МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра морские информационные системы

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

На тему Программная реализация внутренней навигации в здании с помощью API

Исполнитель Гусев Владислав Витальевич

(фамилия, имя, отчество)

Руководитель кандидат технических наук, доцент

(ученая степень, ученое звание)

Попов Николай Николаевич

(фамилия, имя, отчество)

«К защите допускаю»

Заведующий кафедрой

(подпись)

доц. техн. наук, доц

(ученая степень, ученое звание)

Загородних Надежда Алекс.

(фамилия, имя, отчество)

«__»_____2018 г.

Санкт-Петербург

2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра морские информационные системы

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

На тему Программная реализация внутренней навигации в здании с помощью API

Исполнитель Гусев Владислав Витальевич

(фамилия, имя, отчество)

Руководитель кандидат технических наук, доцент

(ученая степень, ученое звание)

Попов Николай Николаевич

(фамилия, имя, отчество)

«К защите допускаю»

Заведующий кафедрой

_____

(подпись)

_____

(ученая степень, ученое звание)

_____

(фамилия, имя, отчество)

«___»_____2018 г.

Содержание

Введение

Развитие ГИС-сервисов позволяет решать широкий спектр задач, связанных с навигацией на местности. Однако вопрос навигации внутри зданий и прилегающих к ним территорий стоит особенно остро. Данная проблема затрагивает большую целевую аудиторию:

- студенты и сотрудники университетов, имеющие несколько корпусов
- промышленные предприятия
- транспортные узлы (вокзалы, аэропорты, морские и речные порты)
- медицинские учреждения
- государственные учреждения
- бизнес-центры
- торговые и многофункциональные комплексы

Для составления маршрута до места назначения можно использовать специальные сервисы: Google Maps, Яндекс Карты или 2GIS. В последнее время, на мой взгляд, все более актуальной становится проблема навигации непосредственно внутри помещений. Здания становятся более объемными и часто имеют сложную архитектуру. Нередко люди попадающие в большое здание впервые, зачастую не могут сразу найти необходимый терминал, кабинет, магазин. В попытках найти нужное место на карте или по указателям тратится большое количество времени.

Свое применение система внутренней навигации также находит в морских и речных портах, имеющих огромные территории, на которых спутниковая навигация практически бесполезна.

Актуальность выбранной темы обусловлена тем, что в наши дни большинство предприятий имеет огромные территории с большим количеством корпусов, а также сложной архитектуру построек. В связи с этим новые сотрудники или студенты проходящие практику на таких предприятиях сталкиваются с проблемой ориентирования на их территориях.

Объект исследования – первый и второй корпуса Российского Государственного Гидрометеорологического университета.

Предмет исследования – система внутренней навигации.

В 2015 году к Российскому Государственному Гидрометеорологическому университету была присоединена Государственная полярная академия. На сегодняшний день университет имеет четыре учебных корпуса. Для большинства людей, как для студентов, в особенности первокурсников, так и для сотрудников университета остро встала проблема поиска необходимых аудиторий и кабинетов в новых для них корпусах. В связи со сложной архитектурой зданий, некоторые аудитории находятся в труднодоступных местах, что приводит к частым опозданиям или пропускам занятий, это, в свою очередь, негативно сказывается на статистике посещений.

Главной задачей разработки системы внутренней навигации является создание удобного инструмента, позволяющего ориентироваться в малознакомом месте и существенно сэкономить время.

Основной целью данного проекта является создание приложения, для решения проблемы навигации внутри зданий, а также на территории предприятий имеющих большое количество корпусов и сложную архитектуру построек.

Задачи реализации представленного проекта:

1. Предварительная оценка способов реализации проекта

На данном этапе проводится обследование объекта, детальный осмотр, оценка особенностей и предварительная оценка способов реализации.

2. Реализация проекта.

Реализация проекта включает в себя оцифровку планов зданий и написание кода для корректного отображения карты и основного интерфейса. Оцифровка может быть относительно простой при наличие поэтажных планов здания.

3. Введение приложения в эксплуатацию

Этап включает в себя конфигурацию всех данных в систему, настройку согласно проекта, загрузка технических планов, синхронизация планов с приложением, подключение приложения к серверу

Свое применение система внутренней навигации находит в морских и речных портах, имеющих огромные территории, на которых спутниковая навигация бесполезна.

Глава 1 Предварительная оценка способов реализации проекта

1.1 Исследование предметной области

Объектом исследования являются два корпуса Российского Государственного Гидрометеорологического Университета. Первый расположен по адресу – Малоохтинский пр., д.98, второй – пр. Металлистов, д.3. Данный университет хорошо подходит для примера, так как имеет несколько больших корпусов и сложную архитектуру зданий.

На сегодняшний день существует множество технологий, позволяющих определять положение внутри помещений. Их можно разделить на две группы, каждая из которых имеет свои ограничения и области применения[21]:

- использование мобильных устройств. Позиционирование осуществляется при помощи мобильных сетей и сети Wi-fi, а также различных датчиков, которыми оснащены современные мобильные устройства.
- использование специальных радиометок, используемых в оборудовании или закрепленных на одежде.

Рассмотрим более подробно некоторые из них.

Система позиционирования - механизм определения местоположения объекта в пространстве. Технологии этой задачи существуют в диапазоне от всемирного охвата с точностью метра до охвата места с точностью миллиметра.

Система внутреннего позиционирования **(**Indoor Positioning System) — система, которая используется для определения местоположения объектов или людей внутри зданий, где практически недоступна система спутниковой навигации с помощью радиоволн, акустических сигналов и другой сенсорной информации, собранной с помощью мобильных устройств. Вместо спутников, IPS пользуется близкими к нему точками с уже известными координатами.[30] Созданные для IPS устройства используют оптические, радио и акустические технологии. IPS позволяет "найтись" на карте в мобильном устрой-

стве. Пользователь всегда может открыть приложение и система подскажет, где он находится и покажет, что находится вокруг.

Система локального позиционирования (Local positioning system) — это навигационная система, обеспечивающая информацией о местоположении в пределах зоны действия сети в любую погоду, в любом месте, где есть прямой доступ до трех или более сигнальных маяков. Особым типом LPS является система определения местоположения в режиме реального времени (Real-time LS), которая позволяет в режиме реального времени отслеживать объект или человека в здании. LPS используют набор маяков, имеющих ограниченный диапазон и включающих в себя точки доступа wi-fi, сотовые базовые станции и радиостанции. [29]

Гибридная система позиционирования (Hybrid positioning systems) — системы поиска местоположения мобильного устройства, использующие несколько различных технологий позиционирования. Обычно GPS является одним из основных компонентов таких систем, в сочетании с вышками сотовой связи, беспроводными технологиями, Bluetooth или других локальных систем позиционирования. Эти системы разработаны для преодоления ограничений системы GPS, которая очень точна на открытых площадках, но плохо работает между высокими зданиями и внутри помещений. [29]

Для реализации поставленной цели необходимо подключение определенных интерфейсов, называемых Application Programming Interface (API).

API – это интерфейс программирования и создания приложений. Он представляет собой набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением или операционной системой для использования во внешних продуктах. API определяет функциональность, предоставляемую программой, при этом позволяет абстрагироваться от того, как реализована функциональность. Также предоставляет среду для разработки приложений, используя конкретный язык программирования, и может быть разработан для web-систем, операционных систем и баз данных. В случае веб-приложений API может отдавать данные в отличном от стандартного

HTML формате, благодаря чему ими удобно пользоваться при написании собственных приложений. Сторонние общедоступные API чаще всего предоставляют данные в одном из двух форматов: XML или JSON.[10]

Классификация программных интерфейсов тесно связана с возможностями и назначением приложений, которые через них управляются. В отдельные группы выделяют интерфейсы управления графическими компонентами программных модулей, операционными системами, звуковые, оконные интерфейсы и т.д.

В большинстве процедурных языков программирования API определяет набор функций, выполняющих определенные задачи или имеющих доступ к определенному компоненту программного обеспечения. API интерфейс представляется как коллекция файлов включения, используемых программой. В объектно-ориентированных языках API объекта представляет собой описание того, как данный объект функционирует в данном языке. Обычно имеет представление в виде набора классов со связанными списками методов.

Также необходимым интерфейсом для реализации проекта является Maps API (API карт). Для этого можно воспользоваться API Яндекс Карт, GoogleMapsAPIs или другими. Однако у данных сервисов есть существенные недостатки, которые не позволяют в полной мере использовать данные сервисы: API Яндекс карт не имеет возможности навигации внутри здания, GoogleMapsAPIs имеет данную возможность, но она закрыта для общего доступа.

В связи с этим, мы рассмотрим возможности HERE Technologies API и функционал Indoor навигации.

Indoor навигация (система внутреннего позиционирования) – локальная система определения местоположения внутри зданий и закрытых сооружений, где практически недоступна спутниковая система навигации.

1.2 Современные средства реализации проекта

Достаточное внимание необходимо уделять выбору инструментов и технологий создания web-приложения, так как использование правильных технологий и инструментов может значительно повлиять на процесс разработки и конечный продукт.

Технологии для реализации проекта были выбраны исходя из некоторых принципов:

- программное обеспечение с открытым исходным кодом
- наличие возможности редактирования и тестирования программы на локальной машине

Для реализации проекта, кроме API будут использованы язык программирования JavaScript, языки разметки HTML, каскадные таблицы стилей CSS, а также программа камеральной обработки QGIS.

HTML является языком разметки гипертекстовых документов. Он отвечает за расположение на странице текстов, рисунков, таблиц и других объектов. На данный момент используется несколько версий HTML: наиболее распространена версия HTML 4.01, а более мощная и новая, черновая спецификация HTML5 обретает популярность и получает все большую поддержку в браузерах.[1]

К преимуществам HTML языка можно отнести:

1. возможность отображения HTML-страницы в разных браузерах
2. наличие почти абсолютной невозможности взлома, в связи с отсутствием базы данных и файла конфигурации
3. все теги прописываются создателем сайта
4. отсутствие необходимости делать периодический backup системы
5. полный контроль над сайтом

HTML – не язык программирования, а разметки, он создает систему для идентификации и описания различных компонентов документа, таких как заголовки, абзацы и списки. Команды используемые в коде называются дескрипторами, чаще тегами. Данные команды не видны читателю, зато хорошо

видны браузеру. Команды пишутся с помощью специальных скобок “<...>”.[3]

Ниже приведен код HTML простой веб-страницы

```
<!doctype html>
<html>
<head>
<meta charset = “UTF-8”>
<title>Это заголовок веб-страницы</title>
</head>
<body>
<p>А это абзац этой страницы</p>
</body>
</html>
```

В приведенном примере, как и в любом HTML коде, большинство команд-тегов используются парами, начиная и завершая какой-то фрагмент. Закрывающий тег всегда пишется с прямым слешем (/) после первого символа скобки (<).

На любой веб-странице обычно имеется как минимум четыре основных элемента:[2]

- Самая первая строка содержит объявление типа документа – <!DOCTYPE>. Указать тип текущего документа необходимо, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу, поскольку существует несколько различных версий HTML, различающихся по синтаксису.
- Далее идет тег <html>, он является контейнером, который заключает в себе все содержимое веб-страницы, включая все остальные теги.
- Тег <head> предназначен для хранения других элементов, цель которых – помочь браузеру в работе с данными. Данные элементы невидимы при просмотре веб-страницы. Также внутри контейнера <head> находятся метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и

других данных. Также в этом теге подключаются библиотеки JavaScript и код со стилями CSS.

- Элемент <body>, предназначен для хранения содержания веб-страницы, отображаемого в окне браузера. Информацию, которую следует выводить на странице, необходимо располагать именно внутри контейнера <body>. К такой информации относится текст, изображения, теги, скрипты JavaScript и т.д.

Внутри тега <body>, можно найти следующие теги:

- - открывающий тег начинает абзац, а закрывающий - заканчивает.
- <a> - данный тег предназначен для создания ссылок, при нажатии на которую можно переместиться в другое место веб-страницы или на другой сайт.

В то время как HTML используется, чтобы описать содержимое веб-страницы, каскадные таблицы стилей (CSS) влияют на то, как выглядит приложение. Это означает, что шрифтами, цветами, фоновыми изображениями, интервалами между строками, расположением объектов на страницы и прочим управляет CSS. С помощью последних версий CSS вы можете добавлять на страницу даже простую анимацию.[6]

CSS работает вместе с HTML, но не имеет к нему никакого отношения. Создав стиль один раз, можно применять его к текстовым фрагментам, изображениям, заголовкам и любым другим элементам страницы сколько угодно. Например, вы можете выбрать часть текста и применить к нему стиль, тут же изменяющий размер, шрифт и цвет текста.

До появления CSS дизайнеры веб-страниц были ограничены в возможностях оформления и разметки языка HTML. Если сравнивать страницы написанные до появления CSS и после, будет видна значительная разница в возможностях.[4]

Чтобы создавать веб-страницы на языках HTML и CSS, вполне достаточно обычного текстового редактора, такого как **Блокнот** в Windows или

Text Edit в OS X.[5] Однако существуют специальные редакторы, разработанные для ускорения процесса написания кода вручную.

В настоящее время, HTML все больше теряет свою популярность, новые теги уже не нужны, так как вполне достаточно существующих, к тому же акцент создания веб-страниц сместился на стили, которые дают куда больше возможностей по оформлению.

Естественно, CSS не может заменить собой HTML, но зато позволяет использовать небольшой набор тегов, а вид, положение и различные параметры элементов, задавать через стили.

