Санкт-Петербургский государственный университет

ЯДРИХИНСКАЯ Юлия Сергеевна

Выпускная квалификационная работа

Разработка пилот-проекта приложения по построению пешеходных маршрутов с пользовательскими параметрами

Уровень образования: бакалавриат

Направление: 05.03.03 «Картография и геоинформатика»

Основная образовательная программа: СВ.5020.2015 «Картография и геоинформатика»

Профиль: «Геоинформатика»

Научный руководитель:
доцент кафедры
картографии и геоинформатики,
Институт Наук о Земле,
к.г.н. Штыкова Наталья Борисовна

Рецензент:
начальник ГИС отдела,
ООО «АВИАЦИОННЫЕ РОБОТЫ»,
Аникин Максим Игоревич

Санкт-Петербург
2019

СОДЕРЖАНИЕ:

Введение

Известен факт, что почти каждый человек старается придерживаться здорового образа жизни. В крупных городах люди все чаще предпочитают гулять пешком или использовать велосипедный транспорт вместо автомобильного. Но добраться пешком как можно быстрее из пункта А в пункт Б не единственный вариант прохождения по маршруту. Его можно спланировать так, чтобы добраться до пункта назначения, максимально избегая шума автомобильных дорог. Или построить маршрут, проходящий через озелененные территории города. Пройти через наибольшее количество интересных объектов - популярный вид пешеходного маршрута для городов с богатым культурным наследием. Прохождение последним маршрутом привлекает не только туристов, но и жителей города. Отсюда возникает необходимость в создании веб-приложения, с помощью которого осуществлялись бы построения различных видов пешеходных маршрутов, определяемых пользователем, который в свою очередь мог бы просматривать объекты, встречающиеся по построенному маршруту, и задавать определенные параметры.

На сегодняшний день существуют несколько функционирующих приложений по построению пешеходных маршрутов. Однако все они имеют свои ограничения. Например, отсутствие возможности задать время для прогулки, построить несколько видов маршрутов в одном приложении или просмотреть встречающиеся по пути интересные объекты.

Целью дипломной работы являлось создание пилот-проекта веб-приложения, предоставляющее функцию построения пешеходных маршрутов с пользовательскими параметрами, такими как: выбор вида пешеходного маршрута, задание максимального времени для прогулки, обзор кофеен и достопримечательностей города по маршруту.

Для достижения поставленной цели необходимо было решить следующие задачи:

- проанализировать функциональные возможности существующих приложений по построению пешеходных маршрутов;
- подготовить исходные данные для проекта;
- выбрать алгоритмы для построения пеших прогулок с различными параметрами;
- создать базу данных для реализации построения пешеходных маршрутов с использованием запросов;
- разработать функционирующее приложение, основанное на клиент-серверной архитектуре;

Взаимодействие пользователя с приложением осуществляется с помощью веб-браузера. Исходными данными для построения пешеходных маршрутов послужили материалы курсовой работы, которая выполнялась в прошлом году; данные об объектах,

полученные с помощью запросов к поисково-картографической службе «Яндекс.Карты» и программные коды для определения шумового загрязнения. Тестовая территория пилот-проекта находится в городе Санкт-Петербург и расположена следующим образом: южная граница проходит вдоль Невского проспекта, северо-западная и северо-восточная границы – набережные рек Нева и Фонтанка соответственно.

Глава 1. Анализ существующих проектов по построению пешеходных маршрутов.

1.1. Существующие алгоритмы и приложения для построения пешеходных маршрутов.

В настоящее время разновидностей пешеходных прогулок достаточно велико. Один из самых популярных – добраться из пункта А в пункт Б самым коротким путем. Помимо него набирают популярность пешеходные маршруты, при построении которых учитываются различные параметры. Исследователи предлагают свои теории построения маршрутов, учитывая как безопасность (освещение) городских улиц, число фотографий, сделанных пользователями социальных сетей, так и экологические факторы городской среды.

Разработанная система пешеходных прогулок, описанная Ясуфуми Такамака, имеет своей целью улучшение здоровья человека (Yasufumi, T. et al., 2017). Авторы статьи предлагают строить пешеходные маршруты, учитывая указанные пользователем калории, которые он хочет потратить по пути следования до конечной точки.

Популярные сервисы, такие как Instagram и ВКонтакте, предоставляют возможность анализировать огромный набор данных о фотографиях. Учитывая это, были обнаружены точки притяжения туристов в городе Санкт-Петербург на основе данных Instagram, которые также могут использоваться для построения пешеходных маршрутов (Mukhina, K. et al., 2017). Существует проект сервиса по созданию самых красивых пешеходных маршрутов в городе Санкт-Петербург. Алгоритм будущего сервиса разработан с учетом данных о фотографиях, опубликованных в социальной сети ВКонтакте и имеющих географические координаты (Semenov A.,2018). Исследователи из Yahoo также использовали фотографии пользователей социальных сетей с известными координатами. Они разработали алгоритм для пешеходов, который позволяет создавать самый красивый, тихий и улучшающий эмоциональное состояние человека пешеходный маршрут (Quercia, D. et al., 2014).

Была разработана система, учитывающая уличное освещение для построения пешеходных маршрутов (Miura, H. et al., 2011). Исследователи все больше обращают внимание на ночное освещение городских улиц при разработке алгоритмов для построения пешеходных прогулок, потому что, во-первых, улицы без освещения могут быть опасными; во-вторых, красивые ночные улицы городов всегда привлекали туристов (Guo, Q., et al.,2011).

Еще одна теория построения пешеходных маршрутов была разработана китайскими исследователями (Zhong, W. et al., 2017). Их идея основана на том, что пользователь может совмещать передвижения и пешком, и с использованием велосипедного транспорта. Авторы

предлагают включать в маршрут две точки расположения велопарковочных станций, с учетом которых маршрут разделяется на три части: пешеходная (от начальной точки до велопарковочной станции), велосипедная (между двумя станциями) и пешеходная (от второй велостанции до конечной точки маршрута). Помимо этого, пользователь имеет возможность выбирать три разновидности будущего маршрута, основанные на алгоритме Дейкстры (Кормен, Т. и др., 2005): кратчайший, самый безопасный (алгоритм учитывает число фактов нарушения правопорядка между крайними точками пути) и оптимальный (с учетом двух предыдущих условий). Такой подход для движения по улицам города применим только для крупных населенных пунктов, где достаточно развита сеть велопарковочных станций, расположенных равномерно по всему городу.

На сегодняшний день известно несколько действующих приложений для построения городских пешеходных маршрутов с учетом различных факторов. Алгоритм, учитывающий факторы окружающей среды, был разработан в компании Urbica (https://medium.com/@urbica/walkstreets-5a41b22ae104). Их приложение под названием «Walkstreets» предлагает строить самые экологичные, тихие или чистые маршруты. Одним из важных отличий от других приложений является способность навигатора предлагать пешеходный маршрут до произвольной точки назначения. Это очень полезно, если пользователь просто хочет прогуляться по приятным местам и не заботиться о пункте назначения. Компания контролирует качество воздуха и городской шум, а также использует спутниковые снимки для обнаружения озелененных зон, что учитывается при построении маршрутов. Данное приложение работает только на территории города Москва.

Сервисы, которые удовлетворяют требованию построения туристических пешеходных маршрутов, проходящих через достопримечательности Санкт-Петербурга, называются «Wander» (мобильное приложение) и «Travelpath» (версия, запускаемая из браузера) (http://travelpath.ru/) (Рис.1). По мнению создателя проекта, в базах данных обеих версий собраны наиболее популярные музеи, парки, памятники и церкви, через которые строятся пешеходные маршруты. В обеих версиях проекта есть возможность выбрать опцию для прохождения по отдельности через популярные музеи, парки, памятники, церкви, или выбрать одновременно все пункты для их учета при построении маршрута. Пользователь может посмотреть подробную информацию об этих объектах. В версии, запускаемой из браузера, возможно задание только крайних остановок планируемого маршрута. После его построения не отображается информация о длине и времени прохождения маршрута.

Что касается мобильной версии «Wander», то она доступна только для пользователей платформы Android. В ней предоставлено немного больше опций, чем в версии, запускаемой

из браузера. В добавлении к ним, есть информация о количестве точек (достопримечательностей), времени и длине маршрута (Рис. 2).

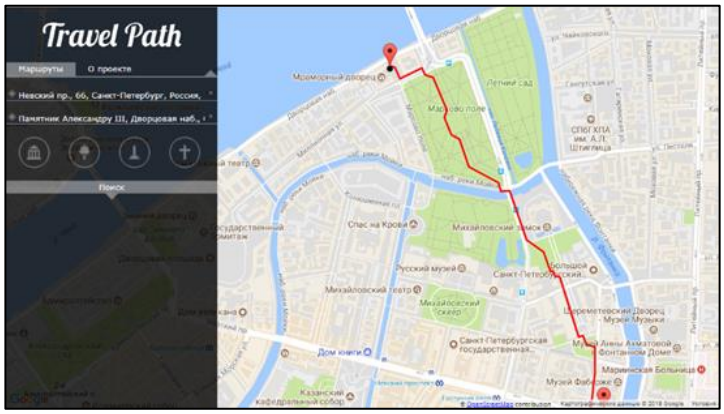


Рис.1. Сервис Travel Path.

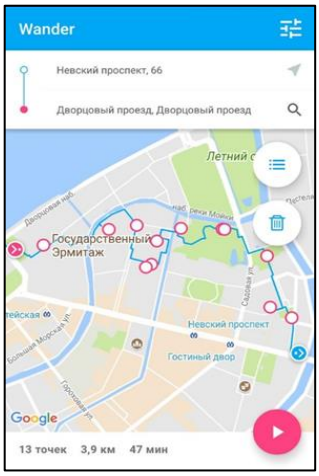


Рис.2. Приложение "Wander"

Реализованный алгоритм в приложениях «Wander» и «Travelpath» учитывает только достопримечательности, локализованные в точке. К недостаткам приложения можно отнести то, что при построении пешеходных маршрутов не учитываются расположения набережных города, которые также могут заинтересовать туристов. Имеются неудобства в использовании, такие как невозможность просмотреть длину и время на прохождение построенного маршрута (для браузерной версии). Отсутствует возможность выбрать дополнительные параметры маршрута, например, указание желаемого времени прохождения маршрута и просмотр местоположений кофеен по пути следования. Последняя опция была бы полезна для пользователей, планирующих продолжительные прогулки. В версии, запускаемой из браузера после построения не отображаются объекты по пути маршрута, указанные в меню программы (парки, музеи и так далее). Это реализовано в мобильной версии, но тип объекта сложно понять, не нажав на него – условные обозначения музеев, парков, церквей и памятников одинаковые.

Таким образом, существующие приложения для построения пешеходных прогулок только по отдельности позволяют строить маршруты с учетом факторов окружающей среды и с прохождением через достопримечательности города. Отсутствует единый сервис, позволяющий строить разные виды маршрутов на территории Санкт-Петербурга. Имеются ограничения в выборе дополнительных параметров при построении маршрутов. Возникают неудобства при пользовании интерфейсом приложений с целью просмотра объектов вдоль построенного маршрута.

1.2. Настольное приложение как инструмент планирования пешеходных маршрутов с учетом достопримечательностей и факторов окружающей среды.

Ранее в рамках курсовой работы был разработан пилот-проект по построению пешеходных маршрутов, проходящих через достопримечательности. Территории нынешнего и сегодняшнего пилот-проектов совпадают. Работа осуществлялась в программном продукте ArcGIS Desktop, при этом были выполнены следующие задачи:

– выгружены общегеографические данные из открытого источника OpenStreetMap (далее OSM) на исследуемую территорию и преобразованы в прямоугольную проекцию Гаусса-Крюгера;

– найдено и исправлено значительное количество топологических ошибок дорожной сети с учетом дальнейшего построения маршрутов для пешеходов;

– присвоены названия пешеходных дорог и созданы локаторов адресов, необходимых для создания путевого листа следования по маршруту;

– разработана цепочка геообработки для построения результирующих маршрутов; при этом сам маршрут проходил через достопримечательности вблизи кратчайшего маршрута.

Работа над этим проектом дала необходимый практический опыт использования сетевой модели данных; позволила создать топологически корректную модель пешеходных дорог, которая используется в нынешнем проекте как основа для построения пешеходных маршрутов. В тоже время ограничениями проекта являлись:

– реализация в специализированном ПО, что ограничивает возможность применения проекта широким кругом пользователей;

– малое количество достопримечательностей, использованных для построения пешеходных маршрутов, проходящих через них;

– возможность строить только один вид маршрутов;

– отсутствие опции задания желаемого времени прохождения маршрута и других параметров.

В связи с этим было принято решение создать версию настольного приложения с использованием свободно распространяемого программного обеспечения, которое позволит строить пешеходные маршруты для прогулок в городе Санкт-Петербург, а именно: «самый интересный» маршрут - строится через популярные достопримечательности; «самый тихий» - создается с учетом распространения шумового загрязнения; «самый экологичный» - проходит через озелененные территории; «самый быстрый» - кратчайший маршрут.

«Самый интересный» маршрут – строится с учетом популярности памятников, фонтанов, зданий с архитектурной ценностью, мостов, набережных города, площадей и озелененных территорий в общем рейтинге, составленном поисково-информационной

картографической службой «Яндекс.Карты» (https://yandex.ru/maps). Такой подход к оценке перечисленных объектов является предельно объективным, поскольку вычисляется с учетом оценивания большого количества пользователей «Яндекс.Карт». «Самый тихий маршрут» - создается на основе данных по «карте» шумового загрязнения на территорию проекта и проходит через наименее шумные участки местности. «Самый экологичный маршрут» - строится с учетом расположения различных озелененных территорий без градации их по популярности. Алгоритм при построении пешеходного маршрута учитывал близко расположенные объекты между крайними точками прогулки. «Самый быстрый маршрут» - кратчайший маршрут между двумя заданными точками.

Отличие разрабатываемого приложения от других заключается в том, что «самый интересный» маршрут включает в себя данные намного большего количества объектов, чем в вышеописанных приложениях (другими словами, не только локализованных в точке, но и линейных и площадных). Помимо этого, пользователь может строить три разновидности маршрутов в одной программе, а также выбирать параметр для выбора просмотра местоположений кофеен вблизи построенного маршрута. Предполагается сделать возможным просмотр обозначенных достопримечательностей города вдоль построенного маршрута. Пользователь также сможет указывать максимально возможное время прохождения маршрута, оценивать расстояние между крайними точками по маршруту. Если пользователь указал время, недостаточное даже для прохождения маршрута самым коротким путем, он сможет узнать время прохождения кратчайшего маршрута и указать уже корректное время для успешного построения маршрута. Также отличительной особенностью будет являться то, что построенный маршрут можно будет сравнивать с кратчайшим, построенным через те же крайние точки.

1.3. Анализ и выбор программного обеспечения для разработки пилот-проекта.

Для достижения поставленной цели необходимо было выбрать не только программное обеспечение для решения различных задач на протяжении всего процесса создания пилот-проекта, но и подходящие языки программирования, применяемые для написания алгоритмов как во время работы с исходными данными, так и при разработке приложения.

Существует множество геоинформационных программных продуктов, среди них наиболее популярными считаются: Quantum GIS (далее QGIS), ArcGIS Desktop (далее ArcGIS), GRASS GIS (далее GRASS GIS), MapInfo, «Панорама», «Нева». Последние три не

подходили для решения задач проекта. Основные причины: ограниченный доступ для пользования программными продуктами, предоставление узкого спектра инструментов геообработки, отсутствие интеграции с созданной базой данных (далее БД) (ГИС «Нева») и возможности запуска написанных на языке программирования геоалгоритмов. Несмотря на свободное распространение программного продукта GRASS GIS, его возможности также не были задействованы, поскольку существуют различные трудности при работе с векторными данными и имеются неудобства в пользовании, как у вышеописанных трех программ.

Для осуществления цепочки геообработки исходных данных и визуализации результатов на различных этапах создания пилот-проекта были задействованы две географические информационные системы (далее ГИС): QGIS и ArcMap (один из продуктов семейства ArcGIS).

QGIS является свободно распространяемой программой с открытым исходным кодом. Для решения задач проекта система была удобна при анализе, редактировании и визуализации как векторных, так и растровых данных. QGIS предоставляет возможность простого подключения к выбранной пространственной БД и работы с ней (что не сказать о работе с БД в ArcGIS): редактирование, просмотр и анализ данных, хранящихся в базе. Есть возможность визуализировать построенные маршруты посредством запросов к БД. что являлось основной причиной работы в этой ГИС.

Инструменты не менее популярного семейства программных продуктов ArcGIS также были применены для решения определенных задач. В отличие от QGIS, ArcGIS не является программой с открытым исходным кодом, поэтому она не была выбрана в качестве основного геоинформационного программного обеспечения. Основная причина использования возможностей ArcGIS – удобство пользования некоторыми инструментами, а именно: пространственное соединение атрибутов векторных данных и вычисление евклидового расстояния на растре. Эти встроенные инструменты были необходимы в процессе подготовки данных для построения всех видов маршрутов. Создание нового инструмента, алгоритм работы которого был написан на языке программирования Python, также осуществлялся посредством возможностей ArcGIS. Новый инструмент требовался для решения задачи создания растровых данных на основе вычисления евклидового расстояния. Написанный программный код инструмента запускался из-под ArcMap и результат по окончанию работы сразу можно было оценить в диалоговом окне программы.

Таким образом, обе ГИС дополняли друг друга как во время работы с исходными данными и созданием производных материалов, так и для просмотра результатов построения

трех видов пешеходных маршрутов. Более того, обе программы поддерживают общий формат векторных данных - .shp, что также было удобно при работе в обеих программах.

В процессе работы над проектом создавались собственные алгоритмы для решения конкретных задач. Часть программных кодов алгоритмов была написана на языке Python, часть на JavaScript (далее JS). Использование обоих языков было нужно для решения разных типов задач. Например, использование языка программирования JS было необходимо для написания кода клиентской части приложения (см. главу 3). Применение возможностей языка Python требовалось для создания собственного инструмента в ArcMap, где другой язык программирования не применим. Кроме того, возможности Python были задействованы при выгрузке данных с помощью «API поиска по организациям» компании Яндекса (см. раздел 1.4) и их последующем представлении в виде shape-файлов, а также для разработки серверной части приложения и обеспечения его функциональности (создания веб-сервиса) (см. главу 3).

Python – высокоуровневый интерпретируемый язык программирования с динамической типизацией (Лутц М, 2011).

