Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра программного обеспечения информационных технологий

К защите допустить:

Заведующий кафедрой ПОИТ

_____ Н. В. Лапицкая

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему

ПРОГРАММНОЕ СРЕДСТВО ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ ПРОДАЖИ КВАРТИР

БГУИР ДП 1-40 01 01 055 ПЗ

Студент	Е. М. Новик
Руководитель	Д. В. Горбачев
Консультанты:	
от кафедры ПОИТ	Т. И. Малиновская
по экономической части	Е. В. Анохин
Нормоконтролёр	В. А. Леванцевич

Рецензент

Минск 2017

РЕФЕРАТ

Пояснительная записка 90с., 18 рис., 1 табл., 0 формул и 1 литературный источник.

ПРОГРАММНОЕ СРЕДСТВО ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ ПРОДАЖИ КВАРТИР

Предметной областью разработки является возможность для продажи кватир, покупки новой недвижимости. Объект разработки – приложение для конечного пользователя, предоставляющее функционал по покупки и продажи недвижимости.

Целью разработки является создание удобного, простого приложения, пригодного для решения практических задач, возникающих при работе в области недвижимости.

При разработке проекта использовалась среда разработки Sublime Text 3 с различными расширениями, такими как, LatexTools, Package Control и т.д. Язык программирования приложения – Java.

Результатом разработки стало простое в использовании приложение, которое может быть легко интегрировано в процесс работы, предоставляющее различные возможности для удобного поиска, покупки, продажи недвижимости. Разработаны диаграмма классов, диаграмма вариантов использования, а также различные схемы алгоритмов.

Разработанное приложение является экономически эффективным, оно полностью оправдывает средства, вложенные в его разработку.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Инициализация – приведение областей памяти в состояние, исходное для последующей обработки или размещения данных.

Программа – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

Программное обеспечение – программы, процедуры, правила и любая соответствующая документация, относящиеся к работе вычислительной системы.

Программирование – практическая деятельность по созданию программ.

Программный модуль – программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

Подпрограмма – программа, являющаяся частью другой программы и удовлетворяющая требованиям языка программирования к структуре программы.

Спецификация программы – формализованное представление требований, предъявляемых к программе, которые должны быть удовлетворены при ее разработке, а также описание задачи, условия и эффекта действия без указания способа его достижения.

Веб-интерфейс – это совокупность средств, при помощи которых пользователь взаимодействует с веб-сайтом или любым другим приложением через браузер.

ООП – объектно-ориентированное программирование

ПС – программное средство

ОС – операционная система

БД – база данных

СУБД – система управления базами данных

JPA – спецификация Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных

DAO – data access object

ВВЕДЕНИЕ

При любом общественном устройстве особое место в системе общественных отношений занимает недвижимое имущество, так как с его функционированием связаны жизнь и деятельность людей во всех сферах бизнеса, управления и организации. Именно недвижимость формирует центральное звено всей системы рыночных отношений. Объекты недвижимости – не только важнейший товар, удовлетворяющий разнообразные личные потребности людей, но одновременно и капитал в вещной форме, приносящий доход. Основным видом сделок с недвижимостью является купля-продажа.

Такой бизнес, как купля-продажа недвижимости, несомненно, нуждается в компьютеризации своей деятельности и переходе от работы с бумажными документами к работе с электронными данными. Помимо этого, поиск необходимой и доступной недвижимости очень сложен, если использовать старые методы как поиск объявлений в газетах, прослушивание новости по телевизору и т.д. Для поиска такой информации, потенциальный клиент тратит много своего времени.

Целям оптимизации процесса в данной ситуации может послужить создание веб-приложения для продажи квартир, с помощью которой риелтор смогут частично (а в будущем полностью) отказаться от ведения записей на бумаге и которая поможет автоматизировать некоторые процессы по покупке недвижимости. А клиенты смогут искать подходящую недвижимость в интернете, используя для этого различные критерии поиска. Более того, человек сможет сам создать объявление для продажи недвижимости, указав всю необходимую информацию для этого.

Актуальность и значимость предлагаемой работы заключена как в теоретическом, так и практическом плане, поскольку операции с недвижимым имуществом стали массовыми и повседневными в предпринимательской деятельности граждан и юридических лиц. В данном дипломном проекте будет разработано ПС для продажи квартир, которое поможет автоматизировать многие процессы для риелторов, а так же предоставит легкий способ для поиска необходимой недвижимости или размещений объявлений для продажи квартир.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе будет произведён обзор предметной области задачи, решаемой в рамках дипломного проекта;

1.1 Предметная область

Объектом исследования данной работы является порядок купли-продажи жилых помещений, который регулирует вопрос прав человека на жилое помещение. Под предметом исследования понимаются современные проблемы регулирования продажи жилого помещения.

Следует отметить, что современный рынок жилья существенно отличается от рынка прошлых лет. Круг традиционных участников договора купли-продажи значительно расширился. Теперь обычно в сделку кроме покупателя и продавца включается третья сторона - посредник, в качестве которого выступает либо риэлтерская фирма, либо частный маклер. В сделках на рынке жилья все большую роль играют коммерческие банки, биржи, страховые компании, инвестиционные фонды и другие рыночные институты.

Учитывая высокую стоимость жилой площади, сумма заключаемых сделок оценивается в больших деньгах. Квартирный бизнес является одним из самых выгодных. Посреднические организации - риэлтерские фирмы и отдельные маклеры готовы оказать любые услуги гражданам по распоряжению принадлежащими им квартирами и домами, получая при этом наибольшие дивиденды.

К договору купли-продажи жилья применяются обязательные требования:

– письменная форма в виде одного документа, подписанного сторонами, с государственной регистрацией сделки и нового собственника;
– указание имени и регистрации по месту жительства;
– определенная (однозначная) характеристика предмета сделки;
– данные о возможных правах третьих лиц;
– цена жилого помещения и оплата расходов по договору;
– срок и порядок передачи имущества.
– Покупатели квартир преследуют различные цели:
– улучшение жилищных условий;
– перемена места жительства;
– вложение капитала в недвижимость для последующей продажи и

получения дохода.

Купля и продажа недвижимости включает в себя сложные процессы, оказывающие различные услуги для клиентов. Такие как обмен, продажа, покупка недвижимости и аренде жилья. В данной бизнес сфере исчерпывающие базы данных, содержащие информацию обо всех актуальных предложениях на рынке недвижимости, что позволяет предоставить клиенту информацию о предлагаемом объекте, полностью соответствующем его индивидуальным запросам. Такая сфера, несомненно, нуждается в компьютеризации своей деятельности, поскольку такой объем информации сложно обрабатывать людьми. Программное средство может автоматизировать данный процесс. Например, предоставить сервис для поиска недвижимости по различным критериям. В этом поиске можно использовать следующие критерии: площадь квар-тир, количество комнат, этаж, и другая полезная информация. Это позволяет экономить время потенциальных клиентов. Помимо этого существуют клиенты, которые хотят продать недвижимость.

1.2 Обзор существующих аналогов

Реалт.бай это программное средство, позволяющее размещать объявления для продажи недвижимости. Помимо этого предоставляет сервис для аренды жилья.