Теги и иерархическая система HTML жестко описана в спецификациях, что накладывает некоторые ограничения. Поэтому все большую популярность набирает XML, с помощью которого можно создавать собственные теги и формировать их структуру. Это не единственное отличие XML от HTML. При работе с HTML браузер пропускает различные мелкие недочеты в структуре кода или то, что неправильно указан атрибут. В XML, если документ сформирован неправильно, браузер выдаст ошибку.[9, 7]

Очень трудно сразу перейти на новый язык программирования, поэтому, чтобы помочь разработчикам изменить их стиль написания кода, а также уменьшить сократить разницу между HTML и XML, был разработан XHTML, как промежуточный этап.

XHTML (EXtensible HyperText Markup Language, расширяемый язык разметки гипертекста) является заменой HTML и считается более строгой версией.[8]

Если подумать о неком идеальном коде веб-страницы, то его можно сравнить с программой, которая не будет запускаться, пока все ошибки не исправлены. XHTML, сохраняя все преимущества HTML, вносит более строгие правила написания кода, поэтому пока документ не соответствует спецификациям, браузер не будет его отображать. Благодаря этому, сайт будет корректно функционировать во всех современных браузерах и на платформах вроде смартфонов, компьютеров, ноутбуков и т.д. [25]

Следующим поколением стандартов HTML 4.01, XHTML 1.0 и XHTML 1.1. - является HTML5. Он представляет собой совершенно новые функциональные возможности, необходимые современным веб-приложениям.

Если XHTML намеревался отбросить все то, что было до него, то HTML5 основывается на уже существующих спецификациях и реализациях. Большая часть HTML 4.01 вошла в HTML5.

Определяющим фактором в разработке HTML5 стало постоянное внутреннее напряжение. С одной стороны, эта спецификация должна быть достаточно мощной, чтобы поддерживать создание веб-приложений. С другой, HTML5 должна поддерживать существующее содержимое даже с учетом того, что большая часть существующего содержимого – неудобоваримая каша. Если спецификация слишком отклонится в одном направлении, ее постигнет та же судьба, что и XHTML 2. XHTML 2.0 стал новым языком разметки, созданный, чтобы обойти ограничения XHTML, а не просто стать новой версией языков разметки. Но если она пойдет слишком далеко в другом направлении, тогда выйдет, что спецификация будет рекомендовать использование тегов <font> и таблиц для разметки – поскольку в конце концов именно с использованием этих приемов построено огромное количество веб-страниц. Здесь нужно выдержать очень тонкий баланс, и это требует прагматического, уравновешенного подхода. [6]

Впервые в истории HTML, спецификация HTML5 объявляет, что браузер должен делать, если ему попались документы с ошибками разметки. Если предыдущие спецификации разметки писались для авторов, то HTML5 написан и для авторов, и для разработчиков реализаций.

Многие языки программирования, например Python, обязывают писать инструкции специфическим образом. Пробелы и переносы строк имеют значение при написании кода. Некоторые языки программирования, такие как JavaScript, не обращают никакого внимания на формирование - совершенно не важно сколько пробелов в начале строки.

Для разметки не важны пробелы и переносы строк. Если вам хочется поставить перенос строки и отступ при каждом вложенном элементе, пожалуйста, но ни браузер, ни валидатор не требуют этого. Однако некоторые версии разметки обязывают к более строгому стилю написания.

До XHTML 1.0 не имело никакого значения, пишете вы теги в верхнем или нижнем регистре. Не имело значения, ставили ли вы атрибуты в кавычки или нет. Для некоторых элементов даже не имело значения, ставите ли вы закрывающий тег.

В XHTML 1.0 вы обязаны следовать синтаксису XML. Все теги должны быть написаны в нижнем регистре. Все атрибуты должны быть в кавычках. Все элементы должны иметь закрывающий тег.

При использовании XHTML, вас обязывали следовать определенному, строгому специфическому стилю. С нетребовательностью синтаксиса HTML5, вы сами обязываете себя писать в том стиле, который считаете необходимым.

В предыдущих версиях HTML, существовал процесс под названием “исключение”. Он заключался в удалении существующего элемента или атрибута. Веб-разработчикам рекомендовалось не использовать данные элементы.

HTML5 не имеет исключенных элементов или атрибутов, однако есть огромное количество устаревших элементов. “Устаревший” имеет иное значение, нежели “исключенный”. Это приводит к немного странным ситуациям, когда авторам говорят не пользоваться данным элементом, а браузеры должны отображать его.

Одним из самых ранних добавлений к HTML стал элемент img, и это очень сильно изменило веб-приложения. Появление JavaScript позволило вебу стать более “живой” средой. Наконец, быстрый рост количества Ajax-приложений позволил вебу стать средой, в которой возможны полноценные приложения.

В настоящее время, веб-стандарты продвинулись настолько, что можно создать практически любое приложение, используя лишь JavaScript, HTML и CSS.

В разнообразном наборе веб-стандартов существуют пробелы. Например если вы хотите добавить текст и картинки, вам достаточно HTML и CSS. Однако если вам нужно аудио и видео, вам понадобится использовать специальные плагины, такие как Flash или Silverlight.

HTML5 заполняет эти пробелы. По сути, язык становится конкурентом таких технологий, как Flash и Silverlight. Но вместо того чтобы быть зависимыми от плагинов, мультимедиа-элементы в HTML5 являются встроенными в браузеры.

В данном проекте мы будем использовать HTML 4.01, так как на данный момент это самая распространенная версия.

Еще одним необходимым элементом создания приложения является язык программирования JavaScript.

JavaScript – язык сценариев, который наделяет веб-страницы интерактивностью и вариантами поведения. Данный язык используется для управления элементами на веб-странице, примененными к ним стилями, или даже самим браузером. Существуют и другие языки веб-сценариев, однако JavaScript стандартизирован и наиболее широко распространен. Программы на нем называются скриптами. В браузере они подключаются напрямую к HTML и выполняются вместе с загрузкой страницы. [11]

Для выполнения программ в браузерах для JavaScript применяется подход интерпретации. Интерпретация – это когда исходный код программы получает другой инструмент, который называют «интерпретатор», и выполняет его «как есть».[14] При этом распространяется именно сам исходный код (скрипт).

Современные интерпретаторы перед выполнением преобразуют JavaScript в машинный код, оптимизируют, а уже затем выполняют. И даже

во время выполнения стараются оптимизировать. Поэтому JavaScript работает быстро.

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице. Но, разумеется, JavaScript можно использовать не только в браузере. Это полноценный язык, программы на котором можно запускать и на сервере, и на любом мобильном устройстве, где установлен соответствующий интерпретатор.

Современный JavaScript – это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.[22]

Большинство возможностей JavaScript в браузере ограничено текущим окном и страницей. (Рисунок 1.1)

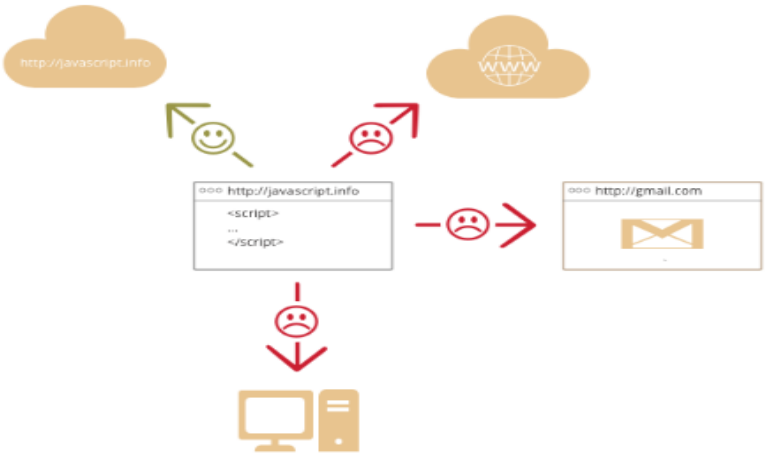

Рисунок 1.1

- JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе.
- JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).

- Из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т. к. и здесь есть ограничения безопасности.

Что же касается возможностей – они зависят от окружения, в котором запущен JavaScript. В браузере JavaScript умеет делать всё, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:[12]

- Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.
- Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.
- Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX").
- Получать и устанавливать cookie, запрашивать данные, выводить сообщения и т.п.

Сам язык JavaScript совершенствуется. Современный стандарт ECMAScript 5 включает в себя новые возможности для разработки, ECMAScript 6 будет большим шагом в улучшении синтаксиса языка.

Современные браузеры улучшают свои движки, чтобы увеличить скорость исполнения JavaScript, исправляют баги и стараются следовать стандартам.

Очень важно, что новые стандарты HTML5 и ECMAScript сохраняют совместимость с предыдущими версиями. Это позволяет избежать ошибок с уже существующими приложениями.

Помимо JavaScript, на страницах используются и другие технологии. Связка с ними поможет добиться более интересных результатов в тех местах, где возможностей JavaScript может быть не достаточно.[26]

JAVA. Java – язык общего назначения, на нем можно писать самые разнообразные программы. Для интернет-страниц есть особая возможность – написание *апплетов.*

Апплет – это программа на языке Java, которую можно подключить к HTML при помощи тега applet.

В браузере Java-апплет, выполняет все тоже самое что и программа установленная на компьютере. Однако данная технология имеет несколько недостатков:

1. Для загрузки требуется больше времени
2. Для корректной работы, на компьютере пользователя должна быть установлена среда выполнения Java, включая плагин для браузера
3. Java-апплет выполняется отдельно от HTML-страницы

Плагины и расширения для браузера.

Используя JavaScript или C, есть возможность написать плагины (программный модуль компилируемый независимо и динамически подключаемый к программе для расширения ее возможностей). Данные плагины могут как отображать содержимое определенных форматов, так и взаимодействовать со страницей.

Вместе с JavaScript на на веб-страницах можно часто увидеть систему AJAX. Эта технология позволяет обращаться к серверу без перезагрузки страницы.[13]

Примеры использования AJAX:

- Элементы интерфейса

AJAX крайне полезен для кнопок и форм, связанных с элементарными действиями, например добавление в корзину.

В настоящее время, такие действия на сайтах осуществляются без перезагрузки страницы.

- Динамическая подгрузка данных
- Живой поиск

Живой поиск является классическим примером использования данной технологии. Когда пользователь начинает печатать запрос, JavaScript предлагает возможные варианты, получая список вероятных вариантов с сервера

- С помощью AJAX, вы можете обмениваться любой информацией с сервером

Обычно используются форматы:

- JSON – для отправки и получения структурированных данных, объектов.
- XML – если ваш сервер работает в формате XML
- HTML/текст – можно и просто загрузить с сервера код HTML или текст для показа на странице.
- Бинарные данные, файлы – гораздо реже, в современных браузерах есть удобные средства для них.

Также существует термин описывающая различные техники получения данных по инициативе сервера. Такой термин называется COMET.

AJAX отправляет запрос и после этого получает результат, в то время как COMET - это непрерывный канал, по которому приходят данные.

Примеры COMET-приложений:

- Чат – пользователь сидит и смотрит, что пишут другие. Новые сообщения приходят без обновления окна чата.
- Аукцион – человек смотрит на экран и видит, как обновляется текущая ставка за товар.
- Интерфейс редактирования – когда один редактор начинает изменять документ, другие видят информацию об этом. Возможно и совместное редактирование, когда редакторы видят изменения друг друга.

Сценарий DOM или сценарий объектной модели документа, используемый в отношении JavaScript. Технология DOM является сокращением термина объектная модель документа и обращается к стандартизированному списку элементов веб-страницы, к которым можно получить доступ и управлять ими, используя JavaScript.

Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами. Для представления текста создаются узлы

с типом «текст». DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript.

Например для данного документа

<!DOCTYPE HTML>

<html>

<head>

<title>О лосях</title>

</head>

<body>

Правда о лосях

</body>

</html>

Дерево DOM будет выглядеть следующим образом (Рисунок 1.2).

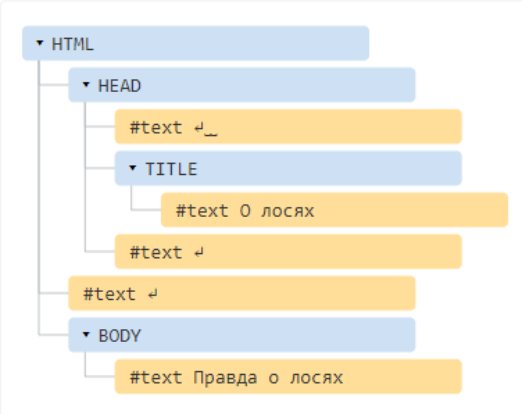

Рисунок 1.2

В этом дереве выделено два типа узлов.

- Теги образуют *узлы-элементы* (element node). Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.
- Текст внутри элементов образует *текстовые узлы* (text node), обозначенные как #text. Текстовый узел содержит исключительно строку текста и не может иметь потомков, то есть он всегда на самом нижнем уровне.

Всё, что есть в HTML, находится и в DOM.

Даже директива <!DOCTYPE…>, которую мы ставим в начале HTML, тоже является DOM-узлом, и находится в дереве DOM непосредственно перед <html>. На иллюстрациях выше этот факт скрыт, поскольку мы с этим узлом работать не будем, он никогда не нужен. Даже сам объект document, формально, является DOM-узлом, самым-самым корневым.