Его преимущества, которые учитывались при выборе именно этого языка программирования:

- свободное распространение с открытым исходным кодом;
- высокоуровневость - простота в использовании, поскольку он приближен больше к человеческому языку, чем к машинному - залог высокой производительности труда;
- интерпретируемость – выполняет написанный код построчно, есть возможность быстрого редактирования при возникновении ошибок;
- динамическая типизация – не нужно заранее объявлять типы данных используемых переменных;
- автоматическое управление памятью – язык Python автоматически распределяет память под объекты и освобождает ее, когда объекты больше не используются;
- огромное количество встроенных библиотек, позволившие решить разнообразные задачи в процессе разработки пилот-проекта;
- возможность интеграции с программными компонентами языка JS (Доусон М., 2014).

Во время создания проекта были использованы следующие библиотеки языка Python:

- ogr – создание shape-файлов;

– fiona – удаление из shape-файла лишних выгруженных объектов;
– shapely – работа с shape-файлами.

Использованные модули языка:

– arcpy – запуск написанных алгоритмов в ArcMap;
– json – работа с одноименным форматом;
– os - работа с операционной системой;
– urllib – получение описания используемой проекции при создании shape-файла;
– re - работа с регулярными выражениями;
– request – работа с http-запросами;
– shutil - обработка файлов;
– collections – работа со словарями.

Помимо перечисленных выше применений языка на всех этапах создания приложения, он также использовался для построения «карты» шумов. Программные коды для создания этой «карты» были написаны на языке Python и предоставлены разработчиками приложения «Walkstreets» компании Urbica (https://github.com/urbica/noisemap).

В рамках данной работы язык JS применялся при написании клиентской части приложения. JS – это высокоуровневый интерпретируемый язык программирования. Иногда JS называют языком сценариев, поскольку его основное применение (в отличии от языка Python) – обеспечивать работу приложения на стороне клиента. Любой веб-браузер может работать с JS кодами, поскольку каждый из них содержит уже установленный интерпретатор этого языка. Этот факт облегчил работу над созданием приложения.

JS поддерживает объектную модель документа (Document Object Model или DOM), которая вместе со сценариями позволили обращаться к объектам веб-страниц и модифицировать их свойства и отображение; изменять поведение веб-страницы приложения для придания интерактивности, тем самым обеспечивая взаимодействие пользователя с ее содержимым (Флэнаган Д., 2008). Например, интерактивность заключается в добавлении маркеров и их перемещении, выборе вида маршрута и желаемого времени его прохождения, нажатии на кнопку, что вызывает функцию построения пешеходного маршрута и так далее.

Помимо встроенных функций, были задействованы возможности библиотеки языка Leaflet, которая позволила осуществить следующие: отображение карты и географических координат курсора мыши, добавление и перемещение маркеров (крайних остановок маршрута), определение их адресов, визуализация пешеходных маршрутов и отображение объектов вдоль них (кофеен, фонтанов, мостов и так далее). Инструменты библиотеки jquery обеспечили обращение к элементам веб-страницы и их свойствам.

Вся функциональность приложения входит в понятие «сервис». Для его проектирования была выбрана REST архитектура. REST (Representational state transfer - передача состояния представления) – это стиль архитектуры программного обеспечения для взаимодействия компонентов приложения.

Основные причины выбора REST архитектуры для проектирования веб-сервиса:

– производительность и эффективность сети с точки зрения пользователя;

– простота в использовании. Запросы передаются на сервер в понятном виде, нет необходимости разбираться в его формате и осуществлять дополнительные действия, в отличии от запросов веб-сервисов SOAP и XML-RPC;

– данные серверу передаются без изменения их формата. Например, при использовании веб-сервисами SOAP и XML-RPC данные для передачи нужно конвертировать в XML формат.

Ресурсы приложения (см. главу 3) являются ключевым понятием в REST архитектуре (Gaston C. Hillar., 2016). Она определяет доступ к ним, который осуществлялся через метод GET (получение информации о ресурсе) – единственный метод, использовавшийся для отправки запросов на сервер приложения.

Одна из возможностей сервера приложения – работа с базой данных. В целях применения существующих алгоритмов для построения пешеходных маршрутов, а именно Дейкстры и k-кратчайших путей (см. главу 3), было использовано свободно распространяемое программное обеспечение pgRouting, которое является расширением БД PostGIS и поддерживает функции маршрутизации и другого сетевого анализа (https://pgrouting.org/). PostGIS - расширение системы объектно-реляционной базы данных PostgreSQL (https://postgis.net/). PostGIS позволяет хранить пространственные объекты и поддерживает функции для их анализа и обработки. QGIS поддерживает взаимодействие с PostGIS, что также определило выбор обеих программ для разработки проекта. Кроме дорожного графа для построения маршрутов, база данных была необходима для хранения объектов, обозначенных вдоль любого построенного маршрута в приложении.

В результате анализа возможных инструментов для разработки проекта, были выбраны две взаимодополняющие друг друга геоинформационные системы: QGIS и ArcGIS, система управления базой данных PostGIS, несколько языков программирования (JS и Python) для решения как подготовительных задач перед разработкой приложения, так и для создания клиентской и серверной его частей. Обозначено направление проектирования веб-сервиса как функциональной составляющей приложения.

1.4. Источники исходных данных для построения пешеходных маршрутов.

Исходными данными для проекта послужили элементы общегеографической основы и информация о кофейнях, достопримечательностях, набережных и озелененных территориях: название, координаты местоположения и рейтинговая оценка.

Элементами общегеографической основы послужили shape-файлы озелененных территорий и топологически корректной дорожной сети, подготовленные в рамках курсовой работы. Дорожная сеть требовалась для построения пешеходных маршрутов, shape-файл озелененных территорий – для построения «самых экологичных» маршрутов.

В качестве исходных материалов использовались программные коды, предоставленные компанией Urbica (https://github.com/urbica/noisemap). Эти программы были необходимы для создания «карты» шумового загрязнения.

В целях пользования картой, расположенной на веб-странице приложения, и получения данных о достопримечательностях были задействованы API источников этих данных.

API (англ. Application Programming Interface) — это интерфейс программирования приложений. API сервиса предоставляет набор готовых процедур, функций и структур, с помощью которых разработчики могут создавать свои программы, приложения для работы с сервисом (https://tech.yandex.ru/direct/doc/dg/concepts/about-docpage/). Другими словами, API — это конструктор, в котором есть набор функций, методов и правил их использования, с помощью которых можно создать свое собственное приложение (рис.3).

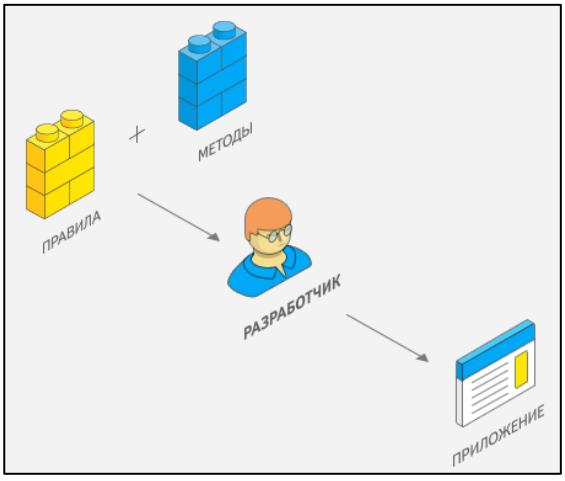


Рис.3. Представление об API. Источник: https://tech.yandex.ru/direct/doc/start/intro-docpage/ - API Яндекс.Директа.

При работе через API разработчики одной программы имеют возможность составить запрос к серверу другой программы и получить желаемый ответ. Так, API онлайн-сервиса

Mapbox предоставил возможность отображения карты OSM на веб-странице разработанного приложения. Для этого был создан аккаунт на сайте Mapbox, далее в программном коде JS клиентской части приложения был составлен запрос к онлайн-сервису с указанием кода доступа. Отображаемая карта приложения является результатом запроса, выполняемого каждый раз при загрузке веб-страницы (рис.4).

Рис. 4. Отображение карты в приложении с использованием возможностей Mapbox API.

Информация о достопримечательностях и кофейнях была получена из поисково-информационной картографической службы «Яндекс.Карты» при помощи API-поиска по организациям. API-поиск по организациям - сервис, который используется для поиска географических объектов и организаций (https://tech.yandex.ru/maps/doc/geosearch/concepts/about-docpage/). Сервис позволяет искать улицы, дома, достопримечательности, кафе и другие объекты.

Для задач проекта требовалось получить названия, координаты и оценки достопримечательностей по тексту поискового запроса – прямой поиск. Его результаты возвращались в формате .json.

Формат запроса. Начало запроса: 'https://searchmaps.yandex.ru//v1//. Далее указывались параметры запроса (под номерами 1,2,4 являлись обязательными):

1. [apikey] - ключ доступа к сервису;
2. [text] - текст поискового запроса. Использовалось название трех типов объектов (см. раздел 2.1);
3. [type] - типы возвращаемых результатов. Использованное значение: biz — организации;
4. [lang] - язык ответа. Выбирался русский язык;
5. [ll] - центр области поиска, задавался долготой и широтой в градусах в виде десятичной дроби, разделенных запятой;

6. [spn] - размер области поиска, указывался в виде долготы и широты в градусах в виде десятичной дроби, разделенных запятой;

7. [results] - количество возвращаемых объектов. Максимальное значение — 500;

Формат ответа. Сервер возвращал найденные объекты, количество которых в одном поисковом запросе не могло быть больше 500. Ответ был представлен в виде вложенных словарей с ключами. Названия некоторых из них:

- results - максимальное количество возвращаемых результатов;
- skip - количество пропускаемых результатов;
- found - количество найденных объектов;
- type - тип геометрии;
- coordinates - координаты объекта;
- boundedBy - границы области показа найденных объектов;
- geocoderMetaData - подробная информация о найденном объекте;
- text - полный информация об объекте.

Таким образом, в ходе анализа существующих приложений по построению пешеходных маршрутов были сформулированы функциональные особенности разрабатываемого приложения, которые будут преимуществом перед разработанными приложениями по построению пешеходных маршрутов. Основные из них: построение нескольких видов маршрутов, указание времени для прогулки и дополнительных параметров. Были определены источники исходных данных и геоинформационные программные продукты для их обработки (QGIS и ArcGIS). Обозначено направление создания веб-приложения, а именно: разработка его клиентской и серверной частей (в последнюю входит взаимодействие с БД и проектирование веб-сервиса для построения пешеходных маршрутов). Обозначены области применения функциональных возможностей двух языков программирования на всех этапах разработки проекта.

Глава 2. Модели и алгоритмы, использованные при создании пилот-проекта приложения.

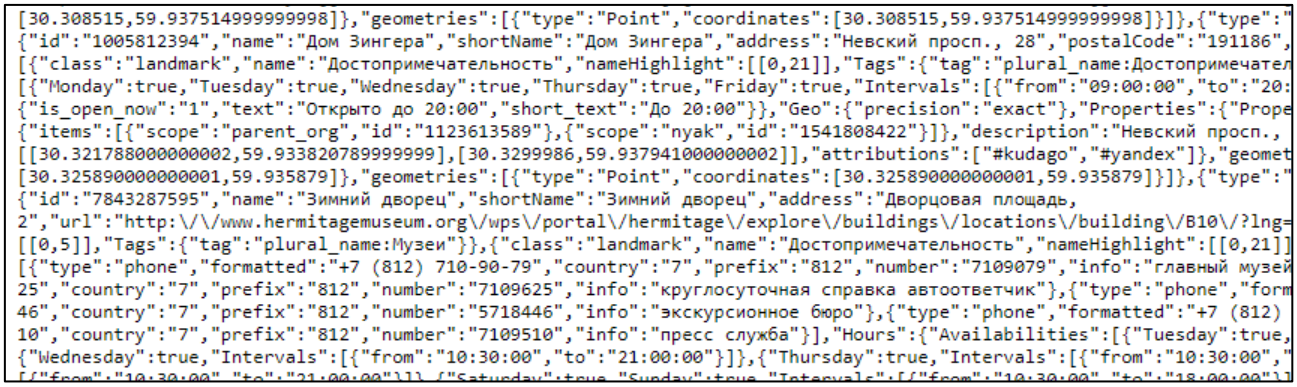
2.1. Получение и представление исходных данных в нужном формате для дальнейшего использования.

Для получения информации о достопримечательностях и кофейнях был выбран источник «Яндекс.Карты». Сначала была создана электронная почта с доменом «Яндекса». Затем получен API-ключ для доступа к пространственным данным, который позволил за один поисковый запрос получать до 500 объектов, что было достаточно для решения поставленной задачи.

Выгрузка объектов производилась путем нескольких запросов к серверу, состоявших из:

1. текста поискового запроса. Например, «достопримечательность», «фонтаны», «памятники»;
2. координат точки – центра области поиска в формате «долгота/широта»;
3. размера области поиска по долготе и широте, «0.04,0.04»;
4. типа возвращаемых результатов (выбран «biz» - организации);
5. языка возвращаемых объектов (выбран параметр «ru_RU» - русский);
6. количества возвращаемых объектов – 500;
7. API-ключ.

Запрос выглядел следующим образом: 'https://searchmaps.yandex.ru//v1//?text=1&ll=2&spn=3&type=4lang=5&results=6&apikey=7', где цифры – параметры запроса, описанные выше. Результат запроса представлял собой вложенные словари с ключами (рис.5), значения которых необходимо было извлечь для получения координат объектов, их названий и оценок. Поскольку объектов было достаточно много чтобы вручную извлекать вышеперечисленные параметры, был написан программный код на языке Python.



[30.308515,59.937514999999998]},"geometries":[{"type":"Point","coordinates":[30.308515,59.937514999999998]}]},{"type":"
{"id":"1005812394","name":"Дом Зингера","shortName":"Дом Зингера","address":"Невский просп., 28","postalCode":"191186",
[{"class":"landmark","name":"Достопримечательность","nameHighlight":[[0,21]],"Tags":{"tag":"plural_name:Достопримечател
[{"Monday":true,"Tuesday":true,"Wednesday":true,"Thursday":true,"Friday":true,"Intervals":[{"from":"09:00:00","to":"20:
{"is_open_now":"1","text":"Открыто до 20:00","short_text":"До 20:00"}},"Geo":{"precision":"exact"},"Properties":{"Prope
{"items":[{"scope":"parent_org","id":"1123613589"},{"scope":"nyak","id":"1541808422"}]},"description":"Невский просп.,
[[30.321788000000002,59.933820789999999],[30.3299986,59.937941000000002]],"attributions":["#kudago","#yandex"]},"geomet
[30.325890000000001,59.935879]},"geometries":[{"type":"Point","coordinates":[30.325890000000001,59.935879]}]},{"type":"
{"id":"7843287595","name":"Зимний дворец","shortName":"Зимний дворец","address":"Дворцовая площадь,
2","url":"http:\/\/www.hermitagemuseum.org\/wps\/portal\/hermitage\/explore\/buildings\/locations\/building\/B10\/?lng=
[[0,5]],"Tags":{"tag":"plural_name:Музеи"}},{"class":"landmark","name":"Достопримечательность","nameHighlight":[[0,21]]
[{"type":"phone","formatted":"+7 (812) 710-90-79","country":"7","prefix":"812","number":"7109079","info":"главный музей
25","country":"7","prefix":"812","number":"7109625","info":"круглосуточная справка автоответчик"},{"type":"phone","form
46","country":"7","prefix":"812","number":"5718446","info":"экскурсионное бюро"},{"type":"phone","formatted":"+7 (812)
10","country":"7","prefix":"812","number":"7109510","info":"пресс служба"}],"Hours":{"Availabilities":[{"Tuesday":true,
{"Wednesday":true,"Intervals":[{"from":"10:30:00","to":"21:00:00"}]},{"Thursday":true,"Intervals":[{"from":"10:30:00","

Рис. 5. Результат запроса для получения данных о достопримечательностях

Алгоритм его работы был следующий:

1. импорт необходимых модулей;
2. формирование запроса для выгрузки информации о достопримечательностях в формате .json;
3. создание функции, которая возвращает описание проекции далее созданного точечного .shp файла по коду обозначения систем координат EPSG;
4. создание точечного .shp файла, в атрибутивной таблице которого содержалось два поля – название и оценка достопримечательности.

При работе с «Яндекс.Картами» было замечено, что понятия «достопримечательность», «фонтан» и «памятник» для поисковой службы являлись различными. Учитывая это, были написаны еще два программных кода для выгрузки фонтанов и памятников раздельно. В кодах были изменены только параметры запроса, а именно текст поискового запроса. В shape-файле достопримечательностей содержались объекты – центроиды озелененных территорий. Они были использованы для присвоения озелененным территориям в виде полигонов оценок, которых не было в исходных shape-файлах этих объектов. Последние были взяты из материалов курсовой работы. Присвоение оценок проводилось в программном продукте QGIS с помощью модуля «NNJoin». В дальнейшем, центроиды были удалены из shape-файла достопримечательностей. Достопримечательностям, фонтанам и памятникам, имевшие оценку «0» в «Яндекс.Картах», была присвоена новая оценка со значением «0,5». Это было сделано, для того чтобы наличие этих объектов в дальнейшем учитывалось при построении пешеходных маршрутов. Было произведено устранение дублирующихся объектов. Для этого в программном продукте ArcMap был задействован инструмент «Выборка по местоположению», в диалоговом окне которого выбирался параметр для выделения объектов с идентичными координатами. В результате работы инструмента дублирующиеся объекты были выделены и затем удалены.

В результате, были получены три точечных shape-файла, в которых содержались данные о названии и оценке каждой достопримечательности (далее в это понятие входят фонтаны и памятники). Три точечных shape-файла были объединены в один для дальнейшей работы. Данные о кофейнях были представлены в отдельном shape-файле, в атрибутивной таблице которого содержалось только название этих объектов.

Размер области поиска достопримечательностей был выбран больше, чем тестовая территория. Следовательно, количество объектов в результате выполнения запроса к «Яндекс.Картам» выгрузилось больше. Избыточные данные понадобились для создания растрового изображения суммарного влияния всех достопримечательностей на территории

проекта, что описано далее. Данные за пределами территории позволили создать корректные результаты на ее границах.

Что касается набережных города, то их оценка была так же получена из «Яндекс.Карт» и занесена в атрибутивную таблицу созданного shape-файла этих объектов. Остальные исходные данные проекта в виде элементов общегеографической основы были взяты из материалов курсовой работы, а именно shape-файлы озелененных территорий, упомянутых ранее и топологически корректной дорожной сети.

Для показа объектов вокруг пешеходных маршрутов в приложении были задействованы выгруженные данные из «Яндекс.Карты» и shape-файл зданий, полученный при создании «карты» шумового загрязнения. Их подготовка происходила для каждого вида объектов по отдельности: создавались два файла для каждого вида объектов: один для отображения в приложении, второй для нахождения пересечения пешеходных дорог с объектами (см. раздел 2.6).