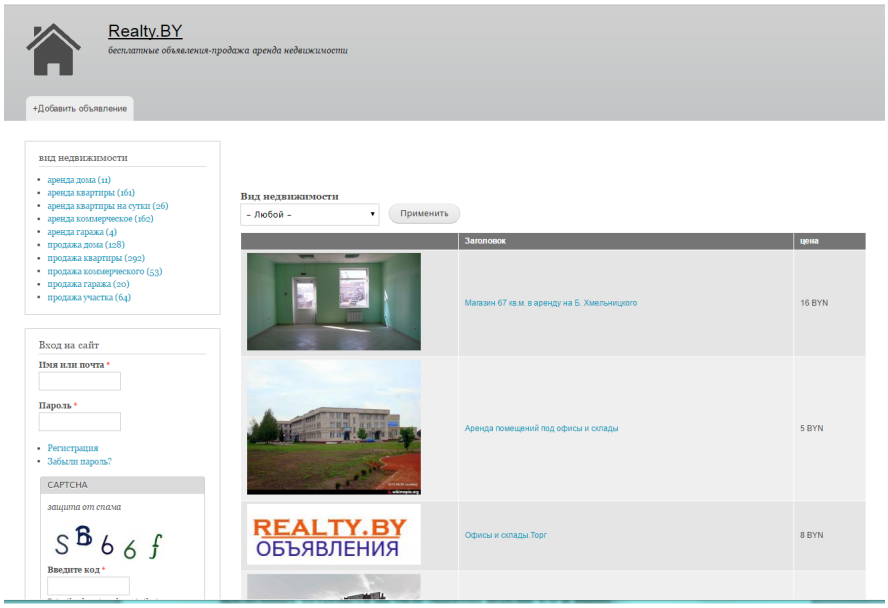

Рисунок 1.1 – Главный страница программного средсва

Реалт.бай это простой способ для размещения объявлений в сети. Веб-

интерфейс простой в использовании, все объявления хранятся в базе данных.

Реалт.бай это портал, на котором собрана вся информация о жилой и коммерческой недвижимости в Беларуси. На сайте можно найти информацию по вопросам аренды квартир и комнат в Минске, продаже квартир.

Прежде чем добавить объявление, необходимо пройти процедуру регистрации на сайте. Операция по добавлению нового объявления выглядит следующим образом. Пользователю необходимо добавить описание, тему заголовка, цену, добавить контактные данные. После этого будет размещено объявление.

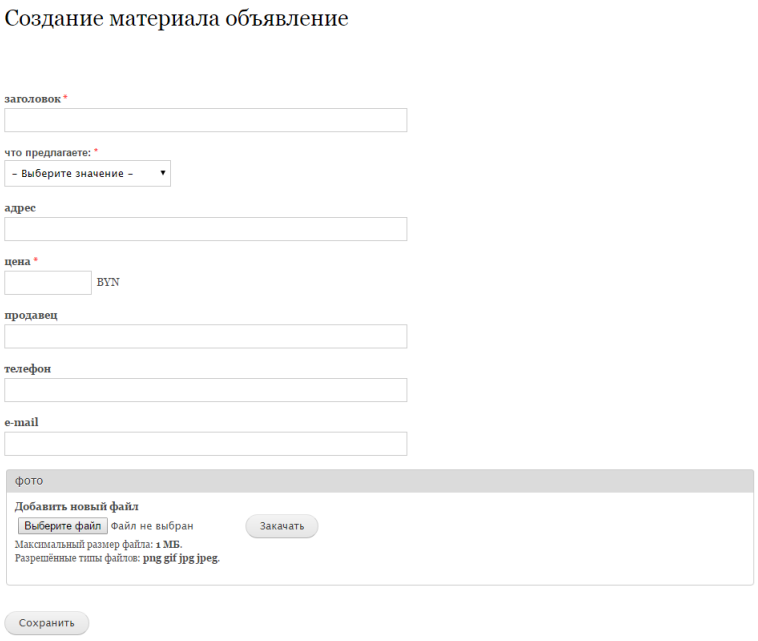

Рисунок 1.2 – Добавление нового объявления

Среди возможностей программного средства реалт.бай можно выделить следующие достоинства:

– понятный и простой интерфейс.

– возможность размещать объявление не только о продажи недвижимости.

– легкая обратная связь между клиентом и риелтором.

Помимо всех плюсов, стоит отметить несколько минусов данного программного средства:

– только один критерий для поиска объявлений.

– нет возможности редактирования информации.

– большинство предложений уже не актуально.

Таким образом, это программное средство подходит в качестве простого приложения, решающее только частично описанные проблемы.

Квартирант.бай это программное средство, предоставляющее следующий перечень услуг:

– аренда квартир;
– аренда комнат;
– аренда гостинец;
– продажа недвижимости;

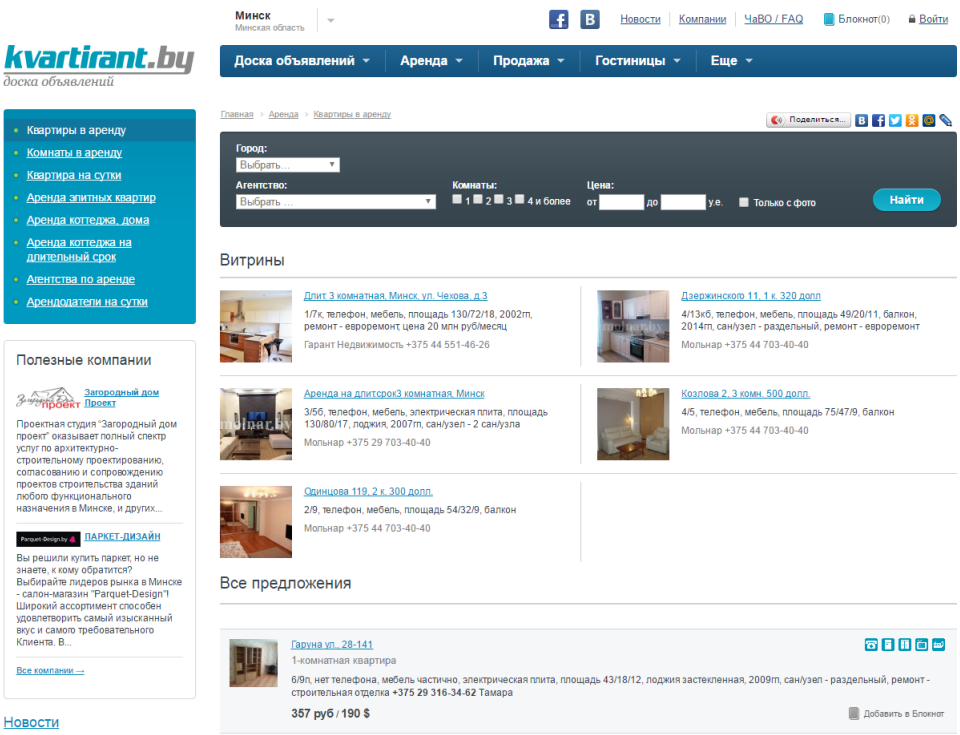

Рисунок 1.3 – Доска объявлений

Проект квартирант.бай стартовал в 2004 году как специализированный сайт по аренде всех видов недвижимости. Доска объявлений портала Квартирант.бай является популярной площадкой для бронирования гостинец, покупки недвижимости, аренды квартир.

Среди возможностей программного средства у можно выделить следующие достоинства:

– понятный интерфейс;
– легкая обратная связь между клиентом и риелтором;
– возможность размещать объявление не только о продажи недвижимости;

Помимо всех плюсов, стоит отметить несколько минусов данного про-

граммного средства:

– реклама;

– нет возможности редактирования информации;

– нет возможности оставить отзывы;

Таким образом, это программное средство подходит в качестве приложения, решающе не все описанные проблемы.