Всего различают 12 типов узлов, но на практике мы работаем с четырьмя из них:

- Документ – точка входа в DOM.
- Элементы – основные строительные блоки.
- Текстовые узлы – содержат, собственно, текст.

Комментарии – иногда в них можно включить информацию, которая не будет показана, но доступна из JS.

Выводы.

В данном разделе проводится исследование предметной области и анализ современных средств. В результате анализа были выбраны необходимые для реализации проекта современные средства, такие как HTML, CSS и JavaScript. Каждый из этих элементов имеет набор необходимых функций для корректной работы приложения.

Также в результате анализа был выбран инструмент для быстрой и точной оцифровки чертежей этажей первого и второго корпусов РГГМУ. Данный инструмент позволяет просматривать и накладывать друг на друга векторные и растровые данные в различных форматах и проекциях без преобразования во внутренний или общий формат. С помощью данной функции мы можем добавлять чертежи каждого этажа, обрабатывать их, а затем накладывать друг на друга.

Глава 2. Реализация проекта

2.1 Концептуальная схема Web-приложения

Наиболее известными методами проектирования и разработки web-приложений являются:

- WebML - метод разработки и язык Web Modeling Language.
- WSDM - один из первых методов разработки веб-приложений Web Site Design Method.

Метод WebML является подходом к разработке web- приложений на основе модели. Его основной вклад состоит в разработке набора понятий, обозначений и методик для создания web- приложений, активно использующих данные, которые могут применяться командами разработчиков для поддержки всех видов работ жизненного цикла приложений – от анализа до развертывания и развития.[27]

Методика WebML объединяет традиционные приемы, хорошо известные разработчикам, такие как сценарии использования на языке UML и концептуальное проектирование данных с помощью модели Entity-Relationship, с новыми понятиями и методами для проектирования гипертекстов, которые являются важными для web-приложений.

Рисунок 2.1. Этапы модели разработки WebML

В процессе WebML разные этапы повторяются и уточняются до тех пор, пока не будут получены результаты, полностью соответствующие требованиям к приложению. В связи с этим жизненный цикл продукта проходит в

несколько повторений, на каждом из которых создаются прототипы или частичные версии приложения. На каждом повторе текущая версия приложения проверяется и оценивается, а затем расширяется или модифицируется, чтобы удовлетворять уже собранным и вновь появившимся требованиям.

Анализ требований WebML состоит в выполнении следующих этапов:

- Выявление групп пользователей, для которых разрабатывается приложение.
- Спецификация функциональных требований, которые связаны с функциями, предоставляемыми пользователям.
- Выявление базовых информационных объектов, т. е. основных информационных активов, к которым может быть предоставлен доступ пользователям и которыми он может манипулировать.
- Разные web-страницы, спроектированные таким образом, чтобы удовлетворять хорошо описанному набору функциональных требований и требований пользователей.

Проектирование web-приложения включает в себя виды работ:

- Проектирование данных: соответствует преобразованию базовых информационных объектов, выявленных в ходе анализа требований, в полную и согласованную схему данных.
- Проектирование гипертекста: создаются схемы представлений на основе ранее описанной схемы данных.

2.2 Процесс камеральной обработки

Процесс камеральной обработки проходит в программе по обработке пространственно-временных данных QGIS. Данная программа работает на Windows, Linux, Mac OSX и Android, обладает широкими возможностями и поддерживает множество векторных и растровых форматов, баз данных. Целью создания QGIS было сделать использование геоинформационных систем легким и понятным для пользователя. Программа имеет крайне простой в ис-

пользовании интерфейс, а также бесплатный доступ к большинству функций, в отличие от аналогов (например ArcGis).

Основные возможности системы:

1. Поддержка таблиц PostGIS с пространственными данными;
2. Поддержка форматов shapefiles (шейпфайлы), покрытий ArcInfo, файлов Mapinfo, и других форматов, доступных через OGR;
3. Поддержка растров;
4. Идентификация объектов;
5. Отображение атрибутивных таблиц;
6. Возможности выбора объектов;
7. Экспорт в map-файл Mapserver.

В программе есть возможность просматривать и накладывать друг на друга векторные и растровые данные в различных форматах и проекциях без преобразования во внутренний или общий формат. С помощью данной функции мы можем добавлять чертежи каждого этажа, обрабатывать их, а затем накладывать друг на друга. После обработки данных есть возможность привязать чертежи к геопозиции. Программа поддерживает множество форматов для сохранения, в том числе .geojson, что позволяет в дальнейшем добавлять эти данные на карту. [24]

Процесс оцифровки каждого этажа в программе QGIS включает в себя несколько основных этапов.

Для начала необходимо установить дополнительные модули: QuickMapServices (расширение, позволяющее удобно и быстро работать с подложками, полученными из разных интернет-сервисов) и Advanced Digitizing Tool (расширенный инструмент оцифровки). С помощью первого загружаем карту мира, находим нужный город и место (Рисунок 2.2, 2.3), второй необходим для точной оцифровки.

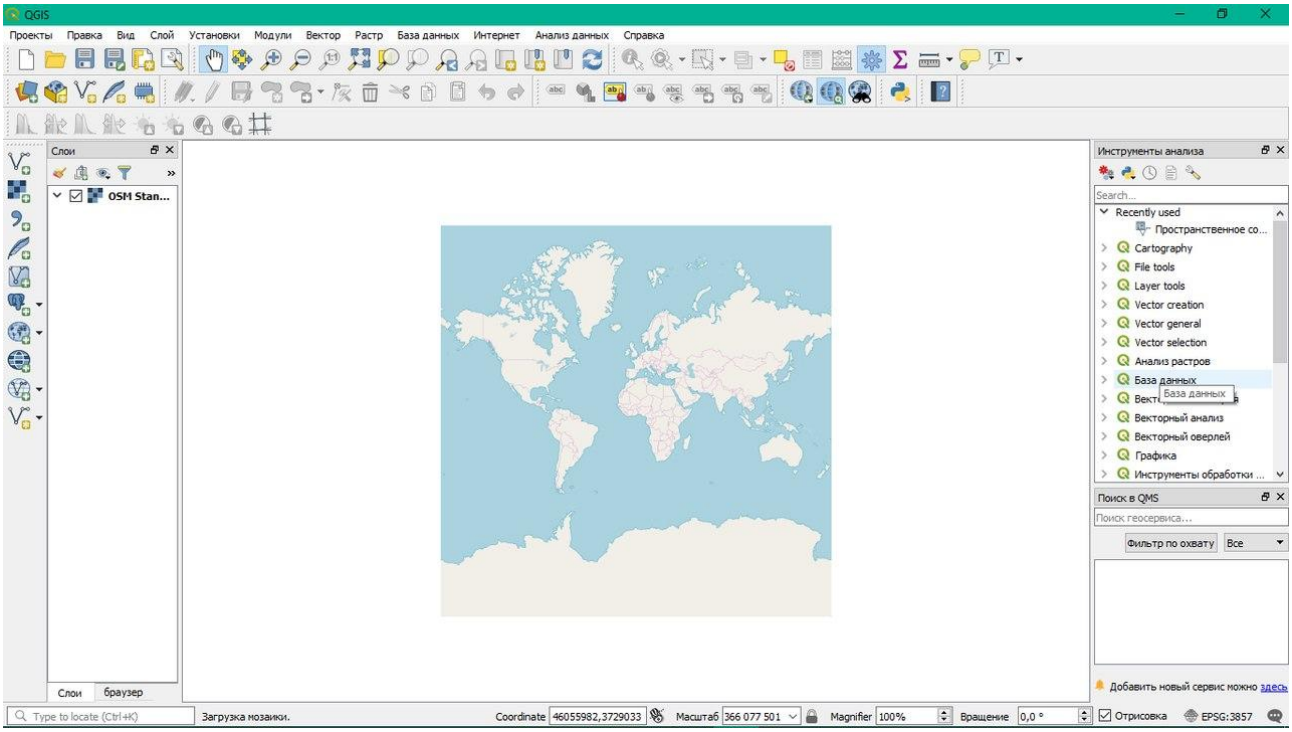

Рисунок 2.2.

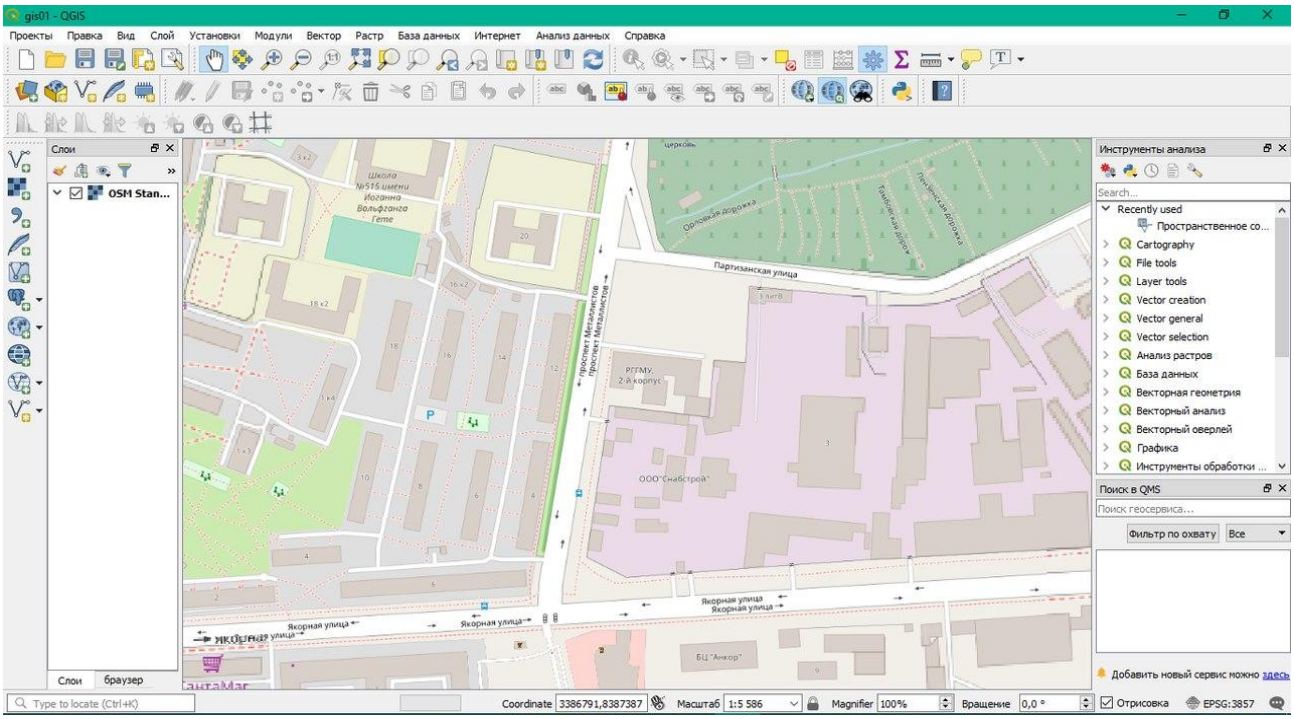

Рисунок 2.3

Плюсы использования QuickMapServices:

1. имеет готовый список адресов сервисов;
2. умеет добавлять подложки в один клик;

3. не испытывает проблем с масштабированием надписей на нестандартных масштабах;

4. при необходимости, решает проблему отображения подписей на нестандартных масштабах;

5. дает возможность легко расширить список сервисов путем добавления простых ini файлов.

Далее выполняем следующие этапы:

1. Подгружаем план здания (Рисунок 2.4)

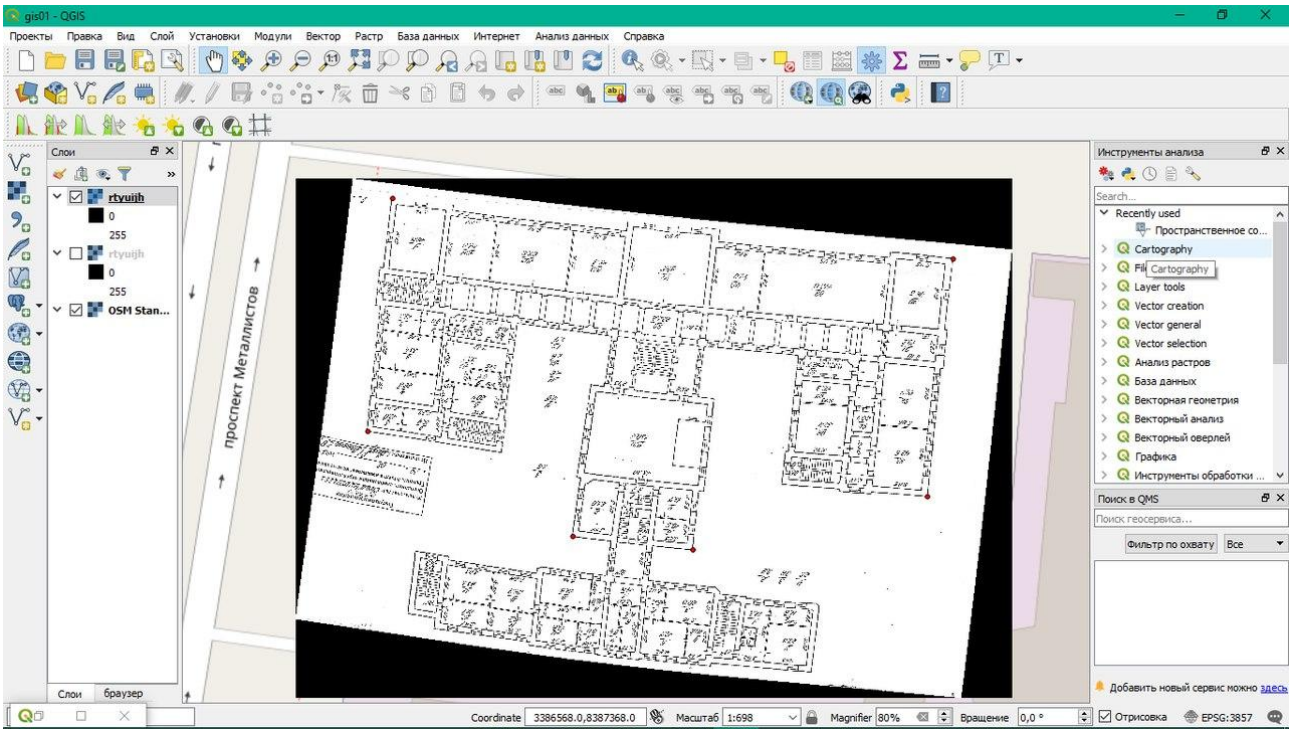

Рисунок 2.4

2. С помощью функции привязки растра добавляем точки с координатами (координаты берем с самой карты), получаем наложение плана здания на карту мира (Рисунок 2.5)

Благодаря данной функции мы можем подключать к нашему проекту любые растровые данные.