Архитектурные достопримечательности, обозначающие вдоль строящихся маршрутов в приложении, включают в себя объекты в зданиях или сами здания, плюс Дворцовую площадь. Для получения итогового точечного shape-файла производились следующие действия. Сначала была ответворизована Дворцовая площадь, которая была добавлена в shape-файл зданий. Затем при работе инструмента «Spatial Join» в программе ArcMap были присоединен атрибут «имя» точечного shape-файла достопримечательностей (без учета памятников и фонтанов; этот shape-файл был сформирован после выгрузки данных с «Яндекс.Карты» на территорию, большую проекта) к обновленному файлу зданий. Отметим, что некоторые здания геометрически содержали несколько достопримечательностей или находились в десятках сантиметров от них, относясь при этом к зданиям. Учитывая все это, в параметрах используемого инструмента были указаны радиус поиска достопримечательностей – 50 сантиметров и разделение запятой названий достопримечательностей, геометрически содержащихся в одном и том же здании (рис.6).

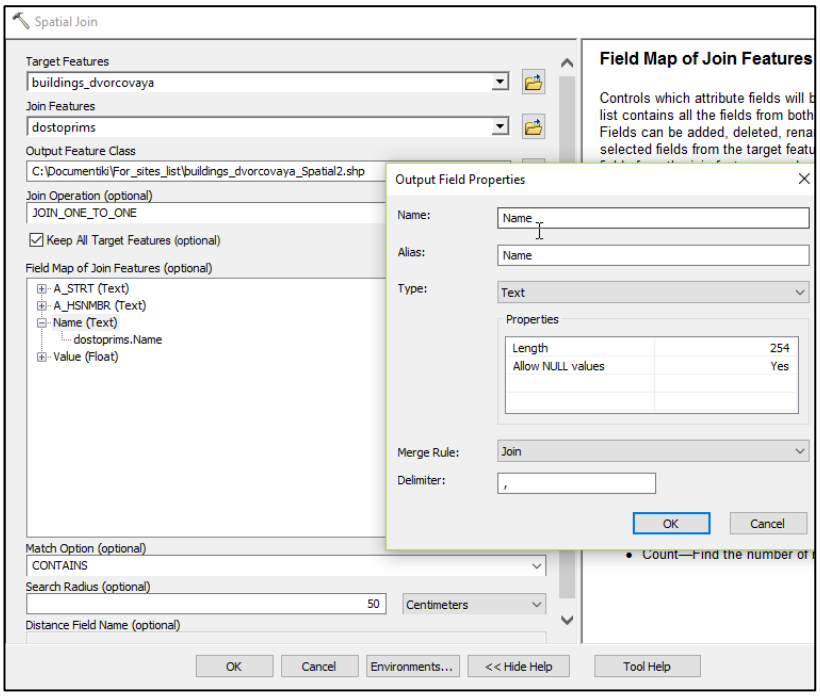


Рис.6. Параметры инструмент «Spatial Join» в программе ArcMap.

Далее были удалены объекты из shape-файла зданий+Дворцовая площадь, которые не заимствовали каких-либо имен достопримечательностей. С помощью инструмента «Polygon centroids» в программе QGIS были получены центроиды объектов в shape-файле зданий и Дворцовой площади, которые использовались при отображении архитектурных достопримечательностей в приложении. Для нахождения пересечения построенных маршрутов с архитектурными достопримечательностями был создан еще один файл – буферные зоны шириной 20 метров вокруг объектов в shape-файле зданий. Такая ширина была выбрана исходя из средней ширины городских улиц.

Для отображения фонтанов и памятников вдоль построенного маршрута в приложении был использованы shape-файлы этих объектов, полученные ранее. Для них были созданы буферные зоны шириной 50 метров для пересечения их с маршрутами; буферные зоны для памятников и фонтанов в Летнем саду строились шириной не 50, а 15 метров из-за особенностей сада. Отличие составляли озелененные территории и мосты, для которых буферные зоны не строились, центроиды получались также инструментом «Polygon centroids» в программе QGIS. Для кофеен, которые также отображаются на карте в приложении, были созданы буферные зоны шириной 30 метров. Далее по тексту совокупность точечных объектов в виде кофеен, памятников, фонтанов, достопримечательностей, центроидов мостов и озелененных территорий объединяется в понятие «точечные объекты».

Таким образом, были подготовлены shape-файлы для отображения шести видов объектов вдоль какого-либо пешеходного маршрута и различные данные для построений трех видов маршрутов, полученные из «Яндекс.Карты», а также заимствованы необходимые материалы предыдущих исследований.

2.2. Автоматизированное создание «карты» шумового загрязнения тестовой территории.

Для получения данных об уровне шума в Санкт-Петербурге был применен алгоритм создания «карты» шумового загрязнения, разработанный компанией Urbica. Их разработка является улучшенной версией алгоритма аппроксимации уровня шума, объясненного Лукасом Мартинели (http://lukasmartinelli. ch/gis/2016/04/03/openstreetmap-noise-pollution-map.HTML). Основным усовершенствованием алгоритма являлось рассмотрение изменений распределения шума в зависимости от плотности зданий (Mostafa Refat, I., 2014). Помимо плотности зданий, алгоритм Мартинели учитывает базовый уровень шума в зависимости от типа источника шума. Алгоритм создает полигоны вокруг каждого источника – буферные зоны с тремя уровнями шума: 45, 55 и 65 децибел (дб) (табл.1). Самые шумные объекты получают более высокое значение буферной зоны, а менее шумные объекты – более низкое.

Таблица 1

Уровень шума для каждой зоны

Зона	Уровень шума, дб
L1	≥ 65
L2	55 – 64,9
L3	45 – 54,9

Автор также приводит список объектов, которые являются источником шума:

1. шоссе, магистрали, основные и второстепенные дороги являются шумными;
2. железные дороги;
3. торговые и промышленные зоны;
4. любые магазины и рестораны;
5. места развлечения людей;
6. места временного пребывания людей (отели, гостиницы, хостелы);
7. спортивные учреждения;
8. туристические здания.

Из них в качестве источников шума на территории проекта располагаются только некоторые объекты. Каждый из них имел определенные тэг в атрибутивных таблицах этих

объектов (табл.2). Набор тэгов и их значений для источников, а также уровень их шума был получен из ресурса (https://github.com/urbica/noisemap).

Таблица 2

Ширина зон уровня шума в метрах

Тэг / Зона,дб	L1	L2	L3
Дорожная сеть			
highway=motorway	60	220	500
highway=trunk	50	190	400
highway=primary	35	160	300
highway=secondary		80	125
highway=tertiary		35	65
Промышленные и торговые зоны			
landuse=retail		70	180
Магазины и рестораны			
shop=[any]		30	65
amenity=[bar,bbq,café,..]		35	75
Места развлечения людей			
amenity=[cinema,casino,nightclub,..]	40	70	150
Туристические объекты			
leisure=[beach,resort,zoo,..]	35	55	75

Создание «карты» шумов началось с извлечения данных OSM с использованием сервиса overpass turbo (http://overpass-turbo.eu/). Перед извлечением каких-либо данных из этого источника составлялся запрос в формате .txt. Результат любого запроса возвращался формате .geojson. Сначала был составлен запрос (рис.7) для выгрузки источников шума - данные о дорогах, зданиях и объектов, локализованных в точке (например, туристические объекты) с определенными тэгами из табл.2. В дальнейшем, результат запроса был необходим для его использования в программных кодах при составлении «карты». Полученный файл с данными содержал точечные, линейные и полигональные объекты – возможные источники шума (рис.8).

```
  // query part for: “highway=*”
  node["highway"]({{bbox}});
  way["highway"]({{bbox}});
  relation["highway"]({{bbox}});
);
// print results
out body;
>;
out skel qt;
// gather results
(
  // query part for: "building=*"
  node["tourism"="viewpoint"]({{bbox}});
  way["tourism"="viewpoint"]({{bbox}});
  relation["tourism"="viewpoint"]({{bbox}});
  node["historic=*"]({{bbox}});
  way["historic=*"]({{bbox}});
  relation["historic=*"]({{bbox}});
  node["tourism"="gallery"]({{bbox}});
  way["tourism"="gallery"]({{bbox}});
  relation["tourism"="gallery"]({{bbox}});
  node["tourism"="artwork"]({{bbox}});
  way["tourism"="artwork"]({{bbox}});
  relation["building:architecture=*"]({{bbox}});
  node["building:architecture=*"]({{bbox}});
  way["building:architecture=*"]({{bbox}});
  relation["building:architecture=*"]({{bbox}});
  node["artwork_type=*"]({{bbox}});
  way["artwork_type=*"]({{bbox}});
  relation["artwork_type"="*"]({{bbox}});
  node["amenity"="arts_centre"]({{bbox}});
```

Рис.7. Фрагмент запроса

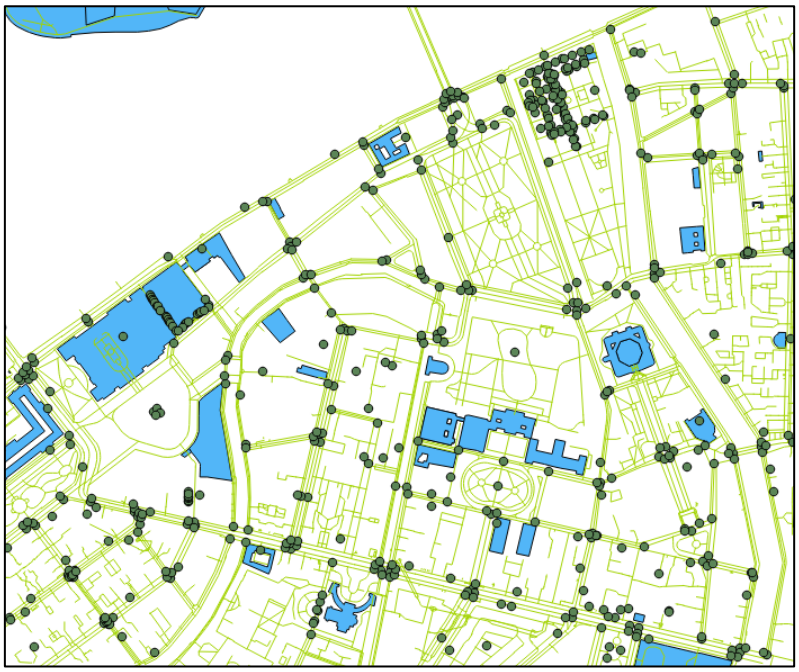


Рис.8. Источники шумового загрязнения

Затем был составлен запрос и выполнена выгрузка геометрии всех зданий внутри территории для учета влияния плотности застройки на распространение шумового загрязнения (рис.9).

Рис.9. Результат запроса выгрузки геометрии зданий.

После запуска программного кода overpass_data_processing.py был получен файл – истинные источники шума, выбранные из ранее выгруженных источников с определенным тэгом и значением уровня шума.

Для определения плотности застройки тестовой территории был выполнен программный код make_density_grid.py. В результате, была получена сетка из полигонов размером 500x500 метров, где каждому ее элементу было присвоено значение плотности застройки (в десятичных долях). На рисунке 10 квадратом красного цвета обозначен один из элементов полигональной сетки и показана плотность застройки территории, ограниченной выделенной областью. Красная граница на рисунке – территория пилот-проекта.

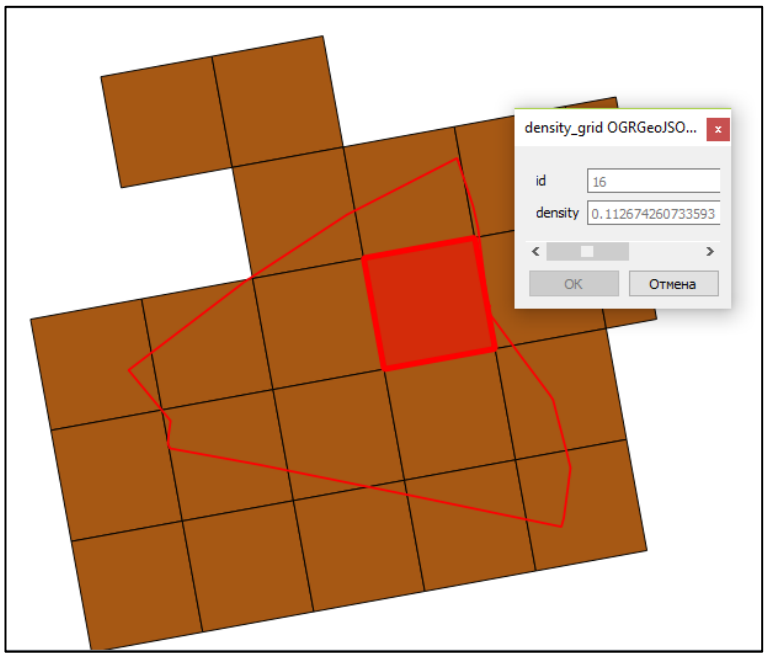


Рис.10. Создание полигональной сетки плотности застройки.

После подготовки всех данных для создания «карты» был запущен алгоритм noise_mapping.py, который создал полигональные объекты с тремя зонами распространения шумового загрязнения (от 45 до 65 дб) (рис.11).

Рис.11. «Карта» шумового загрязнения в дб

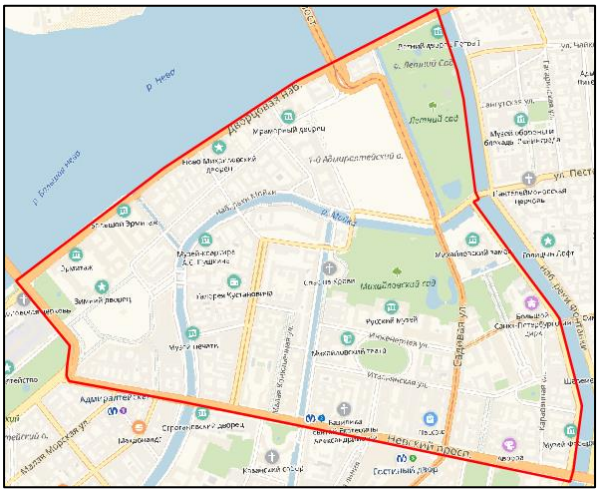


Рис.12. Территория пило-проекта

При сравнении построенной «карты» шумов (рис.11) и карты территории пилот проекта, выделенной красным на рис.12 видно, что вокруг автомобильных дорог уровень шума уменьшается при удалении от них; внутри парков уровень шума минимальный. Вокруг источников шума, показанных на рис.8 уровень шума принимает наибольшие значения.

На «карте» шумов полигональный объект со значением шума 45 дб геометрически включал в себя объекты с уровнем шума 55 и 65 дб, в то время как объект с уровнем 55 дб геометрически включал объект с уровнем шума 65 дб. Необходимо было устранить наложения между тремя объектами. Для этого был созданы три полигональных слоя со значением шумов 45, 55 и 65 дб. Затем, в результате оверлея, был получен промежуточный векторный слой со значениями 45 и 55 дб; аналогично был получен промежуточный shape-файл для геометрической разности объектов со значениями 55 и 65 дб. Первый векторный слой – распространение шумового загрязнения со значением 45 дб, второй – со значением 55 дб. Затем эти два shape-файла вместе с файлом со значением уровня шума 65 дб были объединены в один векторный слой при помощи инструмента «Merge vector layers» в программном продукте QGIS. Именно последний полученный shape-файл был использован для подготовки данных к построению «самых тихих маршрутов».

2.3. Создание растровых данных весовых значений.

Для построения трех видов пешеходных маршрутов требовалось присвоить три вида весовых значений пешеходным дорогам. Это было необходимо для построения «самого интересного», «тихого» и «экологичного» пешеходных маршрутов. Сначала пешеходные дороги, взятые из материалов курсовой работы, были разделены на отрезки, максимальная длина которого составляла 20 метров. Для разделения был применен инструмент «v.split.lenght» в программном продукте QGIS. Из-за дорожных перекрестков не все отрезки пешеходных дорог имели длину 20 метров (рис.13).

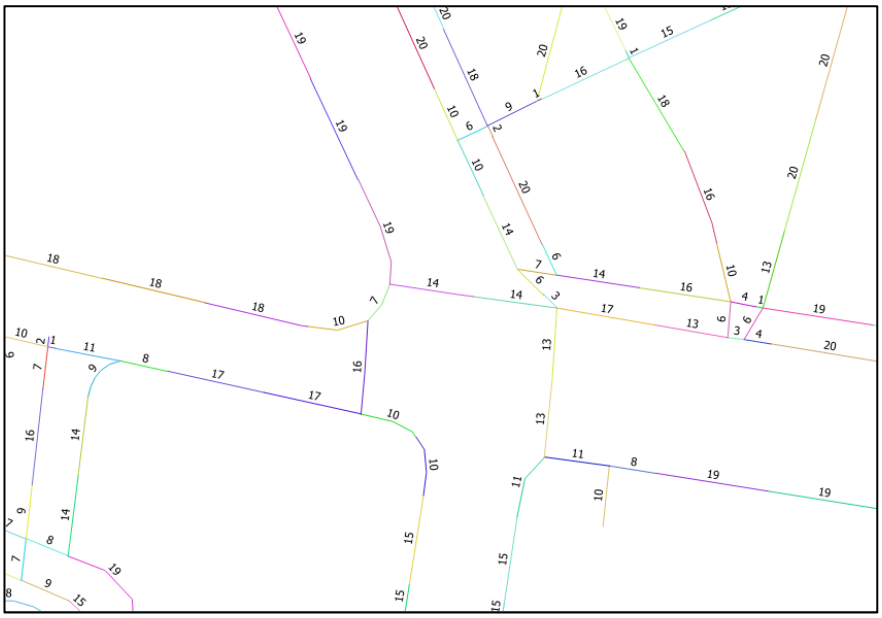


Рис.13. Фрагмент изображения пешеходного дорожного графа в программе QGIS. Цифрами обозначены длины каждого ребра графа.

Что касается весовых значений пешеходных дорог для построения «самого интересного» маршрута, то они зависели от трех параметров – оценок достопримечательностей, набережных и озелененных территорий. Оценка отдельного объекта говорила о его значимости среди других объектов. Следовательно, «притяжение» объекта прямо пропорционально его оценке. Чем выше была оценка, тем больше должен был отклоняться маршрут от кратчайшего с целью построения «самого интересного» маршрута через «притягательный» объект. Были выбраны параметры «Оценка объекта – Максимальное расстояние в метрах», где первый – оценка, взятая из источника «Яндекс.Карты»; второй – максимальное расстояние, при котором его объект будет вносить свой вклад в результирующий вес дороги (табл.3).

