МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____Г.И. Радченко
«____»_____2021 г.

Разработка веб-приложения «Карта помещений ЮУрГУ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к. т. н., доцент каф ЭВМ
_____Е.С. Ярош
«_____»_____2021 г.

Автор работы
студент группы КЭ-405
_____О.В. Севостьянов
«_____»_____2021 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____С. В. Сяськов
«_____»_____2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2021 г.

ЗАДАНИЕ
на выпускную квалификационную работу студента
студенту группы КЭ-405
Севостьянову Олегу Валерьевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. Тема работы: «Разработка веб-приложения «Карта помещений ЮУрГУ»» утверждена приказом по университету от 26 апреля 2021 года №714-13/12 (приложение №73)

2. Срок сдачи студентом законченной работы «5» июня 2021 г.

3. Исходные данные к работе

- кроссплатформенное web-приложения;
- возможность поиска помещения по критериям номер аудитории, наименованию, ФИО сотрудника;
- интерактивное взаимодействие с графическим интерфейсом.

4. Перечень вопросов, подлежащих разработке

- постановка и анализ задачи;
- анализ-обзор существующих решений, в том числе зарубежных;
- определение требований к web-приложению;
- разработка архитектуры web-приложения.

5. Дата выдачи задания «1» декабря 2020 г.

Руководитель_____
(подпись) (И.О. Ф.)

Задание принял к исполнению_____
(подпись студента) (И.О. Ф.)

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2021	
Разработка модели, проектирование	01.04.2021	
Реализация системы	01.05.2021	
Тестирование, отладка, эксперименты	15.05.2021	
Компоновка текста работы и сдача на нормоконтроль	24.05.2021	
Подготовка презентации и доклада	30.05.2021	

Руководитель работы _____ /Е. С. Ярош_____/

Студент _____ /О. В. Севостьянов_____/

Аннотация

Севостьянов О. В. «Разработка веб-приложения «Карта помещений ЮУрГУ»». – Челябинск ФГАОУ ВО «ЮУрГУ» (НИУ) ВШЭКН; 2021, 87 с., 21 ил. Библиографический список – 29 наименований.

Работа посвящена разработке web-приложения «Карта помещений ЮУрГУ». Оно предназначено для облегчения ориентации внутри кампуса НИИУ «ЮУрГУ».

Данная работа состоит из введения, пяти глав, заключения, библиографического списка.

В первой главе представлен обзор существующей литературы на тему интерактивных карт, web-приложений и геоинформационных систем. Во второй главе – анализ аналогов, сравнение картографических сервисов. В третьей главе описано планирование, сценарии использования, выбор технологий. В четвёртой главе описано проектирование web-приложения и взаимодействие компонентов. В пятой главе – реализация программного продукта. В шестой главе описано тестирование полученного приложения.

В заключении проведено описание результатов работы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность Темы. По статистике с портала WebCanape о состоянии интернета на начало 2021 года в мире [1]:

- интернетом пользуются 4,66 миллиарда человек, что на 7,3 % больше, чем в прошлом году;
- мобильным телефоном пользуются 5,22 миллиарда человек - 66,6 % мирового населения, что на 1,8 % больше, чем в прошлом году.

Эти показатели будут только расти со временем, так как количество людей и доступность мобильных устройств тоже увеличиваются.

На текущий момент интернет является незаменимым инструментом в повседневных задачах, поэтому количество сайтов и web-сервисов неумолимо растёт. Это решает также проблему отсутствия единой платформы устройств, так как web является кроссплатформенным (работает на любой платформе, будь то Android или IOS).

«Интерактивная карта – это карта, которая работает в двустороннем режиме, то есть человек может задействовать какие-то функции и передавать какие-то данные» [2]. Эти карты получили своё широкое распространение относительно недавно. Когда интерактивными картами стало пользоваться большое количество людей, начали появляться новые направления использования геоданных: построение маршрутов (как локальных, в городах, так и глобальных, по континентам), визуализации данных на местности (например, распределение плотности населения), бизнес (карта предприятия на сайте этого предприятия) и т. п.

Разработка интерактивной карты актуальна для различных производственных предприятий, учебных заведений, спортивных комплексов, торгово-развлекательных комплексов, с большой площадью. НИУ «ЮУрГУ»

входит в этот перечень, так как в нём имеется 18 учебных корпусов, не считая общежитий и филиалов. А количество помещений порядка 2,5 тысяч.

Цель работы. Цель выпускной квалификационной работы – разработка web-приложения «Карта помещений ЮУрГУ».

Задачи работы. В соответствии с целью были сформулированы следующие задачи:

- определить основные потребности потенциальных пользователей;
- освоить основные средства создания web-приложения;
- изучить основные средства создания интерактивных карт;
- изучить основные средства работы с БД (база данных);
- определить инструменты взаимодействия БД и интерфейса интерактивных карт;
- реализовать БД;
- реализовать интерактивные карты;
- разработать интерфейс взаимодействия пользователя с интерактивной картой.

Значимость работы. Теоретическая значимость проекта состоит в изучении всего стека web-технологий и принципов работы с БД для реализации поставленных задач.

Практическая значимость проекта состоит в разработке системы навигации по интерактивной карте НИУ «ЮУрГУ» с использованием всего выше перечисленного стека технологий.

Выпускная квалификационная работа состоит из введения, обзора литературы, сравнения отечественных и зарубежных технологий, основной части, заключения, библиографического списка и приложения.

1. ОБЗОР ЛИТЕРАТУРЫ

В текущей главе проведём исследования на предмет упоминания темы работы в других источниках. Это прежде всего нужно для формулировки, реализации исследовательского процесса и исключения случаев создания полного аналога уже существующего продукта [3].

1.1. ОБЩИЕ СВЕДЕНИЯ О WEB-КАРТОГРАФИИ

Быков и Пьянков в учебном пособии «Web-картографирование» [4] детально описали все этапы создания web-карт, начиная с понятия о вычислительных сетях, их архитектуры и механизмов работы. Из этого источника мы можем сделать ряд выводов о каждом этапе создания и проектирования web-карты.

Основой web-карты должно быть web-приложение с присущими ему незаменимыми частями:

- СУБД и сервер базы данных;
- архитектура клиент-сервер;
- web-сервис;
- HTML.

В следующей главе рассмотрены принципы разработки web-приложений. Автор выделяет следующие навыки для разработки web-приложений:

- знание языка разметки HTML;
- знание языка стилей страниц CSS;
- знание языка сценариев JavaScript;
- навыки разработки серверной части;
- навыки обслуживания серверной части.

В следующих главах описаны фундаментальная информация о ГИС (рассмотрим позже) и примеры уже существующих сервисов в общем доступе.

1.2. ИСТОРИЯ WEB-КАРТОГРАФИИ

Немного о истории создания web-карт. В 1993 году появился первый картографический сервер на основе стандарта CGI (стандарт интерфейса, используемого для связи внешней программы с web-сервисом), написанный на языке Perl от компании Xerox Corporation, который назывался Xerox PARC Map Viewer. Он позволял переопределять стили и определять разные экстенты (часть области региона, показанная на карте) карты. 1996 год ознаменовал появление инструментов для создания пользовательских web-карт. С 2004 года начали появляться некоммерческие карты, как OSM (OpenStreetMap), которые создаются силами обычных пользователей, и по сей день количество таких карт только растёт. Основные события по годам описали Кацко и Кикин в своём труде «Состояние и проблемы веб-картографировании на современном этапе развития единого геоинформационного пространства» [6], они представлены в таблице №1.

Таблица 1 - Хронология развития web-картографии

Дата	Событие
1993 – 1994 гг.	Выход первых картографических веб-приложений раннего поколения (Xerox PARC Map Viewer, Национальный атлас Канады)
1996 – 1999 гг.	Появление и бурное развитие интерактивных картографических веб-систем (Mapquest, MultiMap, Geomedia WebMap 1.0, UMN MapServer 1.0, Terraserver USA). Дальнейшее развитие систем раннего поколения (US Online National Atlas Initiative). Выход гигантов мировой ИТ-индустрии на рынок вебкартографии (проект

	Terraserver USA был создан и внедрен при активном участии Microsoft и HP)
2000 – 2003 гг.	Начало эпохи распределенных картографических веб-платформ (UMN MapServer 3.0-3.5-4.0, ESRI ArcIMS 3.0-4.0) и сервисов (ESRI Geography Network, NASA World Wind)

Продолжение таблицы 1

Дата	Событие
2004 г	В апреле Стив Кост запустил проект Open Street Maps. Google и Yandex начали разработку своих распределенных картографических веб-сервисов
2005 г	Выпущен первый релиз картографического веб-сервиса Google Maps, предоставивший доступ к масштабируемым картам всего земного шара через интерактивный навигационный интерфейс. Вышел первый релиз картографической платформы Microsoft Virtual Earth и ее веб-интерфейса
2006 г	В мае Андрей Корякин и Евгений Савельев запустили проект WikiMapia. В ноябре Microsoft впервые в области веб-картографии добавила возможность интерактивного просмотра трехмерных изображений объектов на карте в своем веб-сервисе
2007 г	В мае запущен веб-сервис Yahoo! Map

Продолжение таблицы 1

Дата	Событие
2008 г.	В начале года Microsoft переименовала свой картографический веб-сервис в Live Search Maps, одновременно интегрировав его в свою глобальную систему веб-сервисов Live SearchB апреле 2008 г. основатели сообщества Open Street Maps получили инвестиции на развитие компании Cloudmade. Миссией Cloudmade является создание широкого спектра картографических приложений для настольных и мобильных устройств, использующих данные и инфраструктуру сообщества Open Street MapsB августе Cloudmade в сотрудничестве с компанией Cogniance выпустила собственный Web API, который обеспечивает сторонним разработчикам доступ к картографическим данным сообщества Open Street Maps и интеграцию динамических картографических изображений в свои веб-решения. В сентябре «Яндекс» объявил о поддержке его сервисом «Яндекс Карты» карт всего мира

1.3. ГИС

На самом деле появление web-карт обусловлено новой вехой развития использования ГИС (географическая информационная система). Эти системы созданы для сбора и анализа географических и связанных с ними данных. Однако представление таких данных в виде стандартной таблицы были не очень удобны, и именно поэтому отображение геоданных на местности было так важно.

И что же, всё-таки, представляют из себя эта система? В общем случае ГИС не имеет точного определения, потому как это набор подсистем, объединённых в одно название, этот термин в свою очередь может менять своё значение в зависимости от целей использования. Однако можно дать определения в контексте этой работы. ГИС – «это инструменты для обработки пространственной информации (ДеМерс М. Н.)» [5]. Как говорилось выше в состав ГИС входит ряд подсистем, таких как:

- подсистема сбора данных – собирает и проводит предварительную обработку данных их различных источников;
- подсистема хранения – организует пространственные данные;
- подсистема манипуляции – выполняет различные задачи на основе этих данных;
- подсистема вывода – отображает всю базу данных или её часть.

1.4. ТИПЫ WEB-КАРТ

Первую классификацию предложил Menno-Jan Kraak в 2001 году в авторской книге «Web Cartography» [7]. Он различал статические и динамические веб-карты, а также разделял каждый тип на интерактивные и web-карты «только для просмотра» (view only). Схема разделений приведена на рисунке 1.

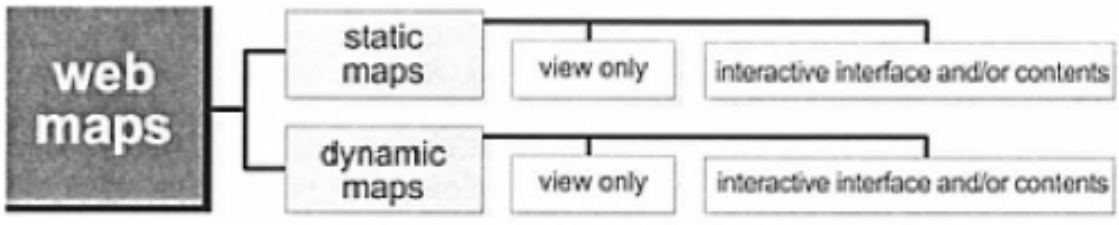

Рисунок 1 - Типы web-карт

Однако на сегодняшний день количество типов web-карт значительно увеличилось. Выделяют несколько основных типов web-карт [8]:

- аналитические web-карты. Они предлагают анализ ГИС. Геоданные могут быть статическими или требовать обновления. Части анализа могут быть выполнены сервером геоданных ГИС;
- realtime web-карты. Карты в реальном времени показывают ситуацию с различными явлениями. Обычно они анимированные. Данные собираются датчиками, а карты создаются или обновляются через определенные промежутки времени или по запросу;
- общественные web-карты (Collaborative web maps). Пользователи совместно работают над созданием и улучшением web-картографии;
- онлайн-атласы;
- статические web-карты. Эти файлы создаются один раз, часто вручную и нечасто обновляются.

1.5. ПРЕИМУЩЕСТВА WEB-КАРТ

По сравнению с бумажными картами web-карты имеют следующие преимущества [9]:

- актуальность информации. Если карты создаются автоматически из баз данных, они могут отображать информацию практически в реальном времени. Их не нужно печатать, осваивать и распространять;
- низкая стоимость. Программная и аппаратная инфраструктура для веб-карт стоит недорого. Аппаратное обеспечение веб-сервера дёшево, существует множество инструментов с открытым исходным кодом для создания веб-карт;

– лёгкость распространения обновлений. Обновления продукта можно легко распространять, поскольку веб-карты распределяют как логику, так и данные при каждом запросе или загрузке;
– лёгкость объединения распределённых источников данных. Используя открытые стандарты и документированные API, можно интегрировать различные источники данных, если система проекции, масштаб карты и качество данных совпадают;
– персонализация. Используя профили пользователей, персональные фильтры и персональные стили и символы, пользователи могут настраивать и создавать свои собственные карты, если системы веб-картографии поддерживают персонализацию;
– обеспечение совместного картографирования, за счёт технологий DHTML / Ajax, SVG, Java, Adobe Flash и т. д., что позволяет получать распределённые данные и организовать коллективную работу. Примерами таких проектов являются проект OpenStreetMap или сообщество Google Earth;
– поддержка гиперссылок на другую информацию в Интернете. Как и любая другая веб-страница или вики, веб-карты могут действовать как указатель другой информации в Интернете.

1.6. ВЫВОД

В данной главе мы рассмотрели литературу, посвящённую интерактивным картам, ГИС и немного углубились в web-представление конечного продукта. Также стоит отметить, что история создания web-картографии довольно обширна и web-карты давно не являются новшеством.

2. АНАЛИЗ АНАЛОГОВ И ПОСТАНОВКА ЗАДАЧИ

2.1. ПРИМЕРЫ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Чтобы иметь более полное представление о предмете исследования, стоит рассмотреть некоторые уже существующие аналоги web-карт.

2.1.1. ЯНДЕКС.КАРТЫ

Яндекс.Карты - это картографическая служба компании Яндекс. Представлена обществу в 2004 году и является самой часто используемой в СНГ. Из основного функционала есть: поиск по карте, загруженность дорог, отслеживание городского транспорта, построение маршрутов и панорамы улиц крупных городов.

Для разработчиков имеется API Яндекс.Карты, который предоставляется бесплатно, если не превышать лимита запросов. Он позволяет наносить на карту свои маркеры и маршруты. Также благодаря ему можно разместить виджет на своём сайте. Интерфейс и визуальный вид самой карты представлен на рисунке 2.

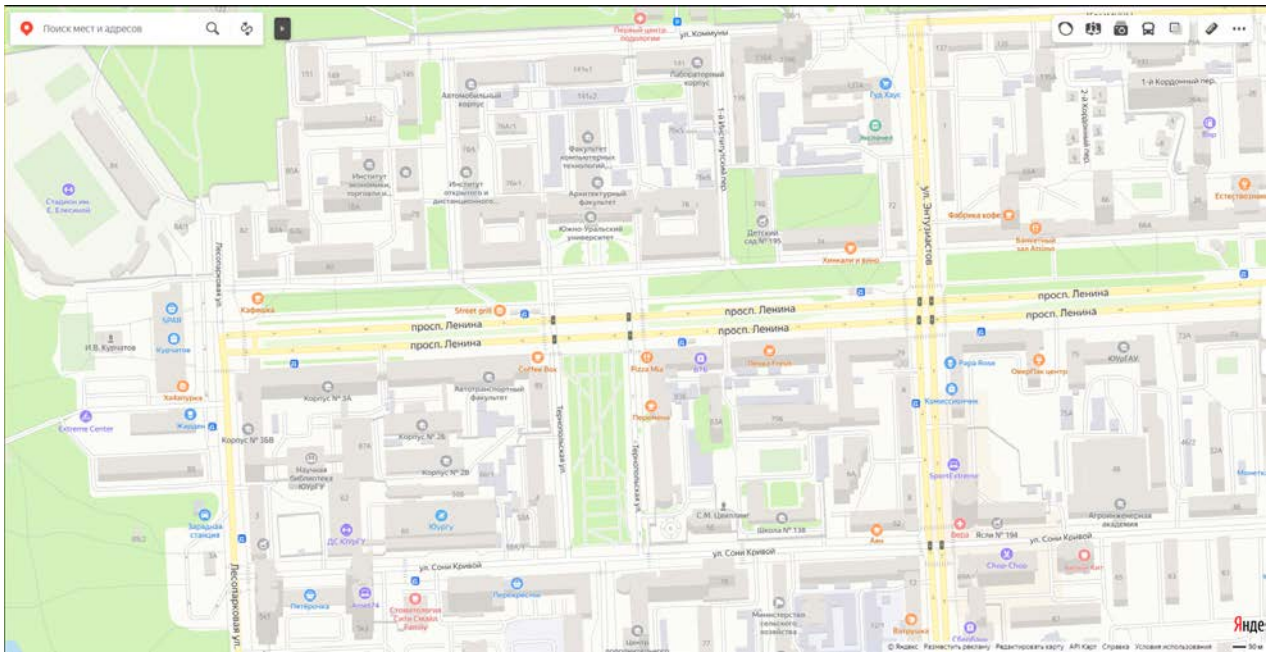

Рисунок 2 - Яндекс.Карты

2.1.2. GOOGLE MAPS

Google Maps – набор приложений, построенных на основе бесплатного картографического сервиса и технологии, предоставляемых компанией Google. Созданы в 2005 году. Сервис подставляет собой карту, построенную на основе спутниковых снимков Земли. Также в него интегрирована карта дорог с поиском маршрута (рисунок 3). Google Maps тесно связан с проектом Google Earth. Отличительной чертой от других картографических сервисов является Google Street View, который позволяет «прогуляться» по любым улицам городов.