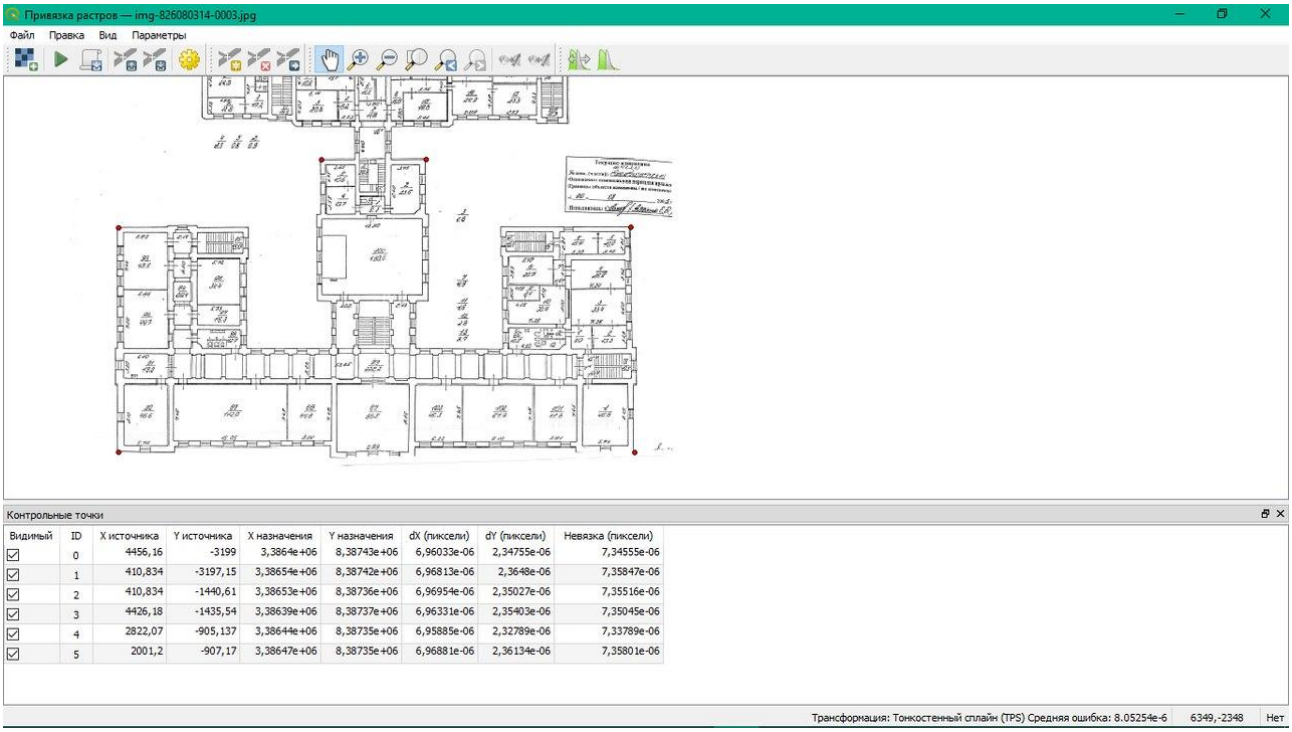

Рисунок 2.5

3. Добавляем растровый слой полигон (Рисунок 2.6). Используя кнопки “Создать shape-файл”, задаем параметры: Имя файла - {‘name’}; Тип геометрии - полигон; Список полей для атрибутивной таблицы - тип поля и количество символов.

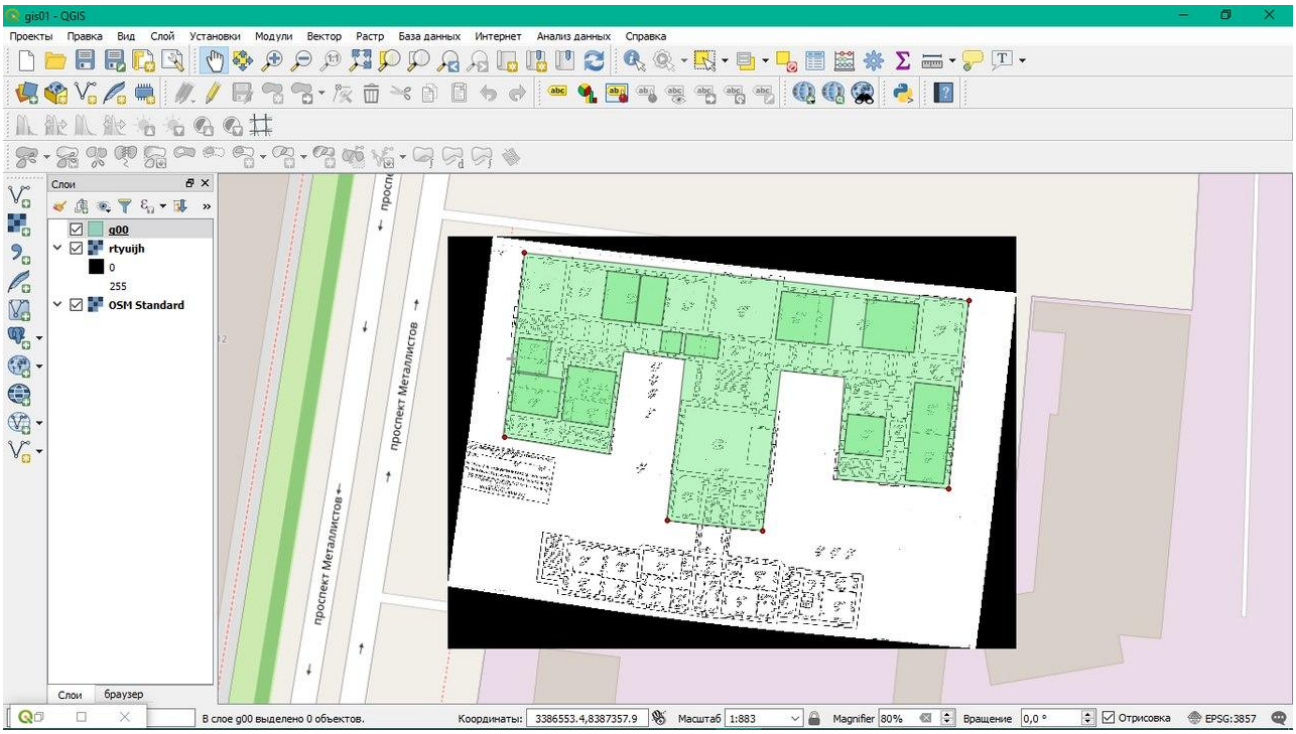

Рисунок 2.6

Далее оцифровываем периметр здания, чтобы оно было единым целым, с помощью кнопки “Добавить полигон” в режиме редактирования объекта. Затем в этом же слое создаем новые объекты: полигоны аудиторий.

Для каждого полигонального слоя продумываем поля атрибутивной таблицы: Название аудитории, Номер этажа, Номер корпуса. Данные из атрибутивных таблиц будут использованы при интеграции оцифрованных чертежей на карту.

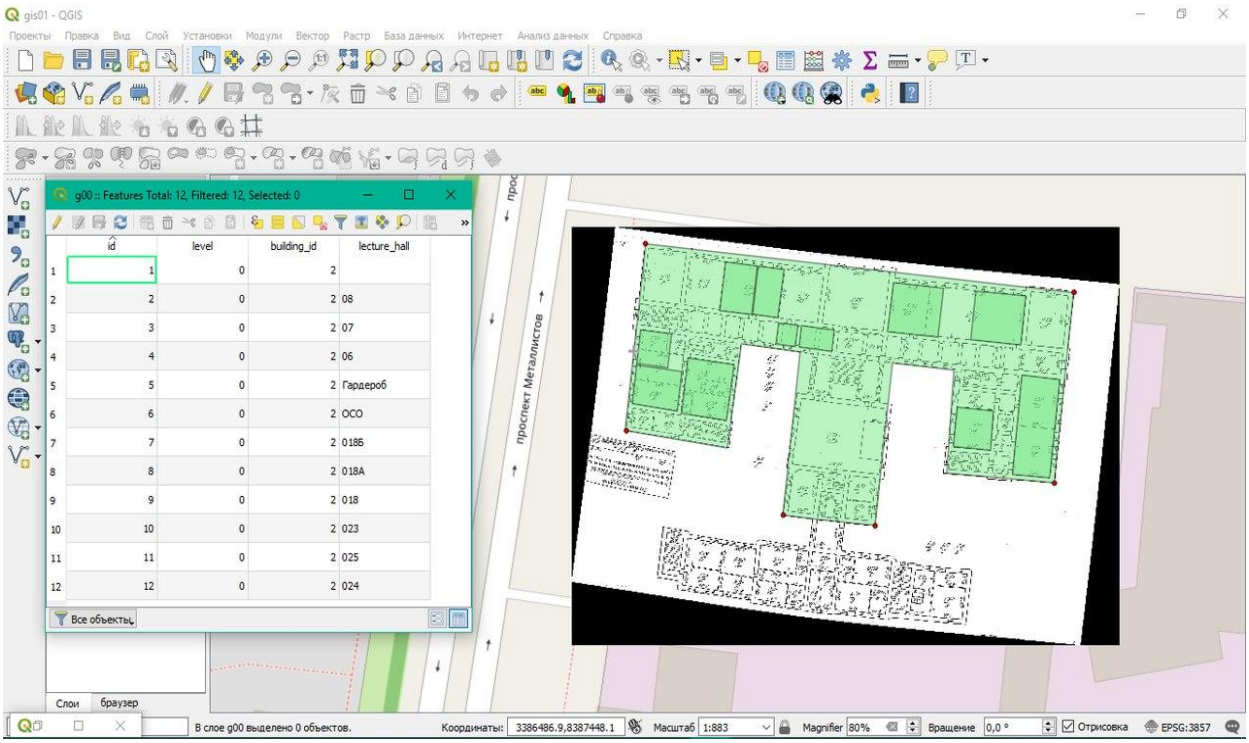

Рисунок 2.7 Таблица атрибутов

Выполнив все необходимые действия, получаем готовую подложку. (Рисунок 2.8)

Для остальных этажей выполняем описанные выше действия. Оцифровав каждый этаж получаем единый план, привязанный к точке на карте. После оцифровки всех этажей, нам необходимо сохранить все слои в формате .geojson. Это необходимо для корректного добавления данных слоев на карту.

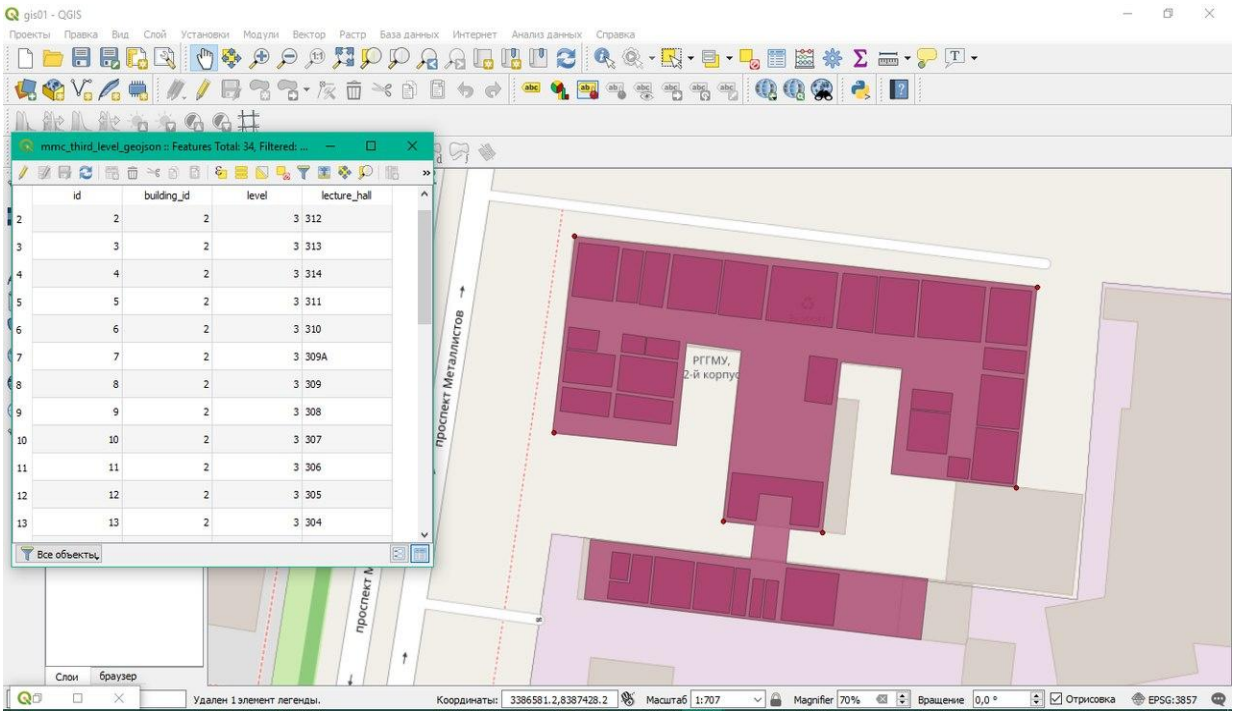

Рисунок 2.8

2.3 API и MapsAPI

Одним из наиважнейших элементов проекта является API. Аббревиатура расшифровывается как Application Programming Interface, или интерфейс для программирования приложений.