Таблица 3.

Параметры «притяжения» объектов

Оценка объекта	Максимальное расстояние в метрах
5	400
4	330
3	260
2	190
1	120
Менее 1	70

Результирующие веса дорогам для «самого интересного» маршрута зависели от трех видов объектов (достопримечательностей, набережных и озелененных территорий).

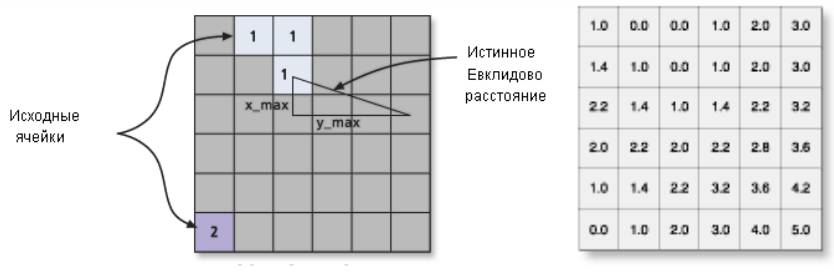
Необходимо было создать общий массив данных, где учитывались бы влияния всех объектов.

Для решения этой задачи применялся расчет по евклидовому расстоянию. Евклидово расстояние — расстояние между двумя точками евклидова пространства, вычисляемое по теореме Пифагора. Для точек с пространственными координатами: $x = (x_1, x_2, \ldots, x_n)$ и $y = (y_1, y_2, \ldots, y_n)$ евклидово расстояние вычисляется по формуле:

$$d = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2} \quad (1)$$

Одно из применений евклидового расстояния – решение задач путевого анализа растровых изображений, когда каждая ячейка несет значение расстояния до источника. Источниками могут быть как точечные, линейные, полигональные объекты, так и растровые представления объектов: памятников, кафе, набережных, парков и так далее.

Расчет евклидового расстояния на растре реализован в программном продукте ArcMap. В случае, когда на вход подается источник в векторном виде, он автоматически преобразуется в растр. В растре содержатся ячейки источника, образованные при растеризации; остальные ячейки имеют значение NoData. Евклидово расстояние на растре вычисляется от центра ячейки источника до центра каждой ячейки. Алгоритм работает следующим образом (https://desktop.arcgis.com/ru/arcmap/latest/tools/spatial-analyst-toolbox/euclidean-distance.htm) (Рис.14а): расстояние до каждой ячейки рассчитывается как гипотенуза треугольника, катетами которого являются x_{max} и y_{max}. Такое вычисление является истинным евклидовым расстоянием, а не расстоянием между ячейками. По завершении работы алгоритма создается растр, значения в каждой ячейке которого есть расстояния от центра каждой ячейки до центра ближайшего источника (Рис.14б). Расстояния измеряются в единицах проекции источника. Вычисленные расстояния не учитывают наличие дорожной сети.



а) б)

Рис. 14. Определение истинного евклидового расстояния на растре. а) – исходный растр, б) – выходной растр. Источник: https://desktop.arcgis.com/ru/arcmap/10.3/tools/spatial-analyst-toolbox/understanding-euclidean-distance-analysis.htm - справочник по инструментам ArcGIS.

Для решения задачи создания общего массива данных весовых значений рассматривалось два подхода:

1. построить несколько буферных зон вокруг каждого объекта; значение внутри каждой зоны зависело бы от оценки объекта и расстояния до него. По мере удаления от объекта значения бы в каждой буферной зоне уменьшались бы. Чем выше оценка объекта, тем больше было бы число буферных зон и их распространение (Табл. 2).

Данный подход не был реализован, поскольку разница значений между смежными буферными зонами достаточно велика, что привело бы к существенной разнице между весовыми значениями близко расположенных дорог;

2. создать растровое изображение по методу «евклидовое расстояние» которое вычислялось с использованием общей формулы:

$$\text{CellValue} = \begin{cases} \text{A}, \text{при ED} = 0 \\ \frac{\text{A}}{\text{X}}, \text{при } 0 < \text{ED} \leq \text{Border} \\ 0, \text{при ED} > \text{Border}, \end{cases} \quad (2)$$

где $\text{A} - \text{значение веса объекта}$, $\text{cellSize} -$ размер ячейки выходного растра, $\text{ED} -$ $\text{евклидовое расстояние}$, $\text{Border} = (76.1 \cdot \text{A} + 19.56)$ – максимальное расстояние от объекта с весом A, который вносит свой вклад в результирующий вес дороги (см.табл.3).

Что касается этого подхода, то в зависимости от того, какое значение могло бы быть в ячейке изображения, рассматривалось два варианта:

2.1 значение обратно пропорционально евклидовому расстоянию (при X = ED);

2.2 значение в ячейке растра определяется исходя из формулы (при $\text{X} = \text{ED}/\text{cellSize}$ - евклидово расстояние, выраженное в количестве ячеек размером cellSize;

Для реализации обоих подходов был написан программный код, позволяющий строить растровое изображение, где в качестве входных данных использовались точечные (достопримечательности), линейные (набережные с заданными оценками) и полигональные (озелененные территории) объекты. Написанный алгоритм был внедрен в программный продукт ArcMap, поскольку алгоритм можно было запускать как инструмент, задавая входные параметры. После выполнения кода, полученное растровое изображение было отображено в окне программы ArcMap для оценки результатов.

Алгоритм работы программного кода был следующий:

– импорт модуля «arcpy» для работы алгоритма из-под ArcMap;

– задание параметров (исходный shape-файл с линейными, точечными или полигональными объектами, название поля со значениями оценок в этом файле, размер ячейки выходного растра, путь для сохранения выходного растра). Эти же параметры указывались в диалоговом окне нового инструмента в программе ArcMap (рис.15). Размер ячейки был выбран 5 метров для повышения точности результатов;

– создание растра со значениями в каждой ячейки равными нулю;

– на каждой итерации проводилось создание растра для каждого по отдельности объекта входного shape-файла; значение в каждой ячейке растра высчитывалось по одной из формул для случаев 2.1 и 2.2.

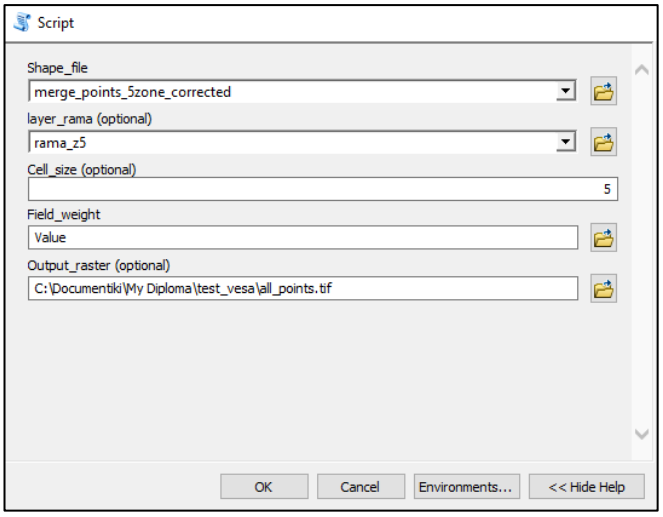


Рис. 15. Окно инструмента в программе ArcMap.

В конце первой итерации цикла создавался новый растр, который получался в результате сложения значений соответствующих ячеек «нулевого» и созданного для отдельного объекта растров. На следующих итерациях вновь создавался растр для следующего объекта из исходного shape-файла и складывался с растром, полученным в предыдущей итерации.

– создание результирующего растрового изображения, значение в каждой ячейки которого было суммарное влияние всех объектов (достопримечательностей). Достопримечательности содержались в входном shape-файле.

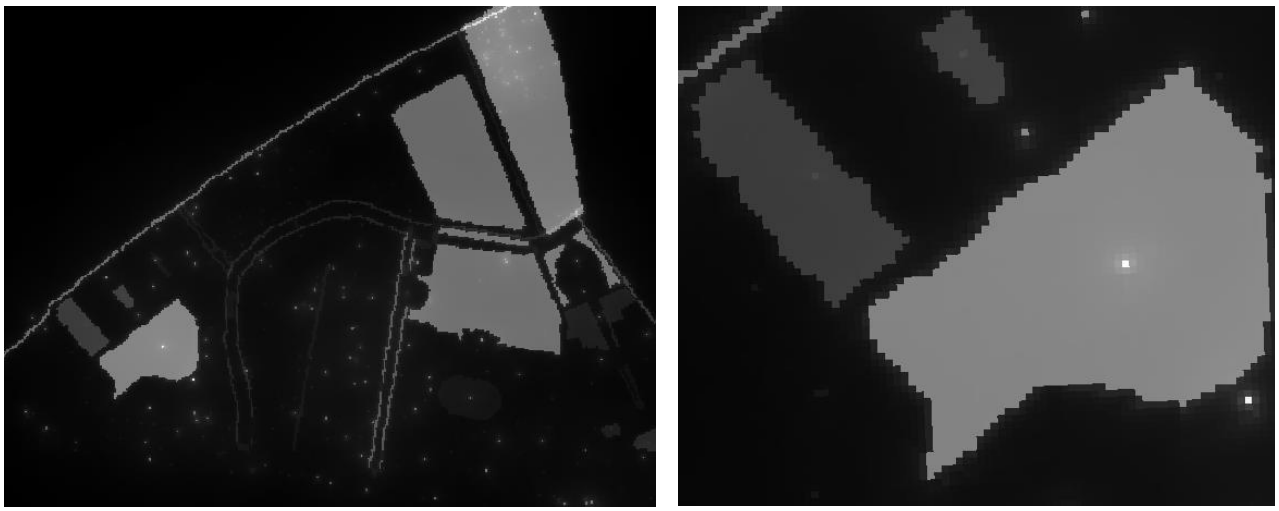
Нужно было учитывать тот факт, что выходное растровое изображение после работы алгоритма всегда будет иметь границы, как у shape-файлов, следовательно, результирующий растр получился бы меньше тестовой территории. Во избежание этого, перед запуском алгоритма были добавлены объекты в shape-файлы за пределами территории проекта с нулевыми весами, для того чтобы результирующий растр покрывал всю территорию проекта. Нулевые веса этих объектов не вносили искажения в результирующие растровые изображения.

В точечном shape-файле достопримечательностей содержались объекты за пределами территории, а именно по другую сторону от Невского проспекта. Они также были важны для правильного присвоения значений ячейкам на границе территории вдоль Невского проспекта, так как пользователь при движении по маршруту вдоль проспекта будет видеть и достопримечательности, находящиеся на другой стороне дороги.

Входной shape-файл не мог содержать точечные, линейные и полигональные объекты вместе. Поэтому были созданы три растровых изображения отдельно для точечных, линейных и полигональных объектов. Каждая ячейка созданных растров имела значение суммарного влияния объектов в радиусе до 400 метров от этой ячейки (табл.3). После получения трех растровых изображений был применен инструмент «Raster Calculator» в программном продукте ArcMap, который позволил поячеисто сложить значения в трех растровых изображениях и создать суммарный растр. При сложении трех растров граница результирующего изображения совпадала с границей растра с наименьшей областью охвата. Следовательно, это факт необходимо было учитывать при добавлении объектов с нулевыми значениями в shape-файлы достопримечательностей, набережных и озелененных территорий (см. выше).

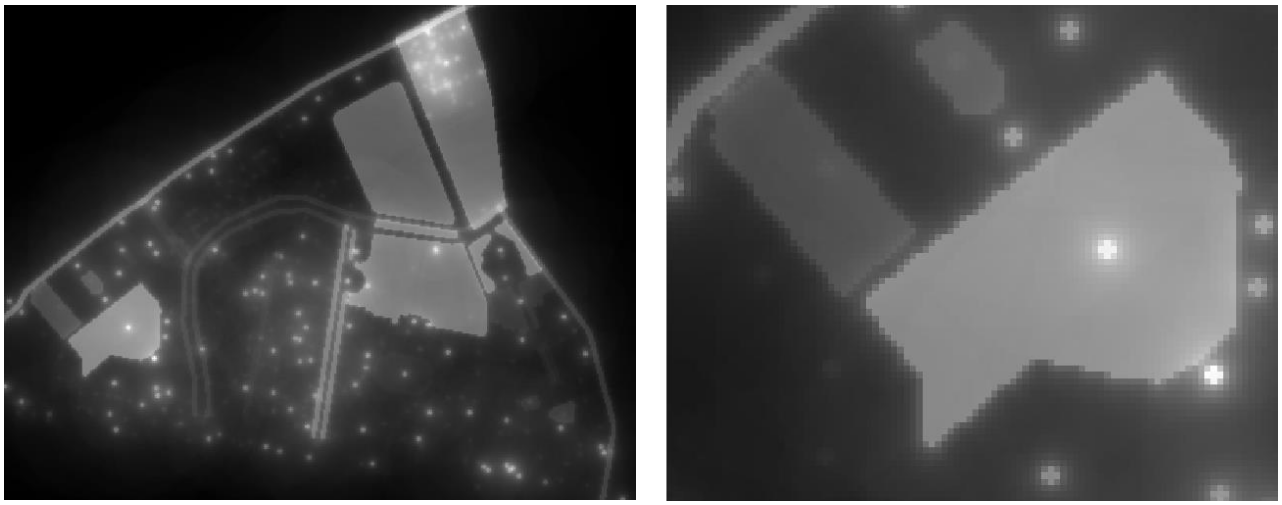
Если сравнить рис.16 и рис.17, то можно заметить, что разница значений в соседних ячейках на первом рисунке больше, чем на втором. На втором рисунке значения в пикселях изменяются плавно, что нельзя сказать о первом изображении. Второе изображение выглядит более естественно, поскольку влияние достопримечательности по мере удаления от нее не может резко уменьшаться, как происходит на первом рисунке. Если бы был выбран первый вариант создания растра, то искомый маршрут незначительно бы отклонялся от кратчайшего, поскольку максимальные значения в ячейках наблюдалось бы только в месте расположения самих объектов. Другими словами, «притяжение» достопримечательностей распространялось бы только внутри самих объектов.

Был выбран способ под номером 2.2 (см. выше) для создания результирующего растрового изображения (далее растр весов) (рис.18).



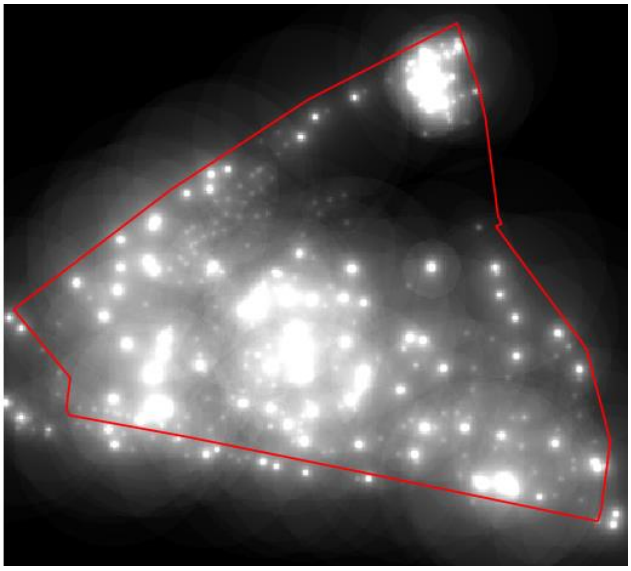
а) б)

Рис.16. Растровое изображение, построенное по принципу из п. 2.1. а) – полный экстент территории, б) – увеличенное изображение растра (Дворцовая площадь). Чем ярче пиксел, тем больше значение в ячейке.

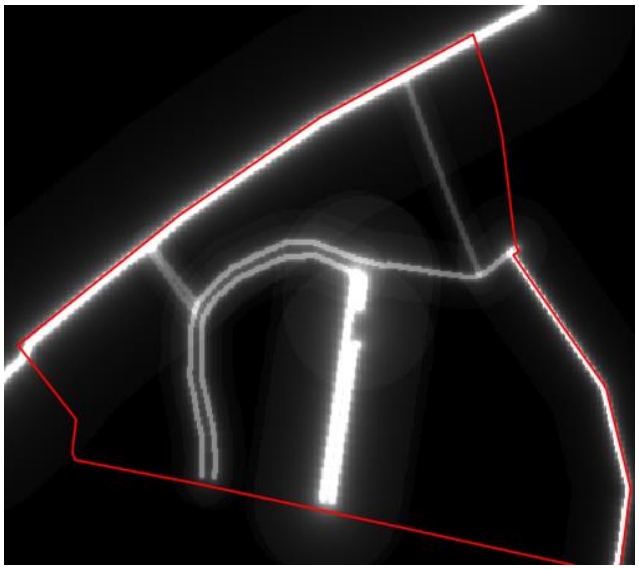


а) б)

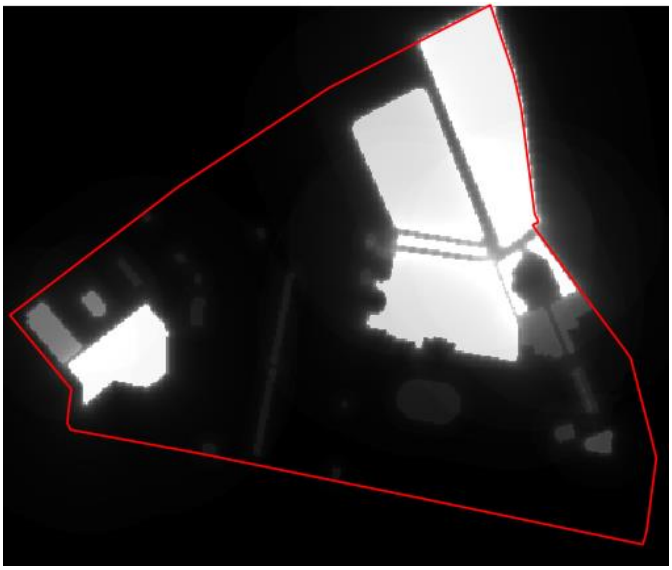
Рис.17. Растровое изображение, построенное по принципу из п. 2.2. а) – полный экстент территории, б) – увеличенное изображение растра (Дворцовая площадь). Чем ярче пиксел, тем больше значение в ячейке.



а)



б)



в)

Рис.18. Итоговые построенные растровые изображения. а) – для точечных достопримечательностей, б) – для линейных объектов (набережных), в) - для площадных объектов (площадей, парков и так далее). Красным полигоном обозначена территория проекта.

2.4. Присвоение весовых значений пешеходным дорогам.

