Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и Информационных Технологий
институт
Информационные Системы
кафедра

УТВЕРЖДАЮЗав. кафедрой ИС
_____ Виденин С. А.
подпись инициалы, фамилия
« _____» _____ 2016г.

ДИПЛОМНЫЙ ПРОЕКТ

230201.65 Информационные системы и технологии

Автоматизированное рабочее пространство для агентств недвижимости

Пояснительная записка

Руководитель Н.В. Молокова
подпись, дата

Выпускник С.А. Пашкевич
подпись, дата

Нормоконтролер Ю. В. Шмагрис
подпись, дата

Красноярск 2016

РЕФЕРАТ

Выпускная квалификационная работа по теме «Автоматизированное рабочее пространство для агентств недвижимости» содержит 45 страниц текстового документа,18 иллюстрации,6 таблиц, 20 использованных источников.

ИНФОРМАЦИОННАЯ СИСТЕМА, ПРОЕКТИРОВАНИЕ, ВЕБ-ПРИЛОЖЕНИЕ, ВЕБ-РАЗРАБОТКА, WEB-SERVICE, SINGLE PAGE APPLICATION, JAVASCRIPT, ANGULARJS, MONGODB, NODEJS, JSON, API.

Цель проекта —спроектировать и разработать веб приложение для усовершенствования поиска и подбора объектов недвижимости для покупки и продажи.

Для достижения поставленной цели были выделены и выполнены следующие задачи:

- Анализ предметной области;
- Выбрать инструменты и технологии для разработки веб приложения;
- Разработать клиентскую часть веб приложения;
- Разработать базу данных;
- Разработать серверную часть веб приложения.

Результатом дипломного проекта является веб приложение, позволяющее повысить взаимодействиеагентов недвижимости с пользователями.

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра «Информационные системы»

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ С. А. Виденин
« _____» _____ 2016г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме дипломного проекта

Студенту Пашкевич Сергею Александровичу

Группа ЗКИ10-02 специальность 230201.65
«Информационные системы и технологии»

Тема выпускной квалификационной работы:
«Автоматизированное рабочее пространство для агентств недвижимости»

Утверждена приказом по университету № 4043/с от 24.03.2016

Руководитель ВКР: кандидат технических наук, доцент кафедры «Информационные системы» ИКИТ СФУ, Молокова Н.В.

Исходные данные для ВКР: техническое задание, методические указания научного руководителя,статьи по программированию, учебные пособия, электронные ресурсы.

Перечень разделов ВКР: аналитическая часть, проектирование и реализация.

Перечень графического материала: презентация, выполненная в MicrosoftOfficePowerPoint 2016.

Руководитель ВКР _____ Н.В. Молокова
подпись инициалы и фамилия

Задание принял к исполнению _____ С.А. Пашкевич
подпись инициалы и фамилия студента

« ___ » _____ 20__ г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По мере повышения интереса к глобальной сети интернет, повышались так же и требования пользователей к содержимому страниц, а именно: оформлению размещенной информации и интерактивному взаимодействию пользователей с сайтом, поэтому стали разрабатываться веб приложения для серверов.

Веб приложения имеют клиент-серверную архитектуру. Создавая веб приложение необходимо рассмотреть некоторые аспекты: архитектуру, подходы к разработке, безопасность информации и сервера. При создании веб приложений универсальной схемы нет, так как разные приложения требуют разных концепций при обработке информации, построения интерфейса и т.д.

С совершенствованием различных технологий создания веб приложений и сайтов, стало очевидно, что существующие доски объявлений уже не удовлетворяют современным стандартам обработки контента. Они предлагают пользователю плохо структурированную информацию и не позволяют её эффективно использовать. В своей работе я рассмотрю решение этой проблемы на примере системы по поиску и подбору жилья.

Цель дипломногопроекта — спроектировать и разработать веб приложение для усовершенствования поиска и подбора объектов недвижимости для покупки и продажи.

Объект проектирования — автоматизированное рабочее пространство для агентств недвижимости.

Цель определяет следующие задачи:

- Анализ предметной области;
- Выбрать инструменты и технологии для разработки веб приложения;
- Разработать клиентскую часть веб приложения;
- Разработать базу данных;

- Разработать серверную часть веб приложения.

Дипломный проект состоит из введения, двух глав, заключения и списка используемой литературы.

В первой главе данной дипломной работы излагаются основные сведения об информационной системе, описывается общая структура информационной системы, даются определения основным понятиям. Большая часть главы посвящена выбору средств разработки и обзору используемых при разработке технологий. Так же в этой главевыявлены требования к системе, которые послужили основой при ее проектировании.

Во второй части была создана структура проекта, разработан интерфейс. В этой части была описанареализация проекта с помощью программных средств. Были разработаны: клиентская часть, база данных, серверная часть веб приложения.

Глава 1 Аналитическая часть

1.1 Модель предметной области

Объектом автоматизации является процесс по привлечению потенциальных клиентов для агентов недвижимости.

Роли в контексте веб приложения:

- Агент недвижимости — работник агентства недвижимости;
- Пользователь — потенциальный покупатель объекта недвижимости.

Веб приложение предоставляет агенту недвижимости следующие возможности:

1. Добавить свои объекты недвижимости в хранилище данных.
2. Смотреть анонимные анкеты пользователей.
3. Редактировать свои варианты недвижимости, указывая все преимущества, с целью заинтересовать конкретного пользователя.
4. Рассылать индивидуальное предложения.

Пользователь в свою очередь, после регистрациив веб приложении и заполнения анкеты имеет возможность:

1. Добавить интересуемые адреса домов в список отслеживания.
2. Просматривать объявления по адресам из своего списка слежения.
3. Получать индивидуальные предложения от агентов недвижимости.

Анонимная анкета пользователя содержит:

- Информацию об адресе работы пользователя;
- Какой вид транспорта пользователь предпочитает;
- Какая инфраструктура пользователя интересует;
- С каким количеством комнат пользователь рассматривает варианты;
- Какую сумму за квадратный метр пользователь готов заплатить;

- Какой этаж пользователь предпочитает или не предпочитает;
- Какой площади варианты интересуют пользователя.

Объявления и индивидуальные предложения содержатследующие поля:

- Фактический адрес объекта недвижимости;
- Число комнат объекта недвижимости;
- Этаж и этажность объекта недвижимости;
- Цена и цена за квадратный метр объекта недвижимости;
- Площадь объекта недвижимости;
- Описание объекта недвижимости;
- Контактные данные продавца объекта недвижимости.

1.2 Автоматизированное рабочее пространство

Деятельность работников сферы управления (бухгалтеров, агентов недвижимости, плановиков, технологов, руководителей, конструкторов, и т.д.) в настоящеевремя ориентирована наиспользованиеразвитых технологий. Организация и реализация управленческих функций требует радикального изменения как самой технологии управления, так и технических средств обработки информации, среди которых главное место занимают персональные компьютеры. Они все более превращаются из системавтоматической переработки входной информации в средства накопления опыта управленческих работников, анализа, оценки и выработки наиболее эффективных экономических решений.

Автоматизированное рабочее место (АРМ) — определяется как совокупность информационно-программно-технических ресурсов, обеспечивающую конечному пользователю обработку данных и автоматизацию управленческих функций в конкретной предметной области.

Создание автоматизированных рабочих мест предполагает, что основные операции по накоплению, хранению и переработке информации возлагаются на

вычислительную технику, а работник сферы управления (экономист, технолог, руководитель и т.д.)выполняет часть ручных операций и операций, требующих творческого подхода при подготовке управленческих решений. Персональная техника применяется пользователем для контроля производственно-хозяйственной деятельности, изменения значений отдельных параметров в ходе решения задачи, а также ввода исходных данных в АИС для решения текущих задач и анализа функций управления.

АРМ создается для обеспечения выполнения некоторой группы функций. Наиболее простой функцией АРМ является информационно-справочное обслуживание. АРМ имеют проблемно-профессиональную ориентацию на конкретную предметную область. Профессиональные АРМ являются главным инструментом общения человека с вычислительными системами, играя роль автономных рабочих мест, интеллектуальных терминалов больших ЭВМ, рабочих станций в локальных сетях.

Локализация АРМ позволяет осуществить оперативную обработку информации сразу же по ее поступлении, а результаты обработки хранить сколь угодно долго по требованию пользователя.

Целью внедрения АРМ является усиление интеграции управленческих функций, и каждое более или менее «интеллектуальное» рабочее место должно обеспечивать работу в многофункциональном режиме.

АРМ выполняют децентрализованную одновременную обработку экономической информации на рабочих местах исполнителей в составе распределенной базы данных (БД). При этом они имеют выход через системное устройство и каналы связи в ПЭВМ и БД других пользователей, обеспечивая таким образом совместное функционирование ПЭВМ в процессе коллективной обработки.

АРМ, созданные на базе персональных компьютеров, — наиболее простой и распространенный вариант автоматизированного рабочего места для работников сферы организационного управления. Такое АРМ рассматривается

как система, которая в интерактивном режиме работы предоставляет конкретному работнику (пользователю) все виды обеспечения монопольно на весь сеанс работы.

Создание АРМ на базе персональных компьютеров обеспечивает:

- Простоту, удобство и дружественность по отношению к пользователю;
- Простоту адаптации к конкретным функциям пользователя;
- Компактность размещения и невысокие требования к условиям эксплуатации;
- Высокую надежность и живучесть;
- Сравнительно простую организацию технического обслуживания.

Эффективным режимом работы АРМ является его функционирование в рамках локальной вычислительной сети в качестве рабочей станции. Особенно целесообразен такой вариант, когда требуется распределять информационно-вычислительные ресурсы между несколькими пользователями.

В наиболее сложных системах АРМ могут через специальное оборудование подключаться не только к ресурсам главной ЭВМ сети, но и к различным информационным службам и системам общего назначения (службам новостей, национальным информационно-поисковым системам, базам данных и знаний, библиотечным системам и т.п.).

Возможности создаваемых АРМ в значительной степени зависят от технико-эксплуатационных характеристик ЭВМ, на которых они базируются. В связи с этим на стадии проектирования АРМ четко формулируются требования к базовым параметрам технических средств обработки и выдачи информации, набору комплектующих модулей, сетевым интерфейсам, эргономическим параметрам устройств и т.д.