Программное средство оказывает услуги в сфере недвижимости. Продажа, покупка, обмен, разъезд, аренда комнат, квартир, коттеджей, дач, земельных участков, коммерческой недвижимости. Приватизация, утверждение перепланировок, вывод в нежилой фонд, консервация и другое.

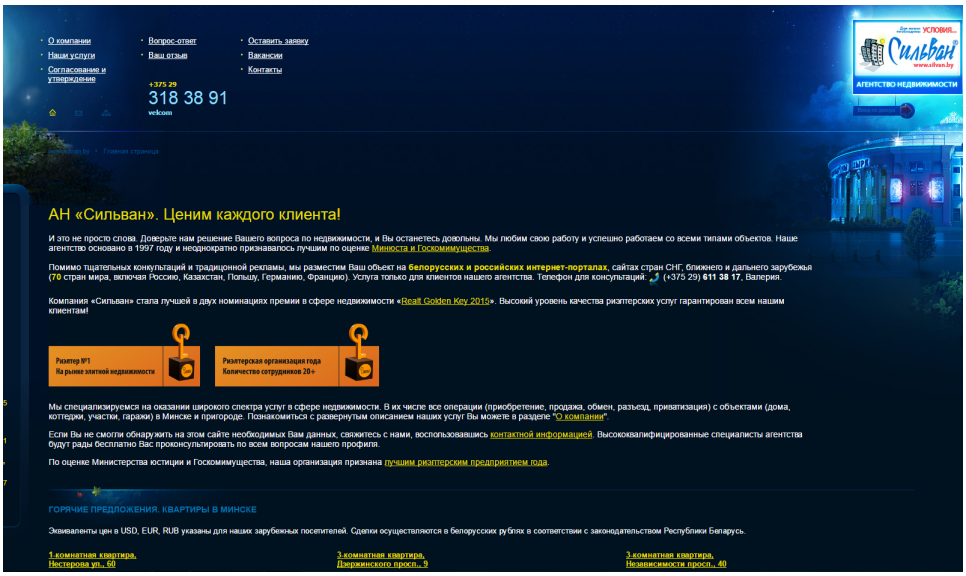

Рисунок 1.4 – Главная форма программного средства

Среди возможностей программного средства у можно выделить следующие достоинства:

– обратная связь между клиентом и риелтором;

– возможность поиска по различным критериям;

Стоит отметить минусы данного программного средства:

– яркий дизайн;

– нет пользователей как таковых

– нет возможности редактирования информации;

– нет возможности оставить отзывы;

– нет возможности добавить объявление;

– все операции делаются в ручную, клиент должен обратиться в офис компании;

Таким образом, данное программное средство не решает множество описанных проблемы. Программное средство представляет из себя частный случай для риэлтерской компании.

1.3 Постановка задачи

В результате выполнения дипломного проекта должно быть разработано программное средство для продажи квартир через веб-интерфейс.

Необходимо выполнить следующие задачи:

– изучить и улучшить знания в web разработке приложения;

– ознакомиться с многопоточными приложениями и особенностями платформы java;

– разработать программное средство для продажи квартир;

– разработать масштабируемое приложение, чтобы была возможность в будущем реализовывать дополнительный функционал;

Программное средство должно иметь два уровня доступа: администратор и пользователь. Пользователь может иметь доступ к своей учетной записи для добавления, поиска, редактирования своих объявлений. Администратор должен иметь возможность настраивать веб-интерфейс и управлять зарегистрированными пользователями.

Должны быть реализованы следующие ключевые функции:

– создание учетной записи пользователя;

– восстановление учетной записи пользователя;

– добавление объявления;

– редактирование объявления;

– удаления объявления;

– добавление фотографий;

– поиск объявления по множеству критериев;

– возможность оставить отзыв;

В процессе дипломного проектирования необходимо разработать приложение, которое будет решать все поставленные задачи и основные функции.

2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ

2.1 Используемые технологии

Выбор технологий является важным предварительным этапом разработки сложных информационных систем. Платформа и язык программирования, на котором будет реализована система, заслуживает большого внимания, так как исследования показали, что выбор языка программирования влияет на производительность труда программистов и качество создаваемого ими кода.

Ниже перечислены некоторые факторы, повлиявшие на выбор технологий:

– разрабатываемое ПО должно иметь возможность запускаться под платформами Windows(7,8,10) и Linux(Ubuntu, Arch Linux, Linux Mint);

– ПО работает в совокупности с другими средствами описания апарутры интегральных схем и должно иметь возможность запускаться в форме скрипта;

– среди различных платформ разработки имеющийся программист лучше всего знаком с разработкой на платформе;

– дальнейшей поддержкой проекта, возможно, будут заниматься разработчики, не принимавшие участие в выпуске первой версии;

– имеющийся разработчик имеет опыт работы с объектоориентированными языками программирования;

Основываясь на опыте работы имеющихся программистов разрабатывать ПО целесообразно с помощью языка Java. Приняв во внимание необходимость обеспечения доступности дальнейшей поддержки ПО, возможно, другой командой программистов, необходимость работы с различными ОС, скриптообразный характер ПО, целесообразно не использовать малоизвестные и сложные языки программирования.

С учетом этого фактора выбор языков программирования сужается до четырех: Python, Ruby и Java. Слабые по сравнению с другими языками механизмы ООП (отсутствие наследования, классов) языка LUA, которые могут быть полезны при разработке ПО, позволяют исключить этот язык из списка кандидатов. Python уступает по удобству использования двум другим кандидатам из нашего списка. Оставшиеся два языка программирования Ruby и Java являются хорошими кандидатами . Тот факт, что Java существует на рынке уже давно, делает этот язык предпочтительным кан-

дидатом.

Таким образом, с учетом вышеперечисленных факторов, целесообразно остановить выбор на следующих технологиях:

– операционные системы: семейство Windows(7,8,10), семейство Linux(Ubuntu, Debian, Arch Linux), Mac os;

– база данных MySql для хранения информации;

– язык программирования Java на котором будет реализована серверная часть программного средства;

– язык программирования Javascript и библиотека react js;

При реализации программного средства встает вопрос, в каком виде должна храниться вся информация и с помощью каких средств её следует обрабатывать. Так как, например, информация о каждом отдельном объявлении представляет собой типичный набор данных (площадь, количество комнат, этаж, год постройки и т. д.), то очевидно, что в этом случае целесообразно хранить их в реляционной базе данных на сервере. Посредством запросов к базе данных пользователь может получать нужные ему сведения, а администратор может добавлять и изменять данные. Выбор конкретной СУБД в качестве сервера баз данных осуществлялся исходя из тех преимуществ, которые она имеет перед другими, а также удобства работы с ней. В данном случае был выбрана клиент-серверная СУБД MySQL

Для реализации поставленной задачи предпочительно использовать на ранних этапах базу данных MySql. В случаи большого количество информации, можно будет слегкость мигрировать на другую СУБД, т.к язык Java использует спецификацию JPA, которая позволяет с легкостью мигрировать на другую СУБД.

Объектно-ориентированный язык программирования Java широко используется для создания серверных приложений. Язык java будет использован для создания высокоуровнего дизайна проложения (иерархия классов и интерфейсов, организация модулей и публичного программного интерфейса), реализации логики приложения, функций и методов, прототипирования различных идей.

Для реализации клиентской части был выбран язык программирования Javascript. Для быстроты и удобства реализации, будет использована библиотека react.js, которая написана на языке программирования Javascript.

Объектно-ориентированный язык программирования Java широко используется для создания серверных приложений.