Поскольку пешеходные дороги были представлены в виде shape-файла, необходимо было растр весов перевести в векторный формат для дальнейшего присвоения пешеходным дорогам весовых значений. В программном продукте QGIS инструментом «Создание полигонов (растр в вектор)» была произведена автоматическая векторизация растра весов. В

результате, были получены два полигональных shape-файла, где каждый объект – полигон со значением, взятого из ячейки исходного растра весов.

Следует напомнить, что все пешеходные дороги были разделены на отрезки, максимальная длина каждого составляла 20 метров. Для присвоения значений векторизованного растра всем частям пешеходных дорог был использован инструмент «Spatial Join» в программном продукте ArcMap (рис.19). Если дорога пересекала несколько полигонов одного из shape-файлов, то ей присваивалось среднее значение пересекающих ее полигонов. После окончания работы инструмента в программе QGIS при помощи «калькулятора полей» весовые значения были возведены в степень -1. Эта операция требовалась для работы выбранных алгоритмов при построении пешеходных маршрутов.

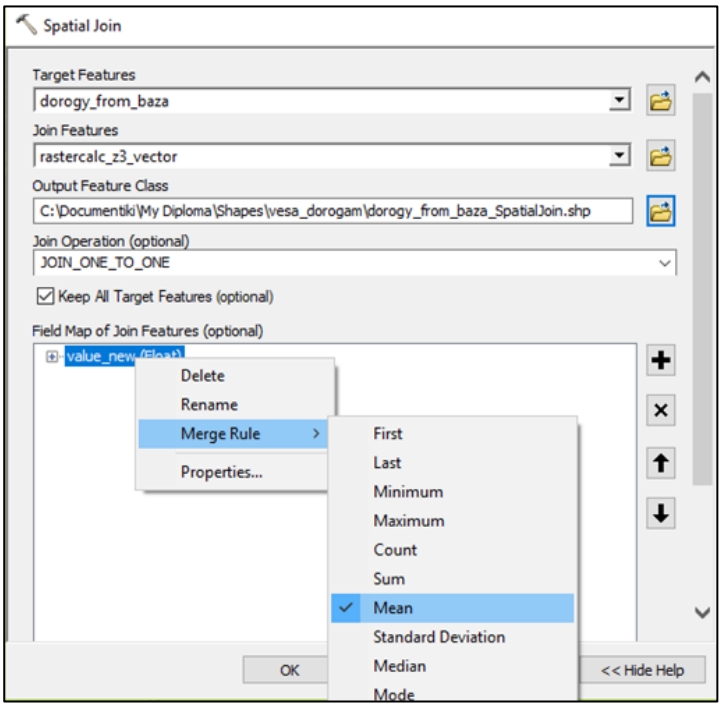


Рис.19. Инструмент «Spatial Join»

Для построения «самых тихих» маршрутов требовалось внести в атрибутивную таблицу пешеходных дорог значения шумового загрязнения для каждого участка дорог. Общая картина загрязнения была получена при создании «карты» шумового загрязнения (см. раздел 2.2). С помощью того же вышеупомянутого инструмента с аналогичными параметрами были получены данные по загрязнению для каждого участка пешеходной дорожной сети.

Для того чтобы построить «самые экологичные» маршруты, необходимо было добавить информацию о пересечении пешеходных дорог с озелененными территориями. Для этого был применен инструмент «Выборка по местоположению» в программном продукте ArcMap. Производился поиск пересечения shape-файла озелененных территорий с файлом

дорожной сети; точность поиска выбиралась 25 метров для включения в выборку дорог, проходящих около озелененных территорий, поскольку рядом расположенные дороги тоже учитывались при построении «самых экологичных» маршрутов.

В результате, в атрибутивную таблицу shape-файла дорог были добавлены три поля – значения шумового загрязнения, влияния достопримечательностей и расположения дорог относительно озелененных территорий. Все три вида значений отображены на рис.20, 21 и 22.

Рис.20. Значения шумового загрязнения для каждого участка пешеходной дороги, дб

Рис.21. Значения весов суммарного влияния всех достопримечательностей для каждого участка пешеходной дороги, оценка

Рис.22. Значения, определяющие расположение дорог относительно озелененных территорий, усл.ед. 0 – дорога, проходящая внутри озелененной территории или на расстоянии 25 метров от нее. 1 – не проходит.

2.5. Выбор алгоритмов для построения пешеходных маршрутов.

В целях построения пешеходных маршрутов требовалось представление дорожной сети в виде неориентированного графа. Неориентированный граф - структура пространственных данных, представляющая собой ребра и вершины, где последние соединяют ребра между собой; при этом ни одному ребру не присвоено направление (Харари Ф., 2003). Любой маршрут, который можно построить в созданном приложении, иначе можно назвать простым путем, представляющим собой последовательность рёбер, где конец одного ребра является началом другого; при этом последовательность ребер не проходит дважды через одну вершину (Кузнецов О. П. и др., 1980). С другой стороны, использованный граф в рамках проекта представлялся как взвешенный, где каждое ребро имело свой вес. В качестве весов принимались длина маршрута (при построении кратчайшего пути), значения «притяжения» достопримечательностей, взятые в степени -1 (во время построения «самого интересного» маршрута), параметры шумового загрязнения (для «самого тихого») и значения, определяющие расположение дорог относительно озелененных территорий (для «самого экологичного»). Пересчет значений весов для достопримечательностей был необходим для того, чтобы ребра графа пешеходных дорог, проходящие через наименее

популярные объекты, имели высокое значение. Выполнение этого условия обеспечило правильную работу алгоритмов при построении «самого интересного» маршрута. Любой пешеходный маршрут в приложении представляет собой путь с наименьшей стоимостью, поскольку он строится с минимально возможным суммарным значением вышеописанных весов.

Существует два наиболее распространенных способа представления графов – списки смежных вершин и матрица смежности. В проекте для реализаций примененных алгоритмов использовалось представление графа именно матрицей смежности (рис.23). В случае взвешенного графа в матрице смежности вместо значений 1, обозначающих вес ребер, использовалось их весовое значение.

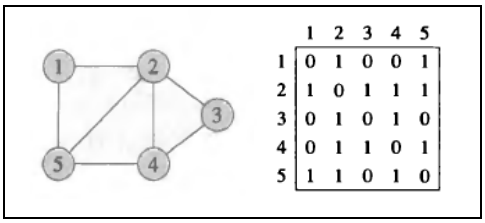


Рис.23. Представления неориентированного графа в виде матрицы смежности. Цифрами обозначены вершины графа. Источник: Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ. Под ред. И. В. Красикова. М.: Вильямс, 2005, 1296 с

Сегодня построение графов можно реализовать с помощью баз геопространственных данных, посредством которых реализуются любые известные алгоритмы для работы с графами. С целью построения различных пешеходных маршрутов в приложении необходимо было проанализировать возможности существующих алгоритмов и выбрать наиболее подходящие. Известные алгоритмы для построения простого пути между точками, которые потенциально могли бы подойти для задач построения пешеходных маршрутов:

– жадный. Путь прокладывается через те вершины, которые на данной итерации расположены ближе всего (по прямой) до конечной точки. Не учитываются веса ребер. Не подходит для задач сетевого анализа (Кормен, Т. и др., 2005);

– в ширину. Находит все возможные варианты прохождения по ребрам графа. Применим, когда не заданы весовые значения каждого ребра сети, поэтому алгоритм не может быть применен для построения нужных маршрутов (Левитин А. В., 2006);

– нахождения кратчайшего пути из одной вершины в ориентированных ациклических графах. Не применим для неориентированных графов, каким являлся граф пешеходной дорожной сети проекта (Кормен, Т. и др., 2005);

– A-Star. Алгоритм использует подлинное расстояние от начала пути по ребрам и расстояние до цели по прямой (жадный алгоритм). Не подходит для задачи

построения маршрутов, где в качестве веса ребра указана неметровая единица измерения (оценка, значение шума и т.д) (Dechter, R. et al., 1985);

– Беллмана-Форда - алгоритм для нахождения кратчайших расстояний между источником (начальной точкой пути) и всем вершинами графа. Время работы алгоритма больше, чем алгоритма Дейкстры (Кормен, Т. и др., 2005);

– Флойда-Уоршелла - алгоритм для нахождения кратчайших расстояний между всеми вершинами графа. Время работы алгоритма больше, чем алгоритма Беллмана-Форда (Кормен, Т. и др., 2005);

В результате анализа, ни один из описанных алгоритмов не мог быть применим для построения пешеходных маршрутов по причинам их продолжительного времени работы или из-за отсутствия возможности взаимодействия со взвешенным неориентированным графом. Был выбран алгоритм Дейкстры, так как обеспечивает выполнение всех вышеописанных условий. Алгоритм был разработан Эдсгером Дейкстра (Edsger Wybe Dijkstra) в 1959 году. Принцип работы алгоритма (рис.24):

– все вершины графа разделяются на два набора: один набор содержит обработанные вершины, а другой еще не обработанные. На начальном этапе все вершины не обрабатываются, и стоимость всех вершин равна бесконечности, кроме стоимости исходной вершины - она равна 0. Эта вершина является источником;

– на каждой итерации алгоритм берет вершину с минимальными затратами перехода к ней и ослабляет каждое ребро с учетом стоимости, получаемой из нее. Вершина, от которой исходят ослабляемые ребра, переходит в набор обработанных.

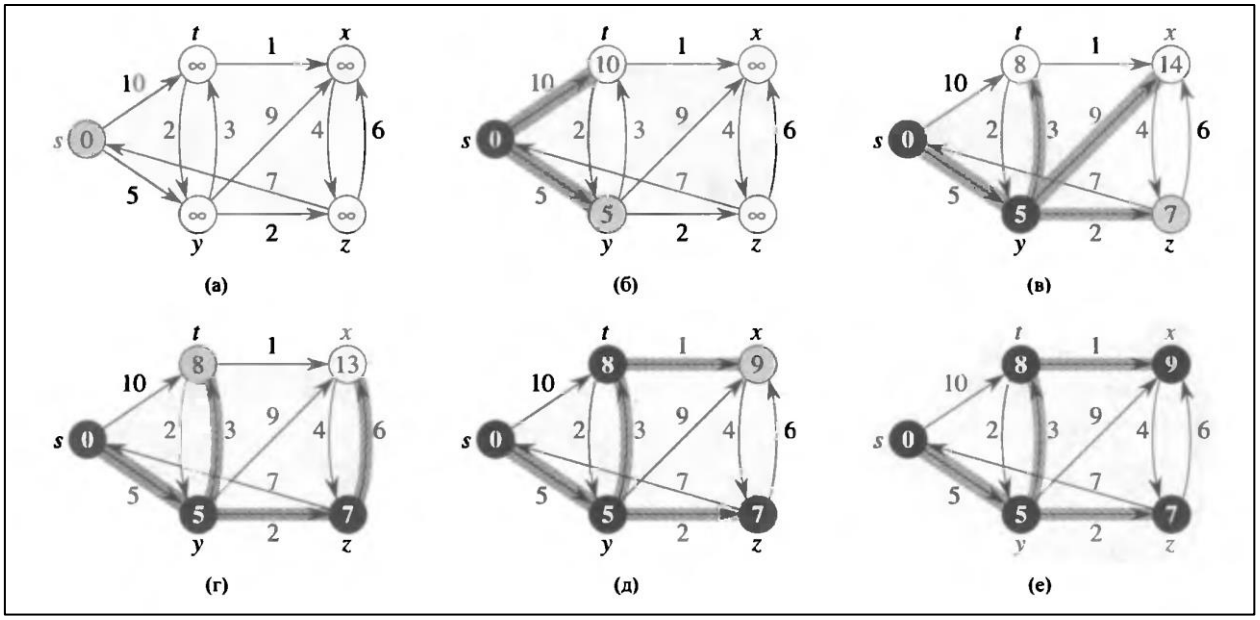


Рис.24. Алгоритм Дейкстры. а) - изначально источник имел значение 0 – минимальное на данный момент. Начало работы алгоритма. б) - на первой итерации

алгоритм ослабляет все ребра, начало которых совпадает с источником. Вершина-источник переходит в набор обработанных вершин. в) - далее действия алгоритма перемещаются в точку, стоимость перехода к которой минимальна и происходит ослабление всех ребер, исходящих от нее. Точка y переходит в набор обработанных вершин. г) - алгоритм снова ослабляет ребра, исходящие из точки z и переприсваивает стоимость перехода к точке x, поскольку на данной итерации она становится минимальной. Точка z переходит в набор обработанных вершин. д), е) - переприсваивание стоимостей ведется до тех пор, пока стоимость перехода к любой вершине на станет минимальной. Источник: Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ. Под ред. И. В. Красикова. М.: Вильямс, 2005, 1296 с.

В случае построения любого вида пешеходного маршрута в приложении (а именно, маршрута с минимальной стоимостью) от точки А до точки Б алгоритм запоминал родителя каждой вершины, т.е. какое именно ребро дало минимальную стоимость при переходе к этой вершине. Алгоритм запоминал движение по ребрам, тем самым путь между заданными крайними точками, при этом сумма стоимостей ребер-родителей давало минимальную стоимость маршрута. Одно из условий работы алгоритма – работа с неотрицательными весами, какими и являлись три вида весовых значений дорожной сети проекта (Кормен, Т. и др., 2005). В приложении для построения пешеходных маршрутов рассмотренный алгоритм Дейкстры применяется только в двух случаях: когда пользователь не указывает максимально возможное время прохождения маршрута и при построении «самого быстрого» маршрута.

Для решения задачи построения трех оставшихся видов маршрутов, когда пользователь указывает максимальноевремя для прогулки, применяется алгоритм k - кратчайших маршрутов, который также работает со взвешенным неориентированным графом. Для понимания его работы был рассмотрен метод работы этого алгоритма, описанный в статье. [Yen, Jin Y. (1971). “Finding the K Shortest Loopless Paths in a Network”. In: Management Science 17.11, pp. 712–716] Все кратчайшие маршруты этого алгоритма находятся по алгоритму Дейкстры. При этом следует понимать, что «кратчайший» означает не только самый короткий по длине (при построении самого короткого маршрута); в случаях построения трех видов маршрутов («самый интересный», «самый тихий», «самый экологичный») имеется в виду путь с наименьшей стоимостью.

Основные этапы работы алгоритма по методу Йена:

– создание контейнера (R) для хранения k-кратчайших путей;

– нахождение кратчайшего пути и добавление его в контейнер;

– оставшиеся k-размер(R) маршруты должны отличаться от маршрутов в контейнере. Для выполнения этого условия итеративно удаляется каждое ребро из каждого маршрута в контейнере, и строится новый кратчайший маршрут без учета удаленного ребра;

– после итеративного удаления каждого ребра формируется промежуточный список маршрутов, из которых выбирается самый кратчайший и заносится в контейнер;

– цикл повторяется, пока количество маршрутов в контейнере не станет равно k.

При работе алгоритма k-кратчайших путей осуществляется поиск маршрута, время на прохождение которого не превышает заданное. При условии, когда заданное время больше, чем время на прохождения маршрута по алгоритму Дейкстры (это маршрут с наименьшей стоимостью, применяется при отсутствии указанного времени), маршрут не перестраивается. Это связано с тем, что при работе алгоритма k-кратчайших путей в контейнере хранится первый путь, идентичный маршруту, строящемуся без задания времени (по алгоритму Дейкстры, маршрут с наименьшей стоимостью).

В случае, когда заданное время меньше, был организован цикл из шести итераций, где на каждой из них алгоритм строит 10*i маршрутов (i – номер итерации). Последующая итерация выполняется только тогда, когда в предыдущей итерации не был найден маршрут, время на прохождение которого меньше или равно заданному пользователем. При нахождении пешеходного маршрута, укладывающегося в рамки заданного времени, но не с наименьшей стоимостью (повторим, что с наименьшей стоимостью маршрут – первый из k-маршрутов или строящийся по алгоритму Дейкстры). Если алгоритм не находит маршрута, удовлетворяющего заданному времени, приложение выводит соответствующее сообщение.

Таким образом, в зависимости от указания или отсутствия желаемого времени для прогулки были выбраны два алгоритма для построения всевозможных пешеходных маршрутов: кратчайшего и трех тематических: Дейкстры и k-кратчайших путей.

2.6. Работа с базой данных.

В связи с тем, что алгоритмы нахождения кратчайшего пути Дейкстры и k-кратчайших путей по методу Йена реализованы в расширении pgRouting программного обеспечения PostGIS, именно эта программа была выбрана для построения маршрутов. Для работы с алгоритмами необходимо было создать базу данных. База данных (БД) была разработана при помощи СУБД PostgreSQL. Для работы с пространственными данными было установлено расширение к СУБД – PostGIS. К геопространственной БД PostGIS было

установлено расширение pgRouting. Работа с БД осуществлялась посредством открытой платформы администрирования pgAdmin 4 (далее pgAdmin).

Как отмечалось ранее, с таблицами созданной БД, можно работать из-под QGIS. Учитывая это, с помощью модуля «Менеджер БД» сначала была создана схема (рис.25), а затем внутри нее и таблица пешеходной дорожной сети, загруженная из shape-файла дорог (рис.26). Аналогично были созданы две схемы для добавления в них таблиц, содержащих отдельно данные для отображения объектов по маршрутам и для пересечения их с линией маршрута. Таблицы формировались из shape-файлов, описание создания которых находится в разделе 1 главы 2.