Первым шагом при создании любого приложения, основанного на API-интерфейсе с JavaScript, является загрузка необходимых библиотек и модулей кода. Для реализации базового варианта использования, нам потребуется два модуля: ядро(core) и сервис, обслуживание (service). Пример программного кода для подключения этих элементов приведен ниже.

```
<script src="http://js.api.here.com/v3/3.0/mapsjs-core.js"
type="text/javascript" charset="utf-8"></script>
<script src="http://js.api.here.com/v3/3.0/mapsjs-service.js"
type="text/javascript" charset="utf-8"></script>
```

Ядро или core (mapsjs-core.js) - это модуль содержащий основной функционал для рендеринга карт, их слоев и объектов, а также утилит, используемых в других модулях.Этот модуль формирует ядро API и от него зависят все другие модули.

Сервисы или service (mapsjs-service.js) - это модуль отвечающий за поиск фрагментов карты, маршрутизацию, геокодирование и т.п.

URL-адрес а атрибуте “src” отображает номер последней версии API. Этот номер изменяется с каждой новой версией, так что необходимо следить, чтобы не возникало ошибок в программе.

Приложение может быть использовано не только на компьютерах или ноутбуках, но и на мобильных устройствах. Чтобы обеспечить оптимальную производительность и корректное отображение на последних, необходимо добавить специальный мета-тег.

```
<meta name="viewport" content="initial-scale=1.0,
width=device-width" />
```

<meta> определяет метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Разрешается использовать более чем один метатег, все они размещаются в контейнере <head>.

Затем нам необходимо инициализировать связь с внутренними службами. Установление данной связи является важной частью создания приложения с использованием API-интерфейсов. В нашем случае, внутренние службы обрабатывают запросы на изображениякарт и пересылку их в приложение для отображения.

Для инициализации внутренних служб нам необходимо получить персональные данные аутентификации и авторизации. После этого необходимо инициализировать переменную Platform используя свои данные ‘app_id’ и ‘app_code’.

```
var platform = new H.service.Platform({
'app_id': '{YOUR_APP_ID}',
'app_code': '{YOUR_APP_CODE}'
});
```

Важно инициализировать объект Platform не только для авторизации и проверки подлинности, но и для использования экземпляра тестирования интеграции клиента платформы HERE или для безопасного HTTP (HTTPS) при взаимодействии с серверной частью. Кроме того, объект предоставляет методы, которые позволяют легко создавать полностью рабочие заглушки сервиса, такие как заглушки листов карты, заглушки сервиса маршрутизации и т.д.

Следующим этапом, будет инициализация карты. Для начала добавим не интерактивную карту, ориентированную на предопределенное местоположение с фиксированным уровнем масштабирования. Реализация включает в себя несколько этапов:

1. Создание элемента контейнера (Container) HTML, в котором можно отобразить карту (например, div)
2. Создаем объект H.Map, указав в нем:

- элемент контейнера карты
- тип карты для использования
- уровень масштабирования для отображения карты
- географические координаты точки, на которую будет центрирована карта

Ниже приведен пример кода, который устанавливает объект карты (H.Map), нормальный тип карты, уровень масштабирования 15 и центр карты - Санкт-Петербург, Россия, по широте 59,921 и долготе 30,408

```
var map = new H.Map(
document.getElementById('map'),
defaultLayers.normal.map,
{
zoom:15,
center:{lat:59.92158020,lng:30.40811749}
}
```

Объединив все элементы кода, мы получаем не интерактивную карту, с центром в определенном месте, указанном с помощью координат. (Рисунок 2.9)

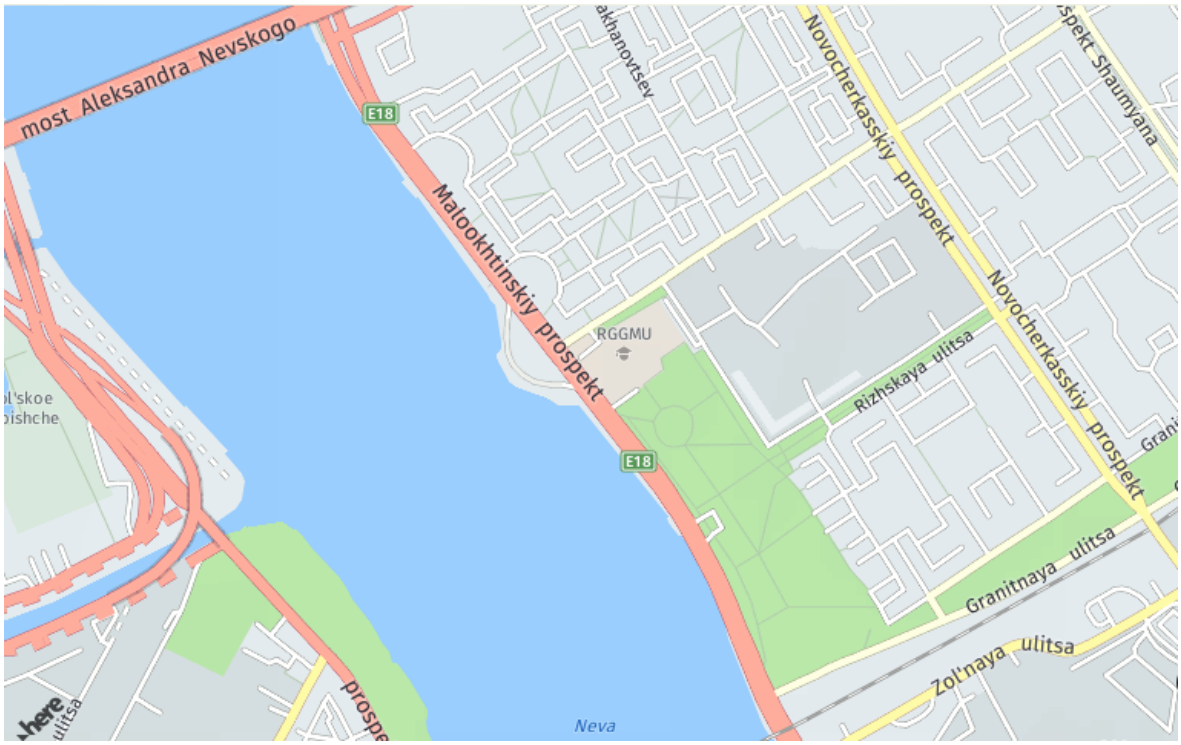

Рисунок 2.9

Далее необходимо сделать карту интерактивной и добавить основной пользовательский интерфейс. Для этого подключаем еще один модуль. API-интерфейс HERE содержит набор готовых элементов управления картой через модуль пользовательского интерфейса (mapsjs-ui.js). Данный модуль предоставляет набор предварительно настроенных компонентов интерфейса браузера, таких как базовые настройки карты, управление масштабированием и масштаб карты, которые можно добавить на карту.

Первый шагом является добавление тега <script> к элементу <head> для загрузки модуля интерфейса API, также добавляется ссылка на файл CSS, который содержит в себе стили данного интерфейса.[28]

```
<script src="http://js.api.here.com/v3/3.0/mapsjs-ui.js"
type="text/javascript" charset="utf-8"></script>
<link rel="stylesheet" type="text/css"
href="http://js.api.here.com/v3/3.0/mapsjs-ui.css" />
```

Затем создаем объект карты со стандартными типами в разделе <script> для правильного отображения логики приложения. Полный код, включая уже написанные элементы, приведен ниже.

```
<script type="text/javascript">
var platform = new H.service.Platform({
'app_id': '{YOUR_APP_ID}',
'app_code': '{YOUR_APP_CODE}'
});
var defaultLayers = platform.createDefaultLayers();
var map = new H.Map(
document.getElementById('mapContainer'),
defaultLayers.normal.map,
{
zoom:15,
center:{lat:59.92158020,lng:30.40811749}
});
var ui = H.ui.UI.createDefault(map, defaultLayers);
</script>
```

Ниже приведена карта, с добавленным интерфейсом. (Рисунок 2.10)

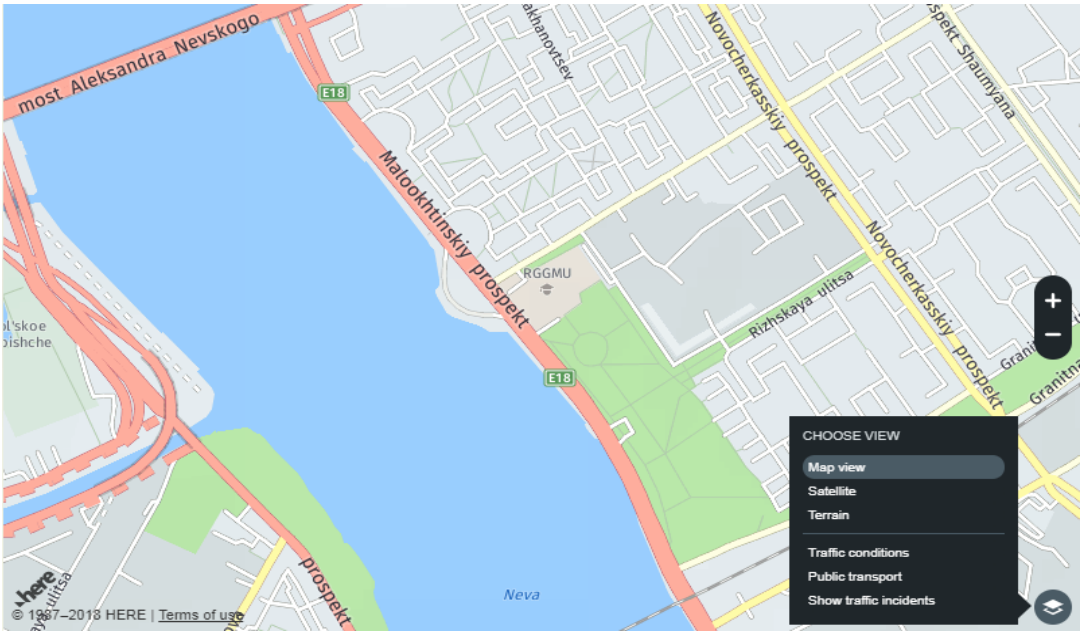

Рисунок 2.10

Также мы можем изменить язык интерфейса на карте. Для этого необходимо поменять идентификатор языка с помощью определенной части кода.

var ui = H.ui.UI.createDefault(map, defaultLayers, 'ru-RU');

Ниже приведен измененный язык интерфейса. (Рисунок 2.11)

Теперь добавим возможность поиска своего местоположения. Для начала нам необходимо создать кнопку, с помощью которой будет работать данный сценарий. С помощью атрибута div создаем контейнер для кнопки и затем задаем ей стили. Ниже приведен код