Система Java создана на основе простого языка программирования, техника использования которого близка к общепринятой и обучение кото-

рому не требует значительных усилий [1].

Java как язык программирования является объектно-ориентированным с момента основания. Кроме того программист с самого начала обеспечивается набором стандартных библиотек, обеспечивающих функциональность от стандартного ввода/вывода и сетевых протоколов до графических пользовательских интерфейсов. Эти библиотеки легко могут быть расширены.

Несмотря на то, что язык C++ был отвергнут, синтаксис языка Java максимально приближен к синтаксису C++. Это делает язык знакомым широкому кругу программистов. В то же время из языка были удалены многие свойства, которые делают C++ излишне сложным для пользования, не являясь абсолютно необходимыми. В результате язык Java получился более простым и органичным, чем C++.

Надежность и безопасность Java существенно облегчает создание надежного программного обеспечения. Кроме исчерпывающей проверки на этапе компиляции, система предусматривается анализ на этапе выполнения. Сам язык спроектирован так, чтобы вырабатывать у программиста привычку писать "правильно". Модель работы с памятью, в которой исключено использование указателей, делает невозможными целый класс ошибок, характерных для C и C++.

В силу того, что Java предназначен для работы в распределенной среде, безопасность становится чрезвычайно важной проблемой. Требования безопасности определяют многие черты как языка, так и реализации всей системы. Компилятор Java производит байт-коды, т.е. модули приложения имеют архитектурно-независимый формат, который может быть проинтерпретирован на множестве разнообразных платформ. Это уже не исходные тексты, но еще не платформно-зависимые машинные коды .

Схема работы системы и набор байт-кодов виртуальной машины Java таковы, что позволяют достичь высокой производительности на этапе выполнения программы:

анализ кодов на соблюдение правил безопасности производится один раз до запуска кодов на выполнение, в момент выполнения таких проверок уже не нужно, и коды выполняются максимально эффективно ;

работа с базовыми типами максимально эффективна, для операций с ними зарезервированы специальные байт-коды;

методы в классах не обязательно связываются динамически;

автоматический сборщик мусора работает отдельным фоновым потоком, не замедляя основную работу программы, но в то же время обеспечи-

вая своевременный возврат свободной памяти в систему;

стандарт предусматривает возможность написания критических по производительности участков программы в машинных кодах;

Каждая из перечисленных характеристик по отдельности может быть найдена в уже существующих программных пакетах. Новым является соединение их в стройную непротиворечивую систему, которая должна стать всеобщим стандартом.

Java полагается на автоматическое управление памятью со стороны исполняющей среды, предоставляя совсем немного средств для управления жизненным циклом объектов. Не смотря на это, в языке все же присутствуют указатели на функции.

Создатели языка Java не являются противниками привнесения в язык новых идей и возможностей. Каждая новая версия интерпретатора языка привносит различные полезные возможности, которые отвечают требованиям индустрии.

Выбор конкретной СУБД в качестве сервера баз данных осуществлялся исходя из тех преимуществ, которые она имеет перед другими, а также удобства работы с ней. В данном случае был выбрана клиент-серверная СУБД MySQL. Её архитектура изображена на рисунке ниже [2].

Рисунок 2.1 – Клиент-серверная архитектура MySQL

Клиент-серверная архитектура MySQL Самая подходящая для MySQL сфера применения - это Интернет, благодаря хорошей системе безопасности этого пакета, стабильной работе и высокому быстродействию. Для создания скриптов был выбран язык программирования PHP, а MySQL – самая популярная СУБД, которая поддерживается этим языком. В PHP

есть множество функций, которые позволяют удобно и эффективно работать с базами данных – и это одна из причин выбора данной СУБД [2].

Рассмотрим преимущества MySQL:

– Быстродействие. Благодаря внутреннему механизму многопоточности быстродействие MySQL весьма высоко. Для разработчиков MySQL скорость всегда являлась ключевым параметром. Новые возможности добавлялись в пакет MySQL только после того, как их удавалось реализовать без ущерба для производительности. Иногда это означало, что некоторые возможности добавлялись не так быстро, как хотелось бы пользователям, но зато всегда гарантировало быструю работу MySQL.

– Безопасность. Довольно высокий уровень безопасности обеспечивается благодаря базе данных mysql, создающейся при установке пакета и содержащей пять таблиц. При помощи этих таблиц можно описать, какой пользователь из какого домена с какой таблицей может работать и какие команды он может применять. Пароли, хранящиеся в базе данных, можно зашифровать при помощи встроенной в MySQL функции password().

– Лицензия.Раньше лицензирование MySQL было немного запутанным; сейчас эта программа для некоммерческих целей распространяется бесплатно.

– Открытость кода. Благодаря этому программист может сам добавлять в пакет нужные функции, расширяя его функциональность так, как ему требуется. За отдельную плату это могут сделать и сами авторы MySQL.

– Простота использования. Для начала работы с MySQL не требуется сложной процедуры конфигурации. MySQL Server начнёт работать соответствующим образом сразу. По умолчанию выбираются значения, соответствующие минимальному использованию ресурсов диска и памяти. Для получения оптимальной производительности и для специальных условий (например, для проверки входа в систему), конечно же, потребуется дополнительная настройка. Чтобы помочь выполнить такую настройку, предлагаются соответствующие примеры файлов типовой конфигурации.

– Сообщество. Как следствие открытости кода, бесплатности программы, стабильной и надежной ее работы образовалось сообщество людей, которые не просто лояльны к MySQL, но и всячески участвуют как в развитии самого пакета, так и в обучении менее опытных людей работе с ним. Существует огромное количество листов рассылки и конференций, где можно получить бесплатную помощь в любое время суток.

В настоящее время существуют версии программы для большинства распространенных компьютерных платформ. Это говорит о том, что вам

не навязывают определенную операционную систему. Вы сами можете выбрать, с чем работать, например с Linux или Windows, но даже в случае замены ОС вы не потеряете свои данные и вам даже не понадобятся дополнительные инструменты для их переноса. Конечно же, как и любое программное средство, СУБД MySQL не избавлена от некоторых недостатков. Например, можно назвать отсутствие вложенных запросов, что приводит к необходимости находить нужные значения отдельно и подставлять их в другой запрос непосредственно в CGI-сценарии, что, несомненно, сказывается на производительности. Несмотря на это, СУБД MySQL была выбрана как наиболее подходящий сервер баз данных для программного средства.

JavaScript — прототипно-ориентированный сценарный язык программирования. Является реализацией языка ECMAScript (стандарт ECMA-262).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности вебстраницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса [3].

Структурно JavaScript можно представить в виде объединения трёх чётко различимых друг от друга частей:

– ядро (ECMAScript),
– объектная модель браузера (Browser Object Model или BOM (en)),
– объектная модель документа (Document Object Model или DOM).

Если рассматривать JavaScript в отличных от браузера окружениях, то объектная модель браузера и объектная модель документа могут не поддерживаться. [3]

Объектную модель документа иногда рассматривают как отдельную от JavaScript сущность, что согласуется с определением DOM как независимого от языка интерфейса документа. В противоположность этому ряд авторов находят BOM и DOM тесно взаимосвязанными.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

JavaScript является объектно-ориентированным языком, но используемое в языке прототипирование обуславливает отличия в работе с объектами

по сравнению с традиционными класс-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания — что придаёт языку дополнительную гибкость.