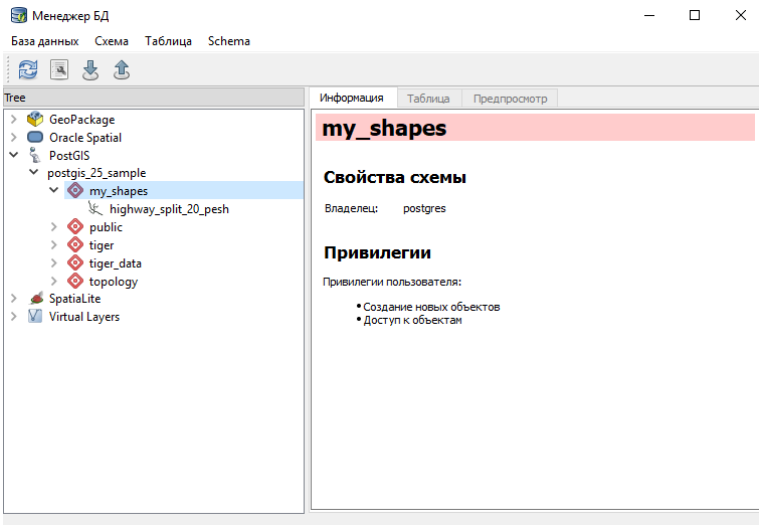


Рис.25. Создание схемы базы данных

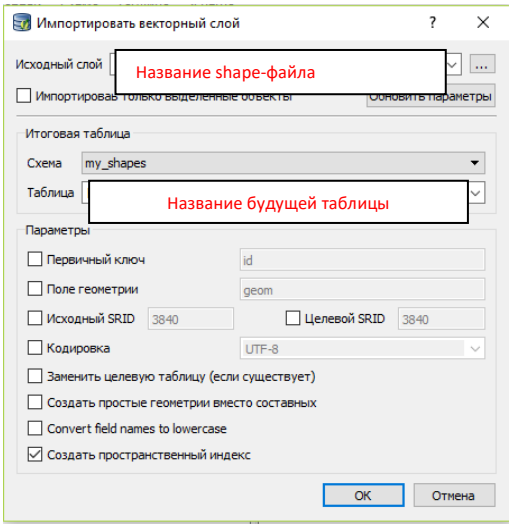


Рис.26. Создание таблицы пешеходных дорог базы данных

Работа с данными в таблицах выбранной базы данных осуществляется с помощью запросов на языке SQL (Beaulieu A., 2005). Перед построением любого пешеходного маршрута, выполняемого с помощью базы данных, необходимо было создать топологию с целью проверки программой на наличие топологических ошибок между элементами дорожной сети. Через pgAdmin была создана топология таблицы дорог с помощью запроса к БД. Как видно из запроса на (рис.27), в таблице дорог были созданы два поля: source и target, в которые записаны идентификаторы соответственно первой и второй вершин каждого ребра графа. После создания топологии ошибки не были найдены, так как были ранее устранены в рамках курсовой работы. Вдобавок, была создана новая таблица – вершины дорожного графа.

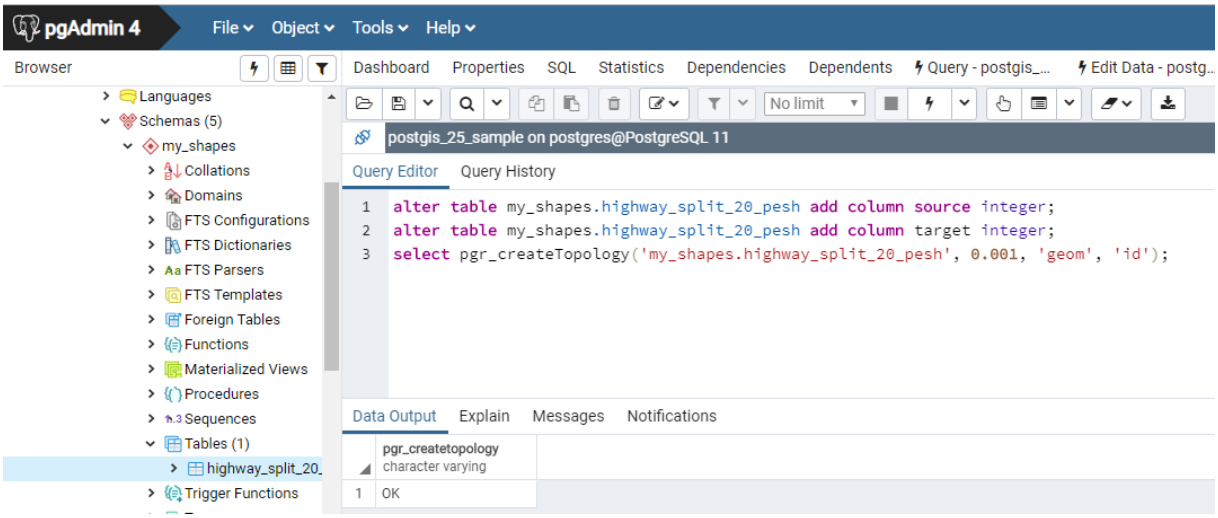


Рис.27. Создание топологии таблицы пешеходных дорог в PgAdmin 4

В зависимости от того, указывает пользователь или нет максимальное время для прогулки, пешеходный маршрут строится по одному из двух алгоритмов (см. раздел 2.5). Для построения каждого пешеходного маршрута по алгоритму Дейкстры составлялся отдельный запрос, общий вид которого представлен на рис.28а. В нем поля «source» и «target» - идентификаторы соответственно первой и второй вершин каждого ребра графа, «cost» - параметр веса: для «самого интересного» маршрута - поле со значениями суммарного притяжения достопримечательностей, «самого тихого» - значения шумового загрязнения, «самого экологичного» - параметр, определяющий положение дороги относительно озелененных территорий, «id_pointA» и «id_pointB» - идентификаторы начальной и конечной точек маршрута.

Запрос к БД для построения маршрута по алгоритму k-кратчайших путей представлен на рис.28б. Параметр «count_marsh» означает количество маршрутов, а «directed:=false» - параметр неориентированного графа, каким является дорожный граф проекта.

```
SELECT * FROM pgr_dijkstra('
        SELECT id AS id,
        source AS source,
        target AS target,
        cost::float8 AS cost
        FROM table_road',
        id_pointA,
        id_pointB,
        false,
        false))
```

а)

```
SELECT * FROM pgr_KSP(
        'SELECT id, source, target,
        cost as cost
        FROM road_table',
        id_pointA,
        id_pointB,
        count_marsh,
        directed:=false)
```

б)

Рис.28. Запрос для построения маршрута. а) - по алгоритму Дейкстры, б) - по алгоритму k-кратчайших путей

Для получение «id_pointA» и «id_pointB» был составлен запрос (рис.29), в котором происходил поиск ближайшего идентификатора к указанной на карте точки. Таблица всевозможных идентификаторов (вершин дрожного графа) была создана в результате создания топологии (см.выше).

```
SELECT id
        FROM table_road_vertices
        ORDER BY ST_Distance(ST_Transform(the_geom,4326),
        'SRID=4326;POINT(pointA pointB)'::geometry) ASC
        LIMIT 1;
```

Рис.29. Запрос для получения идентификатора.

В дополнение к приведенным запросам, необходимы были и другие: для определения длины маршрута (рис.30), нахождения географических координат вершин любого построенного пешеходного маршрута (рис.31) и определения пересечения линии маршрута с буферными зонами кофеен, фонтанов, памятников, архитектурных достопримечательностей, мостами и озелененными территориями – далее они объединяются в понятие «полигональные объекты» (рис.32).

```
SELECT sum(ST_Length(geom))
FROM table_route
```

Рис.30. Запрос для определения длины маршрута

```
SELECT ST_AsText(ST_Transform(the_geom,4326))
FROM table_points
```

Рис.31. Запрос для определения координат точек

```
SELECT *
FROM a, b
WHERE ST_Intersects(a.geom, b.geom))
```

Рис.32. Запрос для нахождения пересечения

Реализация алгоритма отображения объектов вдоль построенного пешеходного маршрута осуществлялась в несколько шагов. Сначала находилось пересечение линии маршрута с «полигональными объектами» с помощью запроса. Этот запрос возвращал идентификаторы «полигональных объектов», встречающихся по маршруту. Далее составлялся второй запрос, в котором объединялись результаты предыдущего запроса и таблица «точечных объектов» (см. раздел 2.1) с целью нахождения последних, которым соответствуют «полигональные объекты». Результат второго запроса состоял из координат и

названий «точечных объектов», отображаемых на карте после построения пешеходных маршрутов.

В результате подготовки исходных данных, полученных из различных источников, их последующего анализа и обработки была получена основа для формирования базы данных. При рассмотрении различных алгоритмов построения пешеходных маршрутов были выбраны два наиболее оптимальных, с применением которых удалось составить различные запросы к содержимому таблиц созданной базы данных. Результаты выполнения запросов являлись одними из главных компонентов для создания приложения по построению пешеходных маршрутов с различными параметрами.

Глава 3. Разработка приложения по построению пешеходных маршрутов.

3.1. Построение клиент-серверной архитектуры приложения.

Созданное приложение основано на клиент-серверной архитектуре. В роли сервера выступает компьютер, а в роли клиента – веб-браузер этого же компьютера. Они взаимодействуют посредством HTTP-протокола. Клиент отправляет серверу запрос на получение ресурсов с помощью URL-адреса. Ресурсы в данном приложении – файлы на компьютере с различными программными кодами, созданными для обеспечения как функциональной части приложения, так и для оформления веб-страницы. В ответ сервер передаёт клиенту запрошенные данные, например, в виде HTML-страницы и набора различных файлов для ее отображения.

Один из основных принципов проектирования REST-архитектуры заключается в том, что веб-сервис должен находиться на сервере. Сервис предоставляет необходимую функциональность клиенту. При отправке клиентом запроса к веб-сервису сервер должен либо отклонить его, либо принять и предоставить соответствующий ответ (Gaston C. Hillar, 2016). Для организации REST-архитектуры (другими словами, клиент-серверной) на рабочем компьютере необходимо было осуществить следующие действия:

– создание отдельной папки на компьютере для хранения всех файлов, связанных с веб-приложением;

– установление виртуального окружения «virtualenv» – «копии» интерпретатора Python и функциональных настроек для работы их в приложении. Установление окружения было необходимо для того, чтобы не затронуть основные установки Python на рабочем компьютере (https://pypi.python.org/pypi/virtualenv);

– установление программного обеспечения flask для создания серверной части веб-приложения на языке Python;

– создание структуры папок приложения, файла инициализации; связывающего клиентскую и серверную части; для запуска веб-сервера приложения.

В результате выполнения программного кода последнего файла происходит запуск сервера, после чего выполняется запрос на получение страницы веб-приложения посредством URL-адреса, вводимого в браузере - http://localhost:5000. Этот порядок действий обязателен для запуска разработанного приложения. В процессе создания необходимых файлов, был создан HTML-документ с командой «Hello world!», запускавшийся при вводе вышеприведенного адреса в браузере. После загрузки веб-страницы по URL-адресу

вышеупомянутая строка была отображена. Это являлось доказательством осуществления связи между браузером и сервером. В дальнейшем, этот файл наполнялся командами HTML-разметки, которые были необходимы для разработки приложения.

Таким образом, был разработан «каркас» для создания приложения. Осуществлена проверка связи между клиентской и серверной частями посредством запроса к серверу.

3.2. Разработка функциональной части приложения на стороне клиента.

Создание клиентской части приложения велась с применением современных языков и технологий: JS, CSS и языка разметки HTML5. Создание интерфейса веб-страницы приложения осуществлялось с применением функциональных возможностей языков таблиц стилей CSS и разметки веб-страниц HTML5. Описание структуры HTML-страницы и отношений между элементами HTML-документа происходили на основе правил языка HTML5. HTML-документ – обычный текстовый документ, в который были добавлены HTML-элементы и текст. Каждый элемент обозначался открывающимся и закрывающимся тэгами и имел атрибуты. Например, <input name="marshrut" value="short" id="short">, где <> - тэги, «name», «value», «id» – атрибуты. Последние содержали свое имя и значение. Любой атрибут позволял изменять свойства и поведение элемента, для которого он задан. Для отображения различных пояснительных надписей (например, названий пешеходных маршрутов) использовался один из атрибутов - текст, заключавшийся внутри HTML-элементов между тэгами. Для просмотра содержимого HTML-документа браузер отображает его в виде HTML-страницы приложения в соответствии с инструкциями, включенных в HTML-документ (таблицы стилей, программные коды) (Fulton St., Fulton J., 2011).

При разработке интерфейса веб-страницы приложения были использованы следующие HTML-элементы (рис.33):

– контейнеры «div» для добавления в них карты, координат курсора маши (номер 1) и оформления страницы (рамки белого цвета вокруг параметров маршрута);

– текстовые метки «label» для отображения адресов крайних точек маршрута (номер 2);

– изображения «img» для вставки двух картинок перед полями для адреса (номер 3);

– переключатели с атрибутами «radio» разновидностей пешеходных маршрутов (номер 4);

– переключатель с атрибутом «checkbox» для просмотра кофеен по построенному маршруту (номер 5);

– «input» с атрибутом «time» для задания пользователем времени для прогулки (номер 6);

– несколько элементов «span» для определения строчных элементов документа: координаты курсора мыши, «время прогулки», «Кофе с собой», «Выберите маршрут», «Время прохождения маршрута» и «Длина» (номер 7);

– таблица «table» для условных обозначений отображаемых объектов по маршруту (номер 8);

– кнопка «reset» для очистки содержимого карты и полей отображения адресов крайних точек маршрута (номер 9);

– ссылка «a» для создания кнопки построения маршрута (номер 10);

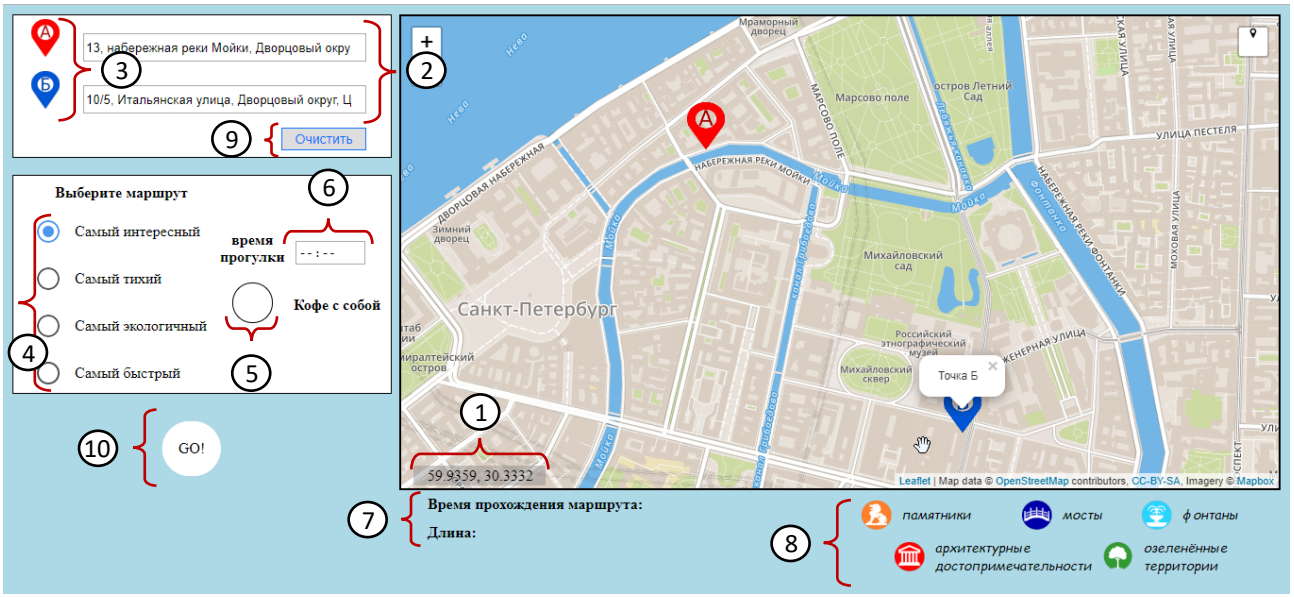


Рис. 33. HTML-элементы веб-страницы приложения.

Определение цветов и шрифтов, их размеров, форм, положения и анимации (далее объединяются в понятие «стили») для HTML-элементов веб-страницы происходило с использованием языка таблицы стилей CSS (Фрейн Б., 2016). Он позволил создавать соответствующие стили, которые были созданы для каждого элемента отдельно. Таблицы стилей языка описывали правила задания отображения элементов с использованием свойств и допустимых значений этих свойств. Для каждого HTML-элемента можно было использовать определенный набор свойств. Например, для контейнера «div», содержащего карту (рис.33), были определены следующие свойства: ширина, высота, стиль и толщина обводки, тип положения, параметры размещения.

Функциональные возможности языка JS также использовались для разработки клиентской части приложения. JS как язык сценариев позволил сделать веб-страницу приложения интерактивной, изменяя её содержимое (Никсон Р., 2016). Например, нажатие

на любую кнопку или передвижение маркеров сопровождалось последующими действиями (или сценариями), прописанные в коде JS. Написанные сценарии хранятся в различных файлах, которые встраивались в HTML-документ как ресурсы с помощью адресов, указанных на эти ресурсы. Выполнение сценариев при пользовании приложением обеспечивается возможностями браузера, в котором уже по умолчанию установлен интерпретатор JS.

При написании программного кода на этом языке было созданы функции, обеспечивающие:

– добавление карты OSM на веб-страницу, которая подгружается с использование API Mapbox (см. раздел 1.4); задание координат окна карты, его масштаба;

– отображение координат курсора мыши при нахождении его на карте;

– добавление маркеров (крайних точек маршрута) на карту, при этом происходит добавление их адресов в HTML-элементы страницы; создание возможности движения маркеров с одновременным обновлением их адреса в HTML-элементах;

– передачу запросов серверу методом «GET» (см. раздел 1.3);

– построение линии маршрута по координатам его вершин, полученных в ответе сервера на запрос. При этом обеспечивается отправление в запросе координат маркеров, вида маршрута, выбранного пользователем и желаемого времени прогулки (при наличии). В случае построения результирующего маршрута реализованы изменение масштаба карты по линии маршрута, отображение времени и его длины. Если маршрут не был построен, то после отображения соответствующего окна пользователь без перезагрузки страницы может изменить параметры маршрута для его построения; в этой же функции реализована перестройка маршрута «на лету», когда пользователь может только передвинуть маркеры без повторного нажатия на кнопку «GO!»; если маршрут был построен без задания времени для прогулки, обеспечена возможность перестройки маршрута с указанием времени без перезагрузки веб-страницы; реализовано отображение объектов по маршруту и высвечивание названия выбранного при нажатии;

– отображение кофеен и высвечивание названия выбранной при нажатии в случае активной кнопки «Кофе с собой» и любого построенного маршрута;

– удаление с карты всех расположенных на ней элементов, сброс адресов маркеров, времени прохождения маршрута и его длины, времени прогулки (при указании) в случае нажатии на кнопку «Очистить»;

Написание программного кода всех описанных функции, применение стилей для HTML-элементов страницы и создание ее разметки с помощью HTML5 позволили завершить разработку клиентской части приложения (рис.34).

```
function outMapMouse(e) {
    document.getElementById('Coordinates').innerHTML = '';
    document.getElementById('coordDiv').style.opacity = 0;
}

function onMapClick(e) {

    var f_id=((!GLOBAL_OPTIONS.markerArray[0])&& 1)|||((!GLOBAL_OPTIONS.markerArray[1])&& 2)||3;

    switch(f_id) {

        case 1:
        case 2:
            var greenIcon = L.icon({
                iconUrl: "static/" + f_id + ".png",
                iconSize:     [40, 50]
                //shadowSize:   [50, 64],
                //iconAnchor:   [22, 94],
                //shadowAnchor: [4, 62],
                //popupAnchor:  [-3, -76]
            });
            var iconOptions = {
                clickable: true,
                draggable: true,
                icon: greenIcon
            }

            var geocoder = new L.Control.Geocoder.Nominatim();
```

Рис. 34. Фрагмент программного кода клиентской части приложения, написанного на языке JS

3.3. Создание серверной части приложения.