Также, как и у Яндекс, Google предоставляет API Google Maps для разработчиков. В его основные возможности входит: построение маршрутов, описание зданий и работа Google Maps под мобильные устройства. Конечно, API Google Maps имеет ограничения для бесплатного использования.

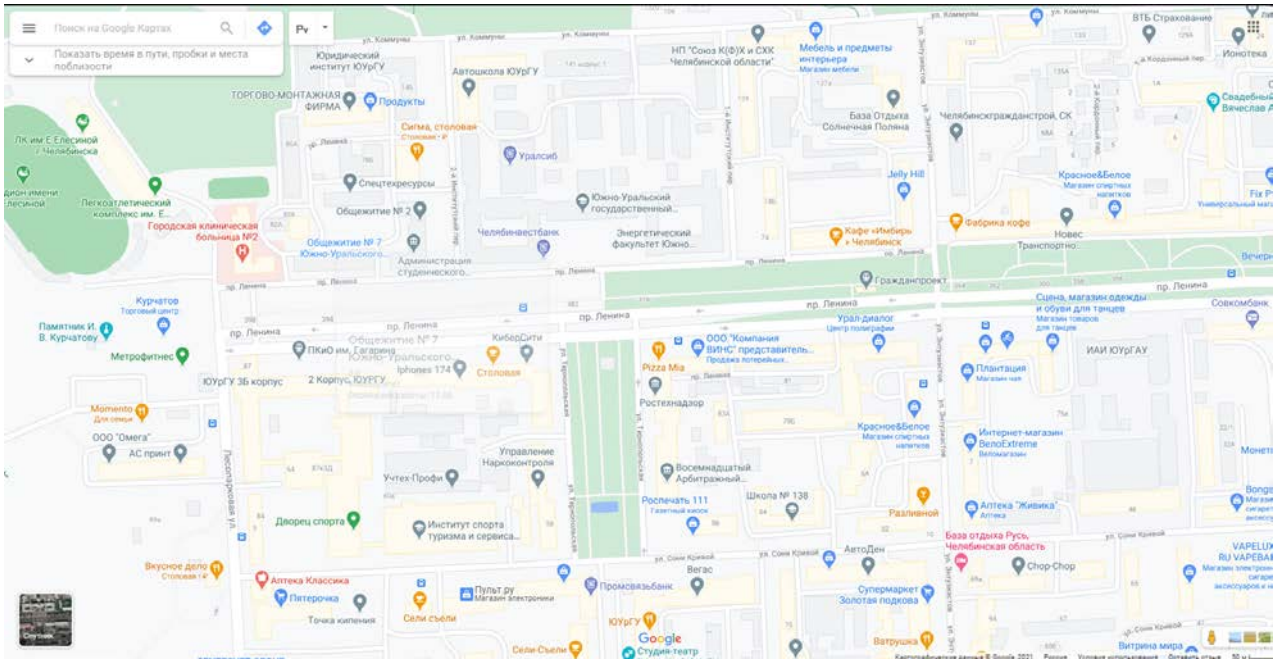

Рисунок 3 - Google Maps

2.1.3. OPENSTREETMAP

OpenStreetMap – некоммерческий web-картографический проект по созданию силами сообщества подробной и бесплатной карты мира, созданный Стивом Костом в 2004 году. В OSM создание карты происходит по принципу википедии, то есть каждый зарегистрированный пользователь может вносить свои изменения на карту. Сам по себе OSM это не карты, а базы данных, содержащие сведения о точках на земном шаре. Благодаря такому устройству OSM может выглядеть как угодно и иметь любой функционал картографического сервиса.

Так как OSM позиционирует себя проектом общественности, внести в него свой вклад можно как в личном кабинете, так и с использованием специальных инструментов. Для разработчиков существует несколько вариантов создания карты для личного или коммерческого пользования. Как и у остальных картографических сервисов, представленных выше, OSM имеет свои библиотеки использования данных. Рассмотрим только 2 самых популярных: Leaflet и OpenLayers.

Leaflet появилась в 2011 году под авторством Владимира Агафонкина. Она представляет собой библиотеку с открытым исходным кодом, написанной на JavaScript, и её главное назначение – отображение карт на web-сайтах. По заявлению автора, он разрабатывал эту библиотеку для простоты использования геоданных без погружения в ГИС.

Основные функции Leaflet: построение маршрута, создание маркера, изменение тайлов на карте и многое другое. Конечно, с помощью этой библиотеки можно создавать свои карты и использовать кем-то созданные.

OpenLayers как и Leaflet – библиотека с открытым исходным кодом, написанная на JavaScript. Её разработка началась в 2005 году, однако только в ноябре 2007 года она увидела свет. Большое время разработки связанно с тем,

что группа разработчиков создала более 150 классов и проверяла качество их на более 1600 модульных тестов. Основные функции OpenLayers: изменение тайлов карты, создание маршрутов, подсчёт расстояния, и т. п. Широкий спектр функционала позволяет работать с картой как угодно. Один из слоёв и графический интерфейс представлен на рисунке 4.

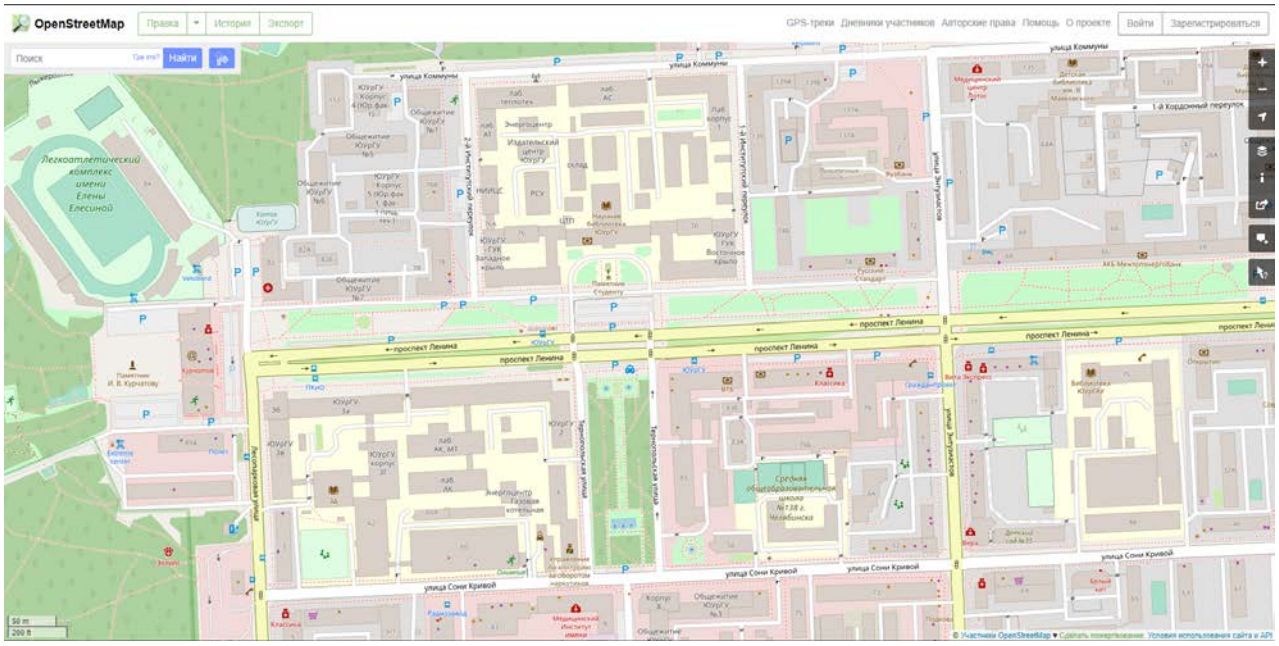

Рисунок 4 - OpenStreetMap

2.1.4. BING MAPS

Bing Maps – это картографический сервис от компании Microsoft. Он появился в 2005 году и уже к 2012 году входил в тройку самых популярных картографических сервисов США, после Google Maps и MapQuest, по данным comScore.

Компания Microsoft предоставляет API для разработчиков. С его помощью можно: изменять представление карты посредством выделения областей, устанавливать маркеры, прокладывать маршрут, и т. д. Визуальный вид представлен на рисунке 5.

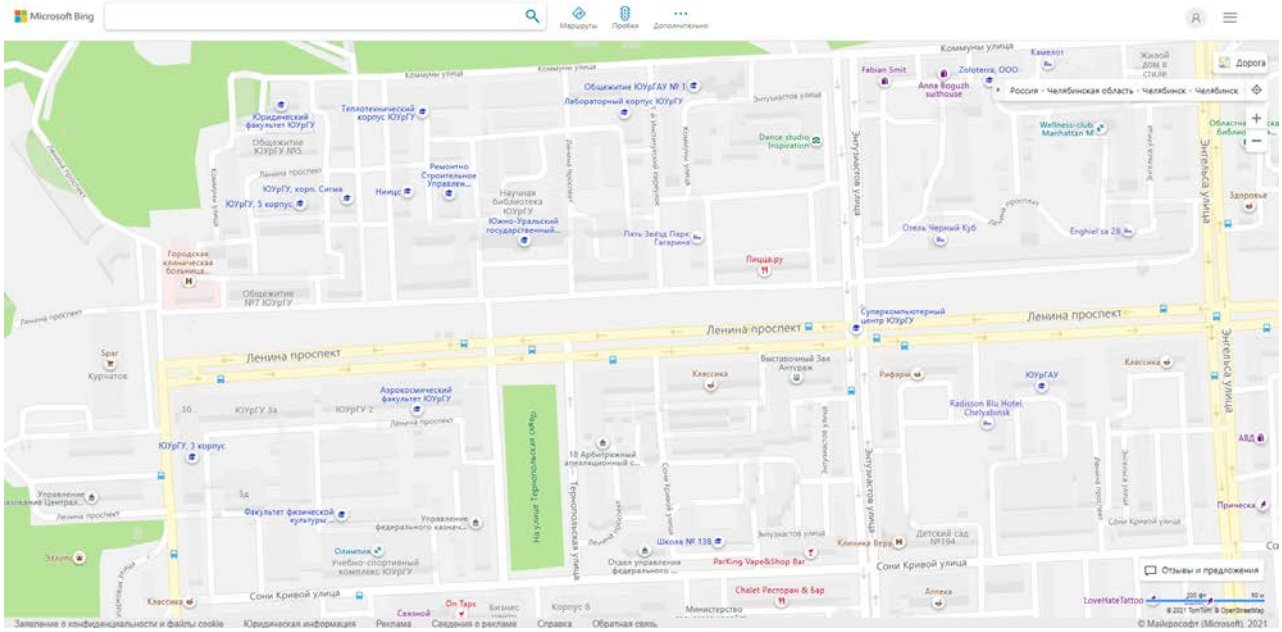

Рисунок 5 - Bing Maps

2.1.5. MAPQUEST

MapQuest – это американский картографический сервис от компании AOL. Сайт был открыт в общий доступ в 1996 году, однако большую популярность сыскал только в начале 2000-х годах. В его возможности входят: просмотр карты местности как с помощью фотографий со спутника, так и с помощью построенных карт векторным способом; установка маркера по координатам; отслеживание пробок (только в США).

Для разработчиков существует API как для web-сайтов, так и для мобильных устройств. Минимальный функционал этого API состоит из основных возможностей: установка маркера, с подробной информацией, построение маршрута, определение области поиска и т. д. Навигационные кнопки и визуальное оформление карты представлено на рисунке 6.

Рисунок 6 - MapQuest

2.2. СРАВНЕНИЕ КАРТОГРАФИЧЕСКИХ СЕРВИСОВ

Мы рассмотрели популярные сервисы web-карт и теперь стоит провести сравнение по некоторым критериям в таблице №2-4, а именно:

- покрытие;
- детализация карт, качество прорисовки;
- построение маршрутов;
- режимы отображения карты;
- условия использования API;
- ограничения количества запросов при бесплатном использовании API;
- документация по использованию API;
- элементы управления;
- средства для вывода большого количества данных.

Таблица 2 - Сравнение web-картографических сервисов Яндекс.Карты и Google Maps

Критерий	Яндекс.Карты	Google Maps
Покрытие	Карта всего мира (но наиболее проработаны карты России, Украины, Белоруссии и Казахстана, а также Европы и Северной Америки)	Карта всего мира (но хорошо прорисованы только наиболее крупные города Северной Америки, Европы, России и др.)
Детализация карт, качество прорисовки	Хорошая или очень хорошая детализация в России, достаточная в других странах.	Средний уровень детализации. Много объектов отсутствует. Объекты хорошо видны только при сильном приближении.
Построение маршрутов	Построение нескольких вариантов маршрута на автомобиле (с учетом пробок), общественным транспортом, пешком. Расчёт предположительного времени в пути. Проигрывает Google.Maps в качестве построения маршрута.	Построение нескольких вариантов маршрута на автомобиле (с учетом пробок), общественным транспортом, пешком, на велосипеде и даже самолетом. Расчёт предположительного времени в пути.
Режимы отображения карты	Режимы «Схема», «Спутник», «Гибрид», панорамы некоторых городов	Режимы «Схема» и «Спутник», панорамы отдельных городов

Продолжение таблицы 2

Критерий	Яндекс.Карты	Google Maps
Условия использования API	Бесплатно для использования в открытых некоммерческих неигровых проектах, не предназначенных для мониторинга и диспетчеризации. Использование ключа и регистрация не обязательна.	Бесплатно для использования в открытых некоммерческих проектах, не предназначенных для мониторинга, диспетчеризации, ведения незаконной деятельности. Обязательна регистрация и получение ключа API.
Ограничения количества запросов при бесплатном использовании API	Число запросов к сервисам геокодирования, маршрутизации и панорам Яндекса не должно превышать 25 000 в сутки.	Число загрузок карт не должно превышать превышает 25 000 в сутки.
Документация по использованию API	Документация очень подробная, с примерами использования большинства функций.	Документация достаточно подробная, но частично на английском языке.
Элементы управления	Элементы для перетягивания карты, увеличения выделенной области, измерения расстояний. Элемент изменения масштаба Переключатель типа карты Масштабная линейка Обзорная карта Поиск по карте Пробки Редактор маршрута	Масштабирование карты Выбор типа карты Элемент управления Street View Элемент управления Rotate для наклона и вращения Элемент перехода в полноэкранный режим Построение маршрутов Пользоват. элементы управления

	Пользовательские элементы управления	

Таблица 3 - Сравнение web-картографических сервисов OpenStreetMap и Bing Maps

Критерий	OpenStreetMap	Bing Maps
Покрытие	Карта всего мира	Карта всего мира (но хорошо прорисованы только наиболее крупные города Северной Америки, Европы, и др.)
Детализация карт, качество прорисовки	Хорошая или очень хорошая детализация за счёт пользователей.	Средний уровень детализации. Много объектов отсутствует. Объекты хорошо видны только при сильном приближении.
Построение маршрутов	Построение нескольких вариантов маршрута на автомобиле (без учёта пробок), пешком и на велосипеде. Все маршруты рассчитаны по наиболее посещаемым местам пользователей.	Построение нескольких вариантов маршрута на автомобиле (без учёта пробок) и пешком.
Режимы отображения карты	Режимы «Стандарт», «CyclOSM», «Велосипедная карта», «Карта транспорта», «ÖPNVKarte», «Гуманитарная»	Режимы «Дорога» и «Гибридный вид»
Условия использования API	Бесплатно для использования в открытых некоммерческих неигровых проектах, не предназначенных для мониторинга и диспетчеризации. Обязательна регистрация и получение ключа API	Бесплатно для использования в открытых некоммерческих проектах, не предназначенных для мониторинга, диспетчеризации, ведения незаконной деятельности. Обязательна регистрация и

		получение ключа API. Так же платно.

