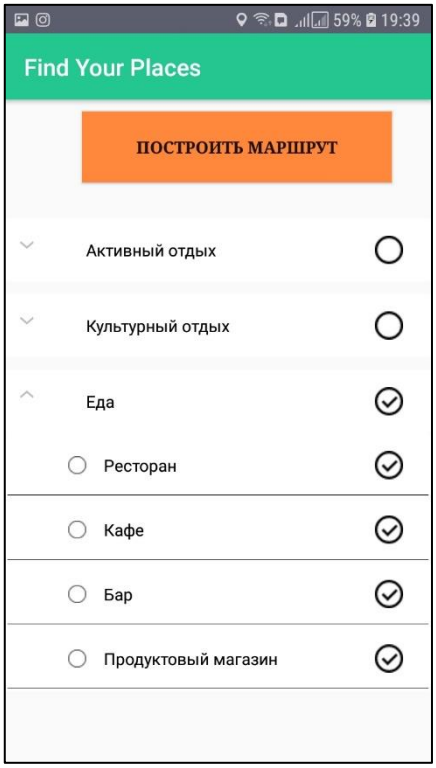


Рисунок 9 – Экран выбора категорий мест

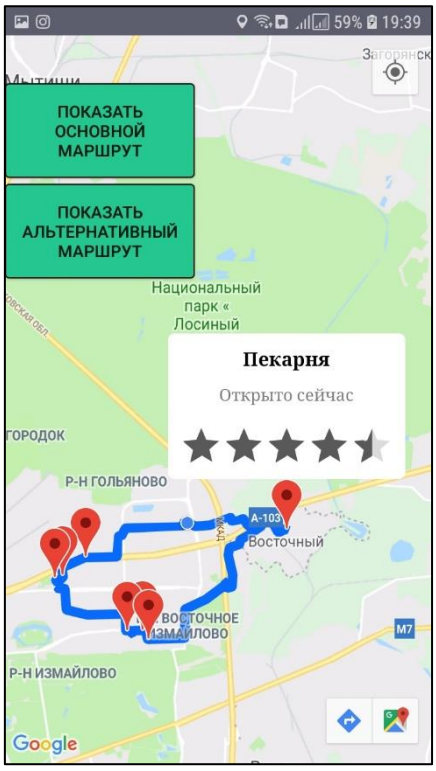


Рисунок 10 – Экран с построенным маршрутом

При тестировании проблем с адаптацией к различным размерам экрана найдено не было. Выявленные локальные ошибки были исправлены при последующей отладке программного обеспечения. В итоге было проведено заключительное тестирование, которое не выявило других ошибок.

Заключение

В процессе работы над проектом были изучены возможности взаимодействия программного обеспечения с различными API картографических платформ, а также основные методы объектно-ориентированного программирования и принципы работы с картами, при разработке приложения в интерактивной среде Android Studio. В итоге был разработан и реализован алгоритм, автоматизирующий построение оптимальных туристических маршрутов. Он позволяет путешественникам не тратить время на составление пути между интересующими культурными и социальными объектами во время поездок, предоставляя на выбор основной и альтернативный маршруты, составленные согласно отведенному времени и персональным предпочтениям пользователя.

Один из недостатков разработанного программного обеспечения – это непосредственная зависимость от сервисов платформы Google. Из-за прямого взаимодействия с различными Google Maps API, программа не сможет осуществлять свою работу в запланированном режиме при возникновении неполадок на стороне платформы. Также стоит отметить, что указанные картографическим сервисом категории мест, которые используются для поиска и составления выборки в ходе работы программы, не всегда совпадают с действительностью, ведь эти данные зачастую сообщаются пользователями, а не владельцами организаций.

В перспективе программное обеспечение может быть доработано и функционально расширено. Для этого можно включить в приложение функцию авторизации и сохранения любимых маршрутов, которые предполагается хранить в базе данных на сервере приложения. Также возможен перенос программного обеспечения на другие операционные системы, такие как, например, iOS.

Список использованных источников

1. Шилдт, Герберт. Java 8: руководство для начинающих, 6-е изд. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2018. – 720 с. : ил.
2. Иванова Г.С. Технология программирования : учебник / Г.С. Иванова. – 3-е изд., стер. – М. : КНОРУС, 2016 – 334 с. – (Бакалавриат).
3. Климов, Александр. Retrofit [Электронный ресурс]. – Режим доступа: http://developer.alexanderklimov.ru/android/library/retrofit.php (дата обращения: 01.01.2019)
4. Яндекс.Карты, 2ГИС или всё же Google Maps? [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/242015/ (дата обращения: 01.01.2019)