Информационное обеспечение АРМ ориентируется на конкретную, привычную для пользователя, предметную область. Обработка документов должна предполагать такую структуризацию информации, которая позволяет

осуществлять необходимое манипулирование различными структурами, удобную и быструю корректировку данных в массивах.

Техническое обеспечение АРМ должно гарантировать высокую надежность технических средств, организацию удобных для пользователя режимов работы (автономный, с распределенной БД, информационный, с техникой верхних уровней и т.д.), способность обработать в заданное время необходимый объем данных. Поскольку АРМ является индивидуальным пользовательским средством, оно должно обеспечиватьвысокиеэргономическиесвойстваикомфортность обслуживания.

1.3 Требования к веб приложению

Веб приложение должно отвечать следующим требованиям:

- Иметь простой интерфейс;
- Предусматривать разделение пользователей на два ролевых модуля (агент недвижимости и пользователь);
- Работать в режиме реального времени.

Пользователь должен иметь возможность:

- Просматривать объявления в системе после регистрации;
- Добавлять адреса в свой список слежения;
- Получать индивидуальные предложения от агентов недвижимости;
- Отвергать индивидуальные предложения;
- Просматривать последние изменения.

Агент недвижимости должен иметь возможность:

- Просматривать анкеты пользователей;
- Составлять индивидуальные предложения;
- Отправлять индивидуальные предложения;
- Добавлять свои варианты недвижимости;
- Просматривать объявления в системе.

1.4 Модель анализа

Для рассмотрения некоторых предпосылок к выбору тех или иных инструментов и технологий, а также к проектированию веб приложения, была разработана диаграмма анализа. Уточним некоторые варианты использования согласно диаграмме. На рисунке 1 представлены классы анализа, которые представляют детализацию варианта использования «Заполнить анкету».

.

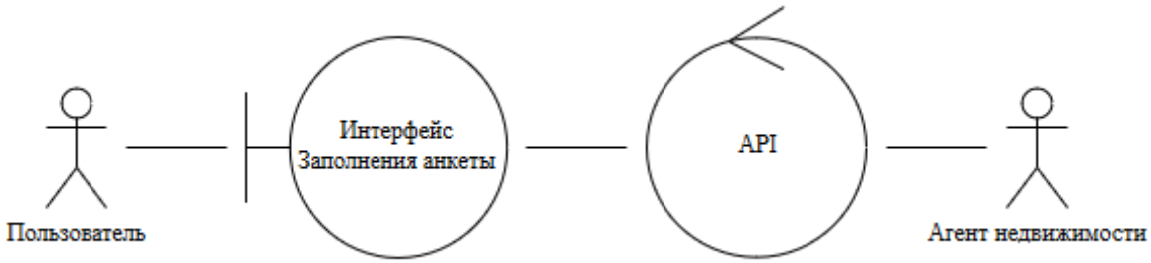

Рисунок 1 —классы анализа, которые представляют детализацию варианта использования «Заполнить анкету»

На рисунке 2 представлены классы анализа, которые представляют детализацию варианта использования «Отправить индивидуальное предложение».

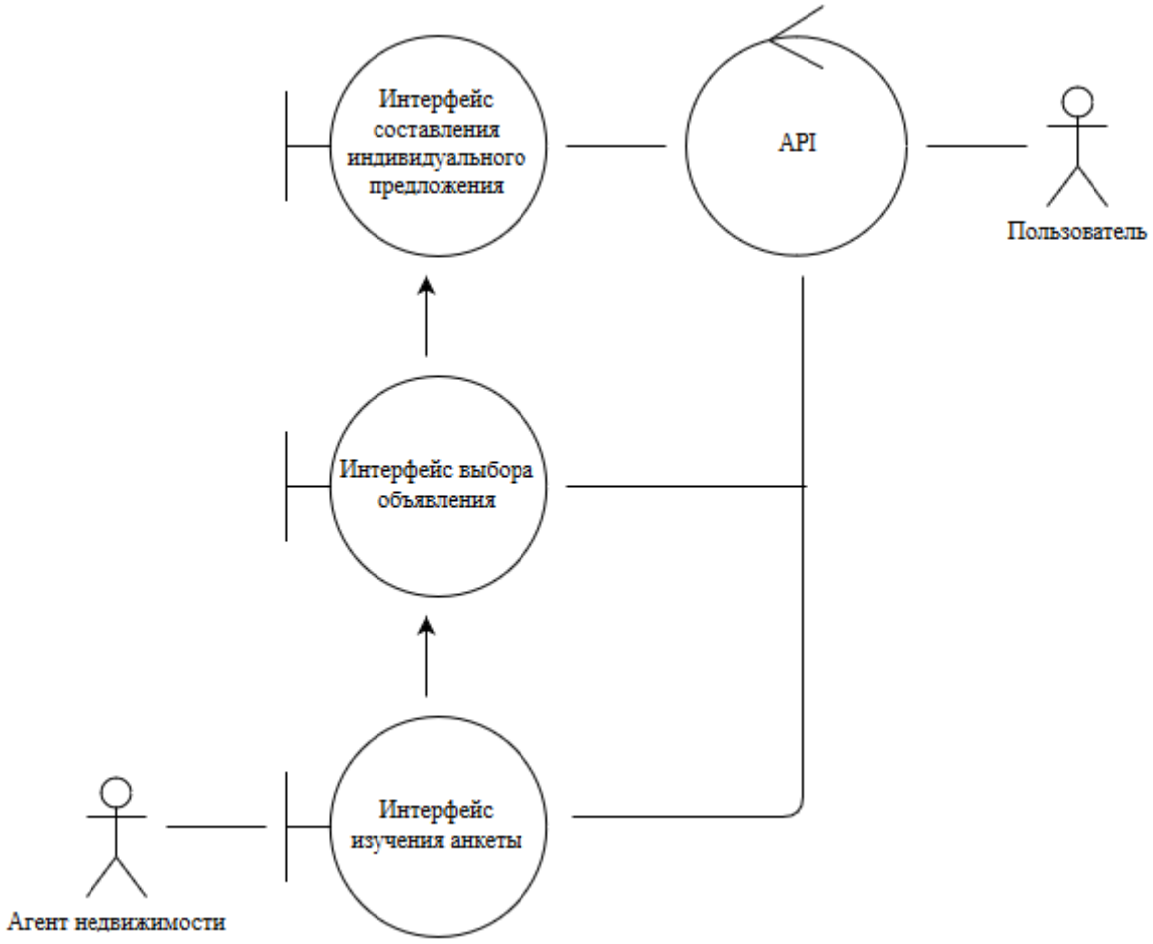

Рисунок 2 — Диаграмма анализа «Отправить индивидуальное предложение»

На основе структуры классов анализа видно, что API является неотъемлемой частью веб приложения и может предоставлять данные, не зависимо от пользовательского интерфейса. А также на основании того, что имеют место быть классы сущности, управления и граничные, имеет смысл ориентироваться на три ответственных компонента: за представление данных, управление и хранение. В этом случае, традиционно используется типовое архитектурное решение MVC.

1.5 Общая архитектура системы

В основе разрабатываемой системы лежит архитектура «клиент-сервер», в которой задания или сетевая нагрузка распределены между поставщиками

услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами. В качестве среды взаимодействия клиента с сервером используется Интернет, рисунок 3.

Также для решения поставленной задачи можно использовать архитектуру «сервер приложений», однако в данном случае подобный подход будет избыточен в виду отсутствия большой фрагментированности бизнес-логики приложения.

Рисунок 3 — Общая архитектура приложения (концепция взаимодействия)

Основными достоинствами архитектуры «клиент-сервер» являются:

- Возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети. Это позволяет упростить обслуживание вычислительной системы. В частности, замена, ремонт, модернизация или перемещение сервера, не затрагивают клиентов;
- Все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов. На сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;

- Позволяет объединить различные клиенты. Использовать ресурсы одного сервера часто могут клиенты с разными аппаратными платформами, операционными системами и т.п.

Основные недостатки:

- В случае использования централизованной системы, неработоспособность основного сервера может сделать неработоспособным все приложение;
- Администрирование данной системы требует квалифицированного профессионала;
- Высокая стоимость оборудования.

1.6 Общие решения

При выборе средств реализации, отметим следующие особенности:

- Ожидается тенденция к росту количества запросов на единицу времени;
- Один запрос не требует «больших»вычислительных ресурсов сервера.

В силу данных особенностей предпочтительно было ориентироваться на средства по обработке запросов или выполнение кода приложения, которые имеют асинхронную модель работы, основанную на событиях, либо предоставляют такие механизмы. Это обуславливалось тем, что асинхронная модель условно позволяет сократить время отклика веб приложения, по той причине, что она не блокирует ввод-вывод и операционные задачи могут выполняться параллельно, повышая общую производительность веб приложения на каждом уровне.

Условное сравнение скорости выполнения кода приложения при синхронной и асинхронной модели, показаны на рисунках 4 и 5 соответственно.