Продолжение таблицы 3

Критерий	OpenStreetMap	Bing Maps
Ограничения количества запросов при бесплатном использовании API	Число запросов к сервисам не ограниченно.	Число загрузок карт не должно превышать превышает 50 000 в сутки. Только на web сайт в качестве виджета.
Документация по использованию API	Документация очень подробная, с примерами использования большинства функций. Вики, официальные форумы.	Документация достаточно подробная, но полностью на английском языке.
Элементы управления	Элементы для перетягивания карты, увеличения выделенной области, измерения расстояний. Элемент изменения масштаба Переключатель типа карты Обзорная карта Поиск по карте Редактор маршрута Пользовательские элементы управления	Масштабирование карты Выбор типа карты Построение маршрутов Пользоват. элементы управления

Таблица 4 - Сравнение web-картографических сервисов MapQuest

Критерий	MapQuest
Покрытие	США
Детализация карт, качество прорисовки	Хорошая.
Построение маршрутов	Построение маршрута на автомобиле (с учётом пробок, и расчёт потраченного бензина).
Режимы отображения карты	Режимы «Map», «Setelite»
Условия использования API	Бесплатно для использования в открытых некоммерческих неигровых проектах, не предназначенных для мониторинга и диспетчеризации. Обязательна регистрация и получение ключа API
Ограничения количества запросов при бесплатном использовании API	Число запросов к сервисам 15000 в месяц.
Документация по использованию API	Документация очень подробная, с примерами использования большинства функций. Полностью на английском.
Элементы управления	Элементы для перетягивания карты, увеличения выделенной области, измерения расстояний. Элемент изменения масштаба Переключатель типа карты Поиск по карте (только заранее подготовленные группы: отели, заправки и т. д.) Редактор маршрута

2.2. ВЫВОД

Подводя итоги, можно сказать, что каждая карта обладает своими отличительными качествами. Однако в рамках данной работы нас интересует прежде всего разработка web-карты на основе одного из этих web-сервисов. Единственный web-сервис без ограничений на количество запросов – это OSM. А самое малое количество запросов предоставляет MapQuest, всего 15000 в месяц. Также про MapQuest можно сразу сказать, что этот сервис в дальнейшей разработке не пригодится, так как его данные ограничены только

картой США и функционал, который он предоставляет, проигрывает всем остальным сервисам.

Что до остальных карт, то стоит обратить внимание на другие критерии, кроме как количество запросов, так как они примерно одинаковые. Яндекс.Карты имеют лучшее покрытие РФ в отличие Google Maps и Bing Maps. В качестве детализации Bing Maps проигрывает и Яндекс.Карты и Google Maps. Остальные параметры сравнения Яндекс.Карты и Google Maps идентичные. Следовательно, Яндекс.Карты подойдут лучше, чем Google Maps.

Среди оставшихся web-сервисов OSM и Яндекс.Карты. Не мало важную роль играет документация к API. У Яндекс.Карты документация на русском языке, однако не мало пользователи оставили плохие отзывы о работе с этим web-сервисом. В основном проблемы выделяли в урезанном или недокументированном функционале. У OSM по мимо официальной документации, существует OSM вики и официальный форму, где разработчики делятся своим опытом использования API.

Из всего выше сказанного можно сделать вывод, что лучше всего нам подойдёт web-сервис OSM.

3. ПЛАНИРОВАНИЕ

В соответствие с заданием приложение должно быть кроссплатформенным, не имеет значения на какой операционной системе будет исполнятся исходный код проекта. Поэтому выбор операционной системы не имеет смысла.

3.1. ОБОСНОВАНИЕ ВЫБОРА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Исходя из задания работы мы знаем, что язык программирования должен иметь возможность реализовывать web-интерфейс, в частности web-приложение.

Разработку web-приложения разделяют на 2 сферы, которые появились вследствие развития принципа разделения ответственности. Backend отвечает за внутреннее функционирование сервиса, а Frontend отвечает за клиентскую сторону пользовательского интерфейса.

Frontend имеет ряд своих языков, а именно CSS, JavaScript, HTML. Всё визуальное преставление строится на них.

Язык Backend может быть самым разнообразным, так как нет привязки к работе с браузером. Он зависит от сервера, системы управления базами данных и предпочтений разработчика.

Следовательно, потребуется выбрать один язык из следующего набора: PHP, Python, Ruby, C#, Java, VB.

Чтобы выбрать один из представленных языков, проведём сравнение и найдём оптимальный.

3.1.1. PHP

PHP [10] – это скриптовый язык, общего назначения. На данный момент большинство web-сайтов написаны именно на нём. Он сыскал огромную

популярность благодаря наличию большого количества встраиваемых моделей для разработки web-приложений.

Вот небольшой перечень основных:

- автоматическое извлечение POST- и GET-параметров [11], а также переменных окружения веб-сервера в предопределённые массивы;
- взаимодействие с большим количеством различных систем управления базами данных через дополнительные модули (MySQL, MySQLi, SQLite, PostgreSQL, Oracle Database (OCI8), Microsoft SQL Server, Sybase, ODBC, mSQL, IBM DB2, Cloudscape и Apache Derby, Informix, Ovrimos SQL, Lotus Notes, DB++, DBM, dBase, DBX, FrontBase, FilePro, Ingres II, SESAM, Firebird и InterBase, Paradox File Access, MaxDB, интерфейс PDO, Redis);
- автоматизированная отправка HTTP-заголовков;
- работа с HTTP-авторизацией;
- работа с cookies и сессиями;
- работа с локальными и удалёнными файлами, сокетами;
- обработка файлов, загружаемых на сервер;
- работа с XForms.

3.1.2. RUBY

Ruby [12] выделяется среди остальных языков программирования набором особенностей, а именно:

- динамичность – свойство, позволяющее определять типы данных и осуществлять синтаксический анализ и компиляцию кода на этапе выполнения программы;

– рефлективность – свойство, реализующее процесс рефлексии, программа способна самостоятельно отслеживать и модифицировать структуру программы во время выполнения;
– интерпретируемость – свойство, определяющее построчный синтаксический анализ, обработку и выполнение исходного кода во время работы программы;
– высокоуровневость – свойство, которое упрощает и ускоряет использование исходного кода программистом, но замедляет скорость выполнения программы на «железе».

Также нельзя не отметить, что интерпретатор Ruby является кроссплатформенным, что позволяет не привязываться к операционной системе. Ему присуща многопоточная обработка, а так как выше мы уже отметили, что он кроссплатформенный, то реализация многопоточности не зависима от операционной системы.

3.1.3. PYTHON

Python обладает тем же преимуществами, что и Ruby, но он больше ориентирован на производительность разработчика, чем остальные языки программирования, а также является полностью объектно-ориентрованным языком программировании (ООП – [13]). Все его преимущества для использования программистом обернулись малой скоростью выполнения программы и не оптимизированным расходом памяти.

3.1.4. VISUAL BASIC

Visual Basic [14] позволяет работать с достаточно сложными элементами интерфейса пользователя, библиотеками кода (например, COM-серверами) и средствами доступа к данным при минимальных затратах времени и сил. Visual Basic в гораздо большей степени, чем MFC прячет от пользователя

вызовы Win32 API и предоставляет большой набор интегрированных средств быстрой разработки.

Однако у Visual Basic есть и недостатки. Главный из них – это гораздо меньшие возможности, которые предоставляет этот язык по сравнению с C++ [15] (это утверждение справедливо, по крайней мере, для версий более ранних, чем VB.NET).

Visual Basic – это язык для работы с объектами, а не объектно-ориентированный язык в обычном понимании этого слова. В Visual Basic нет классического наследования, нет поддержки создания параметризованных классов, нет собственных средств создания многопоточных приложений. И этот список можно продолжать ещё долго.

3.1.5. JAVA

Язык программирования Java [16] – это полностью объектно-ориентированный язык, который в отношении синтаксиса многое унаследовал от C++. Язык Java в синтаксическом отношении проще и логичнее, чем C++. Java как платформа предоставляет в распоряжение программистов большое количество библиотек (пакетов), в которых содержится большое количество описаний классов и интерфейсов на все случаи жизни. С их помощью можно создавать приложения с возможностью обращения к базам данных, поддержкой передачи почтовых сообщений, с клиентской частью, которой необходим только web-браузер, или наоборот, с клиентской частью, обладающей изощрённым интерфейсом.

Одна из серьёзных проблем языка заключается в том, что при создании сложного приложения на Java придётся использовать только этот язык для создания всех частей этого приложения. В Java предусмотрено не очень много средств для межъязыкового взаимодействия.

3.1.6. C# И ПЛАТФОРМА .NET CORE

C# первый из рассмотренных языков программирования, который является компилируемым. Он относится к семейству C-подобных языков, а, следовательно, как и его «предки» имеет статическую типизацию, возможность оптимизировать память, посредством размещения в памяти данных в строгом порядке, как и C++ является высокоуровневым и объектно-ориентрованным.

На момент создания «Карта помещений ЮУрГУ» платформа .NET Core [17] и программирование на C# [17] уже представляли собой заметное явление в мире программирования. Платформа очень развита для программирования, однако, в отличие от своего предшественника .NET [18], все программы, написанные на .NET Core, являются кроссплатформенными.

.NET Core представляет собой совершенно новый способ создания распределённых, настольных и встроенных приложений. Очень важно сразу отметить, что .NET Core не имеет ничего общего с COM (кроме мощных средств интеграции двух платформ). Для типов .NET Core не нужны ни фабрики классов, ни регистрация в системном реестре. Эти основные элементы COM не скрыты – их просто больше нет.

Специально для новой платформы Microsoft разработала новый язык программирования - C# (Си Шарп). Этот язык, как и Java, очень многое позаимствовал из C++ (особенно с точки зрения синтаксиса). Однако на C# сильно повлиял и Visual Basic 6.0.

В целом можно сказать, что C# впитал в себя многое из того лучшего, что есть в самых разных языках программирования, и если у вас есть опыт работы с C++, Java или Visual Basic, то вы найдёте в C# много знакомого.

Очень важно отметить, что платформа .NET Core является полностью независимой от используемых языков программирования. Можно

использовать несколько .NET Core-совместимых языков программирования даже в рамках одного проекта. Разобраться с самим языком C# достаточно просто. Наибольшие усилия потребуются, чтобы познакомиться с многочисленными пространствами имён и типами библиотеки базовых классов .NET Core. С этими типами (как и со своими собственными, созданными, например, на C#) можно работать из любого .NET Core-совместимого языка.

Из всей платформы стоит выделить ASP.NET Core [19] – это главный фреймворк для разработки web-приложений на языке C#.

3.1.7. ВЫВОД

С учётом вышеизложенного, можно сделать вывод, что наиболее подходящим для разработки является язык C# и платформа .NET Core.

3.2. СРЕДА РАЗРАБОТКИ

Для разработки «Карта помещений ЮУрГУ» использовалась среда Microsoft Visual Studio Enterprise 2019. Visual Studio 2019 – это универсальное решение от компании Microsoft. Среда поддерживает множество шаблонов для создания различных проектов. Возможность создавать как консольные, так и графические приложения.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на

каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

3.3. ПЛАТФОРМА И СТОРОННИЕ БИБЛИОТЕКИ

Как уже отмечалось в пункте 3.1, разработка web-приложения состоит из backend и frontend-частей. Для backend был выбран язык программирования C# в пункте 3.1.7. Однако на чистом C# создание web-приложений не представляется возможным, поэтому компания Microsoft разработала платформу .NET.Будет использоваться новая версия этой платформы – .NET Core, что позволит получить на выходе кроссплатформенное приложение.

Среди компонентов платформы .NET Core для работы с web существует ASP.NET Core. Microsoft для удобства работы с базой данных добавила в платформу .NET Core фреймворк Entity Framework Core как часть ADO.NET Core, который позволяет с помощью LINQ (Language Integrated Query – внутренний язык запросов в реализованный на C#) формировать SQL запросы и работать со всем содержимым базы данных напрямую.

По данным с сайта techempower.com [20], среди всех популярных web фреймворков ASP.NET Core в синтетических тестах занимает почётное 2 место, с результатом обработки 7,01 миллиона запросов в секунду.

Среди преимуществ стоит отметить, что распространение платформы ASP.NET Core происходит по варианту Open source, то есть полностью бесплатно, а также весь исходный код находится в доступе сообщества.

Web-приложение «Карта помещений ЮУрГУ» создаётся для массового пользования, и его дальше развитие, которое выходит за рамки данной выпускной квалификационной работы, имеет большое значение. Как известно, для масштабирования подобных проектов [21] принято разделять блоки различных сервисов на составляющие, и эти составляющие могут выступать самыми разными. Web-приложение реализованное на основе платформы ASP.NET, можно развернуть где угодно, что упрощает процесс масштабирования.

Entity Framework Core – это кроссплатформенная версия Entity Framework .NET, которая является ORM (Object-Relational Mapping – технология программирования, связывающая БД с концепциями ООП). Она поддерживает запросы LINQ, отслеживание изменений, обновления и миграции схем данных. EF Core работает со многими СУБД SQLite, MySQL, PostgreSQL, MS SQL Server.

В второй главе были рассмотрены в кратком формате библиотеки для работы с OSM, теперь рассмотрим их с технической стороны.

Leaflet позволяет разработчику, не знакомому с ГИС, легко отображать растровые карты, состоящие из маленьких фрагментов – тайлов, с, возможно, дополнительными слоями, накладываемыми поверх основного. Слои могут быть интерактивными, например, отображать подсказку при клике по маркеру.

Leaflet поддерживает слои Web Map Service (WMS), GeoJSON, векторные и тайловые слои. Многие другие типы слоёв поддерживаются дополнительными модулями.

Как и в других картографических веб-библиотеках, в Leaflet реализована следующая модель: отображается базовая карта с, возможно, растровыми и векторными слоями, накладываемыми поверх неё.

Также Leaflet сравнивают с проприетарной закрытой Google Maps API (впервые вышла в 2005) и Bing Maps API – они обе используют значительную часть на стороне сервера для предоставления таких услуг, как геокодировние, прокладка маршрутов, поиск и интеграция с дополнительным ПО, таким как Google Earth. Google Maps API даёт скорость и простоту вместе с гибкостью, однако предоставляет доступ только к сервисам Google Maps. Впрочем, DataLayer – часть Google’s API – позволяет использовать внешние данные.

OpenLayers позволяет очень быстро и легко создать web-интерфейс для отображения картографических материалов, представленных в различных форматах и расположенных на различных серверах. Благодаря OpenLayers разработчик имеет возможность создать, к примеру, собственную карту, включающую слои, предоставляемые WMS (и WFS) серверами, такими как Mapserver, ArcIMS или Geoserver, и данными картографических сервисов Google. Библиотека является разработкой с открытым исходным кодом и разрабатывается при спонсорской поддержке проекта MetaCarta, который использует OpenLayers в своих разработках. Тем не менее, OpenLayers является независимым свободно распространяемым продуктом.

Эти платформы часто сравнивают. Обе являются открытым ПО, обе – клиентские библиотека на JavaScript. Leaflet заметно компактнее, содержит около 7 тысяч строк против 230 тысяч у OpenLayers по состоянию на 2021. Leaflet занимает меньше места, чем OpenLayers (около 123 кБ против 423), не в последнюю очередь благодаря модульной структуре. Код новее и использует свежие возможности JavaScript, HTML5 и CSS3.

Нашим выбором будет JavaScript библиотека Leaflet. Одним из преимуществ является то, что Leaflet гораздо более простая и легкая

библиотека (около 123 кБ против 423). В самой идее Leaflet заложена простота. Она содержит только самые необходимые функции. Остальное легко подключается плагинами, которых на данный момент великое множество и при должном желании и навыке каждый может написать плагин и опубликовать его для того, чтобы облегчить другим жизнь и развить библиотеку. Для представления данных выбран формат данных GeoJSON, который позволяет представить наше здание как набор геометрических фигур, которые будут описаны с помощью географических координат и иметь различные свойства для дальнейшей обработки их в нашем web-приложении. Такой формат позволит также легко хранить свойства помещений и корпусов для дальнейшего развития системы. Он поддерживается множеством систем и сервисов, а также легко хранится в базе данных [22].

3.4. ВЫБОР СУБД

Выбор СУБД будет производить из перечня, который указали выше [23].

3.4.1. SQLITE [24]

SQLite – это компактная встраиваемая СУБД, распространяемая в формате Open source. Одна из лучших СУБД, если требуется развернуть базу данных быстро и внести изменения, так как она встраиваемая. Термин «встраиваемая» означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы. Однако из-за своей архитектуры SQLite имеет всего 4 типа данных, а именно: INTEGER; REAL; TEXT; BLOB, что значительно

усложняет работу с большими и сложными базами. Также количественные ограничения SQLite не удовлетворяют требованиям данной работы.

3.4.2. MYSQL [25]

MySQL – свободная реляционная система управления базами данных от корпорация Oracle. Продукт распространяется как под GNU (General Public License), так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей.

Достоинства:

- распространяется бесплатно;
- прекрасно документирована;
- предлагает много функций, даже в бесплатной версии;
- пакет MySQL включён в стандартные репозитории наиболее распространённых дистрибутивов операционной системы Linux, что позволяет устанавливать её элементарно;
- поддерживает набор пользовательских интерфейсов;
- может работать с другими СУБД, включая DB2 и Oracle.