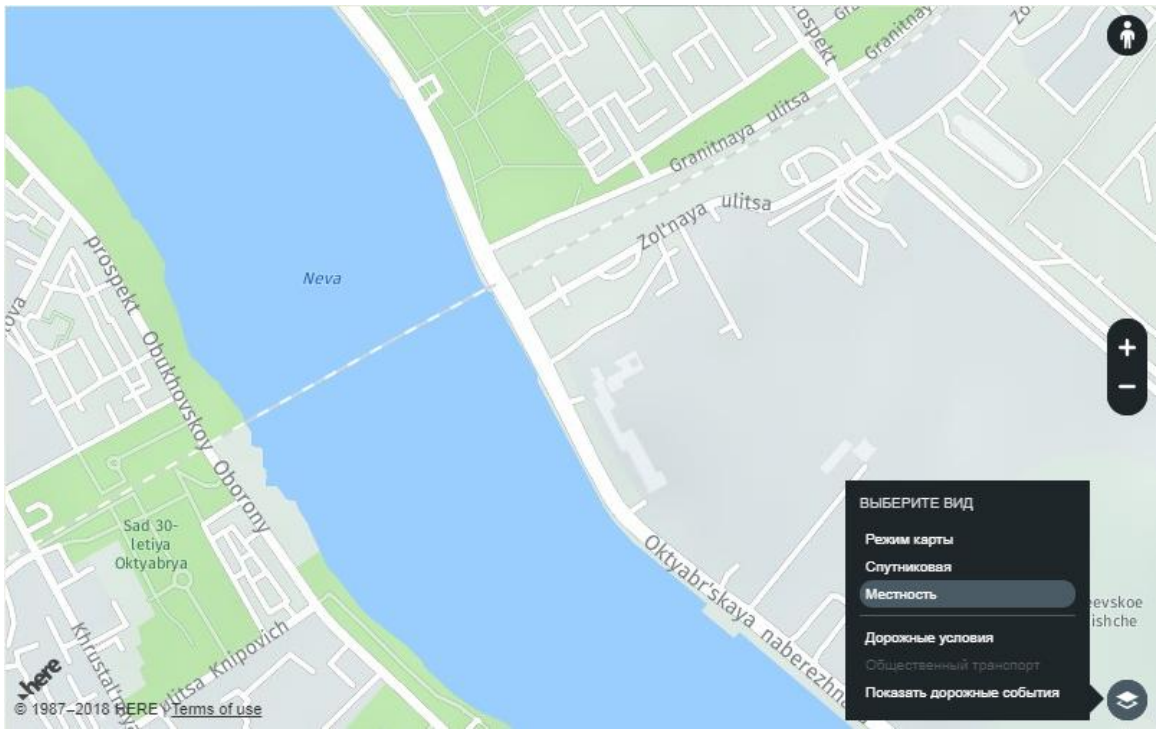

Рисунок 2.11

** Geo **

#geolocation{

position: absolute;

top: 780px;

right: 625px;

width: auto;

height: 20px;

background-color: #1f262a;

border-radius: 22px;

```
color: #fff;
padding: 2px;
cursor: pointer;
}
#geolocation:hover{
color: #1f262a;
background-color: white;
border: 1px solid #1f262a;
}
.active{
color: #1f262a !important;
background-color: white !important;
border: 1px solid #1f262a !important;
}
```

Далее пишем программу, при выполнении которой, при нажатии на кнопку, будет определяться наше местоположение. Пример программы приведен ниже.

```
var onClick = function(){
if (geoloc.className != "active") {
geoloc.className = "active";
navigator.geolocation.getCurrentPosition(function(position){
window.marker = new H.map.Marker
({lat:position.coords.latitude,lng:position.coords.longitude,});
map.addObject(marker);
})}
else{
geoloc.className = "";
map.removeObject(marker);
}};
geoloc.addEventListener("click",onClick);
```

В результате оцифровки этажей, получаем 4 файла для трех этажей и подвала: «first_level.geojson», «second_level.geojson», «third_level.geojson», «underground_level.geojson». Теперь необходимо выбрать, каким образом лучше хранить пространственные данные.

Этот вопрос зависит от уровня системы, если нам необходимо написать полноценный проект с сервером и базой данных, то можно использовать фреймворк GeoDjango и надстройку PostGis для PostgreSQL.[23]

Однако есть более простой вариант - использовать бибилотеки nodejs и mongodb для хранения файлов в формате .geojson.

Для дальнейшей работы с данными, которые мы подготовили, создадим четыре пустых файла: «first_level.js», «second_level.js», «third_level.js», «underground_level.js». Затем присваиваем данным из каждого файла переменные, чтобы использовать их в основном проекте.

После этого имеем файлы с данными об этажах, теперь их необходимо добавить в проект. У нас имеется следующая структура проекта (Рисунок 2.12)

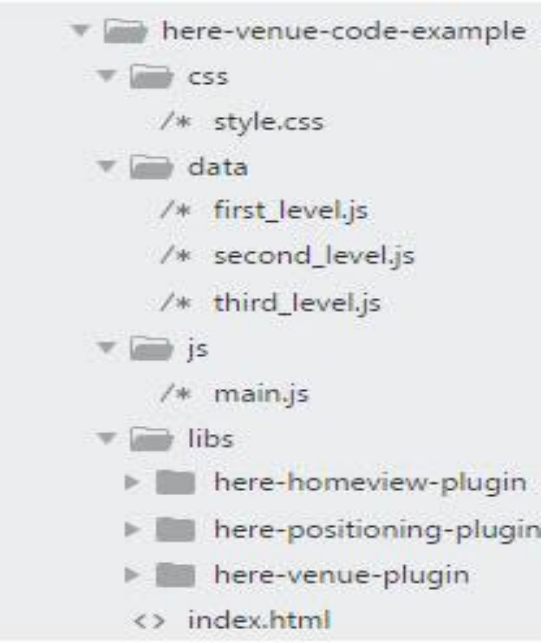

Рисунок 2.12

В папке data содержатся файлы этажей которые требуется подключить в проект.

Добавление слоев прописываем следующими командами:

<script type="text/javascript" src="data/first_level.js"></script>

<script type="text/javascript" src="data/second_level.js"></script>

<script type="text/javascript" src="data/third_level.js"></script>

<script type="text/javascript" src="data/underground_level.js"></script>

Далее нам необходимо подключить классы и методы добавления поэтажной навигации.

Сначала подключаем плагин Venue

<script type="text/javascript" src="libs/here-venue-plugin/src/here-venue-control/here-venue-control.js">

</script>

<link rel="stylesheet" type="text/css" href="libs/here-venue-plugin/src/here-venue-control/here-venue-control.css">

Теперь нам доступен конструктор - IndoorControl.

Таким образом мы подключили три файла с этажами и теперь нам доступны четыре переменные: first_level, second_level, third_level, underground_level. Теперь мы можем создать базовую навигацию по этажам здания. В main.js пропишем следующий код:

```
function geojsonParser (layer) {
let reader = new H.data.geojson.Reader();
reader.parseData(layer);
return reader.getParsedObjects()[0];
}
var venues = [
{id:1, data: geojsonParser(first_level)},
{id:2, data: geojsonParser(second_level)},
{id:3, data: geojsonParser(third_level)},
];
```

var indoor_control = new IndoorControl("top-left", venues, first_level_id=1);

ui.addControl("indoor", indoor_control);

В результате на карту добавляется первый этаж нашего здания и панель с кнопками переключения между этажами.

Полный код разработанного проекта приведен в ПРИЛОЖЕНИИ А.

2.4 Аппаратное обеспечение

Для удобства при разработке приложения рекомендуется использовать следующее оборудование:

- Современный и надежный компьютер. Можно использовать компьютер под управлением любой ОС. Сотрудники Больших компаний предпочитают работать на OS X (Apple). Файлы веб-приложений не требуют большого количества ресурсов, так что для программирования не требуется сверхпроизводительный компьютер. Если только вы не планируете заниматься обработкой звуковых файлов и видеомонтажом.
- Дополнительная оперативная память. В связи с тем что вам придется обрабатывать данные в QGIS, а также переключаться между многими, приложениями, нужно установить достаточный объем оперативной памяти, чтобы запускать одновременно несколько программ, задействующих значительные ее объемы.
- Мобильные устройства. Крайне важно протестировать внешний вид и производительность приложения в мобильном браузере на смартфоне или планшете. Одно и тоже приложение может совершенно по-разному отображаться в браузере на компьютере и на мобильном устройстве.

Выводы.

Данный раздел посвящен описанию современных средств, которые используются при создании приложения. Подробно рассматривается разработанное приложение: освящается используемая программная архитектура, вы-

бранные технологические решения, приводятся примеры разработанных интерфейсов.

Был написан основной код HTML страницы, а также стили к ней. С помощью языка программирования JavaScript в проект была добавлена карта и создан основной интерфейс (поиск местоположения, возможность приближать и отдалять карты, а также различные варианты отображения карты.). Были добавлены слои, содержащие оцифрованные чертежи корпусов в формате .geojson и написан интерфейс для переключения между этажами в здании.

В результате действий совершенных в данном разделе была создана карта с основным интерфейсом, а также в приложение были интегрированы дынные с оцифрованными чертежами первого и второго корпуса РГГМУ. На рисунке 2.12 приведен результат проделанной работы.

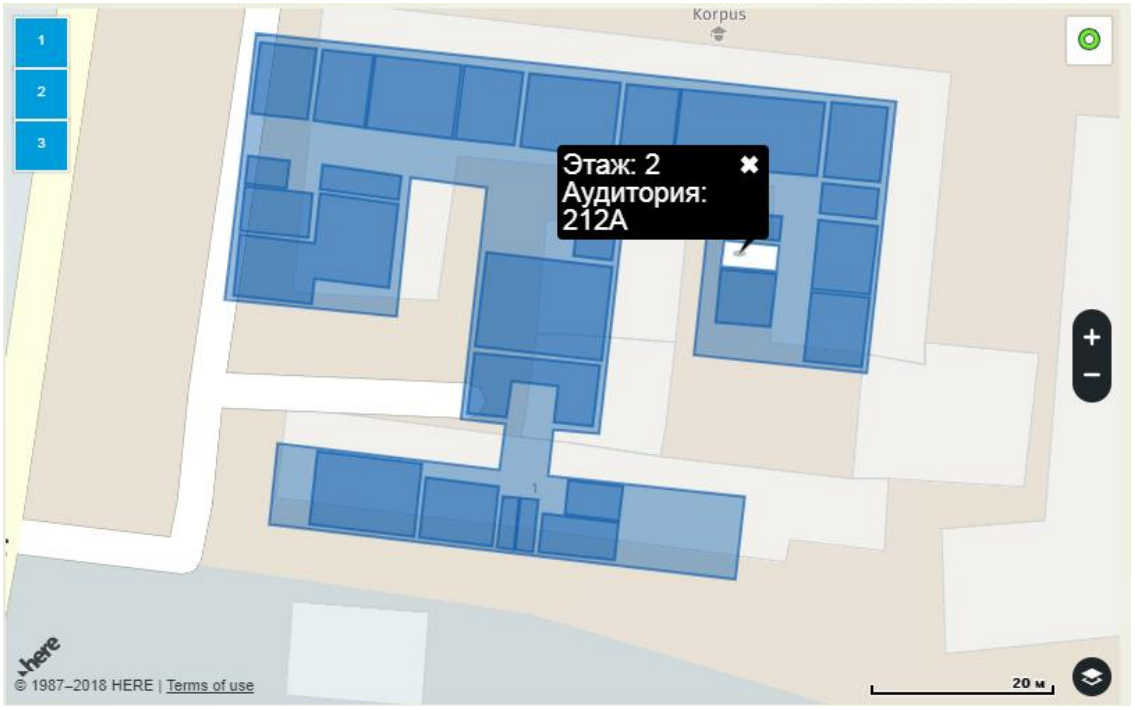

Рисунок 2.12

Глава 3 Введение приложения в эксплуатацию

3.1 Размещение приложения

Процесс развертывания приложения зависит от выбранных типа приложения и сценария развертывания.

Тип приложения можно задать в конструкторе приложений.

Существуют следующие сценарии развертывания:

- 2-уровневое классическое клиентское приложение. В этом сценарии развертывания создается приложение, которое выполняется на компьютере пользователя под управлением Windows. Компоненты базы данных и сервера запускаются с подключенного к сети компьютера.
- 3-уровневое классическое клиентское приложение. В этом сценарии развертывания создается приложение, которое выполняется на компьютере пользователя под управлением Windows. Компоненты базы данных и сервера запускаются на сервере со службами IIS или на платформе Microsoft Azure.
- 3-уровневое веб-приложение. В этом сценарии развертывания создается приложение, которое выполняется в веб-браузере пользователя. Компоненты базы данных и сервера запускаются на сервере со службами IIS или на платформе Microsoft Azure.
- Только служба. В этом сценарии развертывания создается веб-служба OData, к которой могут обращаться другие приложения.

Также исходя из требований аппаратной и программной составляющих спроектированного приложения, можно выделить следующие способы размещения приложения в сети интернет:

- Виртуальный хостинг - вид хостинга, при котором множество веб-сайтов располагается на одном сервере. Данный вид хостинга является самым экономичным и подходит для небольших проектов. Обычно каждый веб-сайт расположен на своем собственном разделе сервера, но они все вместе пользуются одним и тем же программным обеспечением.

Данное решение имеет ряд достоинств:

1. Невысокая средняя стоимость;
2. ПЗУ до 5 Гб;
3. Средний уровень обеспечения безопасности:
4. Ежедневное резервное копирование пользовательских данных.

Также есть и недостатки:

1. Отсутствие доступа к большинству настроек программного обеспечения;
2. Отсутствие возможности установки дополнительного программного обеспечения;

- VDS/VPS — VPS (англ. Virtual Private Server) или VDS (англ. Virtual Dedicated Server) - данный вид услуги, предоставляет пользователю так называемый Виртуальный выделенный сервер. В плане управления операционной системой по большей части она соответствует физическому выделенному серверу. На данный момент существует две разновидности такой услуги:

- Фиксированный виртуальный выделенный сервер — пользователь заказывает одну из возможных конфигураций VPS с фиксированными характеристиками (объем ОЗУ, ПЗУ, процессор). В среднем, стоимость такого решения составляет 8400-9600 р. в год за сервер с 400 МГц, 256 ОЗУ, 8 Гб ПЗУ и от 250 Гб разрешенного трафика в месяц.

- Сервер на основе «облачных» технологий (инфраструктура как услуга) — пользователь заказывает вычислительные единицы с фиксированными характеристиками — ноды. Приобретенные ноды возможно объединять в один сервер или создавать на их основе разные серверы с разным количеством нодов, также возможно динамически подключать ноды к существующим серверам. Стоимость одной ноды с конфигурацией 400 МГц, 356 ОЗУ, 10 Гб ПЗУ и от 250 Гб разрешенного трафика в месяц составляет в среднем 8000-9000 р. в год.