Несмотря на схожий с Си синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

– объекты, с возможностью интроспекции;
– функции как объекты первого класса;
– автоматическое приведение типов;
– автоматическая сборка мусора;
– анонимные функции;

В языке отсутствуют такие полезные вещи, как:

– JavaScript не предоставляет возможности управлять зависимостями и изоляцией областей видимости;
– отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода-вывода, базовых типов для бинарных данных;
– стандартные интерфейсы к веб-серверам и базам данных;
– система управления пакетами которая бы отслеживала зависимости и автоматически устанавливала их;

React является библиотекой, написаной на JavaScript с открытым исходным кодом для создания пользовательских интерфейсов. Библиотека была написана разработчиками из Facebook, Instagram и сообществом индивидуальных разработчиков и корпораций.

React позволяет разработчикам создавать крупные веб-приложения, которые используют данные, которые могут меняться со временем, без перезагрузки страницы. Его основная цель - быть быстрым, простым и масштабируемым.

Клиентская часть программного средства, будет разработана с помощью языка Javascript и библиотеки Ract.

2.2 Функциональное моделирование

Для функционального моделирования программного средства для продажи квартир использована диаграмма вариантов использования, являющаяся частью унифицированного языка моделирования (UML) [4]. Общий вид обобщенной диаграммы вариантов использования удаленного управления мультимедиа представлен ниже.

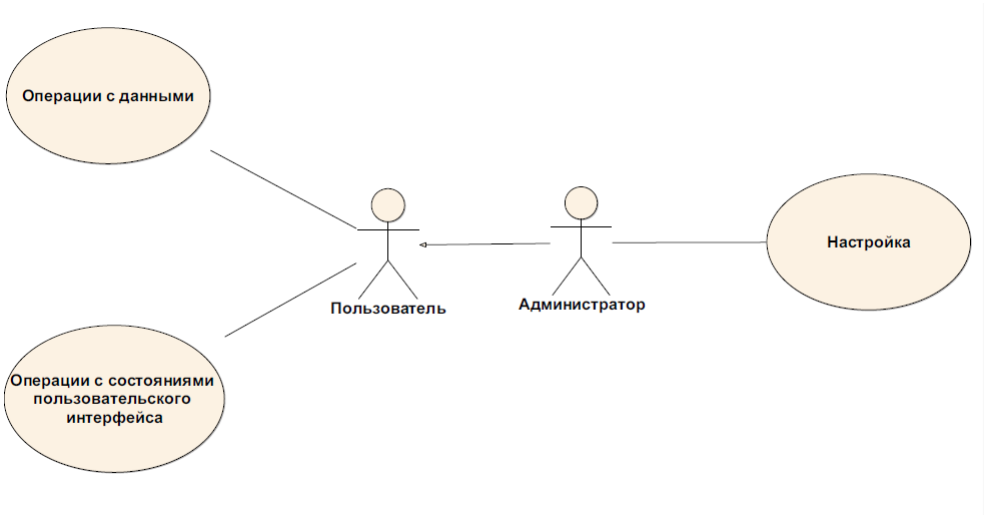

Рисунок 2.2 – Диаграмма вариантов использования

Как видно из диаграммы, для программного средства необходимы два актера:

– пользователь – это любой конечный пользователь из круга лиц, имеющий доступ к удаленному управлению данными на устройствах;

– администратор – это такой пользователь, который кроме всех обычных функций, доступных пользователю, имеет еще специфические функции настройки данных; следует отметить, что администратор наследуются от пользователя.

Среди основных функций пользователя можно выделить:

– операции с данными: все функции, связанные с отображением и управлением мультимедийных данных, дополнительные специфические функции с мультимедиа;

– операции с состояниями пользовательского интерфейса: все функции, предназначенные для управления состояниями пользовательского интерфейса.

У администратора следует выделить функции настройки программного средства, предназначенные настройки всех компонент, связанных с коректной работой пользователя.

Рассмотрим необходимые функции для работы с объявлениями. диаграмма вариантов использования «Работа с объявлениями» представлена ниже.

Как видно из диаграммы, функция «Работа с объявлениями» включает в себя следующие возможности:

– «Создать объявления»: данная функция дает возможность пользова-

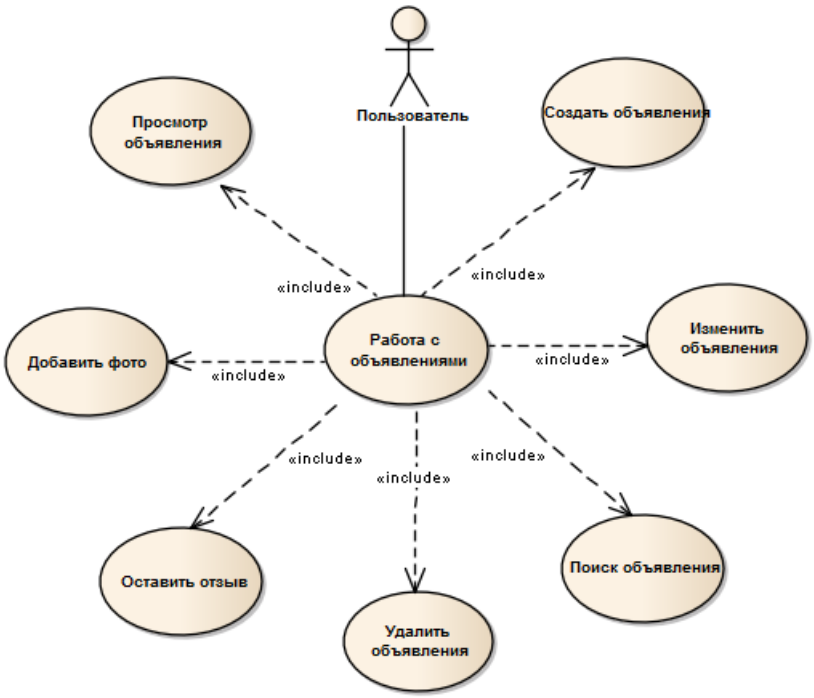

Рисунок 2.3 – диаграмма вариантов использования «Работа с объявлениями»

телю создавать объявления для возможности выставить свою недвижимость на продажу.

– «Изменить объявление»: функция означает возможность пользователем изменять ранее созданные объявления. Включает возможность редактирования информация, которая была описана при создании объявления.

– «Поиск объявления»: такая функция означает возможность пользователя поиска объявления для покупки или аренды недвижимости. Данная функция включает в себя множество критериев для поиска.

– «Удалить объявления»: функция означает возможность удалить выбранное объявление.

– «Оставить отзыв»: функция дает возможность пользователем добавлять комментарии, задавать дополнительные вопросы на счет недвижимости.

– «Добавить фото»: функция означает возможность пользователю добавить фото о недвижимости.

– «Просмотр объявления»: функция дает возможность пользователям просматривать объявление о недвижимости.

Рассмотрим подробнее функцию «Создать новое объявление». Диаграмма вариантов использования представлена ниже.