Созданное веб-приложение выполняет основную функцию – построения пешеходных маршрутов. До этого просмотр любого построенного маршрута, иными словами, результата выполнения запроса к созданной базе данных, осуществлялся в программе QGIS с использованием модуля «pgRoutingLayer» (рис.35). Для отображения маршрута на карте в приложении необходимо было обеспечить связь базы данных и клиентской части приложения. Для создания сервиса в целях обеспечения основной функции приложения заключался в написании на языке Python нескольких функций, которые возвращают:

1. номер идентификатора вершины в дорожном графе. Аргументами являются географические координаты маркера. Программа находит ближайшую к маркеру вершину графа по их географическим координатам и возвращает номер идентификатора (далее id) найденной вершины;
2. координаты точек маршрута, его длину и список id вершин-источников ребер графа, по которым проходит маршрут. Функция отправляет запрос к базе данных на получение списка координат точек маршрута, строящегося по одному из двух алгоритмов, указанных в списке ее аргументов;

3. список координат объектов по маршруту с их названиями, который формируется в результате выполнения запроса к базе данных (см. раздел 2.6);
4. координаты точек маршрута, объектов вдоль этого маршрута (координаты + название), время прохождения маршрута, id ошибки. Основные шаги работы функции:

   – в случае отсутствия заданного пользователем времени для прогулки функция возвращает результаты выполнения функций под номерами 2 и 3;

   – при указании времени для прогулки. Вычисляется время самого короткого маршрута. Если оно больше, чем заданное пользователем, функция возвращает только id ошибки. Если оно меньше, функция обращается за получением результатов выполнения функции под номером 2, и только тогда, когда не возвращается длина маршрута (при использовании алгоритма k-кратчайших, см. раздел 2.5), функция возвращает единственное значение - id ошибки.

   Любой id возможных ошибок, кроме обозначающего возвращение функцией координат точек маршрута, вызывает появление сообщения на веб-странице. Одно из них показано на рис.45.

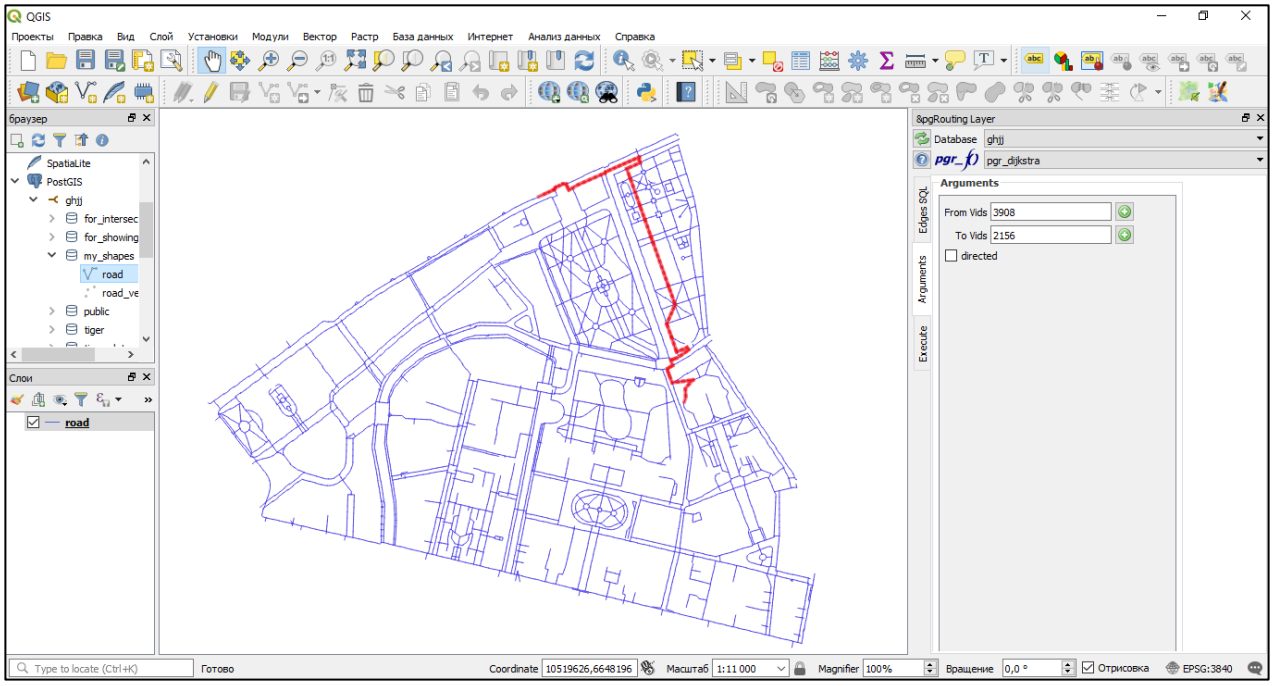


Рис.35. Построения одного из видов пешеходных маршрутов в программе QGIS.

Функция под номером 4 является основной, поскольку значения, полученные в результате ее выполнения, отправляются как ответ сервера на запрос веб-браузера. Веб-браузер перед этим формирует запрос в виде географических координат точек маркеров, тип маршрута и максимально возможное время прогулки (при задании). Перечисленные

параметры используются как аргументы четырех описанных функций. Веб-браузер получает ответ на составленный запрос и использует его для создания линии маршрута, используя полученные координаты вершин этого маршрута. Также в ответе содержатся данные для отображения объектов вдоль маршрута, времени и длины его прохождения. В случае ответа, в котором содержится только один из id ошибки, на веб-странице выводится соответствующее сообщение. Описанные четыре функции были выделены в отдельный модуль, который подключался в определенном файле, созданном в результате действий, описанных в разделе 3.1 (рис.36).

```
def get_id (point,db):
    id_point = "SELECT id " \
        "FROM my_shapes.road_vertices_pgr " \
        "ORDER BY ST_Distance(ST_Transform(the_geom,4326), 'SRID=4326;POINT(%(point[1])s %(point[0])s)'::geometry) ASC " \
        "LIMIT 1;" % {'point[1]': point[1], 'point[0]': point[0]}

    return db.query(id_point)[0][0]

def Route(pointA,pointB,cost,length,type_algor,count_marsh,db):

    # ВОЗВРАЩАЕТ КООРДИНАТЫ И ДЛИНЫ РЕБЕР САМОГО .. МАРШРУТА
    e1 = "SELECT ST_AsText(ST_Transform(the_geom,4326)),(ST_Length(geom)), id2 " \
        "FROM ((SELECT *FROM pgr_dijkstra(' " \
        "SELECT id AS id, " \
        "source AS source, " \
        "target AS target, " \
        "%(cost)s::float8 AS cost " \
        "FROM my_shapes.road', " \
        "%(pointA)s, " \
        "%(pointB)s, " \
        "false, " \
        "false))  as DJ1 INNER JOIN my_shapes.road as road " \
        "ON DJ1.id2 = road.id) AS DJ INNER JOIN my_shapes.road_vertices_pgr AS Tochki  " \
        "ON DJ.id1 = Tochki.id" % {'cost': cost, 'pointA': get_id(pointA, db), 'pointB': get_id(pointB, db)}

    #ВОЗВРАЩАЕТ КООРД И ДЛИНЫ РЕБЕР МАРШРУТА ИЗ К-КРАТЧ
    e2 = "WITH route AS " \
        "(SELECT * FROM pgr_KSP( " \
        "'SELECT id, source, target, " \
        "%(cost)s as cost " \
        "FROM my_shapes.road', " \
        "%(pointA)s, " \
        "%(pointB)s, " \
        "%(count_marsh)s, " \
```

Рис.36. Фрагмент программного кода созданного модуля

Таким образом, был создан модуль, который обеспечивает получение данных для построения пешеходных маршрутов. Осуществлено получение запроса от браузера, генерирование ответа (посредством работы функций в модуле) и отображение одного из четырех видов пешеходных маршрутов на карте приложения – все это является доказательством функционирования приложения.

3.4. Примеры построений пешеходных маршрутов в приложении.

После создания клиентской и серверной частей приложения необходимо было проверить достоверность полученных результатов построения всех видов пешеходных маршрутов. Для этого сначала были выбраны две крайние точки маршрута с адресами (рис.37).

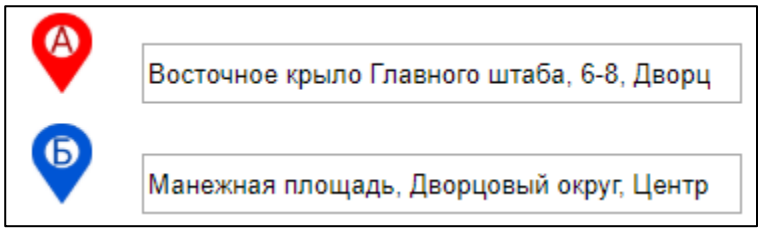


Рис.37. отображение адресов точек маршрута при помещении маркеров на карту

Затем были построены четыре вида маршрута. Для оценки правильности построения трех из них было произведено сравнение «самого интересного», «самого тихого» и «самого экологичного» маршрутов с графом пешеходной дорожной сети проекта, где каждое ребро было окрашено в цвет значения определенного веса.

Оценивая результаты построения «самого интересного» маршрута (рис.38) можно сказать, что алгоритм выбрал прохождение маршрута вдоль набережной реки Мойка (рис.39) (номер 1) вместо прохождения маршрута вдоль Невского проспекта (номер 2), поскольку, исходя из рисунка, вес «притяжения» набережной больше. Далее маршрут был построен через следующую область «притяжения», расположенную около храма Спаса на Крови (номер 3) и Михайловского сада (номер 4), что является правильным выбором алгоритма, поскольку за пределами этой области, веса дорог в окрестности имеют меньшее значение. Следующая область притяжения – территория рядом с Михайловским замком (номер 5) и площадь Петра Великого (номер 6), через которую также был проложен маршрут, поскольку значения весов элементов дорожного графа рядом проходящей Садовой улицы (номер 7) меньше.

Рис.38. «Самый интересный» маршрут

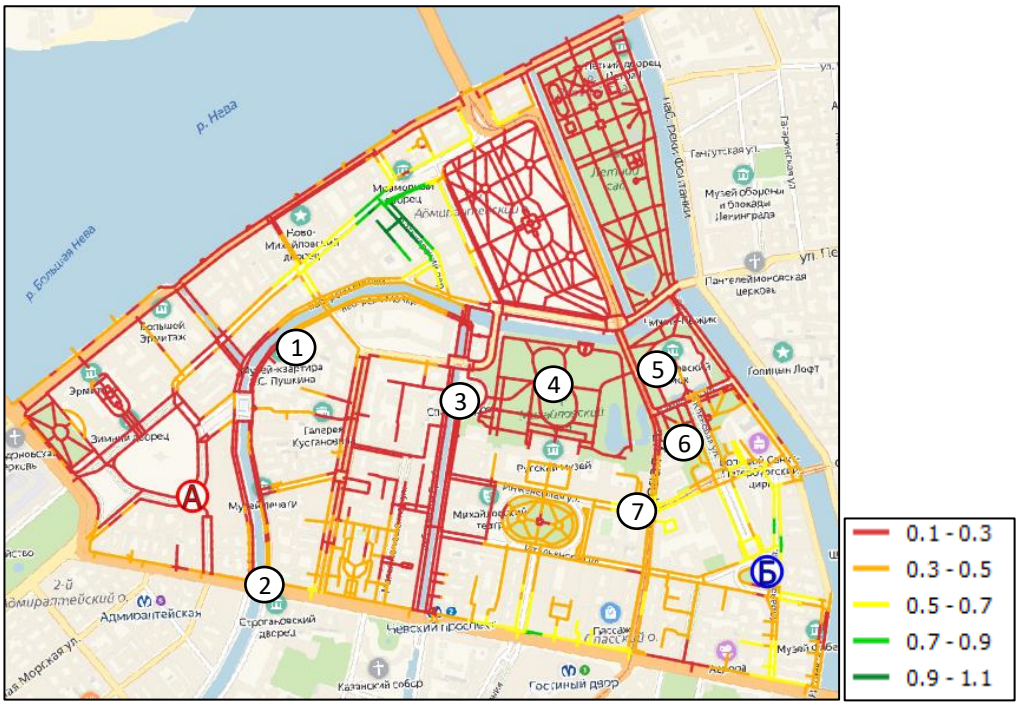


Рис.39. Изображение графа пешеходных дорог с отображением весовых значений «притяжения» объектов

Анализируя построение «самого тихого» маршрута (рис.40) можно заметить, что в его начале программа выбрала прохождение по левой стороне улицы (рис.41) (номер 1) с меньшим значением уровня шума, а не по правой с большим. Далее маршрут проходит по левой стороне набережной реки Мойка (номер 2), а не по правой по той же причине. Затем маршрут строится по Аптекарскому переулку (номер 3) вместо продолжения движения по набережной, вдоль которой по-прежнему сохраняется относительно большое значение уровня шума. Марсово поле (номер 4) – область с низким уровнем шума, следовательно, маршрут был построен через него. Поскольку на Садовой улице (номер 6), по данным изображения, более шумно, чем на площади Петра Великого (номер 5), алгоритм построил маршрут через последнюю.

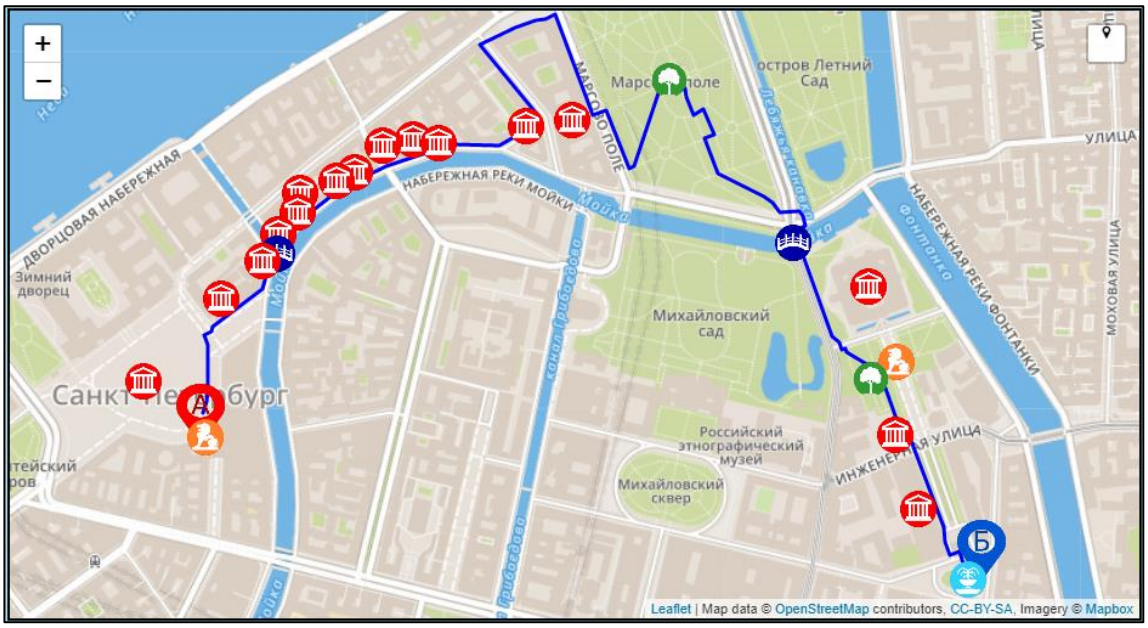


Рис.40. «Самый тихий» маршрут

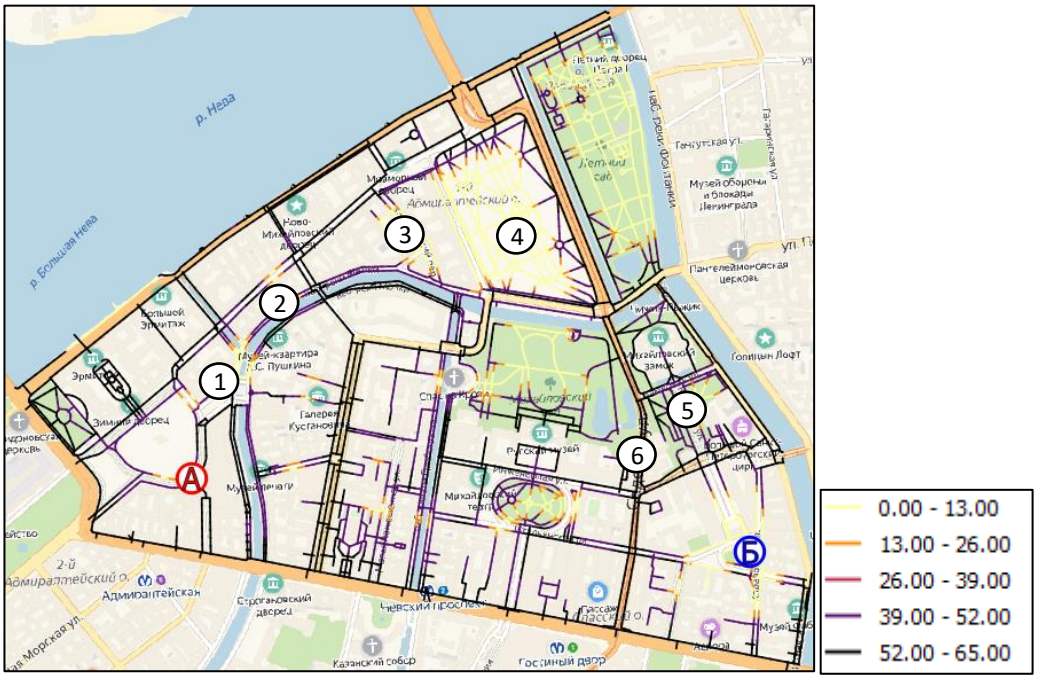


Рис.41. Изображение графа пешеходных дорог с отображением весовых значений шумового загрязнения

Что касается, «самого экологичного» маршрута (рис.42), то он был построен с целью прохождения через наибольшее количество озелененных зон между крайними точками маршрута (рис.43): Большая Конюшенная улица (номер 1), Михайловский сад (номер 2), экологичные зоны около Михайловского замка (номер 3), Площадь Петра Великого (номер 4) и Кленовую улицу (номер 5).