Недостатки:

- придётся потратить много времени и усилий, чтобы заставить MySQL выполнять несложные задачи, хотя другие системы делают это автоматически, например: создание инкрементных резервных копий;
- для бесплатной версии доступна только платная поддержка.

3.4.3. POSTGRESQL [26]

PostgreSQL является одним из нескольких бесплатных популярных вариантов СУБД, часто используется для ведения баз данных веб-сайтов. Это

была одна из первых разработанных систем управления базами данных, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использован, на большинстве основных платформ, включая Linux. Прекрасно справляется с задачами импорта информации из других типов баз данных с помощью собственного инструментария.

Достоинства:

- является масштабируемым и способен обрабатывать терабайты данных;
- поддерживает формат json;
- существует множество предопределённых функций.

Недостатки:

- документация туманна, поэтому, возможно, ответы на некоторые вопросы придётся искать в интернете;
- конфигурация может смутить неподготовленного пользователя;
- скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения.

3.4.4. MS SQL SERVER [27]

Ещё одной из популярных СУБД является программный продукт Microsoft SQL-сервер. Это система управления базами данных, движок которой работает на облачных серверах, а также локальных серверах, причём можно комбинировать типы применяемых серверов одновременно. Вскоре после выпуска Microsoft SQL сервер 2016, Microsoft адаптировала продукт для операционной системы Linux, а на Windows-платформе он работал изначально.

Достоинства:

- продукт очень прост в использовании;

- текущая версия работает быстро и стабильно;
- движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов;
- можно получить доступ к визуализации на мобильных устройствах;
- очень хорошее взаимодействие с другими продуктами Microsoft;
- для физических лиц распространяется на бесплатной основе.

Недостатки:

- даже при тщательной настройке производительности корпорация SQL Server способен занять все доступные ресурсы;
- сообщается о проблемах с использованием службы интеграции для импорта файлов.

Microsoft SQL сервер является основной СУБД, на которой реализована корпоративная система ЮУрГУ Универис.

3.4.5. ВЫВОД

Из выше сказанного можно сделать вывод, что для реализации «Карта помещений ЮУрГУ» лучше всего подойдёт MS SQL Server, так как распространяется на бесплатной основе для физических лиц, может работать, что на Windows, что на Linux, является самой стабильной и производительной, принята в качестве корпоративной СУБД ЮУрГУ.

Подытоживая эту главу перечислим все выбранные продукты, технологии и общий стек. Наш главный язык программирования – это C#, платформа – ASP.NET Core, IDE – Visual Studio 19, СУБД – MS SQL Server.

4. ПРОЕКТИРОВАНИЕ

4.1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

Можно выделить два типа пользователей приложения: конечный пользователь и администратор. В рамках данной работы рассмотрены только сценарии конечного пользователя, так как на создание административных сценариев не хватило времени.

События, возникающие в системе с точки зрения конечного пользователя, изображены с помощью диаграммы использования в нотации UML (Рисунок 7).

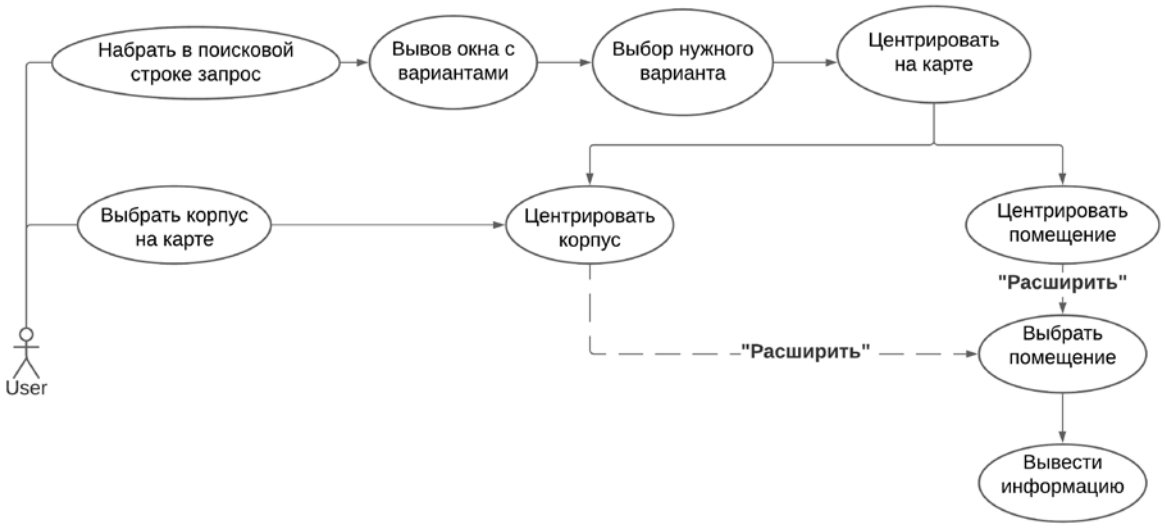

Рисунок 7 - Диаграмма использования

4.1.1. ПОИСК

Начальной точкой этого сценария является ввод значения в строку поиска пользователем. После этого система считывает строку. Далее обрабатывает её и определяет совпадения. Все совпадения выводятся пользователю, как «похожие» варианты. Пользователь выбирает вариант. Теперь, в зависимости от выбранного объекта, производится центрирование этого объекта на карте. Если было выбрано помещение, то на плане нужного

этажа центрируется помещение. Если был выбран корпус, то на глобальной карте центрируется нужный корпус.

4.1.2. РУЧНОЙ ВЫБОР

Пользователь выбирает кликом корпус. Этот корпус центрируется на карте. Далее открывается план 1 этажа (по умолчанию) этого корпуса. Пользователь может как выйти из плана этажа, так и продолжить выбор, только уже среди этажей или помещений. Если пользователь хочет найти помещение не на 1 этаже, то он может воспользоваться панелью переключения этажей. Выбрав нужный ему этаж, пользователь может сделать выбор помещения, чтобы посмотреть подробную информацию о нём.

4.2. ОСНОВНЫЕ АРХИТЕКТУРНЫЕ РЕШЕНИЯ

Архитектура web-положении очень обширная тема, мы познакомимся с ней в кратком формате с помощью документации Microsoft [28].

Есть несколько подходов к построению архитекторы web-приложения:

- монолитное приложение;
- комплексное приложение;
- слои;
- чистая архитектура.

Рассмотрим каждый подход отдельно, чтобы решить, какая архитектура подходит нам больше всего.

4.2.1. МОНОЛИТНОЕ ПРИЛОЖЕНИЕ

Монолитное приложение полностью замкнуто в контексте поведения. Во время работы оно может взаимодействовать с другими службами или хранилищами данных, однако основа его поведения реализуется в собственном процессе, а все приложение обычно развёртывается как один

элемент. Для горизонтального масштабирования такое приложение обычно целиком дублируется на нескольких серверах или виртуальных машинах.

4.2.2. КОМПЛЕКСНОЕ ПРИЛОЖЕНИЕ

Архитектура приложения содержит как минимум один проект. В таком случае вся логика приложения заключена в одном проекте, компилируется в одну сборку и развёртывается как один элемент.

В сценарии с одним проектом разделение задач реализуется с помощью папок. Используемый по умолчанию шаблон включает отдельные папки для обязанностей шаблона MVC (модели, представления и контроллеры) [29], а также дополнительные папки для данных и служб. При такой организации детали презентации данных в максимально возможной степени размещаются в папке представлений (Views), а детали реализации доступа к данным должны быть ограничены классами, содержащимися в папке данных (Data). Бизнес-логика при этом размещается в службах и классах, находящихся в папке моделей (Models). Представление в схематичном виде шаблона MVC представлено на рисунке 8.

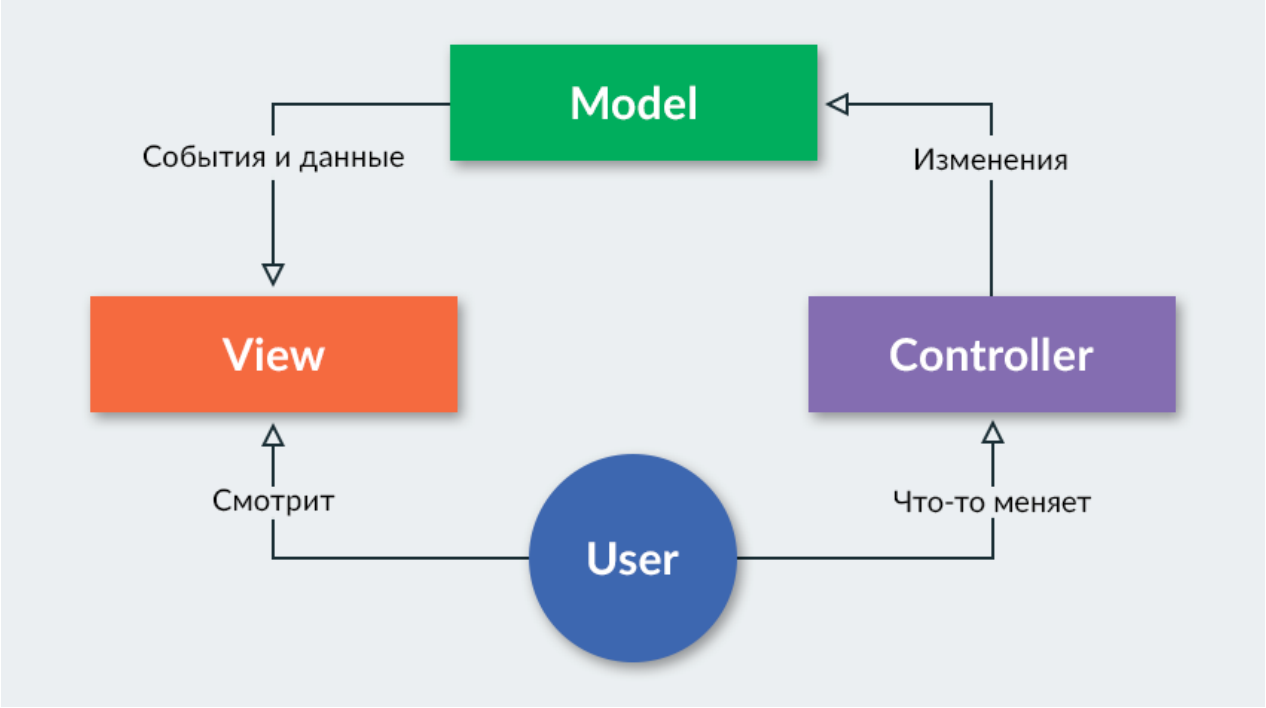

Рисунок 8 - Схема разделения ответственности MVC

4.2.3. СЛОИ

Как правило, в приложении определяются слои пользовательского интерфейса, бизнес-логики и доступа к данным. В рамках такой архитектуры пользователи выполняют запросы через слой пользовательского интерфейса, который взаимодействует только со слоем бизнес-логики. Слой бизнес-логики, в свою очередь, может вызывать слой доступа к данным для обработки запросов. Слой пользовательского интерфейса не должен выполнять запросы напрямую к слою доступа к данным и какими-либо другими способами напрямую взаимодействовать с функциями изменения данных. Аналогичным образом, слой бизнес-логики должен взаимодействовать с функциями изменения данных только через слой доступа к данным. Таким образом, для каждого слоя чётко определена своя обязанность. Общепринятая организация логики приложения по слоям показана на рисунке 9.

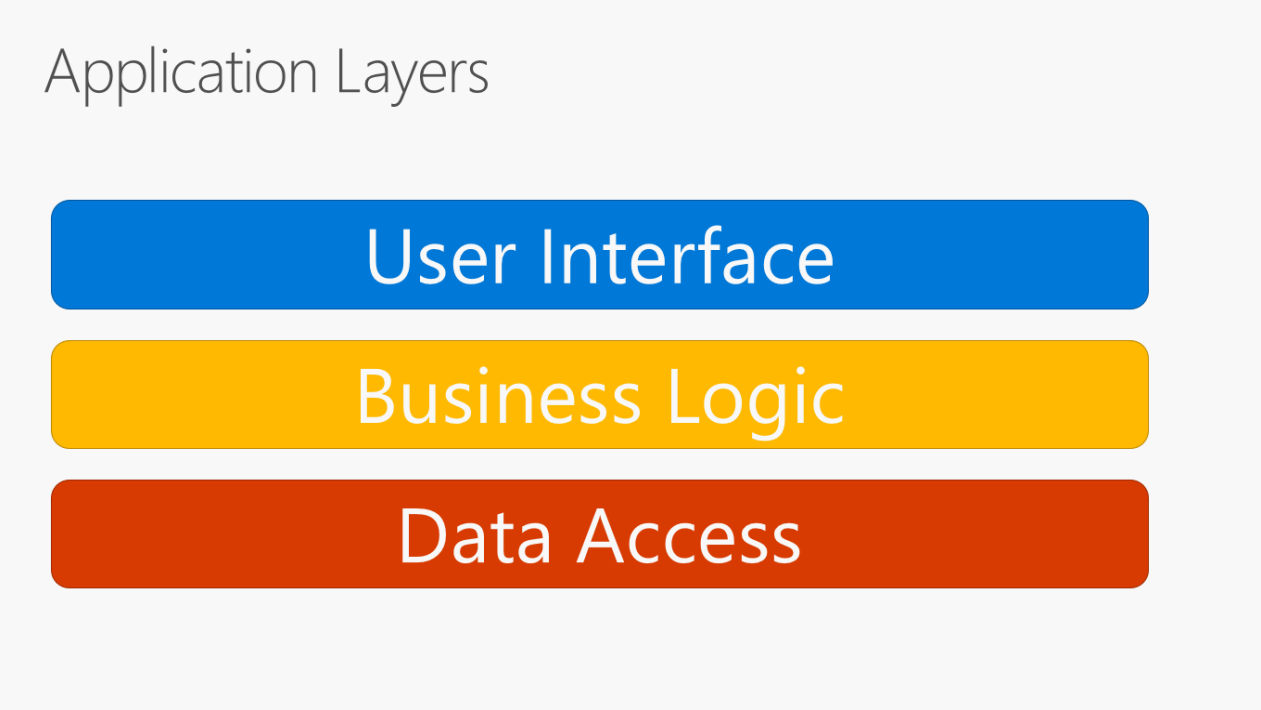

Рисунок 9 - Приложение с многослойной архитектурой

4.2.4. ЧИСТАЯ АРХИТЕКТУРА

В рамках чистой архитектуры центральным элементом приложения являются его бизнес-логика и модель. В этом случае бизнес-логика не зависит от доступа к данным или другим инфраструктурам, то есть стандартная зависимость инвертируется: инфраструктура и детали реализации зависят от ядра приложения. Эта функциональность достигается путём определения абстракций или интерфейсов в ядре приложения, которые реализуются типами, определёнными в слое инфраструктуры. Такую архитектуру обычно рисуют в виде серии окружностей с общим центром, которая внешне напоминает срез луковицы. На рисунке 10 показан пример такого стиля представления архитектуры.

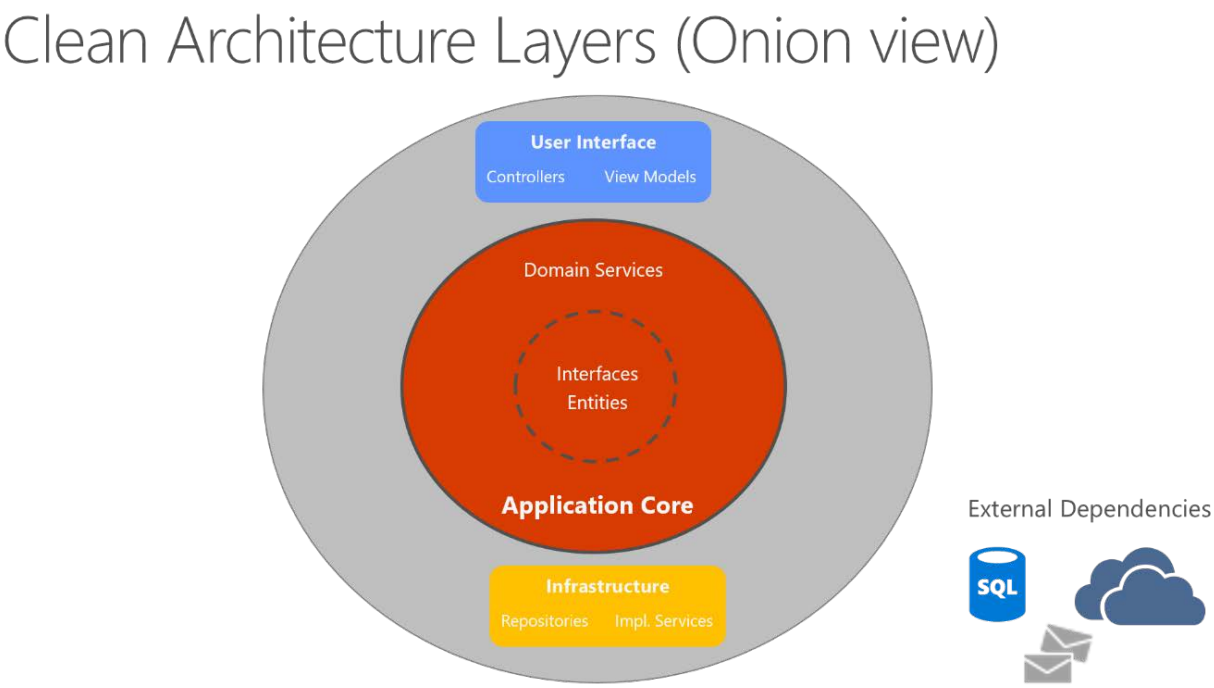

Рисунок 10 - Разделение на слои в чистой архитектуре

На этой схеме зависимости направлены из самой внутренней окружности. Ядро приложения называется так потому, что находится в самом центре этой схемы. Как видно на схеме, ядро приложения не имеет зависимостей от других слоёв приложения. Сущности и интерфейсы

приложения находятся в самом центре. Сразу после них, но все ещё в пределах ядра приложения, расположены доменные службы, которые обычно реализуют интерфейсы, определённые во внутренней окружности. За пределами ядра приложения располагаются слои пользовательского интерфейса и инфраструктуры, которые зависят от ядра приложения, но не друг от друга (обязательно).