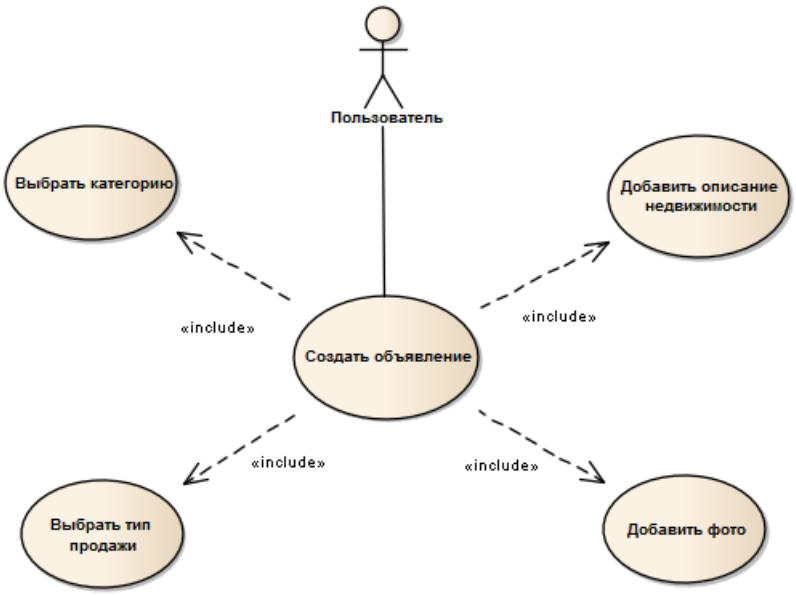

Рисунок 2.4 – Диаграмма вариантов использования «Создать объявление»

Как видно из диаграммы, функция «Создать объявления» включает в себя следующие возможности:

– «Добавить описание недвижимости»: данная функция дает возможность пользователю добавить необходимую информацию о своей недвижимости. Данная информация будет видна другим пользователям.

– «Добавить фото»: функция означает возможность пользователю добавить фотографии о недвижимости. Например пользователь может добавить фотографию планировки квартиры, фотографию гостиной, фотографию кухни и т.д .

– «Выбрать тип продажи»: эта функция дает возможность пользователя выбрать тип продажи недвижимости. Будет это аренда жилья, чистая продажа недвижимости или обмен с доплатой.

– «Выбрать категорию»: эта функция дает возможность пользователя выбрать категорию объявления. В зависимости от категории, будущее объявление будет размещено в нужной секции. Это функция облегчает поиск для других клиентов.

Рассмотрим необходимые функции для роли «Администратор». Диа-

грамма вариантов использования представлена ниже.

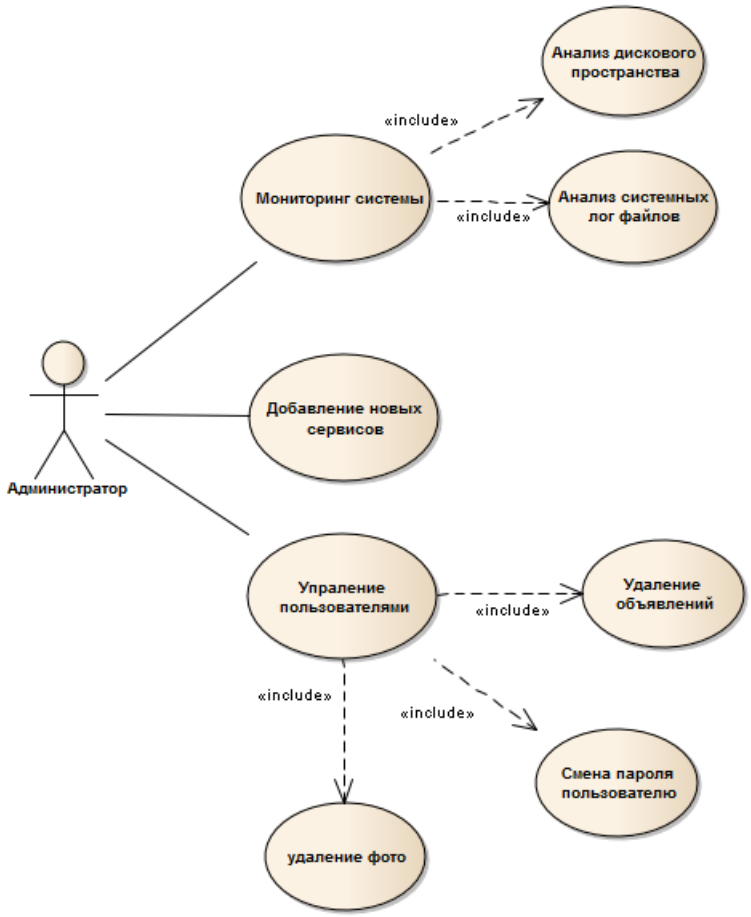

Рисунок 2.5 – Диаграмма вариантов использования для роли «Администратор»

Данная диаграмма показывает, какими функциями должен обладать пользователь, с ролью администратор. Рассмотрим каждую функции в подробном описании:

– «Анализ дискового пространства»: данная функция дает возможность администратору смотреть свободное место дискового пространства. Данная функция необходима, чтобы периодически чистить устаревшую информацию, которая будет храниться на диске.

– «Анализ системных лог файлов»: функция означает возможность администратору просматривать системные лог файлы на наличие ошибок. В случаи сбоя приложения, пользователю необходима узнать по какой причине произошёл сбой.

– «Добавление новых сервисов»: эта функция дает возможность администратору добавить новый сервис. данная функция предназначена для расширения функционала будущего программного средства.

– «Удаление объявления»: эта функция дает возможность администратору удалить любое объявление. Данная функция предназначена для удаления устаревших и неактуальных объявлений, в случаи, если пользователь сам не удалил объявление.

– «Удаление фото»: эта функция дает возможность администратору удалить любые фотграфии. Данная функция предназначена для удаления устаревших и неактуальных фотографий в случаи, если пользователь сам не удалил объявление.

– «Смена пароля пользователю»: функция дает возможность администратору сменить пароль пользователю. Например, если пользователь не смог восстановить пароль с помощью стандартных методов, он может обратиться в сервисный центр.

Для возможности пользоваться программным средством, необходима учётная запись.

Учётная запись это хранимая в компьютерной системе совокупность данных о пользователе, необходимая для его опознавания (аутентификации) и предоставления доступа к его личным данным и настройкам. Учётная запись, как правило, содержит сведения, необходимые для опознания пользователя при подключении к системе, сведения для авторизации и учёта. Это идентификатор пользователя (login) и его пароль. Пароль, как правило, хранится в зашифрованном или хэшированном виде для обеспечения его безопасности. Для использования учётной записи (другими словами, для входа в систему под чьим-то именем) необходимо использовать логин и пароля. Для регистрации необходимо использовать дополнительное поле email [**?**].

Рассмотрим необходимые функции для работы с учетной записью. Данная диаграмма показывает, что пользователь может делать с учетной записью.

Диаграмма вариантов использования представлена ниже [5].

Рассмотрим каждую функцию подробнее:

– «Создание учетной записи»: данная функция дает возможность пользователю завести учетную запись, чтобы использовать возможности программного средства.

– «Редактирование учетной записи»: данная функция позволяет пользователю изменить или дополнить информацией свою учетную запись.

– «Удаление учетной записи»: данная функция позволяет пользовате-

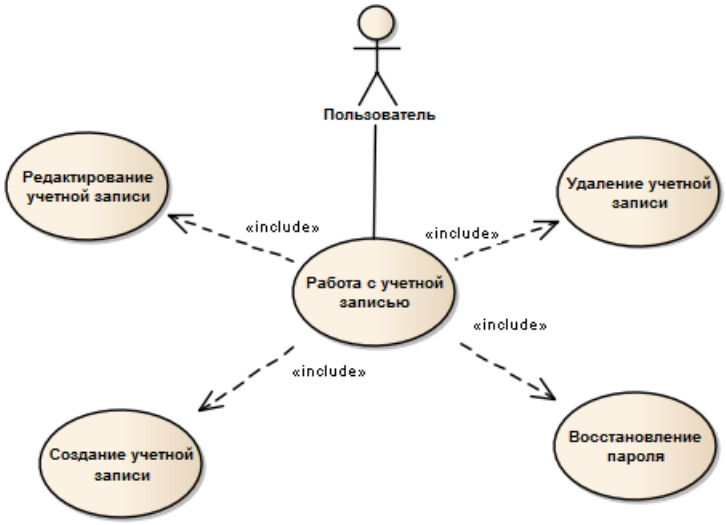

Рисунок 2.6 – Диаграмма вариантов использования «Работа с учетной записью»

лю удалить свою учетную запись.