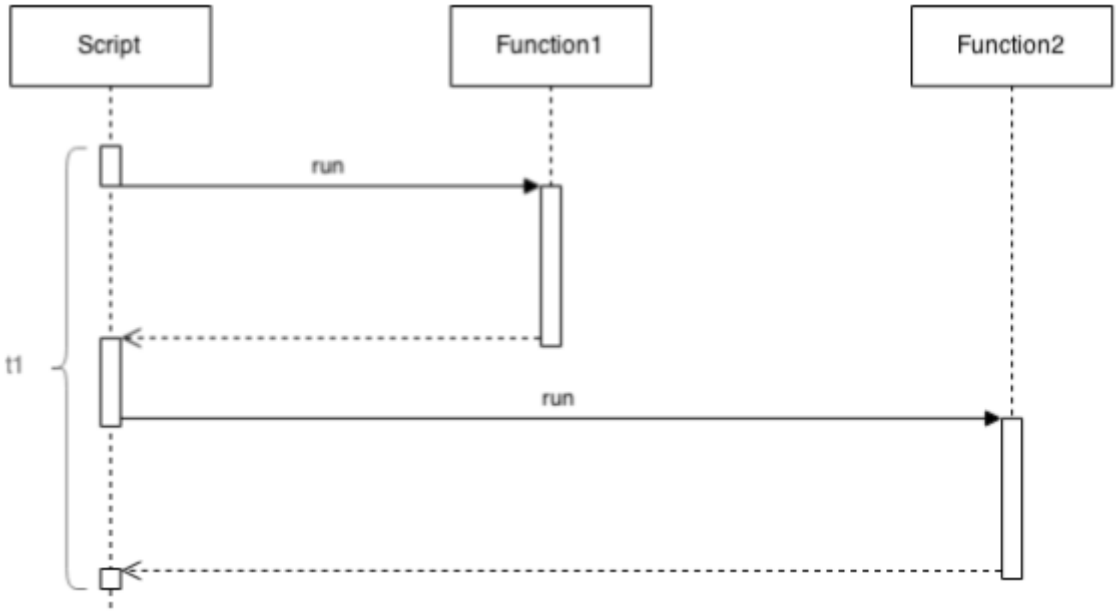

Рисунок 4 — Синхронная модель обработки запросов

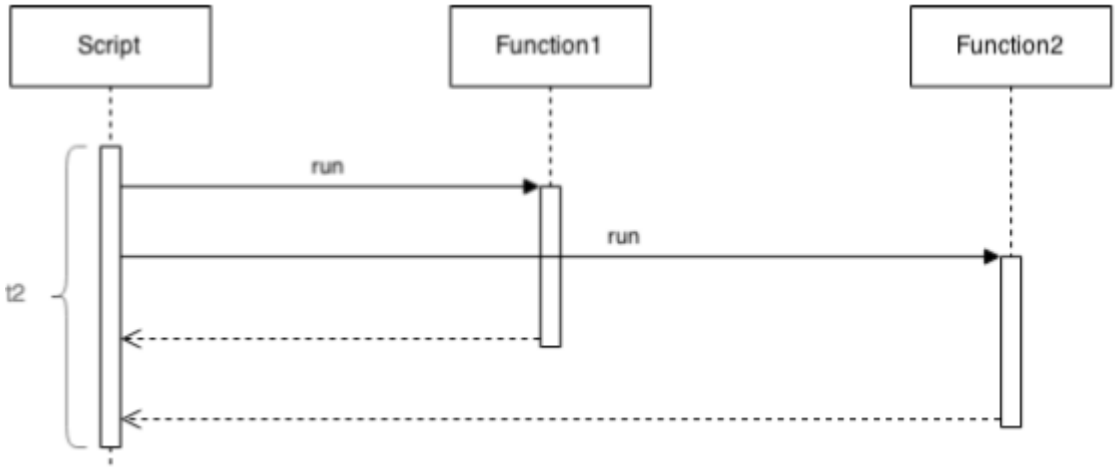

Рисунок 5 — Асинхронная модель обработки запросов

Из рисунков видно, что t1 > t2. Это говорит о том, что время обработки запроса или выполнения кода при асинхронной модели меньше. Зачастую, на рисунках, асинхронная модель предполагает демонстрацию функций обратного вызова, однако для лучшего восприятия они будут опущены [4]. Далее, на диаграммах, работа асинхронной модели будет показываться как на рисунке 5.

Расширяемость веб приложения достигается за счет:

- Использования типовых архитектурных решений, модулей;
- API.

Для каждого типа учетной записи предусмотрен свой рабочий стол и свой набор инструментов, поэтому разграничение прав целесообразно реализовать средствами самого веб приложения, на основе ролей, так как:

- Решение достаточно известное и простое;
- Роли планируется создавать вручную;
- Предполагается наличие только двух ролей: агент недвижимости и пользователь.

Для того чтобы увеличить контроль над доступностью веб приложения и мониторингом работоспособности, был выбрана система мониторинга Zabbix в силу того, что она:

- Свободная;
- Готовое решение, функционал «из коробки».

Zabbix позволяет настраивать уведомления по email при каких-либо ошибках системы.

1.7 Клиентская часть

Клиентская часть предполагает работу веб приложения в браузере пользователя. Для оптимального удовлетворения большинства требований было принято решение разработать веб приложение с использованием технологии Single Page Application.

Single Page Application (SPA) — концепция одностраничного веб приложения, предполагающая использование архитектурного решения «Толстый клиент» (предоставляет собой функциональный интерфейс пользователя с выполнением бизнес-логики) и API серверной части для обмена данными [5].

Особенность «Толстого клиента» в данном случае заключается в том, что практически все промежуточные действия для получения документа происходят на клиенте и только данные принципиальных действий

отправляются на сервер, где в качестве ответа приходят необходимые данные для достижения пользовательской цели. Таким образом, это уменьшает зависимость от качества связи между клиентом и сервером. Без обращения клиента на сервер, смена представлений пользовательский интерфейс меняется практически мгновенно согласно UX [6]. В случае обращения, обмен данными происходит в асинхронном режиме, что позволяет продолжать работу пользователя без блокирования клиента в ожидании ответа сервера. За счет вынесения части бизнес-логики и обработки представлений на клиента, а также отделение представления от данных, — достигается слабая связанность клиентской и серверной части веб приложения. Таким образом, в сравнении с традиционным подходом организации веб приложения, работа клиентской части веб приложения характеризуется высоким быстродействием (однако имеет зависимость от аппаратных характеристик клиента) и возможностью корректной работы при некачественном соединении. SPA является оптимальным решением для удовлетворения требований. В качестве примера использования SPA можно рассматривать сервис электронной почты Gmail, популярные социальные сети и т.п., там, где не требуются вычислительные мощности, а требуется получить небольшую часть информации. SPA можно представить, как набор некоторых технологий и компонентов:

- Клиентское приложение.

Приложение, которое выполняется в браузере, изменяя или расширяя его возможности и организуя уровень абстракции для работы с бизнес-логикой, логикой представления, сервисом предоставляющим данные. Для такой организации работы клиента зачастую используют некоторые браузерные технологии, например, Java-апплеты, Silverlight, Flash, JavaScript.

В данном случае, в качестве средства реализации клиентского приложения, было принято решение выбрать язык JavaScript [7]. Это связано с тем, что JavaScript:

– Интегрирован в браузер;

– Не требует значительных вычислительных ресурсов по сравнению с другими технологиям;

– Распространен.

Для достижения гибкой и богатой функциональности клиентского приложения на JavaScript, целесообразно использовать фреймворк. В нашем случае, было принято решение выбрать JavaScript-фреймворк AngularJS, так как изначально согласуется с принципами SPA [8][9] и корректно работает в браузерах.

- API.

В качестве способа организации обмена данными с API, оптимальным решением было выбрать шаблон проектирования Representational State Transfer (REST), так как:

– Имеет простую, не избыточную семантику;

– Обладает универсальным интерфейсом;

– Позволяет использовать различные HTTP-клиенты.

При использовании REST, URL определяет семантику запросов. REST-API является современной альтернативой более сложным способам обмена данными, например, таким как SOAP или RPC.

- Формат обмена данными.

В качестве формата данных для обмена между клиентом и сервером, было решено использовать JSON, так как:

– Имеет встроенную поддержку в популярных браузерах, вышедших после 2009 года;

– Простой, не избыточный;

– Распространенный;

– Нативный формат JavaScript.

Данные в формате JSON, согласно RFC 4627, могут быть: JavaScript-объекты, массивы, строки в кавычках, числа, булевы значения, null.

- Транспорт.

Для обмена данными между клиентом и сервером, было решено использовать протокол HTTP, как общепринятый и доступный для запросов к API. А также объект XmlHttpRequest для использования технологии AJAX, которая имеет поддержку во множествах клиентских библиотек.

Возможно, другим решением стало бы использование WebSockets, однако, во-первых, на момент анализа решений не было найдено информации о том, что эта технология стандартизирована, что влекло бы дополнительный риск стабильности работы веб приложение. А во-вторых, это повысило бы сложность предоставления данных внешним приложениям.

1.8 Источник данных

Данные веб приложения необходимо систематизировано хранить и обрабатывать, в качестве чего используется база данных. Прежде чем выбрать систему управления базой данных (СУБД), следует рассмотреть предпосылки для выбора таковой. Учитывая бизнес-правила, например, что храниться будут объявления,которые запрещено редактироватьи такие особенности веб приложения как создание измененной копии объявления в личную коллекцию. Самая часто используемая операция при использовании СУБД — чтение данных, затем — добавление. Так как адреса имеют иерархическую структуру и документы могут несколько отличаться друг от друга некоторыми атрибутами, в зависимости от типа, а также что требования могут быть изменены, влача за собой возможное изменение атрибутов, было принято решение о выборе СУБД, где есть поддержка неструктурированной модели данных, — NoSQL [10].

На основании предпосылок и требований, было сделано заключение, что необходима СУБД, которая предлагает:

- Поддержку NoSQL;
- Возможность горизонтальной масштабируемости, в частности

репликации БД.

Плюсом будет поддержка данных типа JSON, так как JSON выбран в качестве формата обмена данными между клиентом и API. А также распространенность и документация. На основании сказанного, в качестве оптимального решения была выбрана СУБД MongoDB.

MongoDB — документно-ориентированная СУБД [11]. Хранит данные в формате BSON, что представляет собой бинарную форму представления JSON. Таким образом, запись в MongoDB является аналогом объектного представления в формате JSON.

MongoDB имеет интерфейс для выполнения операций на языке JavaScript, возможности асинхронной репликации, поддерживает необходимые типы данных, которые могут понадобиться при проектировании БД согласно предметной области, например, массив, дата, объект и другие. Рационально подходит для хранения иерархических данных [12].

1.9 Серверная часть

Начнем проектирование с детализации серверной части приложения.