4.2.5. ВЫБОР АРХИТЕКТУРЫ

Рассматриваемое web-приложение не будет масштабироваться за границы университета, поэтому «чистая архитектура» не имеет смысла. Разделение на слои также не требуется, потому что поток данных не имеет большого объёма, а использование нескольких серверов вместо единого помогут лишь ускорить обработку, но это будет очень дорого. Монолитное приложение не имеет гибкости в реализации, и такой архитектуре плохо даётся масштабирование. Комплексное приложение с MVC является промежуточным решением. Оно достаточно гибкое для разработки и поддержки, и в то же время не является сильно дорогим для реализации. Именно потому мы остановимся на ASP.NET MVC.

5. РЕАЛИЗАЦИЯ

5.1. БАЗА ДАННЫХ

Как мы уже отмечали в пункте 3.4.5. база данных будет создаваться и редактироваться с помощью СУБД MS SQL Server.

Требуется сформировать базу данных для проекта «Карта помещений ЮУрГУ». Эта база должна в себя включать такие таблицы, как:

- помещение;
- словарь оборудования, которое может находится в помещение;
- словарь типов помещений;
- словарь кафедр;
- координаты;
- корпуса;
- словарь типов корпусов;
- этажи;
- сотрудники;
- должности.

База должна быть гибкая к расширению, то есть помимо уже существующих категорий объектов можно создать (например, сигнальные кнопки пожарной безопасности как объект этажа)

Из всего выше перечисленного сформирована схема данных (рисунок 11):

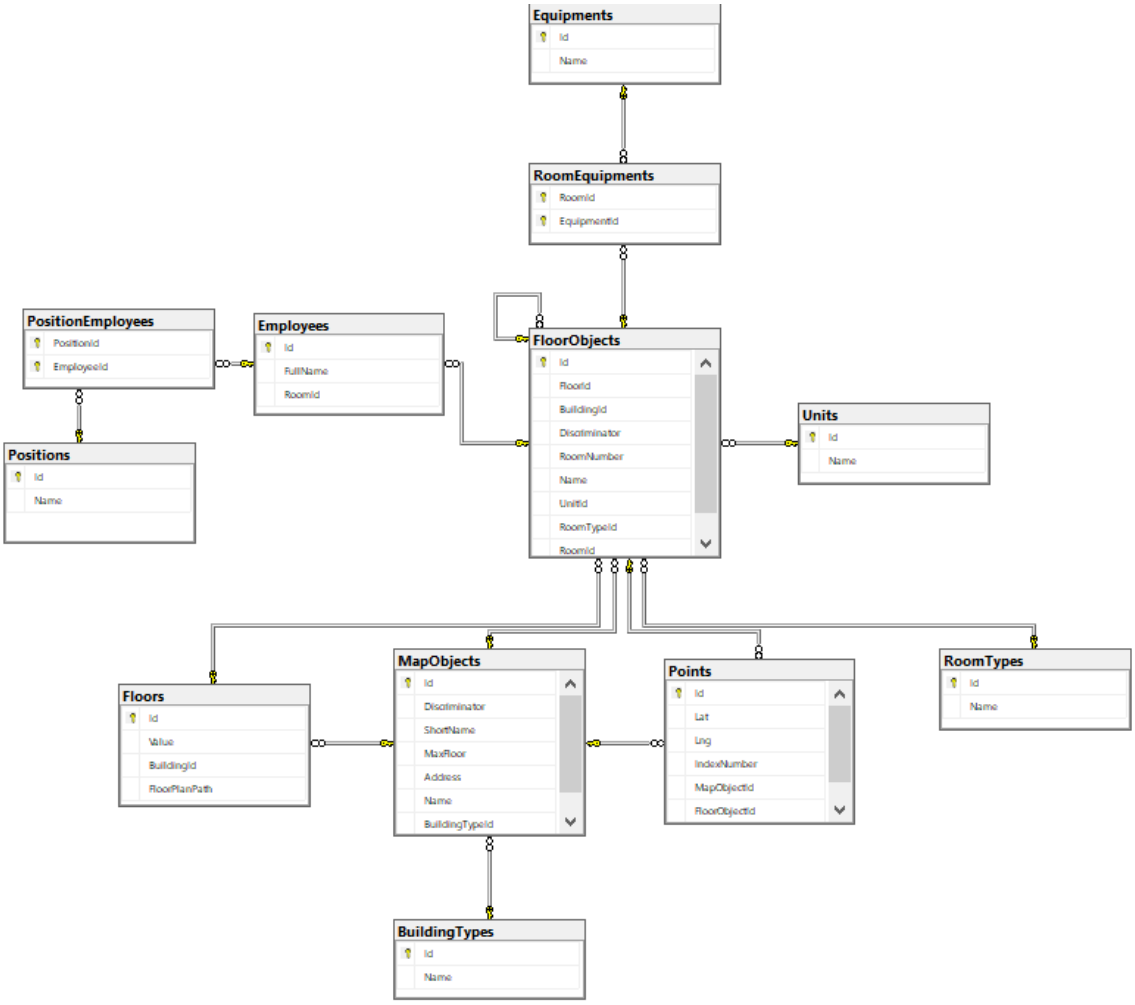

Рисунок 11 Microsoft SQL сервер – Схема данных «Карта помещений ЮУрГУ»

В итоговом варианте получились следующие таблицы:

- FloorObjects – таблица всех объектов, которые могут быть указаны на этаже (помещений, кнопки пожарной безопасности, скамейки и т. д.), определяется вид объекта по полю Discrimenftor, которое хранит в себе название типа этого объекта;
- MapObjects –таблица всех объектов, которые могу быть указаны на глобальной карте (корпуса, памятники, спортивные сооружения и т. д.), определяется вид объекта по полю Discrimenftor, которое хранит в себе название типа этого объекта;
- Points – таблица координат объектов;
- Floors – таблица этажей;
- BuildingTypes – словарь типов корпусов;

- Units – словарь подразделений (кафедры, отделы и т. д.);
- RoomTypes – словарь типов помещений;
- Employees –таблица сотрудников;
- Positions – словарь должностей;
- PositionEmployees – таблица для реализации связи многие ко многим для таблиц Positons и Employees;
- Equipments –словарь оборудования, которое может находится в помещении;
- RoomEquipments – таблица для реализации связи многие ко многим для таблиц FloorObjects и Equipments.

На рисунке 11 представлена схема данных. Для работы нужны правильные запросы. Далее представлены все запросы, которые используются в «Карта помещений ЮУрГУ»:

- SelectAllFloorObjectsWithLocations – формирование выборки всех объектов из таблицы FloorObjects с соответствующими координатами;
- SelectAllEmployeesWithRoomNames – формирование выборки всех сотрудников с соответствующими названиями помещений, к которым они привязаны;
- SelectAllFloorsWithBuildings – формирование выборки всех этажей с соответствующими корпусами;
- SelectAllMapObjectsWithLocations – формирование выборки всех объектов из таблицы MapObjects с соответствующими абсолютными координатами этих объектов;
- SelectAllRoomsWithRoomTypes – формирование выборки всех помещений с их типом;
- SelectBuildingById – поиск корпуса по его полю Id;
- SelectBuildingLocationByName – поиск абсолютных координат корпуса по его имени или номеру;

– SelectEquipmentsByRoomId – полный перечень оборудования, которое находится в помещении с интересующим Id.

Все запросы должны соответствовать критерию оптимальности по времени, чтобы пользователь не ждал слишком долго получения результата. Для проверки оптимизации воспользуемся средствами MS SQL Server. Листинг запросов и результаты анализа оптимизации представлены на листинге 9 и рисунках 12-14, остальные запросы и результаты оптимизации приведены в приложении А.

Листинг 9 – SelectAllFloorObjectsWithLocations

```
SELECT
      fo.Id,
      fop.X,
      fop.Y,
      fo.Name,
      fo.Discriminator as Type
FROM
      FloorObjects as fo
left join
      FloorObjectPoints as fop
on fo.Id = fop.FloorObjectId)
```

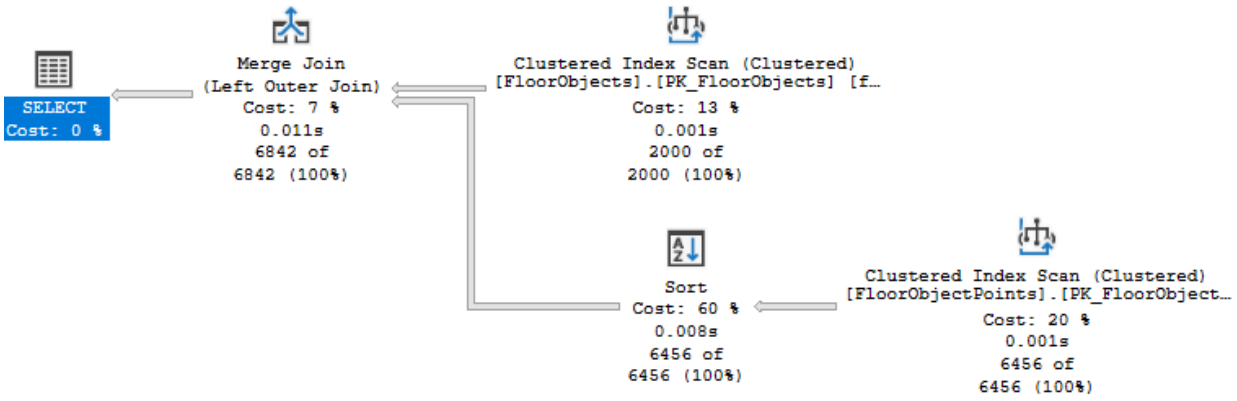

Рисунок 12 - План выполнения запроса

```
Table 'FloorObjectPoints'. Scan count 1, logical reads 83,
Table 'FloorObjects'. Scan count 1, logical reads 56, physi
```

Рисунок 13 - Количество логических чтений запроса

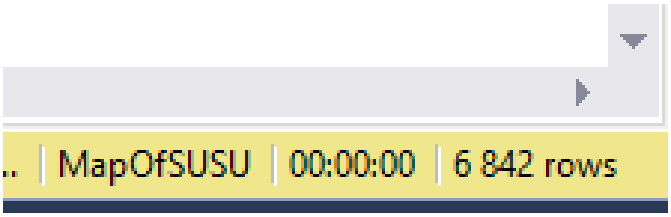

Рисунок 14 - Длительности выполнения запроса

Остальные запросы оформлены в приложении А.

5.2. ОПИСАНИЕ СУЩНОСТЕЙ ENTITY FRAMEWORK CORE

С помощью EF Core были созданы все представленные в пункте 5.1. таблицы. Диаграмма классов представлена на рисунке 15. Полный листинг подключения и преобразования классов в таблицы находится в приложении Б.

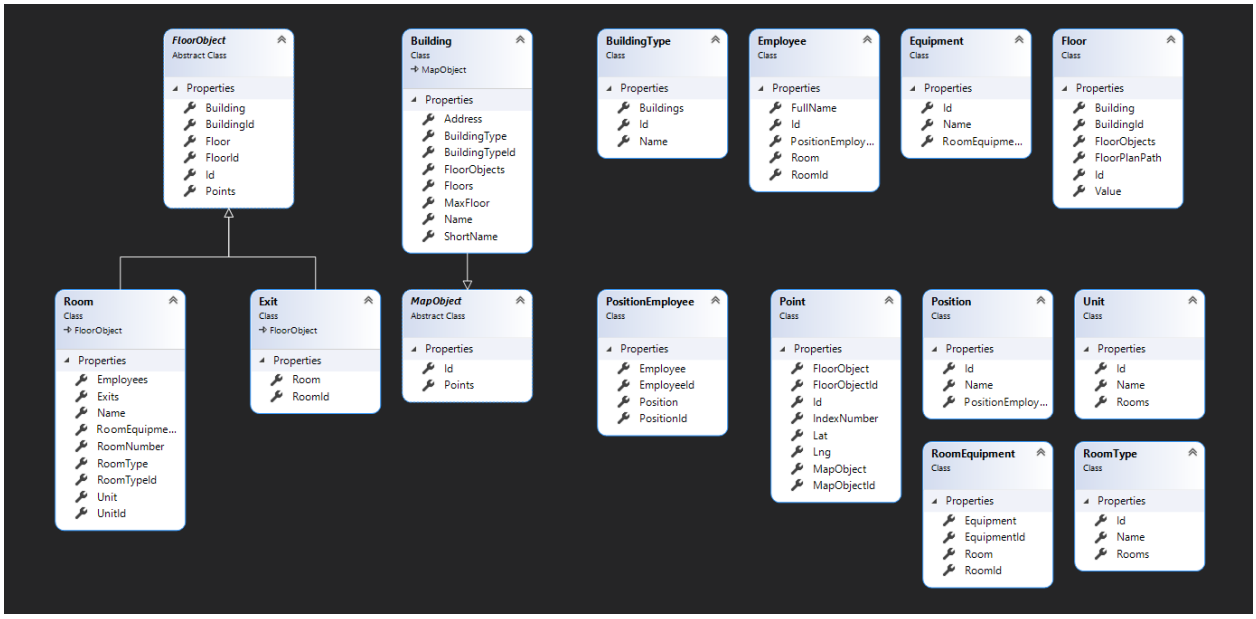

Рисунок 15 - Диаграмма классов

Из рисунка видно, что две таблицы имеют наследников, а именно FloorObject и MapObject. Это сделано для того, чтобы в дальнейшем можно было гибко масштабировать web-приложение. Для примера добавлен наследник Exit, который обозначает выходы помещений. Таким же образом можно добавлять любые объекты на этажи или карту (FloorObject и MapObject соответственно).

5.3. MVC АРХИТЕКТУРА WEB-ПРИЛОЖЕНИЯ

Как мы уже отмечали в пунктах 4.2.2. и 4.2.5, MVC архитектура строится на разделении компонентов на Model, View и Controller. Для карты ЮУрГУ мы создали эти компоненты.

В пункте 4.1 мы отметили, что у пользователя будет 2 сценария использования карты:

- сценарий поиска;
- сценарий ручного выбора объекта;

Для каждого сценария реализованы модели, представления и контроллер.

5.3.1. КОНВЕРТАЦИЯ ДАННЫХ В GEOJSON

Перед началом реализации сценариев нужно реализовать функцию конвертации данных из БД в формат GeoJSON. Для этого были созданы модели, представленные в листинге 10.

Листинг 10 - Модели получения данных о корпусах, этажах и объектов этажа

```
public class BuildingViewModel
    {
        public string Building { get; set; }
        public List<SelectListItem> Buildings { get; set; }
    }

    public class FloorViewModel
    {
        public string Floor { get; set; }
        public List<SelectListItem> Floors { get; set; }
    }
public class FloorObjectViewModel
    {
        public string FloorObject { get; set; }
        public List<SelectListItem> FloorObjects { get; set; }
    }
```

Теперь, когда есть модели можно создать сервис, который будет «вытаскивать» данные из базы и заполнять модели (листинг 11).

Листинг 11 - Сервисы отправки данных из базы в модели

```
public static async Task<Dictionary<Guid, ushort>> GetFloors ()
        {
            await using MapOfSUSUContext db = new MapOfSUSUContext();
            return await db.Floors.OrderBy(x => x.Value)
                .ToDictionaryAsync(x => x.Id, x => x.Value);
        }
```

```
public static async Task<Dictionary<Guid, sting>> GetBuildings ()
        {
            await using MapOfSUSUContext db = new MapOfSUSUContext();
            return await db.Buildings.OrderBy(x => x.ShortName)
                .ToDictionaryAsync(x => x.Id, x => x.ShortName);
        }
public static async Task<Dictionary<Guid, string>> GetFloorObjects ()
        {
            await using MapOfSUSUContext db = new MapOfSUSUContext();
            return await db.FloorObjects.OrderBy(x => x.Name)
                .ToDictionaryAsync(x => x.Id, x => x.Name);
        }
```

Далее можно преобразовать полученные данные в GeoJSON. Так как GeoJSON – это форматированная строка, то нам требуется полученные данные записать в виде свойств в формате GeoJSON (листинг 12).