– «Восстановление пароля»: данная функция дает возможность пользователю востановить пароль от учетной записи.

2.3 Разработка спецификации функциональных требований

На основе функциональной модели ПС и поставленных задач необходимо разработать спецификацию функциональных требований к программному средству для продажи квартир.

Спецификация функционального требования «Создание учетной записи пользователя»:

– Функции создания учетной записи должна быть доступна только клиентам;

– При нажатии пользователем на кнопку «sign up» на экране отображается окно, в котором пользователь должен указать логин, пароль, электронную почту;

– После нажатия на кнопку “create” в системе должен появиться новый пользователь. Должно произойти перенаправление на страницу входа в программное средство;

– Если логин или электронная почта уже используется у другого

пользователя, ПС должно предложить пользователю ввести другой логин и электронную почту, т.к такие данные уже используются;

Спецификация функционального требования «Восстановление учетной записи пользователя»:

– Функции должна быть доступна только клиентам;

– При нажатии пользователем на кнопку «forgot password» на экране отображается окно, в котором пользователь должен указать свою электронную почту для восстановления пароля;

– После нажатия на кнопку “restore” на почту должны прийти инструкции как восстановить пароль. Должно произойти перенаправление на страницу входа в программное средство;

Спецификация функционального требования «Добавление объявления»:

– Функции должна быть доступна только клиентам;

– При нажатии пользователем на кнопку «forgot password» на экране отображается окно, в котором пользователь должен указать свою электронную почту для восстановления пароля;

– После нажатия на кнопку “restore” на почту должны прийти инструкции как восстановить пароль. Должно произойти перенаправление на страницу входа в программное средство;

Спецификация функционального требования «редактирование объявления»:

– Функции должна быть доступна только клиентам;

– Функция редактирования мультимедиа должна быть доступна, после выбора файла и отмечена галочкой;

– Нажатие пользователем во время редактирования клавиши «Cancel» должно отменить редактирование;

– При нажатии пользователем во время редактирования на клавишу «Enter» редактируемое мультимедиа должно быть сохранено;

– После редактирования выбранного мультимедиа страница должна быть обновлена;

Спецификация функционального требования «удаления объявления»:

– Функции должна быть доступна всем;

– При нажатии пользователем на кнопку удаления должно быть показано окно с подтверждением удаления;

– После удаления выбранного мультимедиа страница должна быть обновлена;

Спецификация функционального требования «добавление фотогра-

фий»:

– Функции должна быть доступна только клиентам;

– на странице объявления должна присутствовать кнопка «Add photo». При нажатии на кнопку, появляется диалоговое окно;

– в диалоговом окне должны быть кнопки «choose photo», «add», «cancel»;

– при нажатии кнопки «choose photo» пользователь должен выбрать фото;

– после нажатия кнопки «Add» выбранная фотография появляется на странице объявления;

– при нажатии кнопки «cancel» происходит отмена данной операции;

Спецификация функционального требования «поиск объявления»:

– Функции должна быть доступна только клиентам;

– На главной странице должна присутствовать панель для поиска объявлений;

– На панели должны быть следующие критерии для поиска: ценовой диапазон, город, район, количество комнат, типа продажи, тип категории, кнопка «Search»;

– При нажатии на кнопку «Search» в центре экрана появляются найденные объявления;

Спецификация функционального требования «Добавления отзывы»:

– Функции должна быть доступна только клиентам;

– на странице объявления должна присутствовать кнопка «Add comment». При нажатии на кнопку, появляется диалоговое окно;

– после нажатия кнопки «Add» отзыв появляется на странице объявления;

3 АРХИТЕКТУРА И МОДУЛИ СИСТЕМЫ

Разработанное программное средство представляет из себя клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя. Исходя из выше сказанного, можно сказать, что веб-приложения являются кроссплатформенными сервисами. Серверная часть приложения реализована с помощью языка Java. Клиентская часть реализована с помощью языка Javascript [6].

Серверная часть состоит из следующих модулей:

– модуль доступа к данным (DAO);
– модуль сервисов(Бизнес логика);
– модуль предоставления данных(Форматирование данных);

Клиентская часть состоит из следующих компонентов:

– модуль действий(actions);
– модуль отображений(components);

Рассмотрим каждый компонент по отдельности.

3.1 Модуль доступа к данным

Данный модуль дает возможность доступа к базе данных. Для реализации данной функциональности использовался шаблон проектирования DAO.

Шаблон проектирования DAO – это объект, который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. Определённые возможности предоставляются независимо от того, какой механизм хранения используется и без необходимости специальным образом соответствовать этому механизму хранения. Этот шаблон проектирования применим ко множеству языков программирования, большинству программного обеспечения, нуждающемуся в хранении информации и к большей части баз данных. На данном уровне нет механизма для управления транзакциями, т.к. транзакции как правило принимают участие в бизнес операциях [7].

В качестве выбора данного шаблона, послужили следующие факторы:

– DAO инкапсулирует доступ к источнику данных;

– DAO является реализацией слоя объектно-реляционного отображения;

– DAO более ориентирован на источник данных;

Основная цель данного модуля — упростить процесс взаимодействия с БД. Пример реализации шаблона DAO приведены в листинге 3.1:

Листинг 3.1 – Определение для доступа к данным

```
public interface IDAO<Key extends Serializable, E> {

E create(E entity);

E read(Key key);

void update(E entity);

void delete(E entity);

}

public abstract class AbstractDAOImp<Key extends Serializable, E> implements
    IDAO<Key, E> {

@Autowired
private SessionFactory sessionFactory;

private Class<E> clazz;

protected AbstractDAOImp(Class<E> clazz) {
      this.clazz = clazz;
}

protected Session getSession() {
      return sessionFactory.getCurrentSession();
}

@Override
public E create(E entity) {
      Session session = getSession();
      session.persist(entity);
      session.flush();
      return entity;
}

@Override
public E read(Key key) {
      return getSession().find(clazz, key);
}

@Override
public void update(E entity) {
      getSession().update(entity);
```

```
}

@Override
public void delete(E entity) {
	getSession().delete(entity);
}

}

@Repository("UserRepository")
public class UserDAOImp extends AbstractDAOImp<Integer, User> {

protected UserDAOImp() {
	super(User.class);
}

public List<User> readAllUser() {
	return getSession().createQuery("from users", User.class).list();
}

}
```

3.2 Модуль сервисов

На этом уровне реализована бизнес логика приложения, управление транзакциями, сервис для загрузки изображений и другие.