Поскольку основной целью приложения является предоставление услуг по средствам сети Интернет, необходимо включить в его серверную часть веб-сервер — аппаратно-программный комплекс, предназначенный для обслуживания HTTP-запросов. HTTP запрос — сформированный согласно протоколу HTTP/1.1 запрос на сервер на заранее определенный порт (по умолчанию, порты 80 и 8080) с целью выполнения какого-либо удаленного действия (манипуляции с информацией, выполнения определенных команд и т.д.). Как правило, такие запросы посылает браузер клиента.

Поскольку приложение предполагает большое число операций по чтению, записи и изменению значительного объема данных, наиболее удобным вариантом будет включение в серверную часть технологий баз данных.

На рисунке 6 изображен процесс детализации первоначальной архитектуры приложения, а точнее его серверной части. Серверная часть вмещает себя веб-сервер и сервер баз данных. В задачи веб-сервера входят:

- Получение и ответ на HTTP-запросы;
- Перенаправление запросов на необходимое приложение (сайт), как правило, приписанное к определенному домену или поддомену;
- Предоставление приложениям доступа к необходимым модулям;
- Авторизация и аутентификация пользователей;
- Реализация функций файл-сервера;
- Другие функции.

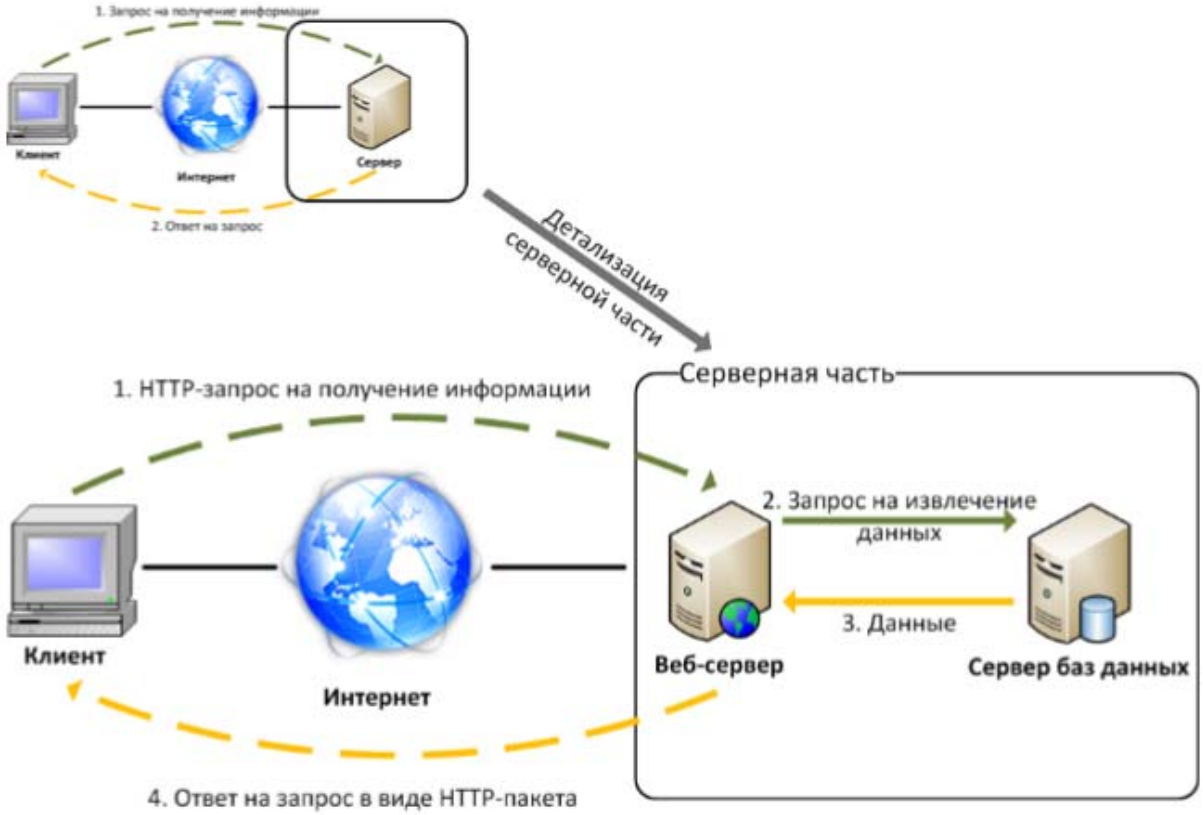

Рисунок 6 — Детализация серверной части приложения

В качестве языка программирования для разработки серверной части был выбран JavaScript по следующим причинам:

- Распространенность и документация;

- MongoDB имеет интерфейс запросов на JavaScript;
- JSON нативный формат JavaScript и используется для обмена данными между клиентской частью и API серверной части;
- Использование одного языка JavaScript и на клиенте и на сервере и при работе с СУБД, позволит упростить и ускорить процесс разработки, использовать одни и те же библиотеки на всех уровнях веб приложения.

JavaScript — прототипно-ориентированный сценарный язык программирования. Другими словами, он является ограниченным объектно-ориентированным, когда нет понятия класса, наследования. В этом случае принципы объектно-ориентированного программирования имитируются, например наследование — это клонирование основного объекта и добавление к нему необходимого поведения или свойств. Отношение строится как «прототип — наследник» без обязательного сохранения структуры прототипа. Таким образом, JavaScript позволяет имитировать объектно-ориентированный подход к разработке. Также существует возможность использовать аналоги JavaScript, такие как TypeScript или CoffeScript. В связи с этим было принято решение придерживаться парадигмы ООП в данной работе.

Для того чтобы выполнить код JavaScript на сервере, он должен быть интерпретирован в машинный код. Для чего был сделан выбор в пользу программной платформы Node.js, как среды выполнения JavaScript, так как:

- Работает согласно асинхронной модели;
- Перспективен и хорошо документирован;
- Имеет в наличии часто используемые библиотеки и модули, написанные на JavaScript.

Node.js является серверной технологией, которая основана на разработанном компанией Google JavaScript-движке V8. Это прекрасно масштабируемая система, поддерживающая не программные потоки или отдельные процессы, а асинхронный ввод-вывод, управляемый событиями. Она идеально подходит длявеб приложений, которые не выполняют сложных

вычислений, но к которым происходят частые обращения. По целям использования Node сходен с фреймворками Twisted на языке Python и EventMachine на Ruby. В отличие от большинства программ JavaScript, этот фреймворк исполняется не в браузере клиента, а на стороне сервера [13].

В JavaScript можно создавать специализированные функции, но — в отличие от Java, Ruby или Perl — нет возможности объединить несколько функций в единый модуль или «пакет», который можно импортировать и экспортировать. Модули — с нетерпением ожидаемая особенность следующей основной версии JavaScript (ECMAScript 6), но до широкого внедрения этой версии Node.js использует свою собственную версию модулей на основе спецификации CommonJS. В первую очередь следует иметь в виду, что в Node.js функции базовых модулей асинхронны. Это означает, что функции ноды не блокируют поток, а исполняются в фоновом режиме. Отсюда сразу возникает понятие callback-функции — процедуры, которая выполняется после завершения исполнения кода в потоке. Node.js прекрасно показывает себя в задачах с большим числом операций ввода/вывода. Например чтение файлов, запись в БД.

Базовым пакетом, для создания http-сервера на Node является express — очень маленький и быстрый серверный веб фреймворк.

Основные возможности express:

- Гибкая система маршрутизации запросов;
- Перенаправления;
- Динамические представления;
- Уточнение контента;
- Особое внимание производительности;
- Обработка представлений и поддержка частичных шаблонов;
- Поддержка конфигураций на основе окружений;
- Оповещения, интегрированные с сессиями;
- Максимальное покрытие тестами;

- Утилиты для быстрой генерации основы приложений;
- Настройки представлений на уровне приложений.

Глава 2 Проектирование и реализация

2.1 Серверная часть

Архитектура серверной части построена на основе веб фреймворка Express. Он инкапсулирует обработку высокоуровневых объектов приложения, например, такие как объекты запроса и ответа, предоставляя их в распоряжение разработчику [17]. Для структурирования серверной части веб приложения были взяты принципы архитектурного типового решения MVC. Рассмотрим основные компонентные представления серверной части веб приложения, которые были разработаны.

Маршрутизатор (Route) — в соответствии с маршрутом делегирует дальнейшую обработку запроса управляющим объектам нижележащего уровня: контроллеру или API.

Контроллер (Controller) — отвечает за обработку запроса, определяет HTML для пользовательского интерфейса. Содержит методы для реализации функциональных требований к веб приложению. Использует модель для получения данных предметной области.

Модель (Model) — отвечает за бизнес-логику непосредственно связанную с предметной областью. Является моделью сущности. Предоставляет интерфейс для работы с сущностями. Инкапсулирует обработку данных соответствующей ей сущности. Mongoose — представляет «обертку» для моделей, предоставляет функциональность для работы с моделями.

Шаблон (Template) — представление пользовательского интерфейса в виде HTML разметки.

Вид (View) — представление, код которого необходимо обработать сервером приложения. API — стандартизированный интерфейс для работы с данными предметной области. Инкапсуляция бизнес-логики за определённым маршрутом. Возвращает все данные в формате JSON. Запрос к API

определяется параметром «api»в строке URL. В соответствии с REST, на примере работы с документами, семантика запросов представлены в таблице 1.

Таблица 1 — Семантика запросов в соответствии с REST

Ресурс	Метод			
	GET	POST	PUT	DELETE
/building	Вернуть список всех предложений	Создать предложение		
/building/:id	Вернуть одно предложение с указанным идентификатором		Обновить предложение с указанным идентификатором	Удалить предложение с указанным идентификатором. Согласно бизнес-правилам объявления удалять запрещено

Аналогично построены интерфейсы запросов и для других сущностей.

Для управления некоторыми сущностями была разработана функция для проверки доступа на выполнение тех или иных действий.

Согласно принципу замыкания (причины для изменения в пакете должны быть одинаковые) классы были объединены в пакеты, учитывая принятое решение MVC, рисунок 7.