Листинг 12 - Конвертация в GeoJSON объекта Building

```
public static async Task ExportBuildingsToGeoJSON(Guid buildingId)
        {
            await using MapOfSUSUContext db = new MapOfSUSUContext();
            var geoJSON = '{' + "type" + ':'
+ "FeatureCollection" + ','
 + "features" + ':'
 + "[{" + "type"
+ ':' + "Feature"
+ ','
+ "properties" + ':' + '{';
            var shortName = db.Buildings.First(x => x.Id == buildingId).ShortName;
            var maxFloor = db.Buildings.First(x => x.Id == buildingId).MaxFloor;
            var address = db.Buildings.First(x => x.Id == buildingId).Address;
            var name = db.Buildings.First(x => x.Id == buildingId).Name;
            geoJSON += '"' + "ShortName" + '"' +  shortName + '"';
```

```
            geoJSON += '"' + "MaxFloor" + '"' + maxFloor + '"';
            geoJSON += '"' + "Address" + '"' + address + '"';
            geoJSON += '"' + "Name" + '"' + name + '"';
            geoJSON += "]}";
        }
```

5.3.2. СЦЕНАРИЙ ПОИСКА

Для реализации этого сценария в первую очередь была размечена область под поисковую строку (листинг 13).

Листинг 13 - Функция обработки ввода в строку поиска

```
 function search() {
    $.get('@Url.Action("GetSearchResult", "Home")', {
       request: $("#searchInput").val()
    }).done(function (data) {
       $("#searchResult").show();
       $("#selectSearchResult").html(data);
    });
     }
```

Из листинга 13 видно, что функция реализована с помощью ajax, следовательно, она работает асинхронно и тем самым не требует обновления. Также видно, что данные предаются с помощью HomeController в модель. В листинге 15 представлена модель.

Эта модель принимает объект, содержит 2 поля: объект, который нужно принять и список, который нужно вернуть из базы данных.

Листинг 15 - Функция ввода и передачи информации в модель

```
public static async Task<SearchResultViewModel> GetSearchResult(string request)
        {
            await using var db = new MapOfSUSUContext();
```

```
                if (string.IsNullOrEmpty(request)) return null;
                var dictionary = await db.Employees
                    .Where(x    =>    !string.IsNullOrEmpty(x.FullName)    &&
x.FullName.Contains(request) &&
                                request.Length >= 3)
                    .Select(x => new
                    {
                        x.Id,
                        Name = x.FullName
                    })
                    .Union(
                        db.Buildings
                            .Where(x  =>  !string.IsNullOrEmpty(x.Name)  &&
(x.Name.Contains(request) ||

x.ShortName.Contains(request) &&

request.Length <= 5))
                            .Select(x => new
                            {
                                x.Id,
                                x.Name
                            })
                    )
                    .Union(
                        db.Rooms
                            .Where(x  =>  !string.IsNullOrEmpty(x.Name)  &&
(x.Name.Contains(request) ||

x.RoomNumber.Contains(request) &&

request.Length <= 15))
                            .Select(x => new
                            {
                                x.Id,
```

```
                        Name = x.RoomNumber
                    })
            )
            .OrderBy(x => x.Name)
            .ToListAsync();

        var model = new SearchResultViewModel
        {
            Objects = dictionary.Select(x => new SelectListItem
            {
                Value = x.Id.ToString(),
                Text = x.Name,
            }).ToList()
        };
        model.Objects.ForEach(item =>
        {
            var room = db.Rooms.FirstOrDefault(x => x.Id == Guid.Parse(item.Value));
            if (room != null)
                item.Text = $"{item.Text}/{room.Building.ShortName}";
        });
        return model;
    }
```

Также нужно модель для работы с данными, которые получены из функции ввода. Модель предоставлена в листинге 16.

Листинг 16 - Модель поиска

```
@model SearchResultViewModel

<select
    id="selectSearchResult"
    class="form-select"
```

```
        aria-label="Не выбрано"
        asp-for="Object"
        asp-items="Model.Objects"
        onchange="selectObject()">
        <option style="display:none">Не выбрано</option>
    </select>
```

На практике это выглядит следующим образом. В поисковую строку вводится, например, номер аудитории. После этого выпадает список с «похожими» значениями (рисунок 16).

Рисунок 16

На рисунке видно, что у нас есть 2 совпадения, после выбора аудитории 811/3б, камера центрируется на выбранной аудитории и выделяется цветом (рисунок 17).

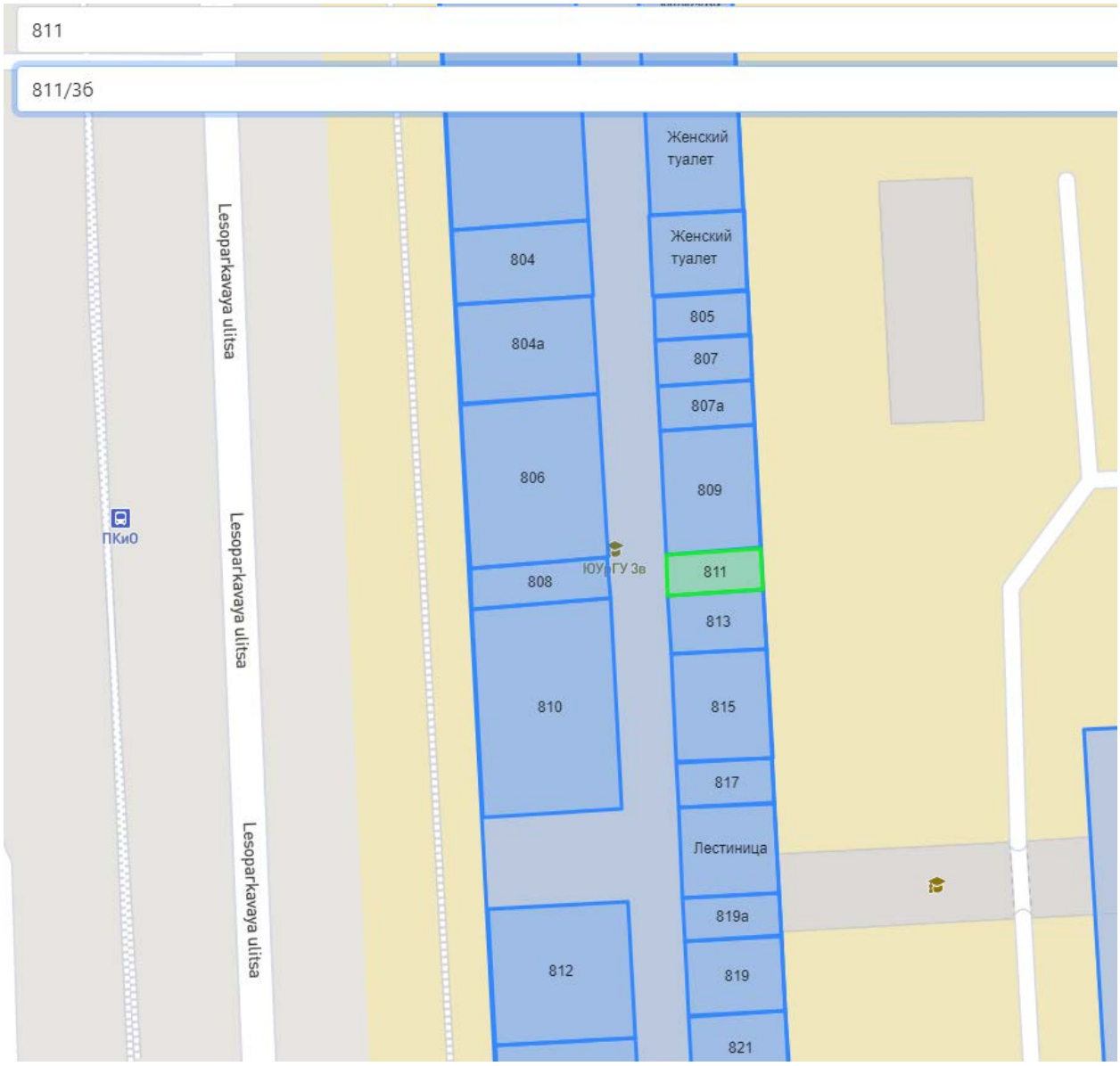

Рисунок 17

5.3.3. СОЗДАНИЕ СЦЕНАРИЯ РУЧНОГО ВЫБОРА

Частично этот сценарий повторяет прошлый, но, чтобы их можно было визуально отличать, при выборе помещения выскакивает popup, который сообщает номер или название помещения.

Модель используется, аналогичная сценарию поиска (листинг 13), однако в данном виде этот сценарий выполняется почти целиком в view (листинг 17).

Листинг 17 - getCenterCoordinate()

```
geo.eachLayer(function (layer) {
    layer.bindPopup(layer.feature.properties.name);
});
```

Дело в том, что эта функция использует сформированный из базы данных GeoJSON объект, и добавляет простой event, за счёт которого по клику на помещение всплывает popup с именем этого помещения (рисунок 18).

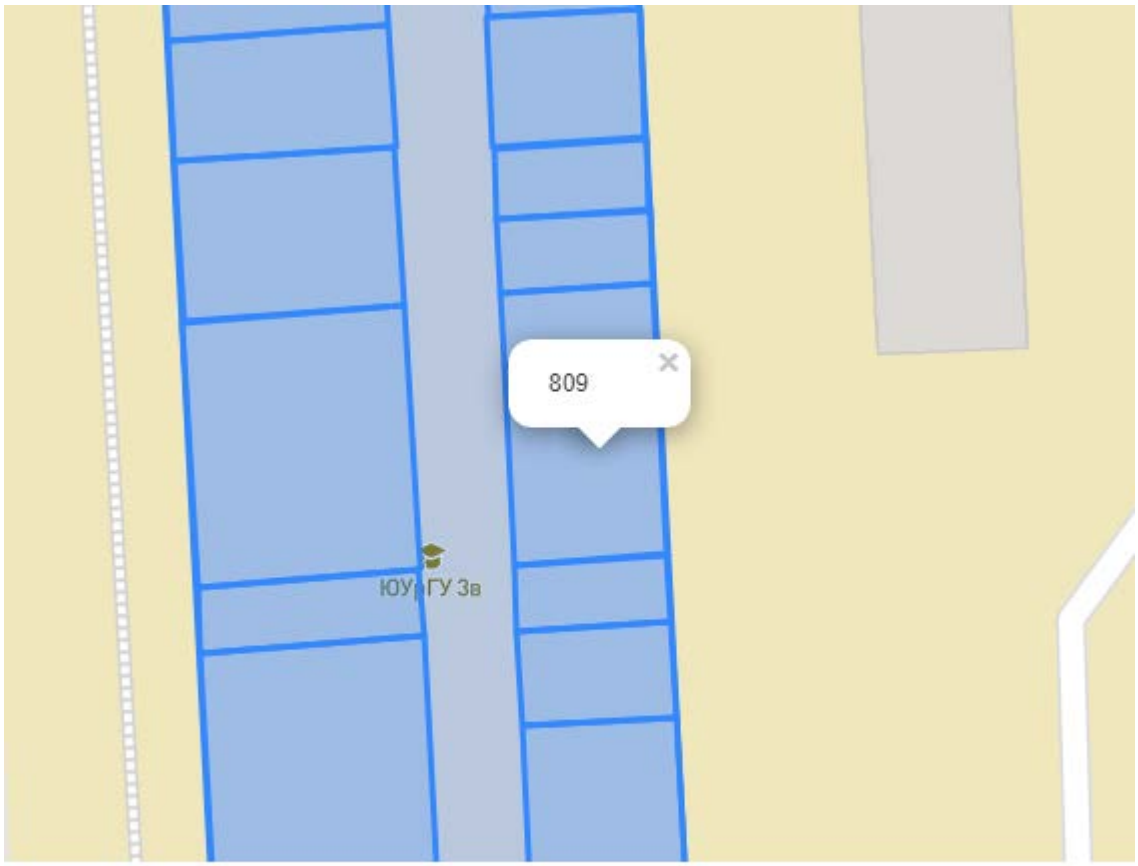

Рисунок 18 - Выбор помещения с номером 809

Тем же способом можно выводить более подробную информацию. Однако не стоит думать, что только с помещениями можно так работать GeoJSON, сформированный из базы данных, может хранить в себе любые поля для любого объекта, следовательно, то же самое происходит с корпусами или другими объектами нашего приложения.

6. ТЕСТИРОВАНИЕ

Из задания было выявлено 2 сценария взаимодействия конечного пользователя с web-приложением. Для проверки их работоспособности проведём ряд тестов.

6.1. ПОИСК

Начнём с проверки обработки введённого значения в строку поиска. Для этого сначала введём случайное значении и посмотрим, что выведет программа (рисунок 19).

Рисунок 19 - В строку поиска введено значение «ывалоры»

Как видно из рисунка, представлено вариантов выбора не было, а, следовательно, дальнейших ошибок, связанных с запросами к базе, не появится.

С другой стороны, введём существующее значение искомого помещениями (рисунок 20).

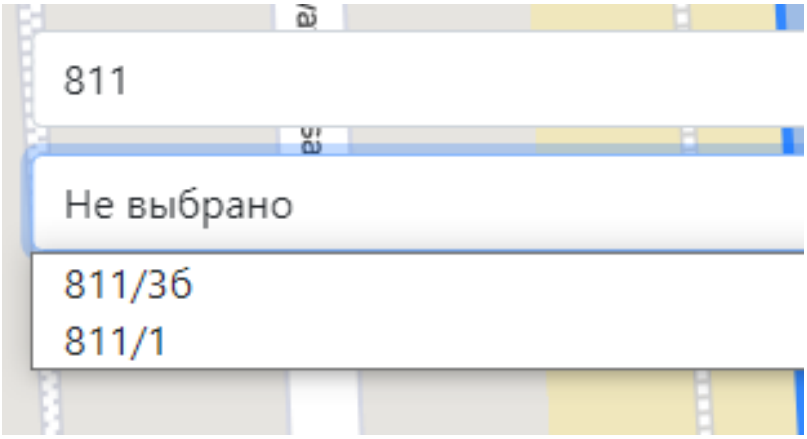

Рисунок 20 - В строку поиска введено значение «811»

На рисунке видно, что по запросу номера помещения «811» выдалось 2 результата поиска, которые оба содержат в себе «811».

6.2. РУЧНОЙ ВЫБОР

Это сценарий нужен для получения дополнительной информации о объекте. Что проверить работоспособность данного сценария, выберем помещение с номером «802» (рисунок 21).

Рисунок 21 - Выделение помещения с номером «802» в ручном режиме

После выбора нужного помещения появился popup с информацией об этом помещении, следовательно, ручной выбор работает исправно.

ЗАКЛЮЧЕНИЕ

Целью работы являлась разработка web-приложения «Карта помещений ЮУрГУ».

Для достижения данной цели были решены следующие задачи:

- определены основные потребности потенциальных пользователей;
- освоены основные средства создания web-приложения;
- изучены основные средства создания интерактивных карт;
- изучены основные средства работы с БД;
- определены инструменты взаимодействия БД и интерфейса интерактивных карт;
- реализована БД;
- реализована интерактивные карты;
- разработан интерфейс взаимодействия пользователя с интерактивной картой.

Поставленные задачи выполнены успешно, цель достигнута. Удобный интерфейс web-приложения позволяет с лёгкостью ориентироваться в кампусе НИИУ «ЮУрГУ». Для достижения цели способствовало грамотное определение составляющих интерфейса взаимодействия, а именно:

- определение местоположения помещений или корпусов с помощью поисковой строки;
- интерактивный выбор корпуса/помещения в ручном режиме;

В ближайшем будущем планируется дальнейшее развитие карты в сторону дополнения административным функционалом.

Разработанный вариант готов к опытной эксплуатации.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. WebCanape [Электронный ресурс]. - Режим доступа - URL:https://www.web-canape.ru/business/vsya-statistika-interneta-i-socsetej-na-2021-god-cifry-i-trendy-v-mire-i-v-rossii/ (дата обращения: 08.03.2021).
2. Что такое интерактивная карта? [Электронный ресурс]. - Режим доступа - URL: https://linuxgid.ru/chto-takoe-interaktivnaya-karta/#:~:text=Интерактивная%20карта%20—%20это%20карта%2C%20которая,для%20интерактивных%20карт%20значительно%20расширяется. (дата обращения: 09.03.2021).
3. Как написать обзор литературы [Электронный ресурс]. - Режим доступа - URL: https://www.academia.edu/3420632/Как_написать_обзор_литературы_по_теме?email_work_card=view-paper (дата обращения: 09.03.2021).
4. Быков, А. В. Web-картографирование: учебное пособие / А.В Быков, С.В Пьянков. - Пермь: Изд-во Перм. гос. нац. исслед. ун. - 2015. - Т. 2015. - 110 с.
5. Кацко, С.Ю. Состояние и проблемы веб-картографии на современном этапе развития единого геоинформационного пространства / С.Ю. Кацко, П.М. Кикин // Материалы Международной конференции «ИнтерКарто. ИнтерГИС». 2014. - 174 с.
6. ДеМерс, М.Н. Географические Информационные Системы. Основы / М.Н. ДеМерс - М.: Изд-во Дата+, 1999. - 490 с.
7. Kraak, M. J. Web Cartography / M. J. Kraak, Allan Brown. - NewYork: Изд-во CRC Press, 2001. - 224 p.