Приложение А

Отправка запроса в Directions API

```
// Все place_id остановок маршрута
String fin_waypoints = waypoints24 + waypoints4;
try {
    fin_waypoints = fin_waypoints.substring(0, fin_waypoints.length() - 1);
} catch (Exception e){
    Toast toast = Toast.makeText(getApplicationContext(),
            text: "Не найдено мест поблизости", Toast.LENGTH_LONG);
    toast.show();
}
// Отправка запроса в Directions API

String url2 = "https://maps.googleapis.com/maps/";
// Строим шаблон Builder и используем библиотеку Retrofit

Retrofit retrofit2 = new Retrofit.Builder()
        .baseUrl(url2)
        .addConverterFactory(GsonConverterFactory.create())
        .build();

RetrofitMaps service2 = retrofit2.create(RetrofitMaps.class);
// Указываем все параметры необходимые для отправки запроса

Call<Example2> call2 = service2.getDistanceDuration( origin: origin_latitude + ","
        + origin_longitude,  destination: origin_latitude
        + "," + origin_longitude,  waypoints: "optimize:true|" + fin_waypoints);


call2.enqueue(new Callback<Example2>() {
    @Override
    public void onResponse(Call<Example2> call2, Response<Example2> response2) {
        try {
            //Удаляем предыдущий маршрут с карты
            if (line[0] != null) {
                line[0].remove();
            }

            for (int i = 0; i < response2.body().getRoutes().size(); i++) {
                // Получаем ответ и находим закодированную ломанную линию,
                // которая будет являться маршрутом
                String encodedString = response2.body().getRoutes().get(0).
                                getOverviewPolyline().getPoints();

                // Декодируем полученное
                List<LatLng> poly = new ArrayList<~>();
                int index = 0, len = encodedString.length();
                int lat = 0, lng = 0;

                while (index < len) {
                    int b, shift = 0, result = 0;
```

```
                    do {
                        b = encodedString.charAt(index++) - 63;
                        result |= (b & 0x1f) << shift;
                        shift += 5;
                    } while (b >= 0x20);
                    int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
                    lat += dlat;

                    shift = 0;
                    result = 0;
                    do {
                        b = encodedString.charAt(index++) - 63;
                        result |= (b & 0x1f) << shift;
                        shift += 5;
                    } while (b >= 0x20);
                    int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
                    lng += dlng;

                    LatLng p = new LatLng((((double) lat / 1E5)),
                            (((double) lng / 1E5)));
                    poly.add(p);
                }

                // Рисуем линию на карте
                line[0] = mMap.addPolyline(new PolylineOptions()
                        .addAll(poly)
                        .width(20)
                        .color(Color.parseColor( colorString: "#076AFC"))
                        .geodesic(true)
                );

            }

        } catch (Exception e){
            Log.d( tag: "error",  msg: "Something went wrong");

        }


    @Override
    public void onFailure(Call<Example2> call, Throwable t) {
        Log.d( tag: "onFailure", t.toString());
    }
});
```

Приложение Б

Начало класса MapsActivity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);

    // Узнаем местоположение пользователя, если приложение уже запускалось недавно
    if (savedInstanceState != null) {
        mLastKnownLocation = savedInstanceState.getParcelable(KEY_LOCATION);
        mCameraPosition = savedInstanceState.getParcelable(KEY_CAMERA_POSITION);
    }
    mFusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
    mapFragment.getMapAsync( onMapReadyCallback: this);

    /* Узнаем радиус
     * Он рассчитывается в зависимости от времени введенного пользователем
     * и передается через другие классы */

    radius = Integer.valueOf(getIntent().getStringExtra( name: "radius"));

    /* Узнаем какие категории мест выбрал пользователь
    * parentItems - это Активный отдых, Культурный отдых, Еда
    * childItems - это категории мест*/
    for (int i = 0; i < MyCategoriesExpandableListAdapter.parentItems.size(); i++) {
        for (int j = 0; j < MyCategoriesExpandableListAdapter.childItems.get(i).size(); j++) {
            String isChildChecked = MyCategoriesExpandableListAdapter.childItems.
                    get(i).get(j).get(ConstantManager.Parameter.IS_CHECKED);
            if (isChildChecked.equalsIgnoreCase(ConstantManager.CHECK_BOX_CHECKED_TRUE)) {
                // Узнаем какие категории выбраны и добавляем их в общий список TYPES
                String tvChild = MyCategoriesExpandableListAdapter.parentItems.get(i).
                        get(ConstantManager.Parameter.CATEGORY_NAME) + " " + (j + 1);
                TYPES.add(getCheckedItems(i, j));

            }
        }
    }
    if (TYPES.size() == 0){
        Toast toast = Toast.makeText(getApplicationContext(),
                text: "Укажите категории мест", Toast.LENGTH_LONG);
        toast.show();
        Intent intent = new Intent( action: ".PlacesListActivity");
        startActivity(intent);

    }

}
@Override
protected void onSaveInstanceState(Bundle outState) {
    if (mMap != null) {
        outState.putParcelable(KEY_CAMERA_POSITION, mMap.getCameraPosition());
        outState.putParcelable(KEY_LOCATION, mLastKnownLocation);
        super.onSaveInstanceState(outState);
    }
}
```

Приложение В

Часть одного из POJO классов

```
public class Result {
    @SerializedName("geometry")
    @Expose
    private Geometry geometry;
    @SerializedName("icon")
    @Expose
    private String icon;
    @SerializedName("id")
    @Expose
    private String id;
    @SerializedName("name")
    @Expose
    private String name;
    @SerializedName("opening_hours")
    @Expose
    private OpeningHours openingHours;
    @SerializedName("photos")
    @Expose
    private List<Photo> photos = new ArrayList<~>();
    @SerializedName("place_id")
    @Expose
    private String placeId;
    @SerializedName("rating")
    @Expose
    private Double rating;
    @SerializedName("reference")
    @Expose
    private String reference;
    @SerializedName("scope")
    @Expose
    private String scope;
    @SerializedName("types")
    @Expose
    private List<String> types = new ArrayList<~>();
    @SerializedName("vicinity")
    @Expose
    private String vicinity;
```

```
    @SerializedName("price_level")
    @Expose
    private Integer priceLevel;

    public Geometry getGeometry() { return geometry; }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public OpeningHours getOpeningHours() {
        return openingHours;
    }
    public String getPlaceId() {
        return placeId;
    }

    public Double getRating() {
        return rating;
    }

    public List<String> getTypes() {
        return types;
    }

    public String getReference() {
        return reference;
    }
```