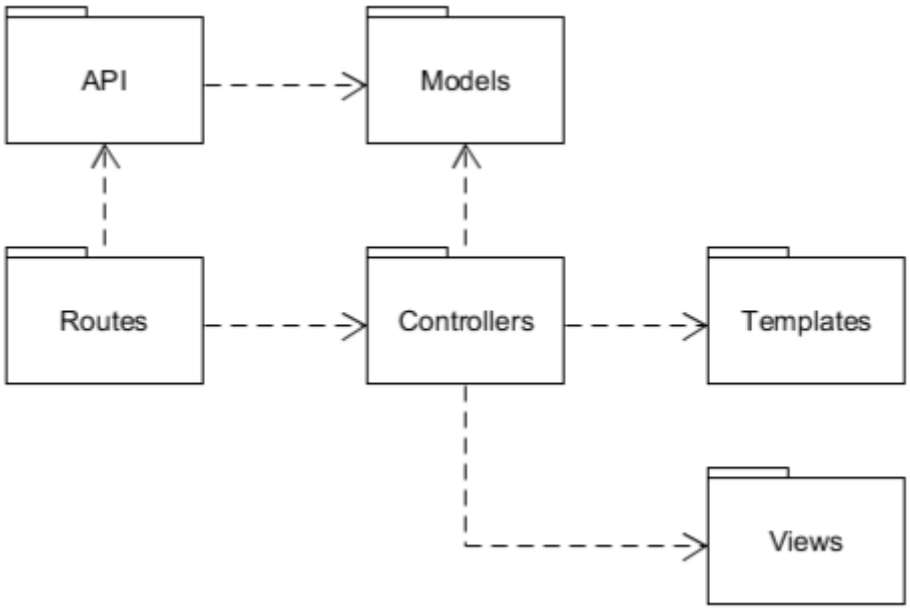

Рисунок 7 — Диаграмма пакетов серверной части веб приложения

В результате проектирования в целом, веб приложение можно разделить на подсистемы, рисунок 8.

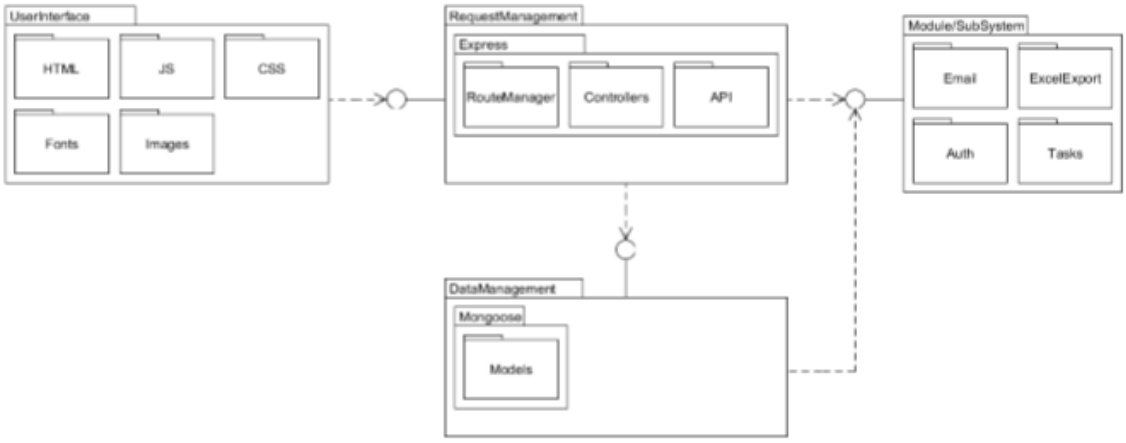

Рисунок 8 — Подсистемы вебприложения

Для удовлетворения требований были реализованы следующие классы и компоненты, представленные в таблице 2 и 3.

Таблица 2 — Классы пакета Routes

Класс	Описание
Server (index)	Отвечает за делегирование обработки запроса к контроллерам
Api	Отвечает за делегирование обработки запроса к API

Таблица 3 — Классы пакета Controllers

Класс	Описание
DocumentController	Обрабатывает запросы, предназначенные для управления объявлением
LanguageController	Обрабатывает запросы для управления языками
OrderController	Обрабатывает запросы для предоставления объявлений, отправки объявлений на почту
LoginController	Обрабатывает запросы ответственные за вход, выход, восстановление пароля агента недвижимости или пользователя
TemplateController	Обрабатывает запросы для получения шаблонов пользовательского интерфейса (HTML)

Для удовлетворения требований были также разработаны представления пользовательского интерфейса, которые располагаются в соответствующих пакетах, таблица 4. В пакете Views храниться HTML для случаев, когда его требуется обработать сервером приложений, таблица 5.

Таблица 4 —Шаблоны пакета Templates.

Шаблон	Описание
Search	Представление поиска объявлений
Order	Представление завершения заказа объявлений
Cart	Представление пользовательской корзины предложений
checkout	Представление ввода данных для заказа
Review	Представление проверки введенных данных заказа
Adress	Представление адреса доставки прайс листа
contact	Представление формы обратной связи

Таблица 5 —Представления пакета Views.

Представление	Описание
Layout	Общее представление пользовательского интерфейса
Login	Представление формы входа
recovery_password	Представление формы восстановления пароля
reset_password	Представление формы сброса пароля
angular	Представление для инициализации клиентского приложения на Angularjs
403, 404, 500	Представление пользовательских ошибок

В работе описаны некоторые из представленных классов. Рассмотрим их отношения и взаимодействия соответствующих им объектов. Ниже, на диаграмме представлено поведение, когда Router делегирует обработку запроса участнику API или контроллеру на основании маршрута, рисунок 9.

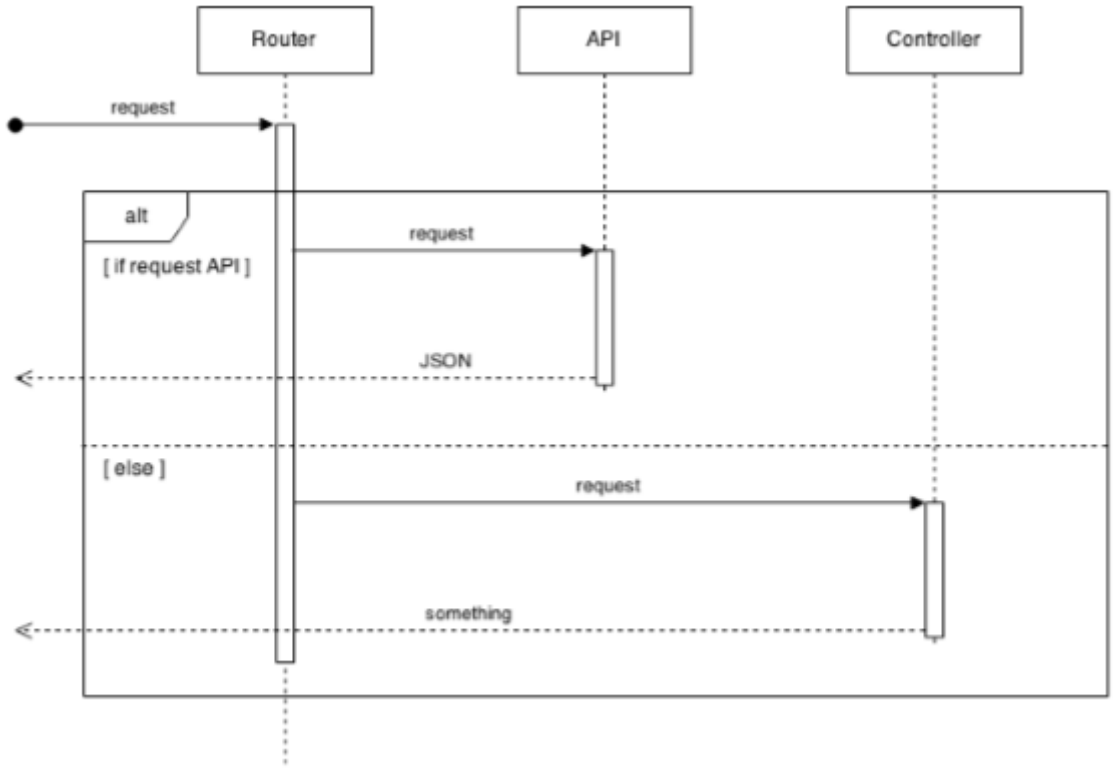

Рисунок 9 — Выбор участника обработки запроса

Взаимодействие объектов при заказе отправки объявлений на почту, представлено на рисунке 10.

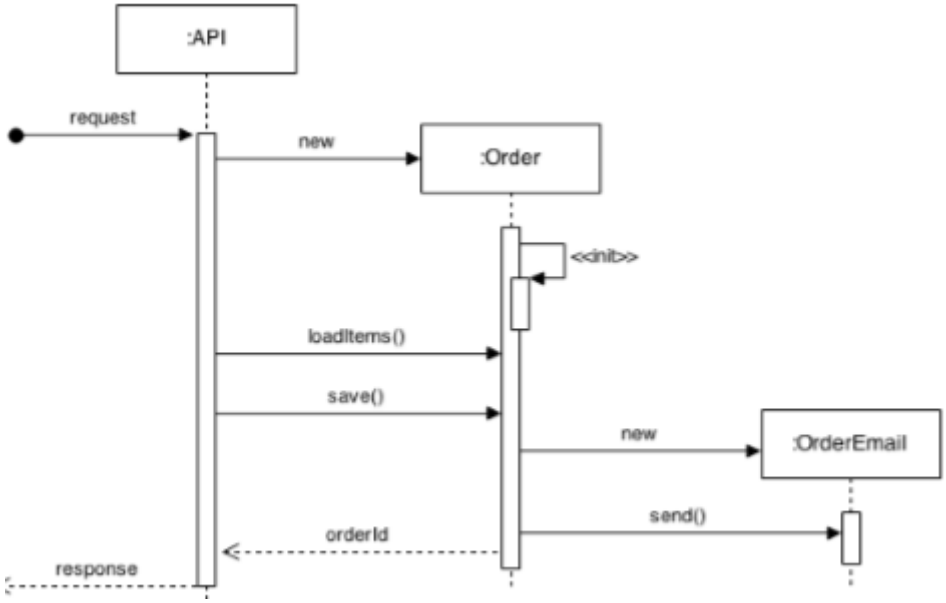

Рисунок 10 — Диаграмма взаимодействия при заказе отправкиобъявлений на почту

Взаимодействие объектов для получения свежего объявления по почте, представлено на рисунке 11.