8. Web mapping литературы [Электронный ресурс]. - Режим доступа - URL:https://en.wikipedia.org/wiki/Web_mapping#Types (дата обращения: 09.03.2021).
9. Evolving paper cartography [Электронный ресурс]. - Режим доступа - URL:https://en.wikipedia.org/wiki/Web_mapping#Evolving_paper_cartography (дата обращения: 09.03.2021).
10. George Schlossnagle, Advanced Php Programming: Developing Large-scale Web Applications With Php 5 / George Schlossnagle. - M: Изд-во Addison-Wesley Professional, 2014. - 700 с.
11. Олифер, В.Г. Компьютерные сети. Принципы, технологии, протоколы: - Учебник для ВУЗов / В.Г. Олифер, Н.А Олифер. - 2-е изд., испр. и доп. Питер, 2019. - 992 с.
12. Документация [Электронный ресурс]. - Режим доступа - URL: https://www.ruby-lang.org/ru/documentation/ (дата обращения 09.03.2021).
13. Карпенко, С. Н. Основы объектно-ориентированного программирования на языке C++ : учебно-методическое пособие / С. Н. Карпенко. - Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2018. - 104 с. - Текст : электронный // Лань : электронно-библиотечная система. - URL: https://e.lanbook.com/book/144808 (дата обращения: 11.03.2021). – Режим доступа: для авториз. пользователей.
14. Maureen, W. Z. Microsoft Visual Basic 6.0 Reference Library / W. Z. Maureen. - NewYork: Изд-во Microsoft Press, 1998. - 3344 с.
15. Бьёрн, С. Язык программирования C++ = The C++ Programming Language / Пер. с англ. - 3-е изд. - СПб.; М.: Невский диалект - Бином, 1999. - 991 с.

16. Шилдт, Г. Java 8. Полное руководство, 9-е издание = Java 8. The Complete Reference, 9th Edition / Г. Шилдт. - М.: «Вильямс», 2015. - 1376 с.
17. .NET documentation [Электронной ресурс]. - Режим доступа - URL:https://docs.microsoft.com/ru-ru/dotnet/fundamentals/ (дата обращения 10.03.2021).
18. Нейгел, К. C# 5.0 и платформа .NET 4.5 для профессионалов / К. Нейгел, Д. Глинн, К. Уотсон. и др. - М.: Изд-во Диалектика 2014. - 1440 с.
19. Introduction to ASP.NET Core [Электронной ресурс]. - Режим доступа - URL: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0(дата обращения 10.03.2021).
20. Web Framework Benchmarks [Электронной ресурс]. - Режим доступа - URL:https://www.techempower.com/benchmarks/#hw=ph&test=plaintext (дата обращения 10.03.2021).
21. Пальти И. Высоконагруженные приложении. Программирование, масштабирование, поддержка. / И. Пальти, А. Тумаркин. – СПБ: Питер. - 640 с.
22. Спецификация формата GeoJSON [Электронной ресурс]. - Режим доступа - URL: http://geojson.org/geojson-spec.html (дата обращения 10.03.2021).
23. Сравнение современных СУБД [Электронной ресурс]. - Режим доступа - URL: https://drach.pro/blog/hi-tech/item/145-db-comparison (дата обращения 10.03.2021).
24. Limits In SQLite [Электронной ресурс]. - Режим доступа - URL: https://sqlite.org/limits.html (дата обращения 10.03.2021).

25. Documentation [Электронный ресурс]. - Режим доступа -URL: https://dev.mysql.com/doc/ (дата обращения 13.06.2021).
26. Documentation [Электронный ресурс]. - Режим доступа -URL: https://www.postgresql.org/docs/ (дата обращения 13.06.2021).
27. SQL Server technical documentation [Электронный ресурс]. - Режим доступа -URL: https://www.postgresql.org/docs/ (дата обращения 13.06.2021).
28. Общие архитектуры веб-приложений [Электронной ресурс]. - Режим доступа - URL: https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата обращения 10.03.2021).
29. Разработка приложений MVC ASP.NET Core [Электронной ресурс]. - Режим доступа - URL: https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/develop-asp-net-core-mvc-apps (дата обращения 10.03.2021).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Листинг А.1 - SelectAllEmployeesWithRoomNames

```
SELECT
            e.Id,
            e.FullName,
            fo.Name
FROM
            Employees as e
left join
            FloorObjects as fo
on e.RoomId = fo.Id
WHERE(fo.Discriminator = 'Room')
```

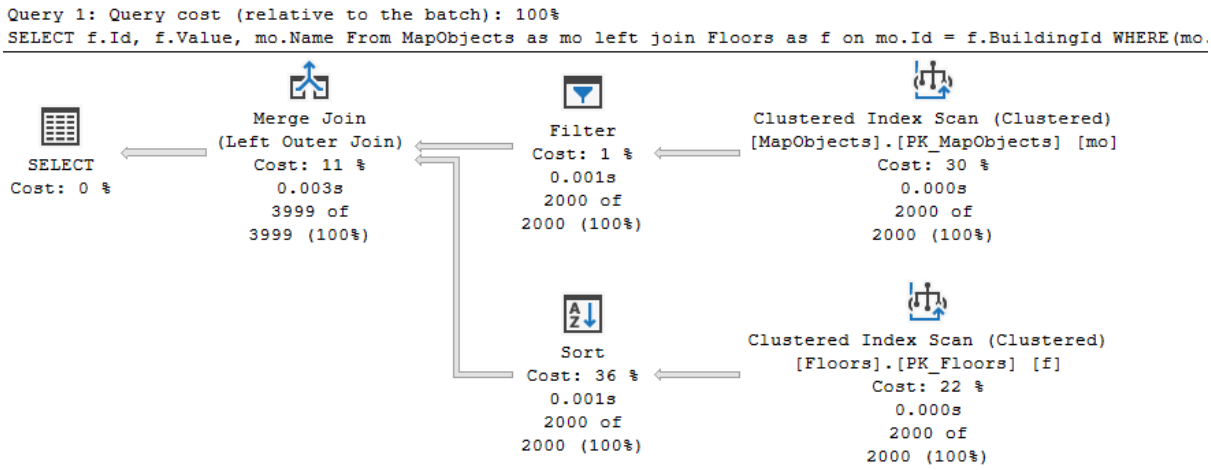

Рисунок А.1 - Execution Plan SelectAllEmployeesWithRoomNames

```
Table 'Employees'. Scan count 1, logical reads 33, ph
Table 'FloorObjects'. Scan count 1, logical reads 3,
```

Рисунок А.2 - Logical reads SelectAllEmployeesWithRoomNames

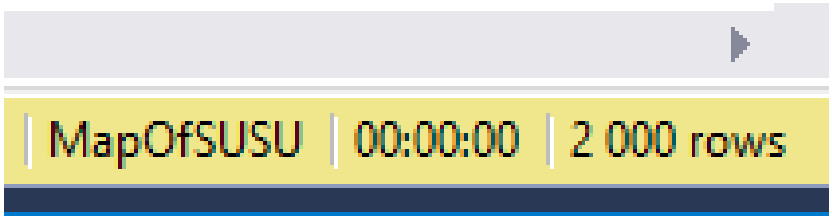

Рисунок А.3 - Длительность выполнения запроса SelectAllEmployeesWithRoomNames

Листинг А.2 - SelectAllFloorsWithBuildings

```
SELECT
            f.Id,
            f.Value,
            mo.Name
From
            MapObjects as mo
left join
            Floors as f
on mo.Id = f.BuildingId
WHERE(mo.Discriminator = 'Building')
```

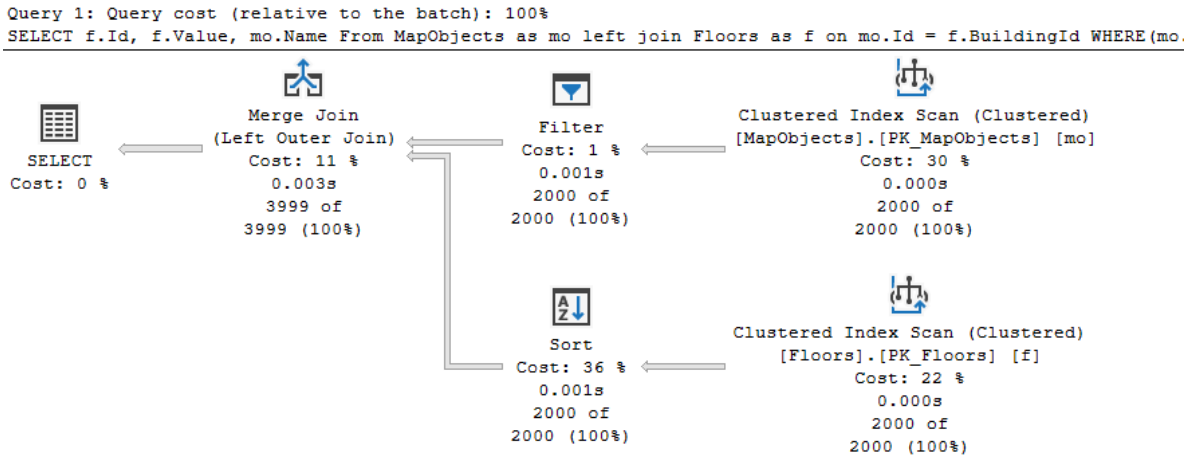

Рисунок А.4 - Execution Plan SelectAllFloorsWithBuildings

```
Table 'Floors'. Scan count 1, logical reads 33, phys
Table 'MapObjects'. Scan count 1, logical reads 48,
```

Рисунок А.5 - Logical reads SelectAllFloorsWithBuildings

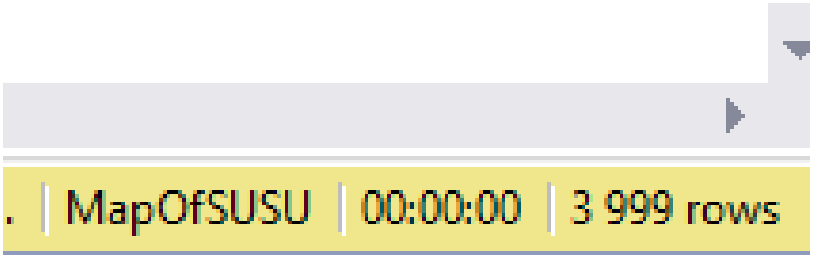

Рисунок А.6 - Длительность выполнения запроса SelectAllFloorsWithBuildings

Листинг А.3 - SelectAllMapObjectsWithLocations

```
SELECT
      mo.Id,
      mop.Lat,
      mop.Lng,
      mo.Name,
      mo.Discriminator as Type
From
      MapObjects as mo
left join
      MapObjectPoints as mop
      on mo.Id = mop.MapObjectId
```

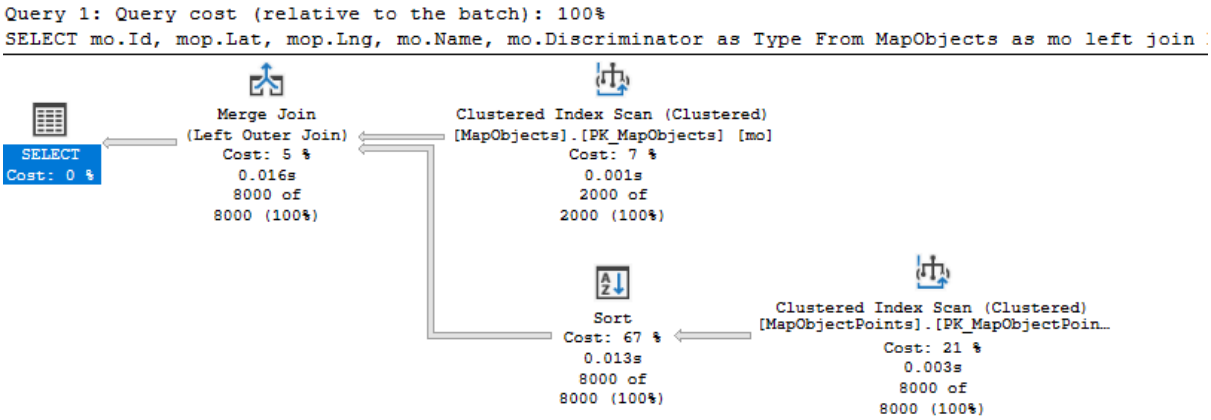

Рисунок А.7 - Execution Plan SelectAllMapObjectsWithLocations

```
Table 'MapObjectPoints'. Scan count 1, logical reads 139,
Table 'MapObjects'. Scan count 1, logical reads 48, physic
```

Рисунок А.8 - Logical reads SelectAllMapObjectsWithLocations

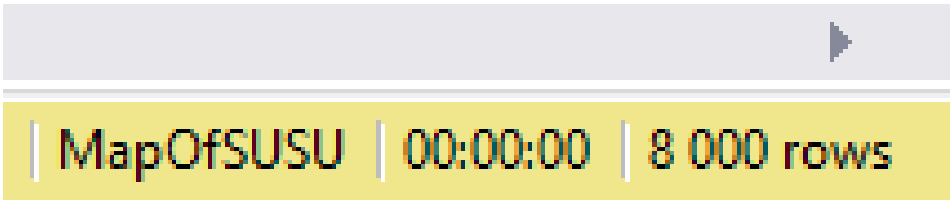

Рисунок А.9 - Длительность выполнения запроса SelectAllMapObjectsWithLocations

Листинг А.4 - SelectAllRoomsWithRoomTypes

```
SELECT
            fo.Id,
            rt.Name,
            fo.Name
FROM
            FloorObjects as fo
left join
            RoomTypes as rt
on fo.RoomTypeId = rt.Id
WHERE(fo.Discriminator = 'Room')
```

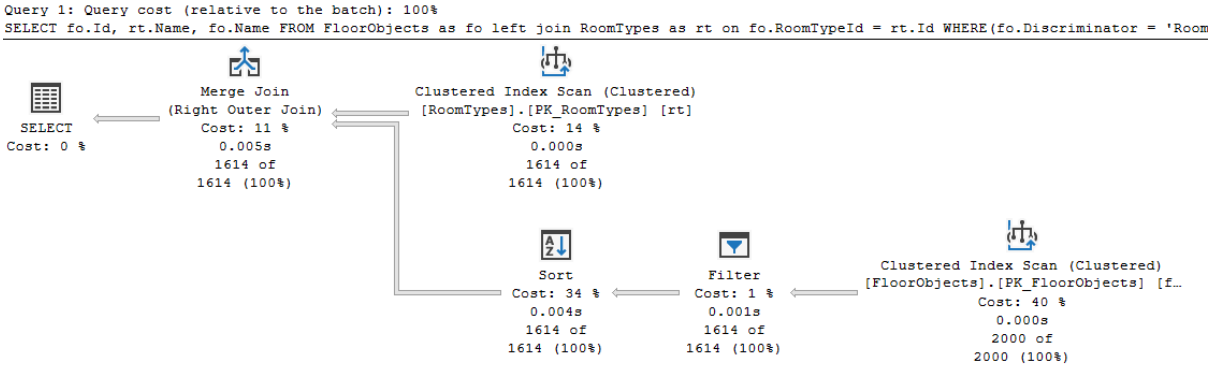

Рисунок А.10 - Execution Plan SelectAllRoomsWithRoomTypes

```
Table 'FloorObjects'. Scan count 1, logical reads 56, p
Table 'RoomTypes'. Scan count 1, logical reads 18, phys
```

Рисунок А.11 - Logical reads SelectAllRoomsWithRoomTypes

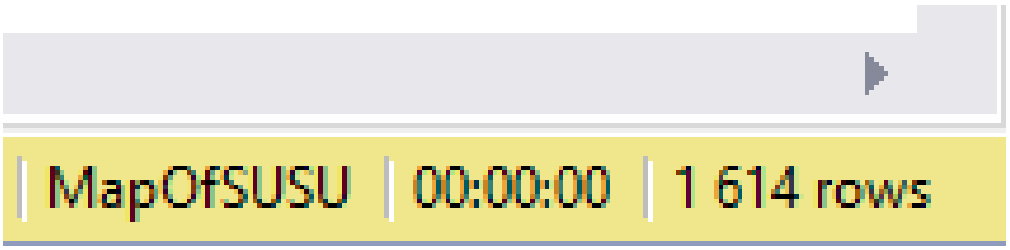

Рисунок А.12 - Длительность выполнения запроса SelectAllRoomsWithRoomTypes

Листинг А.5 - SelectBuildingById

```
SELECT TOP 1 *
            FROM
                  MapObjects as mo WHERE Id = @Id and mo.Discriminator =
                  'Building'
```

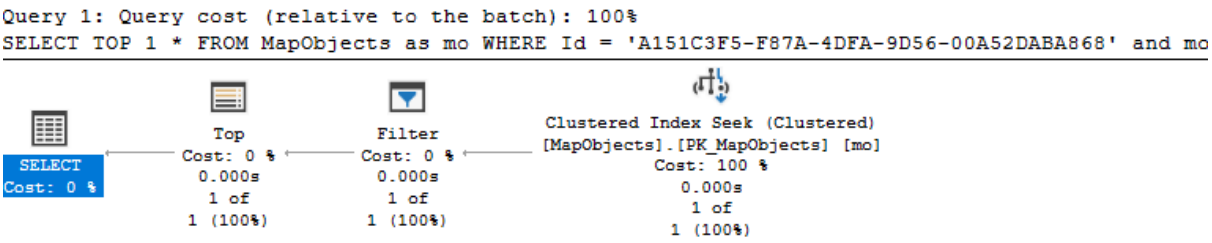

Рисунок А.13 - Execution Plan SelectBuildingById

```
Table 'MapObjects'. Scan count 0, logical reads 2,
```