Плюсы данного метода:

1. Гарантированная производительность, зачастую более высокая, чем при виртуальном хостинге;

2. Практически полный контроль над программной составляющей;

3. В случае использования «облачных» технологий, практически полностью отсутствует вероятность аппаратного сбоя.

Минусы:

1. Высокая стоимость;

2. Необходимость в высокой квалификации администратора сервера для задания оптимальных настроек;

- Колокация, колокейшн (от англ. co-location — расположение рядом) — услуга связи, состоящая в том, что провайдер размещает оборудование клиента на своей территории (обычно в дата-центре) и подключает его к каналам связи с высокой пропускной способностью. Обычно оборудование не принадлежит клиенту, а арендуется им у того же провайдера, в этом случае услуга называется «аренда выделенного сервера».

Плюсы:

1. Полный контроль над программной и аппаратной составляющими;

Минусы:

1. Высокая стоимость решения (от 9000 рублей в месяц);

2. Необходимость в высокой квалификации администратора сервера для задания оптимальных настроек;

3. Существует вероятность критического аппаратного сбоя.

Самым оптимальным для разработанного приложения является решение на основе «облачных» вычислений.

3.2 Проведение тестирования

Тестированием веб-приложений называют оценку разрабатываемых продуктов, с целью проверки их возможностей, способностей и соответствие ожидаемым результатам. Существуют разные методы тестирования, с различными функциями и возможностями.

Тестирование приложения – это не что иное, как испытание куска кода к контролируемым и неконтролируемым условиям эксплуатации, наблюдение за выходом, а затем изучение, соответствует ли он предварительно определенным условиям.[15]

Различные методы тестирования направлены на достижения общей цели – устранение багов и ошибок в коде.

Существует четыре наиболее распространенных метода тестирования:

1. Модульное тестирование
2. Интеграционное тестирование
3. Системное тестирование
4. Приемочные испытания

В первую очередь проводится модульный тест, на наличие ошибок тестируются отдельные программные компоненты. Для данного метода необходимо полное знание программы и каждого установленного модуля.

После этого, отдельные модули, которые были проверены методом модульного тестирования, интегрируются друг с другом, и проверяются на наличие ошибок. Интеграционное тестирование можно осуществлять следуя иерархическому сооружению программы. Такой в первую очередь выявляет ошибки интерфейса.[18]

Далее системным методом тестирования проверяются ожидаемые условия работы программного обеспечения, осуществляя сопряжение аппаратных и программных компонентов всей системы, выполняется проверка.[20]

Последний метод тестирования, проводится перед передачей приложения клиенту. Он проводится, чтобы гарантировать, что программное обеспечение, которое было разработано отвечает всем требованиям заказчика.

Также существуют нефункциональные тесты, такие как:[16, 20]

- Тестирование безопасности

Одной из главных задач разработчика, является безопасность приложения. Тестирование безопасности проверяет приложение на обеспечение конфиденциальности, аутентификации, доступности и безотказности.

- Тестирование на совместимость

Данный тест проверяет совместимость приложения с другими мобильными платформами, а также с различными браузерами.

- Юзабилити-тестирование

Одним из важных аспектов приложения является его практичность и удобство использования. Данное тестирование проверяет, насколько легко пользователь может получить доступ к тем или иным функциям приложения.

- Тестирование эффективности

Данный метод проверяет объем кода и ресурсов, используемые программой на выполнение тех или иных операций.

Веб-приложение должно иметь дружественный пользовательский интерфейс и должно быть достаточно понятным, чтобы пользователь без обладания специальных знаний, мог комфортно его использовать. [17]

Для обеспечения устойчивого функционирования веб-приложения необходимо:

1. осуществлять проверку его совместимости с различными браузерами; осуществлять проверку его совместимости с различными кодировками операционных систем;
2. осуществлять проверку его совместимости с разрешениями экрана;
3. для надежности сохранять данные в сессии, в специальных файлах, которые содержат информацию, хранящуюся в базе данных;
4. обеспечивать восстановление информации утраченной в результате сбоя в работе сервера.

Для эффективной работы веб-приложения необходимы следующие условия:[19]

1. качество работы сервера, где происходит хранение Web-сайта;
2. обеспечивать восстановление информации утраченной в результате сбоя в работе сервера.
3. возможность отката или отмены внесенных изменений, а также возможность перехода к изначальным входным данным.

Выводы.

Были проанализированы различные способы размещения приложения в сети интернет. В результате анализы был выбран способ виртуального хостинга, так как данный метод самый экономичный и подходит для небольших проектов.

Также были проанализированы различные методы тестирования приложения в реальных условиях. В ходе тестирования на разных этапах были найдены и устранены некоторые ошибки связанные с отображением разработанного приложения на различных браузерах и мобильных платформах.

Была описана краткая инструкция по использованию данным приложением.

Заключение

Целью данной выпускной квалификационной работы была разработка и введение в эксплуатацию приложения внутренней навигации. Его использование значительно облегчает навигацию внутри большого количества корпусов Российского Государственного Гидрометеорологического университета, а также является прочным фундаментом для дальнейшего развития этого веб-приложения и для создания похожих по тематике проектов.

Проблема, поставленная в данной дипломной работе, затрагивает большую целевую аудиторию. Большинство крупных предприятий имеет огромную территорию с большим количеством зданий со сложной архитектурой. Работники и студенты сталкиваются с проблемой поиска нужных им помещений. Представленная система внутренней навигации может быть установлена на данных объектах, имеющих огромные территории, на которых спутниковая навигация бесполезна, а также на судах и в портах.

Некоторые виды систем внутреннего позиционирования могут быть установлены на подводных лодках, так как не требуют для определения точного местоположения дополнительной информации от внешних источников. В качестве объекта исследования были выбраны первый и второй корпуса Российского Государственного Гидрометеорологического университета.

В процессе были получены следующие результаты:

1. Выполнен предварительный анализ способов реализации проекта.

Рассмотрены существующие системы внутренней навигации, а также проведено обследование объекта исследования, детальный осмотр, оценка особенностей и предварительная оценка способов и времени реализации.

В результате анализа был выбран метод позиционирование с использованием оцифрованных планов зданий. Данный метод был выбран, так как в отличие от других способов он не требует установки никаких дополнительных устройств. Также в ходе анализа были выбраны компоненты необходимые для корректной работы приложения, такие как язык программирования

JavaScript, язык разметки HTML и таблицы стилей CSS. Для создания интернет страницы был выбран язык разметки HTML, так как это наиболее распространённый инструмент при реализации данных проектов.

В связке с HTML работают таблицы стилей CSS, они имеют огромное множество функций по преображению веб-приложения. Также с помощью CSS можно менять положение и размер различных объектов на странице. Язык программирования JavaScript, создан специально для программирования в веб-сфере. С его помощью мы можем добавить веб-приложению интерактивностью, а также написать специальные программы для определенных объектов, например добавить объект, при нажатии на который будет выполняться заранее написанная программа. Эти элементы являются основными при создании любого веб-приложения.

2. Осуществлена реализация проекта.

Был написан основной код HTML страницы, а также стили к ней. С помощью языка программирования JavaScript в проект была добавлена карта города и разработан основной интерфейс приложения: определение местоположения, возможность увеличивать и уменьшать размер карты, переключаться между этажами, а также различные варианты отображения карты. Были добавлены слои, содержащие оцифрованные поэтажно чертежи корпусов в формате .geojson и написан интерфейс для переключения между этажами в здании.

3. Введение приложения в эксплуатацию.

Был проведен анализ существующих способов размещения приложения в сети интернет. В ходе анализа рассмотрены все достоинства и недостатки существующих способов и в качестве основного варианта был выбран способ Виртуального выделенного сервера. Данный метод представляет собой выделенный сервер в сети интернет. Пользователь использует вычислительные мощности данного сервиса для корректной работы веб-приложения. Это наилучший вариант в соотношении цена/производительность.

Также было проведено тестирование приложения с использованием различных методов. В первую очередь в ходе реализации проекта проводилось модульное тестирование различных компонентов приложения, на данном этапе не было выявлено ошибок влияющих на корректную работу программы.

После реализации проекта было проведено тестирование на совместимость, в ходе которого были выявлены и исправлены ошибки связанные с отображением приложения в различных браузерах и на разных мобильных платформах.

Последним, был проведен тест практичности и удобства использования приложения, по результатам данного теста, было изменено местонахождение некоторых элементов интерфейса.

На данный момент приложение внутренней навигации в первом и втором корпусах университета находится на стадии разработки. В дальнейшем его можно будет найти на официальном сайте Российского Государственного Гидрометеорологического университета, перейдя на специальную вкладку в контекстном меню сайта.

Принцип работы приложения крайне прост. Вам потребуется компьютер или ноутбук с подключенным интернет соединением. Приложение определяет ваше местоположение в корпусе, что значительно облегчает поиск необходимой аудитории.

На карте будет размещен интерфейс необходимый для переключения между этажами (Рисунок 3.1), а также оцифрованный корпус университета.

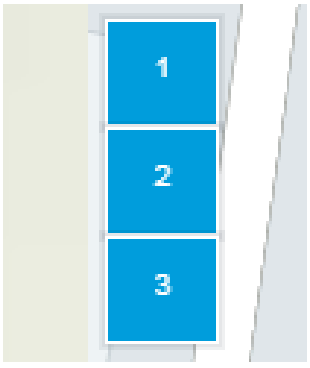

Рисунок 3.1

Переключаясь между этажами, вы сможете найти необходимую аудиторию и, узнав ее месторасположение, идете на занятие.

Список литературы

1. HTML5, CSS3 и JavaScript. Исчерпывающее руководство / Дженнифер Роббинс; [пер. с англ. М. А. Райтман]. — 4-е издание. — М. : Эксмо, 2014. — 528 с.
2. Дэвид Макфарланд Большая книга CSS3. - 3 изд. - СПб.: Питер, 2014. — 608 с.
3. Брайан Хоган HTML5 и CSS3. Веб-разработка по стандартам нового поколения. - СПб.: Питер, 2012. - 272 с.
4. Терри Фельке-Моррис Большая книга веб-дизайна / Терри Фельке-Моррис; пер с англ. Н.А. Райтмона. - М.: Эксмо, 2012. - 608 с.
5. Томас А. Пауэлл. Web-дизайн. – С-Пб.: 2 изд. «БХВ-Петербург», 2004 г.
6. Дженифер Нидерст. Web-мастеринг для профессионалов. – С-Пб.: «Питер», 2001 г.
7. Роберт Рейнхардт, Сноу Дауд. Flash MX. – М.: «Вильямс», 2003 г.
8. Стивен Хольцнер. Dynamic HTML: руководство разработчика. - СПб: BHV, 2000. - 400 с.
9. Брайан Маллой Web API Design: Crafting Interfaces that Developers Love. O'Reilly, 2010. - 100 с.
10. Марк Масс REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. O'Reilly, 2011. - 93 с.
11. Алекс Маккоу Веб-приложения на JavaScript. - СПб.: Питер, 2012. - 288 с.
12. Сэмми Пьюривал Основы разработки веб-приложений; [пер. с англ. М. А. Райтман]. - СПб.: Питер, 2015. - 272 с.
13. Node.js в действии / Кантелон М., Хартер М., Головайчук Т., Райлих Н., Райлих Н. и др. - 2 изд. - СПб.: Питер, 2018. - 432 с.
14. Дженнифер Нидерст Роббинс Web-дизайн. Справочник. КУДИЦ-Пресс, 2008. - 816 с.
15. Гленфорд Майерс Искусство тестирования программ. - М.: Диалектика, 2012. - 270 с.