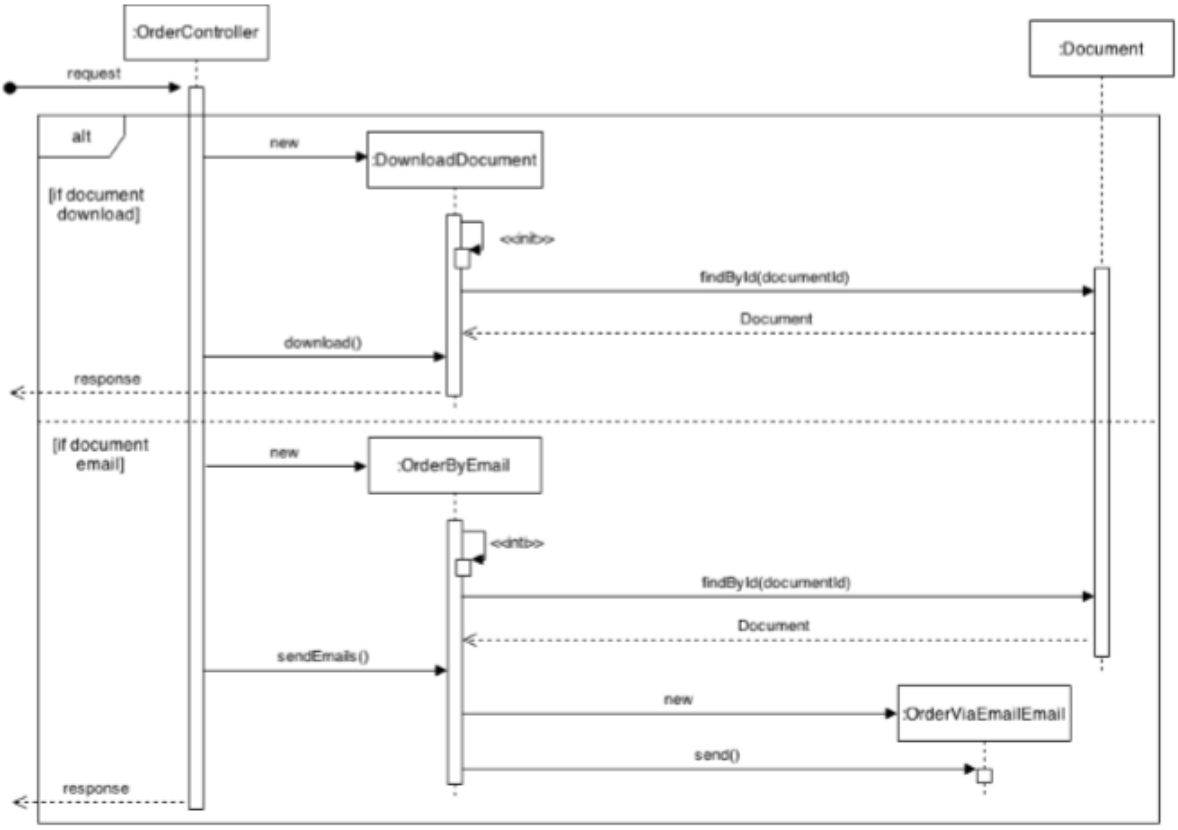

Рисунок 11 — Диаграмма взаимодействия для получения свежего объявления по почте

Для варианта использования «Восстановить пароль» были реализованы следующие классы, рисунок12.

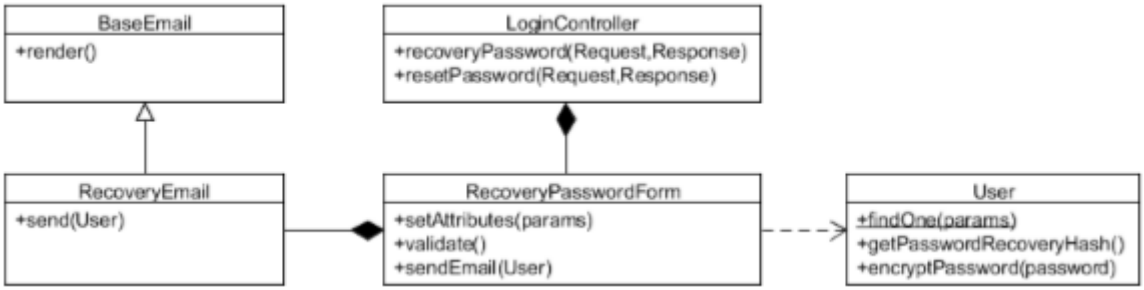

Рисунок 12 — Диаграмма классов для восстановления пароля

Взаимодействие объектов для сброса пароля, представлено на рисунке 13.

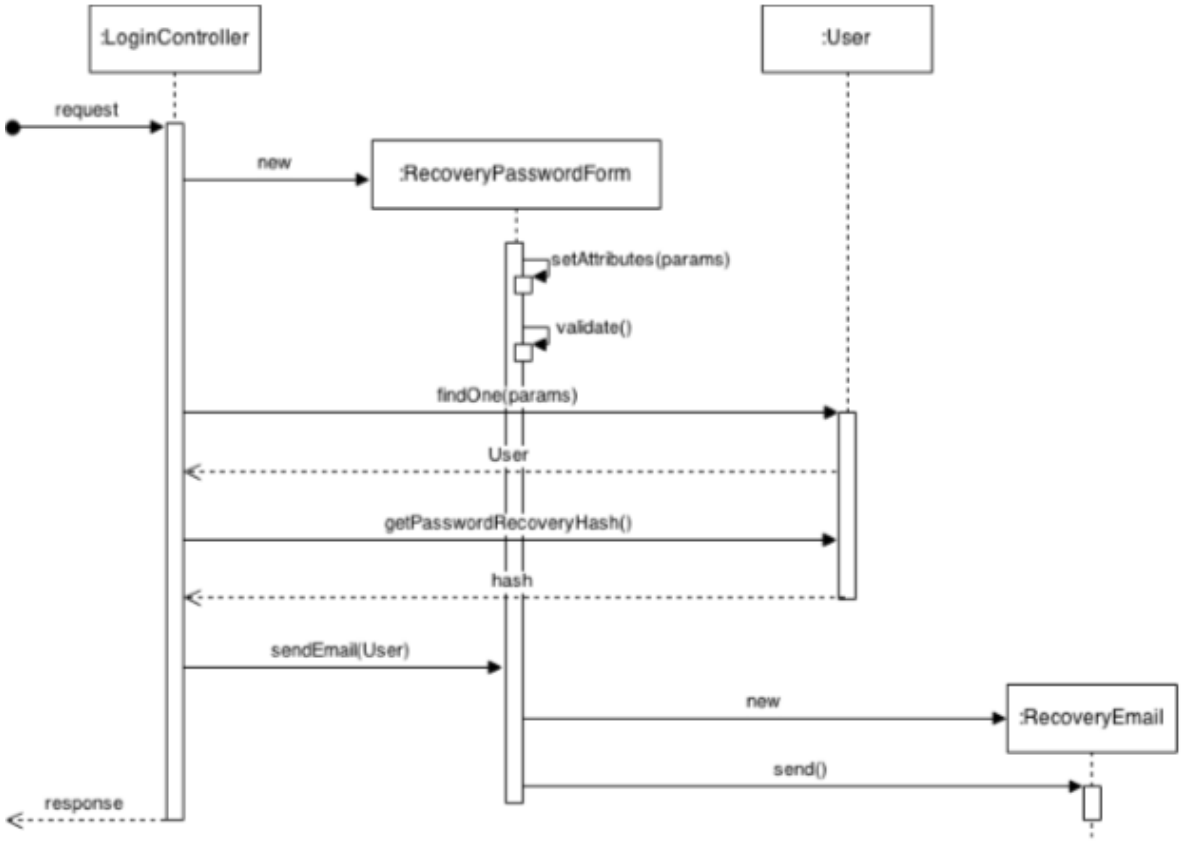

Рисунок 13 — Диаграмма взаимодействия для сброса пароля

Взаимодействие объектов для создания пароля, представлено на рисунке 14.

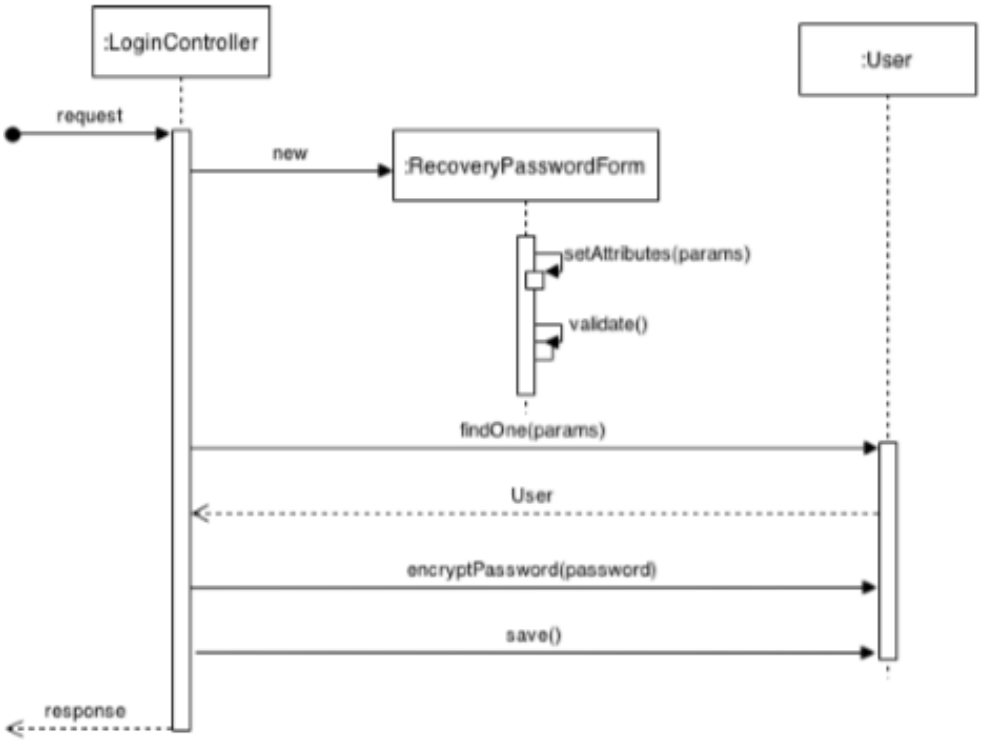

Рисунок 14— Диаграмма взаимодействия для создания пароля

В результате проектирования в целом, ВП можно разделить на подсистемы, рисунок 15.