Рисунок А.14 - Logical reads SelectBuildingById

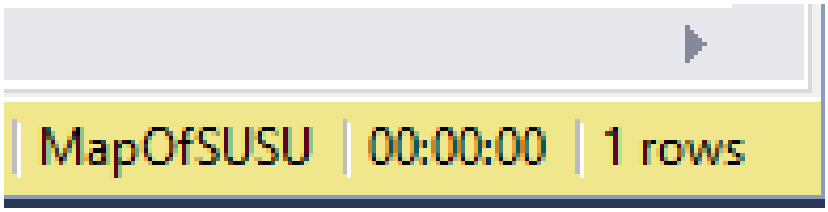

Рисунок А.15 - Длительность выполнения запроса SelectBuildingById

Листинг А.6 - SelectBuildingLocationByName

```
SELECT
            mop.MapObjectId,
            mop.Lat,
            mop.Lng
FROM
            MapObjectPoints as mop
INNER JOIN
            MapObjects as mo
            ON mop.MapObjectId = mo.Id
WHERE mo.Name = @Name or mo.ShortName = @Name
```

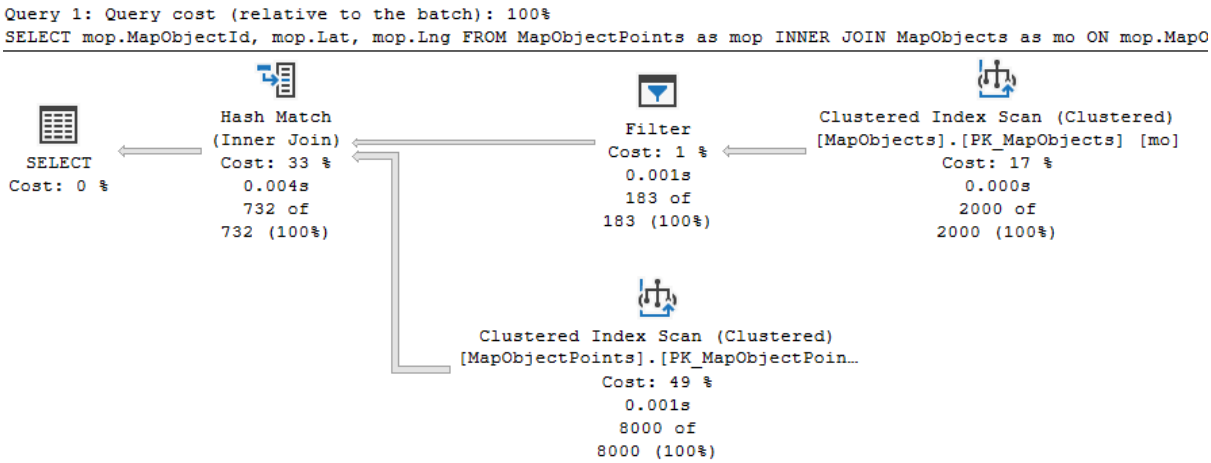

Рисунок А.16 - Execution Plan SelectBuildingLocationByName

```
Table 'MapObjectPoints'. Scan count 1, logical reads 139,
Table 'MapObjects'. Scan count 1, logical reads 48, physi
```

Рисунок А.17 - Logical reads SelectBuildingLocationByName

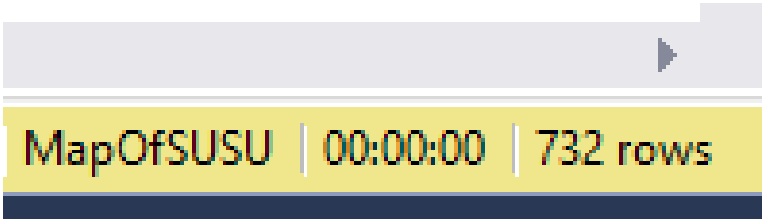

Рисунок А.18 - Длительность выполнения запроса SelectBuildingLocationByName

Листинг А.7 - SelectEquipmentsByRoomId

```
SELECT
            fo.RoomNumber,
            e.Id,
            e.Name
FROM Equipments as e
INNER JOIN RoomEquipments as re
            ON e.Id = re.EquipmentId
INNER JOIN FloorObjects as fo
            ON fo.Id = re.RoomId
WHERE re.RoomId = @RoomId
```

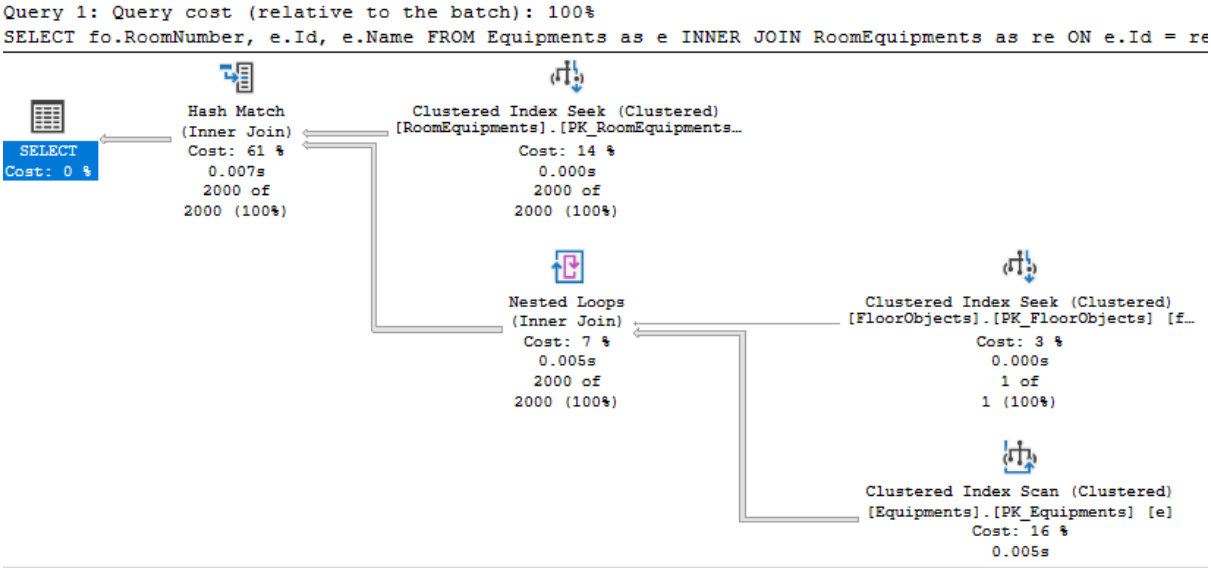

Рисунок А.19 - Execution Plan SelectEquipmentsByRoomId

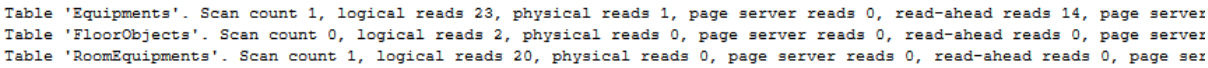

Рисунок А.20 - Logical reads SelectEquipmentsByRoomId

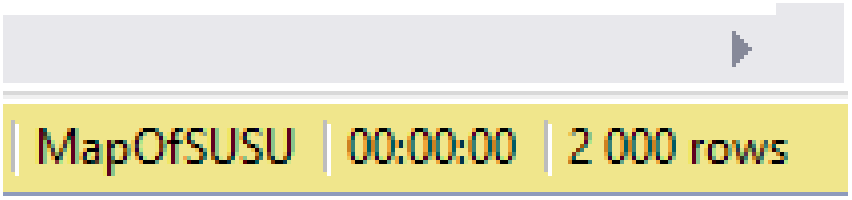

Рисунок А.21 - Длительность выполнения запроса SelectEquipmentsByRoomId

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД MAPOFSUSUCONTEXT.CS

```
public class MapOfSUSUContext : DbContext
    {
        public DbSet<Building> Buildings { get; set; }
        public DbSet<BuildingType> BuildingTypes { get; set; }
        public DbSet<MapObjectPoint> MapObjectPoints { get; set; }
        public DbSet<Unit> Units { get; set; }
        public DbSet<Employee> Employees { get; set; }
        public DbSet<Equipment> Equipments { get; set; }
        public DbSet<FloorObject> FloorObjects { get; set; }
        public DbSet<FloorObjectPoint> FloorObjectPoints { get; set; }
        public DbSet<Floor> Floors { get; set; }
        public DbSet<Position> Positions { get; set; }
        public DbSet<PositionEmployee> PositionEmployees { get; set; }
        public DbSet<Room> Rooms { get; set; }
        public DbSet<RoomType> RoomTypes { get; set; }
        public DbSet<RoomEquipment> RoomEquipments { get; set; }
        public DbSet<Exit> Exits { get; set; }
        public DbSet<MapObject> MapObjects { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder
                .UseLazyLoadingProxies()

.UseSqlServer(@"Server=localhost;Database=MapOfSUSU;Trusted_Connection=True;");

            optionsBuilder.EnableSensitiveDataLogging();
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<PositionEmployee>()
                .HasKey(pe => new { pe.PositionId, pe.EmployeeId });

            modelBuilder.Entity<PositionEmployee>()
                .HasOne(pe => pe.Position)
                .WithMany(p => p.PositionEmployees)
                .HasForeignKey(pe => pe.PositionId)
                .OnDelete(DeleteBehavior.Cascade);
```

```
            modelBuilder.Entity<PositionEmployee>()
                .HasOne(pe => pe.Employee)
                .WithMany(e => e.PositionEmployees)
                .HasForeignKey(pe => pe.EmployeeId)
                .OnDelete(DeleteBehavior.Cascade);


            modelBuilder.Entity<RoomEquipment>()
                .HasKey(re => new { re.RoomId, re.EquipmentId });

            modelBuilder.Entity<RoomEquipment>()
                .HasOne(re => re.Room)
                .WithMany(r => r.RoomEquipments)
                .HasForeignKey(re => re.RoomId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<RoomEquipment>()
                .HasOne(re => re.Equipment)
                .WithMany(r => r.RoomEquipments)
                .HasForeignKey(re => re.EquipmentId)
                .OnDelete(DeleteBehavior.Cascade);
        }
    }
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД INDEX.CSHTML

```
@{
    ViewData["Title"] = "Home Page";
}

<script type="text/javascript"> // Блок функций
        mymap = null;

        floor = null;
        room = null;
        markers = null;

        buildings = null;
        building = null;

        function initBuildings() {
                buildings = L.geoJSON(allBuildings, {
                        style: function() {
                                        return {
                                                color: "#3388FF",
                                                opacity: 1,
                                                fillColor: "#54BEF2",
                                                fillOpacity: 1
                                        }
                                }
                }).addTo(mymap);
        }

    function clearMap(e, selectSearchResultNotClear = false) {
                if (room) {
                        room.removeFrom(mymap);
                        room = null;
        }

        if (floor) {
        floor.removeFrom(mymap);
        floor = null;
                }

        if (markers && markers.length !== 0 ) {
                        markers.forEach(item => item.removeFrom(mymap));
                        markers = null;
        }

        if (buildings) {
                        buildings.removeFrom(mymap);
```

```
                buildings = null;
            }

    if (building) {
                building.removeFrom(mymap);
                building = null;
            }

    if (!selectSearchResultNotClear)
                $("#selectSearchResult").val("Не выбрано");

    $("#searchResult").hide();
    $("#floorPanel").hide();

            initBuildings();
}

function search() {
    $.get('@Url.Action("GetSearchResult", "Home")', {
        request: $("#searchInput").val()
    }).done(function (data) {
        $("#searchResult").show();
        $("#selectSearchResult").html(data);
    });
    }
    function makeIconAnchor(name) {
            if (name.indexOf(" ") !== -1)
            {
                    var arrayOfStrings = name.split(" ");
                    arrayOfStrings[0] = arrayOfStrings[0].length * 3;
        return [arrayOfStrings[0], arrayOfStrings.length * 10];
            }
            else
        return [name.length * 3, 10];
    }

async function selectObject(selectObject) {
    // mymap.setView([39.74739, -105], 13);
    // alert($("#selectSearchResult").val());
            clearMap(null, true);

            let objectDiscriminator = await whatIsItObject(selectObject);

            switch (objectDiscriminator) {
                    case 0:
                            break;
```

```
                        case 1:
                                centerBuilding(selectObject);
                                break;
                        case 2:
                                centerRoom(selectObject);
                                break;
                }
        }

        function whatIsItObject(selectObject) {
            return $.get('@Url.Action("WhatIsItObject", "Home")', {
                        objectId: selectObject
                });
        }

        function centerRoom(selectObject) {
                $.get('@Url.Action("GetFloorsByObjectId", "Home")', {
                                objectId: selectObject
                        }).done(function (data) {
                                $("#floorPanel").show();
                                $("#floors").html(data);
                        });

            if (floor == null) {
                        floor = L.geoJSON(Level8_3b).addTo(mymap);

                        markers = []

                        floor.eachLayer(function (layer) {
                                //layer.bindPopup(layer.feature.properties.name);
                                markers.push(L.marker(layer.getCenter(), {
                                        icon: L.divIcon({
                                                className: 'my-div-icon',
                                                html: layer.feature.properties.name,
                                                iconAnchor:
makeIconAnchor(layer.feature.properties.name)
                                        })
                                }).addTo(mymap));
                        });
                }

                mymap.eachLayer(function (layer) {
                        if (layer.feature?.properties?.id &&
                                layer.feature.properties.id === selectObject)
                                mymap.removeLayer(layer);
                });
```

```
                room = L.geoJSON(Level8_3b, {
                        style: function () {
                                return {color: "#0ff029"}
                        },
                        filter: function (geoJsonFeature) {
                                return geoJsonFeature.properties.id === selectObject;
                        }
                }).addTo(mymap);

                // let layer = room.getLayers()[0];
                // let name = room.getLayers()[0].feature.properties.name;

                //layer.bindPopup(name).openPopup(layer.getCenter());
                mymap.fitBounds(room.getBounds());
        }

        function centerBuilding(selectObject) {
                $.get('@Url.Action("GetFloorsByObjectId", "Home")', {
                    objectId: selectObject
                }).done(function (data) {
                    $("#floorPanel").show();
                    $("#floors").html(data);
                });

            if (buildings != null) {
                building = buildings.getLayers().find(layer =>
                        layer.feature?.properties?.id && layer.feature.properties.id.toLowerCase()
=== selectObject);

                mymap.removeLayer(building);

                building = L.geoJSON(building.toGeoJSON(), {
                        style: function () {
                return {
                        color: "#129b1b",
                                                opacity: 1,
                                                fillColor: "#0ff029",
                                                fillOpacity: 1
                }
                        }
                }).addTo(mymap);

                mymap.fitBounds(building.getBounds());
            }
        }
```

```
        </script>

        <div id="searchForm" class="position-absolute start-50 translate-middle">
                <div class="input-group mb-3">
                        <input id="searchInput" type="text" class="form-control" placeholder="Введите название" aria-label="Введите название" aria-describedby="basic-addon2">
                        <div class="input-group-append">
                                <button id="searchButton" class="btn btn-outline-secondary btn-light" type="button" onclick="search()">Найти</button>
                        </div>
                </div>
        </div>

        <div id="searchResult" class="position-absolute start-50 translate-middle" style="display: none;">
                @{ await Html.RenderPartialAsync("SelectSearchResult", ViewData["SearchResults"] ?? new SearchResultViewModel()); }
        </div>

        <div id="floorPanel" class="position-absolute top-50 translate-middle" style="display: none;">
                @{ await Html.RenderPartialAsync("SelectFloor", ViewData["Floors"] ?? new FloorViewModel()); }
        </div>

        <div class="fill">
                <div id="map"></div>
        </div>

        <script> // Постобработка

                // Execute a function when the user releases a key on the keyboard
                $("#searchInput").on("keyup", function(event) {
                  // Number 13 is the "Enter" key on the keyboard
                  if (event.keyCode === 13) {
                        // Cancel the default action, if needed
                        event.preventDefault();
                        // Trigger the button element with a click
                        $("#searchButton").click();
                  }
                });
        </script>

        <script> // Работа с картой

            mymap = L.map('map', {
```

```
            center: [55.159338, 61.370198],
            zoom: 17
        });

        mymap.on('dblclick', clearMap);


L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token=pk.eyJ1IjoibWFwYm94IiwiY
SI6ImNpejY4NXVycTA2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {
                    maxZoom: 20,
                    attribution:            'Map            data            &copy;            <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, ' +
                            'Imagery © <a href="https://www.mapbox.com/">Mapbox</a>',
                    id: 'mapbox/streets-v11',
                    tileSize: 512,
                    zoomOffset: -1
        }).addTo(mymap);

        initBuildings();

    </script>
```