16. Грэхем Ли Разработка через тестирование для iOS. - М.: ДМК Пресс, 2012. - 272 с.
17. Cem Kaner, Jack Falk, Hung Q. Nguyen Testing Computer Software. - 2 изд. 1988. - 530 с.
18. Д. Дардари, Э. Фаллетти, М. Луизе Методы спутникового и наземного позиционирования. Перспективы развития технологий обработки сигналов. - М.: Техносфера, 2012. - 528 с.
19. Кент Бек Экстремальное программирование: разработка через тестирование. - СПб.: Питер, 2003. - 224 с.
20. Джез Хамбл, Дэвид Фарли Непрерывное развертывание ПО. Автоматизация процессов сборки, тестирования и внедрения новых версий программ. - М.: Вильямс, 2016. - 432 с.
21. К. М. Антонович Использование спутниковых радионавигационных систем в геодезии. - М.: ДМК Пресс, 2006. - 89 с.
22. Д. Кронин, Р. Рейман, А.Купер Алан Купер об интерфейсе. Основы проектирования взаимодействия. Символ-Плюс, 2009. - 688 с.
23. API (Application Programming Interface) // Национальная библиотека им.Н.Э. Баумана [Электронный ресурс]: https://ru.bmstu.wiki/API_ (Application_Programming_Interface)#API
24. ТЕРРИТОРИЯМИ НА ПРИМЕРЕ QUANTUM GIS // МЕЖДУНАРОДНАЯ НАУЧНО-ТЕХНИЧЕСКАЯ ИНТЕРНЕТ-КОНФЕРЕНЦИЯ [Электронный ресурс]: http://kadastr.org/conf/2013/pub/infoteh/qgis-upr-rekr-terr.htm
25. XHTML // Самоучитель HTML [Электронный ресурс]: http://htmlbook.ru/xhtml
26. Основы JavaScript // Современный учебник Javascript [Электронный ресурс]: https://learn.javascript.ru/first-steps
27. Развертывание приложений LightSwitch // Microsoft [Электронный ресурс]: https://msdn.microsoft.com/ru-ru/library/ff872288.aspx
28. API (программный интерфейс приложения) // Википедия [Электронный ресурс]: https://ru.wikipedia.org/wiki/API#API

29. МОДЕЛИ РЕАЛИЗАЦИИ НАВИГАЦИИ ВНУТРИ ПОМЕЩЕНИЯ ПРИ ПОМОЩИ АНАЛИЗА БЕСПРОВОДНЫХ ИСТОЧНИКОВ ДАННЫХ // Компьютерные инструменты в образовании [Электронный ресурс]: http://ipo.spb.ru/journal/content/1786/
30. Разработка системы навигации для мобильных устройств на базе ios // Системы навигации [Электронный ресурс]: http://docplayer.ru/26184039-Razrabotka-sistemy-navigacii-dlya-mobilnyh-ustroystv-na-baze-ios.html

Приложение А

Основной код HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>Diplom</title>
        <meta name="viewport" content="initial-scale=1.0, width=device-width" />
        <link href="style.css" rel="stylesheet" type="text/css" media="all" />
        <script src="http://js.api.here.com/v3/3.0/mapsjs-core.js"
            type="text/javascript" charset="utf-8"></script>
        <script src="http://js.api.here.com/v3/3.0/mapsjs-service.js"
            type="text/javascript" charset="utf-8"></script>
        <script src="http://js.api.here.com/v3/3.0/mapsjs-mapevents.js"
            type="text/javascript" charset="utf-8"></script>
        <script src="http://js.api.here.com/v3/3.0/mapsjs-ui.js"
            type="text/javascript" charset="utf-8"></script>
        <script src="http://js.api.here.com/v3/3.0/mapsjs-data.js"
            type="text/javascript" charset="utf-8"></script>
        <link rel="stylesheet" type="text/css"
            href="http://js.api.here.com/v3/3.0/mapsjs-ui.css" />
        <script type="text/javascript" src="libs/here-venue-plugin/src/here-venue-control/here-venue-control.js"></script>
        <link rel="stylesheet" type="text/css" href="libs/here-venue-plugin/src/here-venue-control/here-venue-control.css">
        <script type="text/javascript" src="libs/here-positioning-plugin/src/here-positioning-control/here-positioning-control.js"></script>
        <link rel="stylesheet" type="text/css" href="libs/here-positioning-plugin/src/here-positioning-control/here-positioning-control.css">
```

```
</head>
<body>
    <h1>Российский государственный гидрометеорологический университет</h1>
    <div id="map"></div>
    <script type="text/javascript" src="data/first_level.js"></script>
    <script type="text/javascript" src="data/second_level.js"></script>
    <script type="text/javascript" src="data/third_level.js"></script>
    <script type="text/javascript" charset="utf-8">
        var platform = new H.service.Platform({
            'app_id': '7gTgV3RA3EZCNlpdn4CW',
            'app_code': 'OeFBygbgqIRwJZnDrrBVPg'
        });
        var defaultLayers = platform.createDefaultLayers();
        var map = new H.Map(
            document.getElementById('map'),
            defaultLayers.normal.map,
            {
                zoom:18,
                center:{lat:59.944382,lng:30.421184}
            }
        );
        var mapEvents = new H.mapevents.MapEvents(map);
        var behavior = new H.mapevents.Behavior(mapEvents);
        var ui = H.ui.UI.createDefault(map, defaultLayers, 'ru-RU');
        function geojsonParser (layer) {
            let reader = new H.data.geojson.Reader();
        reader.parseData(layer);
        return reader.getParsedObjects()[0];
        }
```

```
                var venues = [
                        {id:1, data: geojsonParser(first_level)},
                        {id:2, data: geojsonParser(second_level)},
                        {id:3, data: geojsonParser(third_level)},
                ];
                var indoor_control = new IndoorControl("top-left", venues, first_level_id=1);
                ui.addControl("indoor", indoor_control);
                var geolocation = new GeolocationControl('top-right');
                ui.addControl('geolocation', geolocation);
        </script>
</body>
</html>
```

Каскадные таблицы стилей CSS

```
body{

        background-image: url(https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTEC2HkZgUwe39-RrFFOLhI3MOYfoizxQvoDfXP3l9jHi1A8bILIQ);
        background-size: 100%;
        }
        #map{
                width: 800px;
                height: 500px;
                position: absolute;
                left: 500px;
                top: 350px;
        }
        #geolocation{
                position: absolute;
```

```
                top: 780px;
                right: 625px;
                width: auto;
                height: 20px;
                background-color: #1f262a;
                border-radius: 22px;
                color: #fff;
                padding: 2px;
                cursor: pointer;
            }
            #geolocation:hover{
                color: #1f262a;
                background-color: white;
                border: 1px solid #1f262a;
            }
            .active{
                color: #1f262a !important;
                background-color: white !important;
                border: 1px solid #1f262a !important;
            }
            h1{
                color: #0404B4;
                font-family: "Century Gothic", sans-serif;
            }
.btn-venues{
    background: #fff;
  margin-bottom: 0.8rem;
}
.venue-level{
    width: 17px;
```

```
    height: 15px;
    background: #009ddc;
    color: #fff;
    padding: 10px;
    text-align: center;
    border: 1px solid white;
    box-shadow: 0 0 0 0.1rem rgba(1, 11, 30, 0.1);
}
```

Плагин venue

```
(function () {
        'use strict;'
        class IndoorControl extends H.ui.Control {
                constructor (position="top-left", venues, first_level_id=1) {
                        super();
                        this.setAlignment(position);
                        this.layerGroup = new H.map.Group();
                        map.addObject(this.layerGroup);
                        this.venues = venues;
                        let first_level = this.venues.filter(function (elem) {
                                return elem.id == first_level_id
                        });
                        // debugger;
                        first_level = first_level[0].data;
                        let bubble;
                        map.addEventListener('pointerenter', ev => {
                                first_level.getObjects().forEach(function(el){
                                        el.setStyle({
                                                fillColor: "rgba(0,85,170,.4)",
                                                lineWidth: 2,
                                                strokeColor: "rgba(0,85,170,.6)"
```

```
                    })
                });
                try{
                    bubble.close()
                } catch {
                }
            });
            let holdEvent;
            first_level.addEventListener('pointermove', function (feature) {
                // debugger;
                if (holdEvent != feature) {
                    try {
                        holdEvent.target.setStyle({
                        fillColor: "rgba(0,85,170,.4)",
                        lineWidth: 2,
                        strokeColor: "rgba(0,85,170,.6)"
                       });
                       bubble.close()
                    } catch (err) {
                    }
                };
                holdEvent = feature;
                var coords =
map.screenToGeo(feature.currentPointer.viewportX,feature.currentPointer.viewpor
tY);
                bubble = new H.ui.InfoBubble({
                    "lat": coords.lat,
                    "lng": coords.lng,
                },{
                    content: `
```

```
                                        Этаж: ${fea-
ture.target.getData().properties.level}
                                        <br>
                                        Аудитория: ${fea-
ture.target.getData().properties.lecture_hall}                                    `
                        });
                        feature.target.setStyle({
                                fillColor:"#fff"
                        });
                        ui.addBubble(bubble);
                });
                this.layerGroup.addObject(first_level);
        }
        renderInternal (el, doc) {
                var that = this;
                let control = document.createElement("div");
                control.id = "ctrl-venues";
                control.className = "btn-venue";
                this.venues.forEach(elem => {
                        let button = document.createElement("div");
                        button.id = elem.id;
                        button.className = "venue-level";
                        button.innerHTML = elem.id;
                        control.appendChild(button);
                });
                el.innerHTML = control.innerHTML;
                el.addEventListener('click', this._levelHandler.bind(this), false);
                super.renderInternal(el, doc);
        }
        _levelHandler (evt) {
```

```
this.layerGroup.removeObjects(this.layerGroup.getObjects());
let holdEvent;
let level_id = evt.target.id,
    current_level = this.venues.filter(function (elem) {
        return elem.id == level_id
    });
current_level = current_level[0].data;
current_level.getObjects().forEach(function(el){
    el.setStyle({
        fillColor: "rgba(0,85,170,.4)",
        lineWidth: 2,
        strokeColor: "rgba(0,85,170,.6)"
    })
});
let bubble;
map.addEventListener('pointerenter', ev => {
    current_level.getObjects().forEach(function(el){
        el.setStyle({
            fillColor: "rgba(0,85,170,.4)",
            lineWidth: 2,
            strokeColor: "rgba(0,85,170,.6)"
        })
    });
    try{
        bubble.close()
    } catch {
    }
});
current_level.addEventListener('pointermove', function (feature)
{
```

```
                    if (holdEvent != feature) {
                        try {
                            holdEvent.target.setStyle({
                            fillColor: "rgba(0,85,170,.4)",
                            lineWidth: 2,
                            strokeColor: "rgba(0,85,170,.6)"
                           });
                           bubble.close()
                        } catch (err) {

                        }
                    };
                    holdEvent = feature;
                    var coords =
map.screenToGeo(feature.currentPointer.viewportX,feature.currentPointer.viewpor
tY);
                    bubble = new H.ui.InfoBubble({
                        "lat": coords.lat,
                        "lng": coords.lng,
                    },{
                        content: `
                            Этаж: ${fea-
ture.target.getData().properties.level}
                            <br>
                            Аудитория: ${fea-
ture.target.getData().properties.lecture_hall}                    `
                    });
                    feature.target.setStyle({
                        fillColor:"#fff"
```

```
                    });
                    ui.addBubble(bubble);
                });
                this.layerGroup.addObject(current_level);
            }
        };
        Object.assign(window, {IndoorControl});
}())
```

Плагин геопозиции

```
(function(){
    'use strict;'
    class GeolocationControl extends H.ui.Control {
        constructor (position='top-left') {
            super();
            this.setAlignment(position);
            console.log("hello")
        }
        renderInternal (el, doc) {
            el.innerHTML = `
                <div id="ctrl-geolocation" class="btn-location">
                    <span data-here-map-control-location-svg>
                        <svg xmlns="http://www.w3.org/2000/svg"
width="16" height="16" viewBox="0 0 16 16">

                            <path class="middle_location_stroke"
d="M8 12c-2.206 0-4-1.795-4-4 0-2.206 1.794-4 4-4s4 1.794 4 4c0 2.205-1.794 4-
4 4M8 1.25a6.75 6.75 0 1 0 0 13.5 6.75 6.75 0 0 0 0-13.5">
                            </path>
```

```
                                        <path class="inner_location_stroke"
d="M8 5a3 3 0 1 1 .001 6A3 3 0 0 1 8 5m0-1C5.794 4 4 5.794 4 8c0 2.205 1.794 4
4 4s4-1.795 4-4c0-2.206-1.794-4-4-4">
                                        </path>

                                        <path class="outer_location_stroke"
d="M8 1.25a6.75 6.75 0 1 1 0 13.5 6.75 6.75 0 0 1 0-13.5M8 0C3.59 0 0 3.59 0
8c0 4.411 3.59 8 8 8s8-3.589 8-8c0-4.41-3.59-8-8-8">
                                        </path>

                                        </svg>
                                </span>
                        </div>
                `;
                el.addEventListener ('click', this.getLocation);
                super.renderInternal(el, doc);
        }
        getLocation (evt) {
                var that = this;
                var button = document.getElementById("ctrl-geolocation");
                if (!button.classList.contains("active")) {
                        button.classList.add("active");
                        navigator.geolocation.getCurrentPosition (function (posi-
tion) {
                                var center =
{"lat":position.coords.latitude,"lng":position.coords.longitude};
                                var accuracy = position.coords.accuracy;
                                that.area = new H.map.Circle(center, accuracy, {
                              style: {
                               strokeColor: '#0099bb4f',
```

```
                    lineWidth: 1,
                    fillColor: '#0099bb4f'
                  }
                });
                that.geoGroup = new H.map.Group();
                    that.marker = new H.map.Circle(center, 1.5, {
                  style: {
                    strokeColor: '#245f2f',
                    lineWidth: 1,
                    fillColor: 'rgba(116, 237, 38, 0.78)'
                  }
                });
                    that.marker2 = new H.map.Circle(center, 0.8, {
                  style: {
                    strokeColor: '#245f2f',
                    lineWidth: 1,
                    fillColor: 'rgba(255, 255, 255, 0.74)'
                  }
                });
                    that.geoGroup.addObject(that.marker);
                    that.geoGroup.addObject(that.marker2);
                    map.setCenter(center);
                    map.setZoom(19);
                    map.addObject(that.area);
                    map.addObject(that.geoGroup);
            });
        } else {
            button.classList.remove("active");
            map.removeObject(that.geoGroup);
            map.removeObject(that.area);
```

```
                    }
                }
        };
        Object.assign(window, {GeolocationControl});
}());
```

Интерфейс для геопозиции

```
.btn-location{
        background: #fff;
    padding: 8px;
    box-shadow: 0 0 0 0.1rem rgba(1, 11, 30, 0.1);
    border-radius: 0.2em;
    margin-bottom: 0.8rem;
}
.active {
        background:rgba(255, 255, 255, 0.74) !important;
}
svg path.middle_location_stroke {
    fill: #74ED26 !important;
}
svg path.inner_location_stroke {
    fill: #1C5927 !important;
}
svg path.outer_location_stroke {
    fill: #1C5927 !important;
}
```