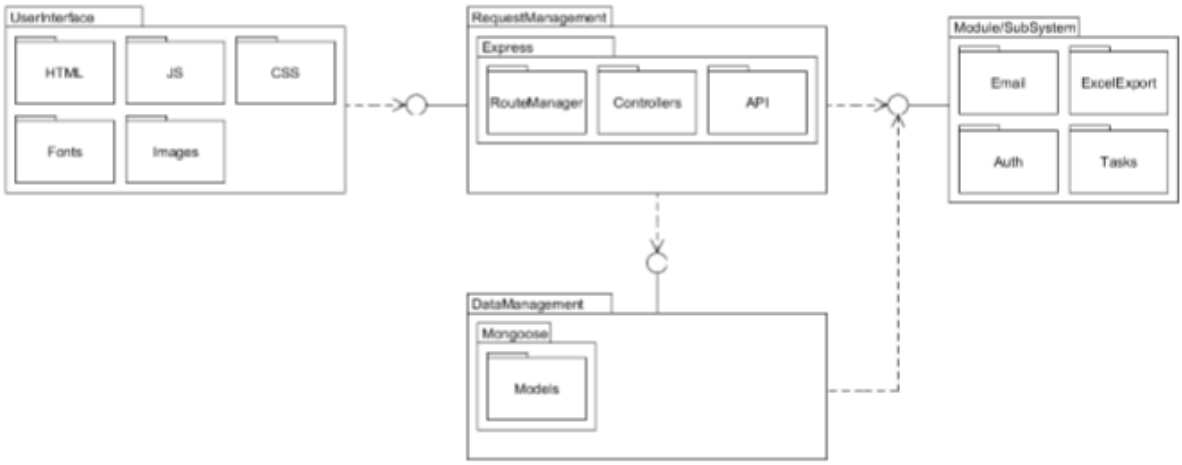

Рисунок 15 — Подсистемы веб приложения

2.2 Клиентская часть

Клиентская часть — это пользовательский интерфейс, который состоит из HTML, CSS, графических файлов и описание логики на языке JavaScript. В нашем случае, именно выполнение логики на JavaScript в браузере и именуется, как клиентское приложение или JavaScript-приложение.

Клиентская часть представляет собой архитектурное решение «Толстый клиент», что предполагает функциональный пользовательский интерфейс, где и выполняется часть бизнес логики веб приложения. Что, в частности, позволяет выполнить некоторые шаги для получения документа, без запроса на сервер. В клиентское приложение было решено вынести ту бизнес-логику, которая не требует обращения к БД.

Клиентское приложение работает в браузере с объектной моделью документа (DOM). DOM является программным интерфейсом для доступа к документам. Чтобы организовать работу должным образом с DOM, API серверной части и прочим, как правило, требуются объекты JavaScript ответственные за бизнес-логику, логику управления, логику представления.

В процессе разработки, были отмечены некоторые особенности клиентской части:

- Разработка логики клиентской части может вестись независимо от серверной части;
- При интерпретации HTML клиент находит указание на то, какие файлы JavaScript приложения необходимо запросить;
- JavaScript переопределяет некоторые стандартные события DOM, например, события перехода по ссылкам, отправки форм и т.п. Перехват JavaScript стандартных событий используется для выполнения той или иной логики;
- HTML изначально пустой, без данных. Данные запрашиваются у API на сервере. JavaScript-приложение использует возможность выполнить

запрос на сервер посредством объекта XmlHttpRequest. После ответа, клиентское приложение решает, что необходимо выполнить на клиенте;

- Данные клиентского приложения хранятся в памяти.

Напомним, что в качестве основы для клиентской части был выбран фреймворк AngularJS. AngularJS гибкий фреймворк [18] и позволяет разработать клиентское приложение, основанное на типовых решениях проектирования. В результате клиентское приложение легко расширяется. Множество решений предоставляется «из коробки» [19], например, AngularJS позволяет автоматически проверять на корректность ввод данных в поля формы. Не конфликтует с другими библиотеками на JavaScript.

Основными объектами и компонентом клиентского приложения являются:

- Шаблон - HTML со специальными маркерами, — выражениями, для интерпретации кода JavaScript. Выражение — доступные переменные и функции из области видимости. Является компонентом.
- Представление — DOM, то, что видит пользователь в окне браузера;
- Директивы — функции для расширения возможностей HTML;
- Маршрутизатор — обеспечивает работу с адресной строкой клиента;
- Контролер — частная бизнес-логика для представления;
- Сервис — общая бизнес-логика, независимая от представления;
- Модель — данные отраженные в представлении;
- Область видимости — контекст, в котором хранятся данные модели, доступные для контроллера и др. компонентов. Иногда именуется ViewModel (модель представления).

Шаблон предоставляет разметку HTML с директивами, выражениями, фильтрами, формами. За счет директив, после обработки контроллером и

моделью, добавляется поведение для представления. Такой механизм позволяет создать динамический DOM.

Существует возможность разбить шаблон на блоки, где каждый блок будет являться независимым от других и самостоятельно запрашивать данные и иметь свое представление для отображения таковых.

Представление отвечает за изменение HTML через DOM, модифицированный в соответствии с шаблоном, контроллером и моделью. Таким образом, достигается смена представлений. Представление связано с моделью через двухстороннюю подписку, поэтому изменение представления изменяет модель и наоборот. Таким образом, они следят за состоянием друг друга. Представление изменяет HTML в тех местах где оно связано как-то с моделью, например пользовательский интерфейс выхода или входа пользователя, в зависимости от того авторизован он или нет, представление будет использовать модель текущего пользователя. Чтобы сгенерировать представление, требуется применить к шаблону функции контроллера связанные с областью видимости, связать свойства модели с областью видимости, а затем сгенерировать модифицированный DOM. Представление знает о контроллере через директивы (функции) и выражения в шаблоне. Представление знает о модели в случае двунаправленной привязке.

Директивы расширяет возможности HTML, а также инкапсулируют логику представления и манипуляций с DOM. Представляют собой HTML синтаксис, который будет обработан клиентским приложением. За счет директив достигается повторное использование кода.

Маршрутизатор наблюдает за состоянием адресной строки, определяет соответствующий контроллер и представление для установленного URL.

Контроллер отвечает за поведение представления или его отдельной части. Представление определяет контроллер. Изменяет состояние модели. Контроллер подписывает представления на изменение модели, которые будут уведомлены после того, как контроллер будет выполнен. Контроллер

предоставляет в области видимости возможные поведения для представления. А также инициализирует данные в области видимости.

Именно один контроллер ответственен за бизнес-логику только одного представления. В случае чрезмерной бизнес-логики или дополнительной, следует перенести в объект сервис, а затем использовать сервис в контроллере. Это позволит не превращать контроллер в сценарий транзакции.

Сервис отвечает за коммуникацию с сервером, в частности за обмен данными с API. Сервис работает с высокоуровневыми объектами браузера, например, такими как XmlHttpRequest для работы с AJAX. Данные, которые приходят с сервера, представляют собой «сырую» модель для клиентского приложения.

Данные модели могут быть от примитивных типов до коллекций объектов.

Чтобы обеспечить актуальность данных в представлении, представление подписывается на события изменения модели. Аналогичным образом представление уведомляет модель, таким образом, обеспечивая целостность данных между представлением и моделью. Объект области видимости, наблюдает за изменениями и событиями в модели и DOM, изменяя поведение клиентского приложения. Именно типовое решение «внедрение зависимостей» позволяет декларативно описывать взаимосвязи между компонентами клиентского приложения.

Свойства области видимости представляет собой модель, которое может получить представление и другие компоненты приложения. Любые типы данных присвоенные области видимости становятся свойствами модели. Область видимости задается в шаблоне, где указывается элемент DOM, который ограничит видимость. Таким образом, можно задавать только часть шаблона, который будет обрабатываться клиентским приложением.

На основе компонентов фреймворка AngularJS, было разработано клиентское приложение. В итоге, если рассматривать процесс получения

списка доступных объявлений, то после выбора минимально допустимого количества критериев поиска, клиентское приложение реагирует на это событие и отправляет запрос на сервер для получения данных.

2.3 Общее представление веб приложения

Веб приложение использует трехуровневую архитектуру. Коммуникация клиентского приложения, сервера приложений и СУБД, осуществляется текстовыми структурированными данными в формате JSON, рисунок 16. Логика клиентской и серверной части реализуются на языке JavaScript.

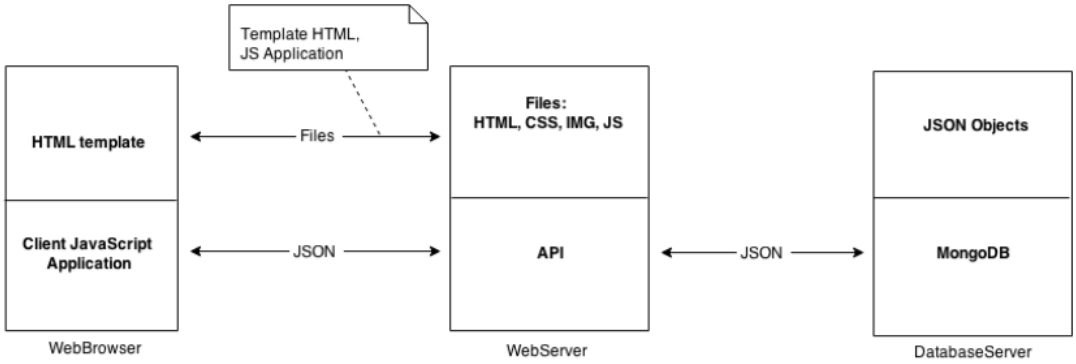

Рисунок 16 — Обмен данными в формате JSON в трехуровневой архитектуре.