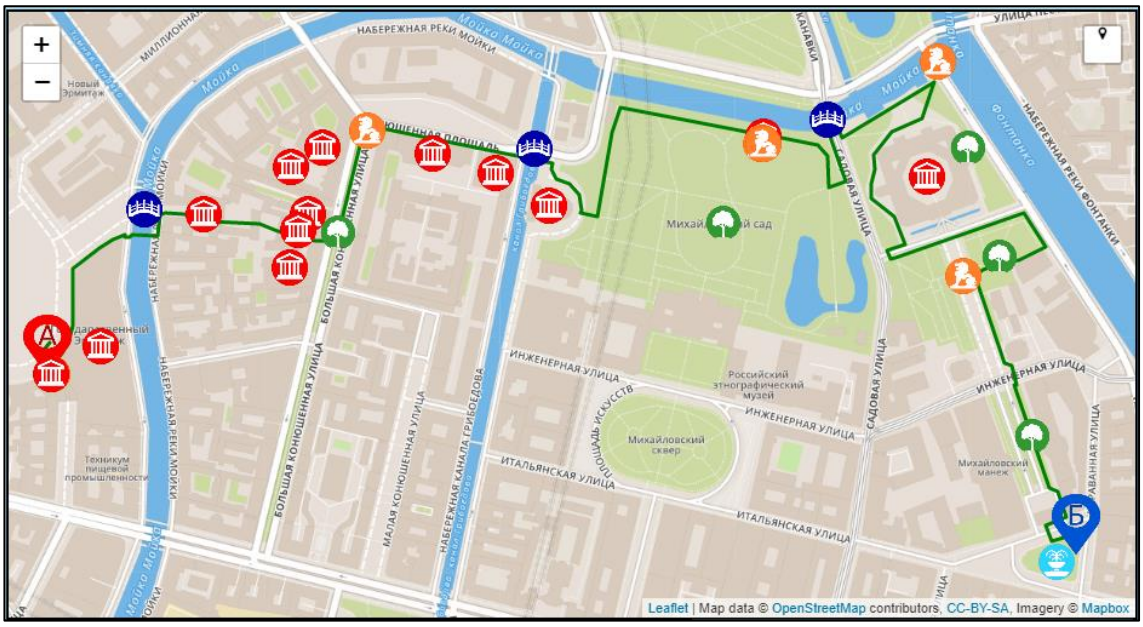


Рис.42. «Самый экологичный» маршрут

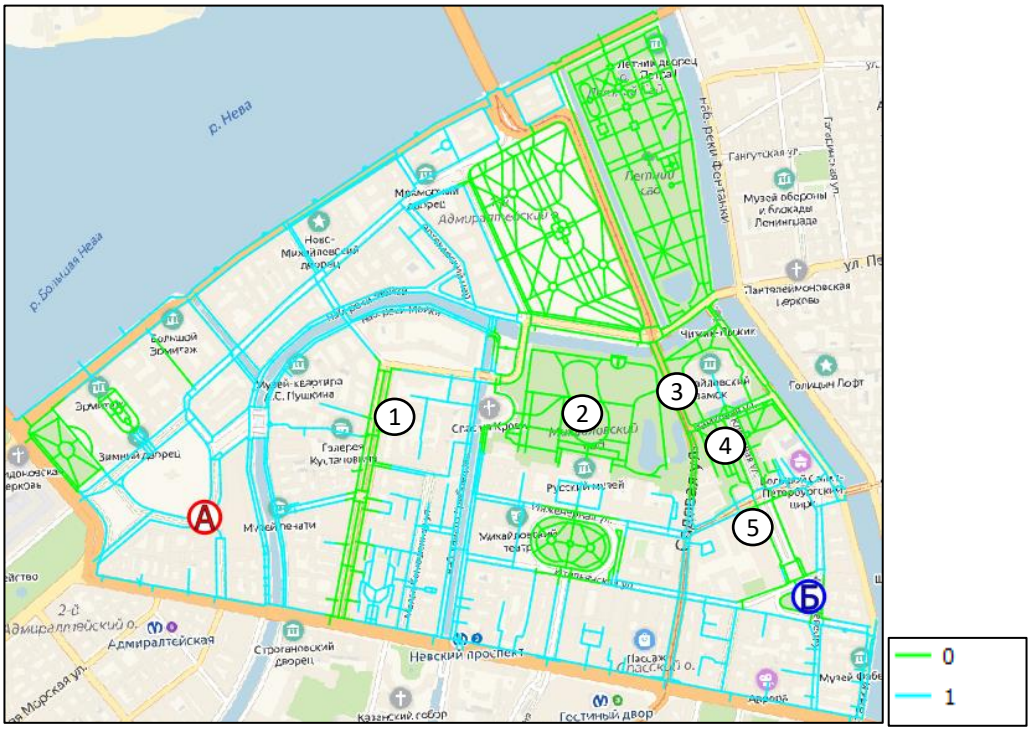


Рис.43. Сеть пешеходных дорог с отображением весовых значений принадлежности ребер графа к озелененным территориям

Построение «самого быстрого» маршрута можно увидеть на рис.44, на котором представлена и веб-страница созданного приложения. Поскольку слева от карты был выбран параметр «Кофе с собой», вдоль маршрута обозначены кофейни, название которых можно просмотреть, нажав на объект на карте. Также показано отображение названия памятника при нажатии на него на рис.38. Более того, можно заметить отображение времени

прохождения маршрута и его длины слева под картой. Справа представлены условные обозначения других объектов по маршруту.

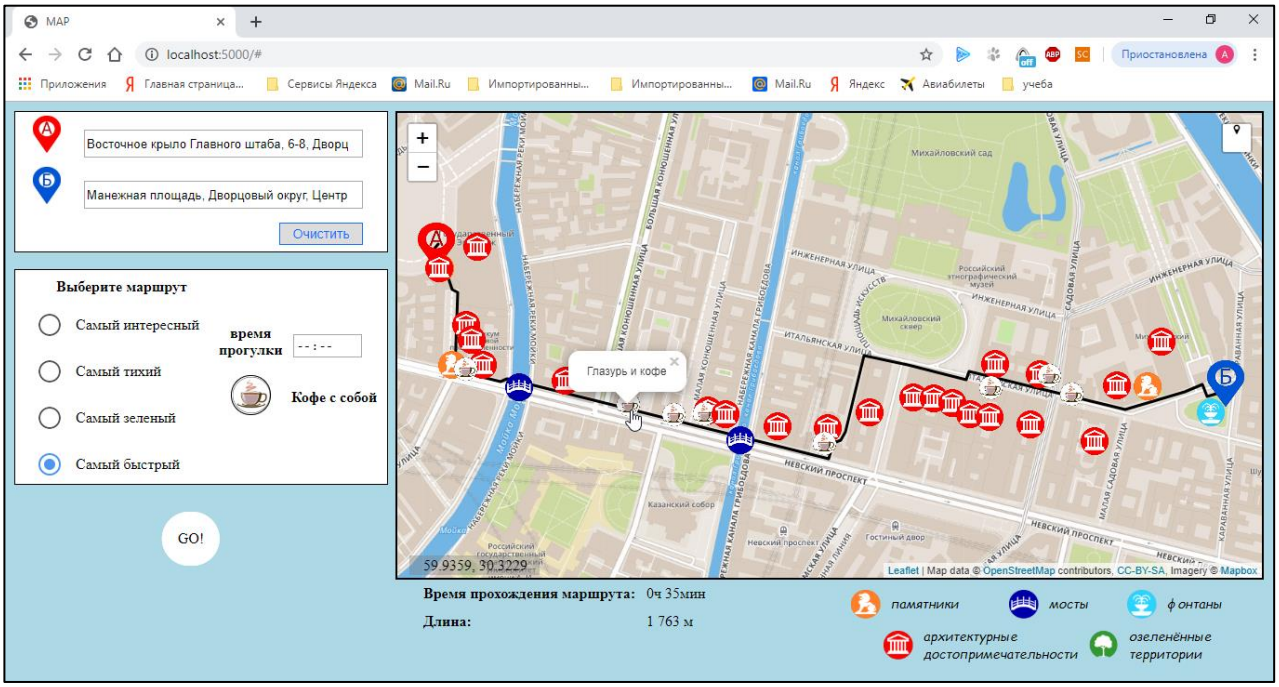


Рис.44. «Самый быстрый» маршрут

В случае, когда время прохождения кратчайшего маршрута (рис.44) больше времени, задаваемого пользователем при выборе, например, «самого интересного маршрута», то возникнет сообщение наверху веб-страницы приложения (рис.45)

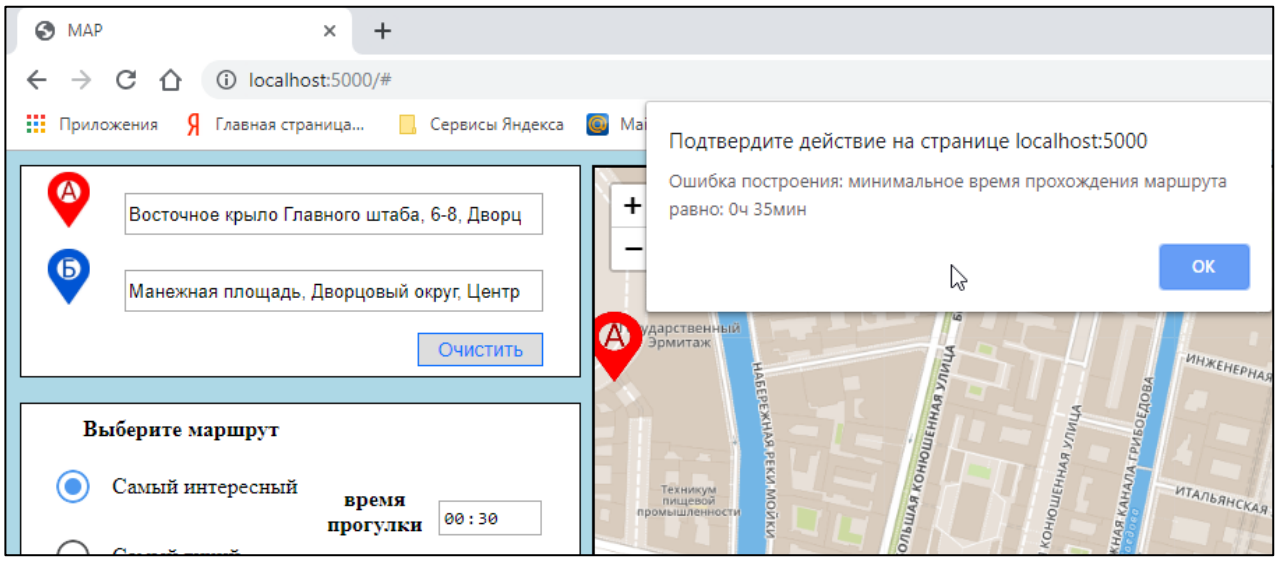


Рис.45. Сообщение об ошибке построения маршрута с заданными условиями

В результате анализа трех видов построенных пешеходных маршрутов в приложении, было подтверждено согласование каждого из них с соответствующими значениями весов ребер пешеходного дорожного графа. Тестовое построение всех видов маршрутов показало его работоспособность. Продемонстрировано изображение оформления

веб-страницы приложения, доказательство ее интерактивности на примере нажатия на объекты карты.

Таким образом, разработка клиент-серверной архитектуры обеспечила создание по отдельности двух частей приложения. Была выполнена разработка клиетской части с использованием возможностей трех языков, каждый из которых выполняет свою функцию в приложении. Создание модуля с обращением в нем к базе данных позволило создать сервис, который выполняет главную задачу созданного приложения – построение различных пешеходных маршрутов. В конечном итоге, построение четырех видов пешеходных маршрутов в приложении является результатом его функционирования.

Заключение

В процессе создания пилот-проекта приложения по построению пешеходных маршрутов с пользовательскими параметрами было выполнено следующее:

– проанализированы существующие приложения по построению пешеходных маршрутов;

– собраны данные по 400 объектам, отображаемым вдоль строящихся пешеходных маршрутов;

– получена «карта» шумового загрязнения территории проекта с применением автоматизированных методов;

– разработаны три вида весовых значений для элементов сетевой модели данных пешеходных дорог;

– выбраны два алгоритма для построения маршрутов с различными входными параметрами: Дейкстры и k-кратчайших путей;

– создана реляционная база данных Postgis;

– составлены запросы к базе данных для решения задач маршрутизации и отображения объектов вдоль пешеходных маршрутов;

– разработана клиент-серверная архитектура приложения;

– написан программный код для функционирования клиентской части приложения средствами JS,CSS и HTML5;

– создана серверная часть приложения;

В результате проделанной работы был создан пилот-проект веб-приложения, позволяющего строить четыре вида пешеходных маршрутов на тестовом участке центра города Санкт-Петербург («самый интересный», «самый тихий», «самый экологичный» и («самый быстрый»). Дополнительно пользователь приложения имеет возможность выбирать параметры маршрута, такие как: задание максимально возможного времени для прогулки и отображение кофеен по пути следования. В процессе работы приложения пользователь получает информацию о необходимом для прогулки времени, длине пути и встречающихся на его пути: мостах, фонтанах, памятниках и архитектурных достопримечательностях, озелененных территориях. Разработанное приложение может функционировать на любом компьютере, где находится необходимая база данных и интерпретатор Python.

Одним из возможных вариантов развития созданного проекта является разработка мобильного приложения. Установка приложения на мобильное устройство позволит учитывать географическое положение пользователя, что, в свою очередь, обеспечит навигацию с помощью мобильного телефона в процессе прогулки. Другой вариант развития

проекта – сохранить приложение в формате веб и обеспечить доступ к нему любого желающего.

Список литературы

1. Beaulieu A. Learning SQL. 2nd ed. Sebastopol, O'Reilly Media Publ., 2005, 405 p.
2. Dechter, R., Pearl, J. Generalized best-first search strategies and the optimality of A* // Journal of the ACM. - 1985. - T. 32, No 3. - P. 505 - 536.
3. Fulton St., Fulton J. HTML5 Canvas. 1st ed. Sebastopol, O'Reilly Media Publ., 2011, 652p.
4. Gaston C. Hillar. Building RESTful Python Web Services. Birmingham, Packt Publ., Ltd, 2016, 412p.
5. Guo, Q., et al. The development of urban night tourism based on the nightscape lighting projects-a Case Study of Guangzhou. // Energy Procedia 5. 2010 International Conference on Energy, Environment and Development. -2011. -P. 477 -481. - ISSN 1876-6102.
6. Miura, H. et al. A Study on Navigation System for Pedestrians Based on Street Illuminations. // Knowledge-Based and Intelligent Information and Engineering Systems. -2011. -P. 49-55.
7. Mostafa Refat, I. A Parametric Study of the Effect of Building Distributions and Size on the Propagation of Sound in the Urban Environment. // Journal of Architectural Engineering Technology 3.1. -2014. -P. 1-8. -ISSN 2168-9717.
8. Mukhina, K., Rakitin, S., Visheratin, A. Detection of tourists attraction points using Instagram profiles. // Procedia Computer Science 108:2378-2382. - 2017. -P. 2378-2382. -ISSN 1877-0509.
9. Quercia, D., Rossano S., and Aiello L. M. The Shortest Path to Happiness: Recommending Beautiful, Quiet, and Happy Routes in the City. // Proceeding HT '14 Proceedings of the 25th ACM conference on Hypertext and social media. -2014. -P. 116-125.
10. Yasufumi, T. et al. Walking Route Recommender for Supporting a Walk as Health Promotion. // IEICE Transactions on Information and Systems. -2017. -P. 671-681.
11. Yen, Jin Y. Finding the K Shortest Loopless Paths in a Network. //Management Science 17.11. -1971. -P. 712-716.
12. Zhong, W., Chen, F. et al. SAFEBIKE: A Bike-sharing Route Recommender with Availability Prediction and Safe Routing. // arXiv:1712.01469. – 2017.
13. Доусон М. Программируем на Python. - СПб.: Питер, 2014, 416 с.
14. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ. Под ред. И. В. Красикова. М.: Вильямс, 2005, 1296 с.
15. Кузнецов О. П., Адельсон-Вельский Г. М. Дискретная математика для инженера. М.: Энергия, 1980, 344 с., ил.
16. Левитин А. В. Введение в разработку и анализ. М.: Вильямс, 2006, 576 с.
17. Лутц М. Изучаем Python, 4-е изд. / пер. с англ. - СПб: Символ-Плюс, 2011, 1280 с.

18. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 4-е изд. СПб.: Питер, 2016, 768 с., ил.
19. Флэнаган Д. JavaScript. Подробное руководство / пер. с англ. - СПб: СимволПлюс, 2008, 992 с., ил.
20. Фрейн Б. HTML5 и CSS3.Разработка сайтов для любых браузеров и устройств. 2-е изд. СПб.: Питер, 2016., 212 с., ил.
21. Харари Ф. Теория графов. - М.: УРСС, 2003, 300 с.

Ресурсы сети Интернет:

22. https://pypi.python.org/pypi/virtualenv-виртуальное окружение Python. (дата обращения: 08.02.2019).
23. https://postgis.net-геопространственная база данных Postgis. (дата обращения: 07.02.2019).
24. https://tech.yandex.ru/direct/doc/dg/concepts/about-docpage/-документация API Яндекс.Директа. (дата обращения: 09.02.2019).
25. https://desktop.arcgis.com/ru/arcmap/latest/tools/spatial-analyst-toolbox/euclidean-distance.htm-Евклидово расстояние в ArcGIS. (дата обращения: 01.04.2019).
26. https://medium.com/@urbica/walkstreets-5a41b22ae104-описание приложения Walkstreets. (дата обращения: 06.02.2019).
27. https://yandex.ru/maps-поисково-информационная картографическая служба «Яндекс.Карты». (дата обращения: 10.02.2019).
28. https://pgrouting.org/-расширение pgRouting. (дата обращения: 01.02.2019).
29. https://tech.yandex.ru/maps/doc/geosearch/concepts/about-docpage/-сервис поиска по организациям компании «Яндекс». (дата обращения: 10.02.2019).
30. http://travelpath.ru/-сервис построения туристических маршрутов TravelPath. (дата обращения: 04.02.2019).
31. https://github.com/urbica/noisemap-создание «карты» шумового загрязнения. (дата обращения: 05.02.2019).
32. http://overpass-turbo.eu/-утилита для фильтрации данных OpenStreetMap. (дата обращения: 03.02.2019).
33. http://lukasmartinelli. ch/gis/2016/04/03/openstreetmap-noise-pollution-map.HTML-Martinelli, Lucas. Global Noise Pollution Map. (дата обращения: 02.02.2019).
34. Semenov A. Development of service suggesting walking routes. // MSc Dissertation. W., Wuhan University, 2018, 45p. URL: https://www.researchgate.net/publication/325567805_Development_of_service_suggesting_walking_routes (дата обращения: 02.06.2019).