Приложения каждого уровня используют объектное представление JSON. Экземпляр сущности предметной области на всех трех уровнях представляет собой JSON-объект.

Протокол транспортного уровня получает данные от представительского уровня, согласно модели OSI, в текстовом (бинарном) формате, именно поэтому при получении этих данных их необходимо привести к объектному виду, чтобы приложения могли использовать их как объекты JSON. Чтобы получить объектный вид достаточно воспользоваться нативными JavaScript-функциями преобразования из строкового вида к объектному, — операция десериализации.

И серверная часть, и клиентская часть веб приложения реализуют бизнес-логику, однако данная ответственность была разделена по принципу обращения к БД или северным возможностям, например, таким как отправление email. В процессе получения документа, это опорные точки которые предполагают выполнение запросов к серверной части, в частности к API:

- Отправить критерии поиска, чтобы получить список объявлений;
- Отправить запрос на получение анкеты пользователя;
- Отправить запрос на отправку индивидуального предложения.

Первый запрос клиента к серверу есть не что иное, как получение клиентского приложения. После чего оно выполняется на клиенте, при необходимости запрашивая данные у API серверной части. API имеет стандартизированный интерфейс для запросов согласно REST. API отражает модель предметной области.

2.4 Проектирование схемы базы данных

В MongoDB база данных состоит из коллекций. Как правило, коллекция отражает сущность предметной области. Коллекция состоит из «документов» (далее запись). Запись представляет собой JSON-объект с обязательным наличием идентификатора ObjectId, который MongoDB задает автоматически и контролирует его уникальность, вплоть до уникальности во всем мире [20]. ObjectId является типом данных. В сравнении с реляционными СУБД, коллекция является таблицей, а запись является строкой в таблице.

Тип хранимых в записи данных - произвольный, из перечисленных ниже:

- String;
- Array (структура данных с доступом по целочисленному индексу);
- Boolean;
- Date;
- Integer;

- Null;
- Object (структура данных, предоставляющая доступ к элементам по ключу).

Целостность связей в MongoDB не поддерживается. Эта задача решается на уровне кода сервера приложений, что и было реализовано с помощью инструмента ODM, — Mongoose. Для того чтобы иметь представление отношений между сущностями, была разработана схема БД, на которой отражены псевдо-связи, рисунок 17, нотация Гордона Эвереста.

Рисунок 17— Схема базы данных с обозначением псевдо-связей

Дополним к схеме, что наименование коллекций и псевдо-связи могут представлять несколько иной смысл, чем таблицы в реляционных БД, так как существуют вложенные объекты. Примером является коллекция с наименованием product_type.

Для описания особенностей проектирования схем в MongoDB, были использованы следующие понятия:

- Вложение — полное содержание объекта(ов) одной сущности в другой.
- Ссылка — уникальный идентификатор ObjectId внешней сущности.

На рисунке 18, представлен пример вложения одного объекта в другой в коллекции product_type.

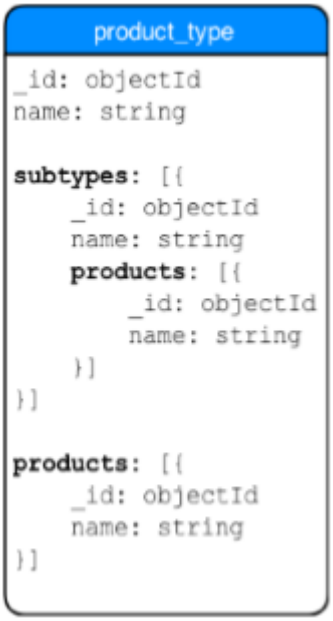

Рисунок 18 — Вложение объекта в коллекции

Использование инструмента Mongoose позволяет автоматически генерировать ObjectId при вставке вложенного объекта. Например, при добавлении продукта в тип продукта.

При проектировании были определены некоторые особенности проектирования схемы БД в MongoDB, которые заключаются в том, что:

- При составных псевдо-ключах резко теряется гибкость работы с данными;
- Хранение только идентификатора объекта внешней сущности потребует выполнить дополнительный запрос для получения всех данных внешней сущности;

- Вложение объекта сущности имеет смысл тогда, когда вложенный объект будет встречаться вместе с родительским;
- Вложение объекта сущности отражает в реляционной схеме связь один ко многим;
- Ссылки обеспечивают большую гибкость при частых изменениях вложенной сущности;
- Массив ссылок объектов сущности отражает в реляционной схеме связь многие ко многим.

Атрибуты основной сущности документ, представлены в таблице 6.

Таблица 6— Атрибуты основной сущности документ

Наименование атрибута	Тип	Описание
title	String	Наименование объекта
address	Object	Адресная информация
space	Object	Площадь объекта недвижимости
price	Number	Цена объекта недвижимости
comment	String	Описание объекта недвижимости
contact	Object	Контактные данные
removed	Date	С какой даты объявление в архиве

При запросе объявления происходит поиск по атрибуту address, и проверяется его актуальность через атрибут removed.Результат должен вернуть объявления по заданному адресу.

ЗАКЛЮЧЕНИЕ

Результатом выполненной работы является веб приложение.

В данной работе представлен процесс анализа, проектированияи разработки системы.

В начале проектирования системы были поставлены задачи,сформированы требования к проекту.Формулировка требований и их утверждение привели к началу этапа построения системы.

В ходе проектирования системы были определены средства для ее разработки. Анализ этих средств помог установить преимущества разработки и определить особенности подобных средств.

Заключительным этапом стало — разработка всех частей веб приложения, был создан интерфейс проекта, разработана база данных.

Спроектированная система позволит получить быстрый и удобный доступ к информации об объектах недвижимости и предоставит персонализированный подбор вариантов с учетом требований пользователя. А также, повысит эффективность взаимодействия организаций с пользователями, предоставляя агентам недвижимости удобное рабочее пространство, с различными преимуществами, а пользователям качественный и ненавязчивый сервис.

В ходе разработки были:

- Получены навыки анализа и проектирования.
- Изучены особенности одностраничных веб-приложений.
- Получены навыки проектирования неструктурированных баз данных.

Полученный опыт будет использован для разработки приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СТО 4.2-07-2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. – Введ. 30.12.2013. – Красноярск : ИПК СФУ, 2013. – 60 с.

2. Штайнер, Г. HTML/XML/CSS : пер с англ. / Г. Штайнер – Москва : Лаборатория базовых знаний, 2005. – 512 с.

3. Угринович, Н.Д. Практикум по информатике и информационным технологиям / Н.Д. Угринович, Л.Л. Босова, Н.И. Михайлова. – Москва : Издательство "БИНОМ. Лаборатория знаний", 2002. – 400 с.

4. Ульман, Д. Основы реляционных баз данных / Д. Ульман, Д. Уид – : Москва : Лори, 2006. – 384 с.

5. Шелдон, Р. MySQL. Базовый курс для начинающих : пер с англ. / Р.Шелдон, Дж.Мойе – Москва : Издательский дом «Вильямс», 2007. – 880 с.

6. Шаши, Ш. Основы построения баз данных : пер с англ. / Ш. Шаши. – Москва : КУДИЦ-ОБРАЗ, 2004. – 336 с.

7. Нильсен, Я. Веб-дизайн: книга Якоба Нильсена : пер с англ. / Я. Нильсен. — Санкт-Петербург : Символ-Плюс, 2000. – 512 с.

8. Глушаков, С.В. Программирование Web-страниц / С.В. Глушаков, И.А.Жакин, Т.С. Хачиров. – Москва : ООО "Издательство АСТ"; Харьков: "Фолио", 2003. – 398 с.

9. Хомоненко, А. Д. Основы современных компьютерных технологий : учеб. пособие / ред.: А.Д. Хомоненко – Санкт-Петербург : КРОНА принт, 1998. – 496 с.

10. Хомоненко, А. Д. Базы данных : учебник для высших учебных заведений. – 4-е изд., доп. и перераб. / под ред. проф.: А.Д. Хомоненко – Санкт-Петербург : КОРОНА принт, 2004. – 736 с.

11. Румянцев, Д. Сам себе Web-программист. Практикум создания качественного Web-сайта : учебное пособие / Д. Румянцев. – Москва : ИНФРА-М, 2001. – 207 с.

12. Зандстра, М. PHP. Объекты, шаблоны и методики программирования : пер. с англ. / Зандстра, М. – Москва : Вильямс, 2015. – 576 с.

13. Контеров, Д. PHP 5 / Д. Котеров. – 2-е изд. – Санкт-Петербург : БХВ-Петербург, 2016. – 1104 с.

14. Скляр, Д. PHP. Рецепты программирования : пер. с англ. / Скляр Д., Трахтенберг А. – Санкт-Петербург : Питер, 2015. – 784 с.

15. Колесниченко, Д. PHP и MySQL. Разработка веб-приложений / Д.Колесниченко. – 5-е изд. – Санкт-Петербург : БХВ-Петербург, 2015. – 592 с.

16. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS : пер. с англ. / Р.Никсон. – Санкт-Петербург : Питер, 2015. – 688 с.

17. Закас, Н. JavaScript. Оптимизация производительности : пер. с англ. / Н.Закас. – Москва : Символ-Плюс, 2012. – 256 с.

18. Дронов, В. Macromedia Dreamweaver MX. Наиболее полное руководство : учебное пособие / В. Дронов. – Санкт-Петербург : БХВ-Петербург, 2003. – 728 с.

19. Флэнаган, Д. JavaScript. Подробное руководство : пер. с англ. / Д.Флэнаган. – Москва : Символ-Плюс, 2009. – 992 с.

20. Белунцов, В. Новейший самоучитель по разработке Web-страниц : учебное пособие / В. Белунцов. – Москва : Издательство "ДЕСС КОМ", 2000. – 448 с.