Для начала рассмотрим сервис, который взаимодействует с модулем DAO. Сервис для взаимодейсвтия с DAO приведен в листинге 3.2:

Листинг 3.2 – Использование DAO в сервисах

```
@Service("UserService")
@Transactional
public class UserService {

@Autowired
private UserDAOImp daoImp;

public User addUser(User user) {
	return daoImp.create(user);
}

public void updateUser(User user) {
	daoImp.update(user);
}

public void removeUser(User user) {
	daoImp.delete(user);
}
```

```
public List<User> findAllUser() {
      return daoImp.readAllUser();
}

public User findUserById(int id) {
      return daoImp.read(id);
}

}

@Service("CommentService")
@Transactional
public class CommentService {

@Autowired
private CommentDAOImp daoImp;

public Post addPost(Comment comment){
return daoImp.create(comment);
}

public Post findCommentById(int id){
return daoImp.read(id);
}

public void updateComment(Post post){
daoImp.update(post);
}

public void removeComment(Post post){
daoImp.delete(post);
}

}
```

Сначала может показаться, что модуль сервис повторяет методы класса DAO. Но эти методы не просто повторяют, они оборачивают методы DAO с использованием транзакций. Транзакция это группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще, и тогда она не должна произвести никакого эффекта. Можно заметить, что модуль сервисов зависит от модуля DAO.

Рассмотрим сервис для загрузки и сохранения изображений. При разработке приложений была обнаружена проблема с быстрой загрузкой графического контента. Если картинки довольно большие, то необходимо обеспечить эффективную работу с памятью для предотвращения злополучной

ошибки OutOfMemoryError. Так же во время загрузки изображения было необходимо показывать маленькое изображение. Именно данная проблематика и подталкивает обратиться к библиотеке с открытым исходным кодом – Universal Image Loader, целью которой является универсализация решения вышеописанной задачи в виде гибкого и конфигурируемого инструмента.

На данный момент библиотеку можно использовать в тех случаях, когда нужно загрузить и отобразить (и можно еще закэшировать) картинку из интернета или из файловой системы. Классические примеры применения ImageLoader'а – это списки, таблицы, галереи, где необходимо отображать изображения из сети.

Среди возможностей ImageLoader'а можно выделить следующие достоинства:

– асинхронная загрузка и отображение изображений из интернета или с SD-карты;

– возможность кэширования загруженных картинок в памяти и/или на файловой системе устройства;

– возможность отслеживания процесса загрузки посредством «слушателей»;

– эффективная работа с памятью при кэшировании картинок в памяти;

– широкие возможности настройки ImageLoader'а.

К глобальным настройкам ImageLoader'а можно отнести:

– максимальный размер кэшируемых в памяти картинок;

– тайм-аут для установки соединения и загрузки картинки;

– максимальное количество потоков для загрузки изображений, работающих одновременно;

– приоритет потоков по загрузке и отображению картинок;

– программная реализация дискового кэша (можно выбрать одну из готовых реализаций или создать свою собственную);

– программная реализация кэша в памяти (можно выбрать одну из готовых реализаций или создать свою собственную);

– опции загрузки изображения по умолчанию.

Опции загрузки изображения (применяются к каждому отдельному вызову ImageLoader.displayImage(...)) предоставляют возможность указать:

– отображать ли картинку – заглушку в ImageView, пока реальная картинка грузится (если да, то нужно указать эту «заглушку»);

– отображать ли какую-либо картинку в ImageView, если URL картинки был передан пустым (если да, то нужно указать эту картинку);

– кэшировать ли загруженную картинку в памяти;

– кэшировать ли загруженную картинку на файловой системе;

– тип декодирования изображения (максимально быстрый или максимально экономный для памяти).

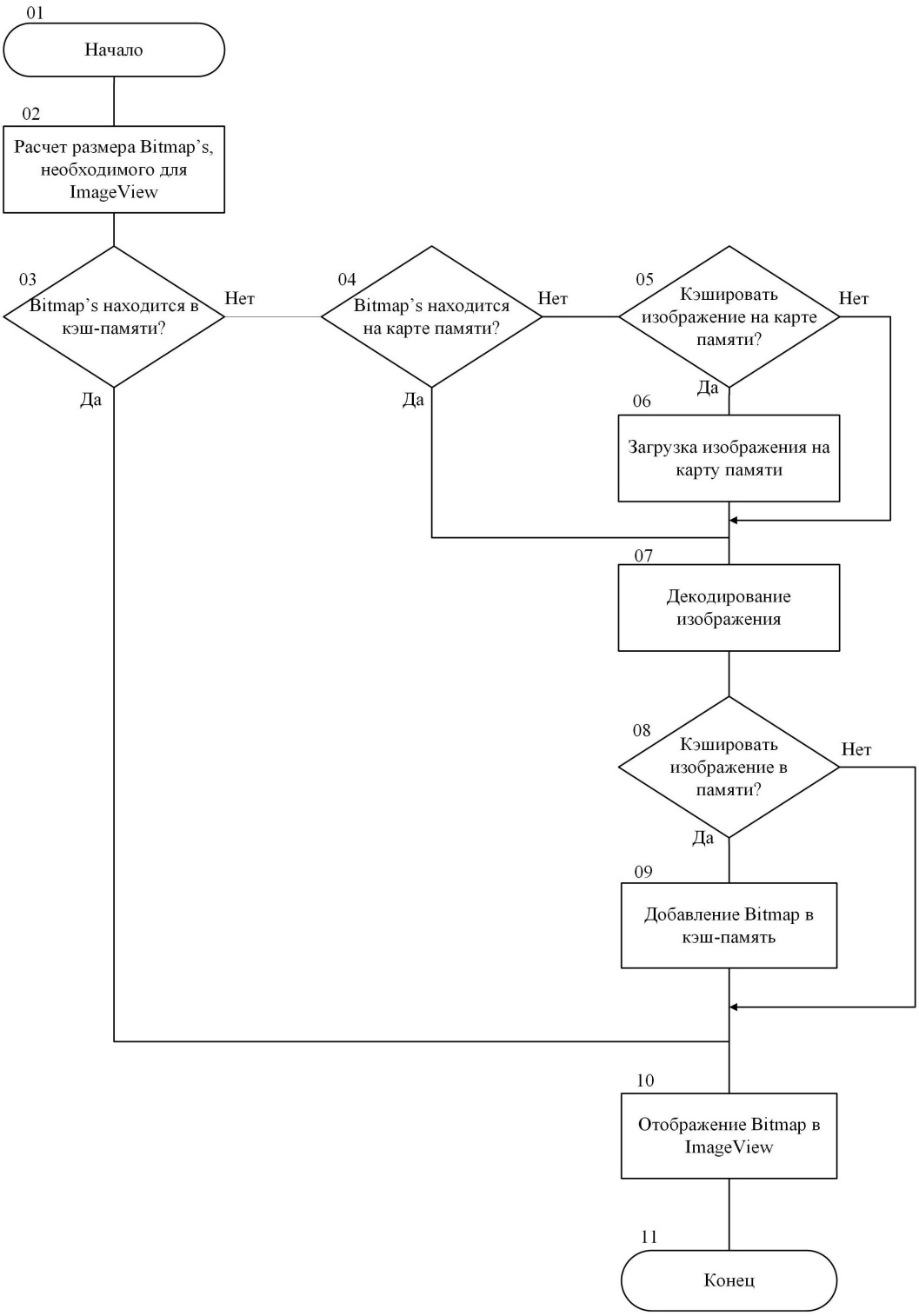

Рисунок 3.1 – Схема алгоритма ImageLoader

Как уже упоминалось, программные реализации дискового кэша и кэша в памяти можно подключить свои. Но, скорее всего, будет достаточ-но готовых решений, которые в большинстве своем представляют собой кэши, ограниченные по какому-либо параметру (размер, количество файлов) и имеющие свою логику самоочищения при превышении лимита (FIFO, самый старый объект, самый большой объект, наиболее редко используемый) [8].

Каждая задача на загрузку и отображение картинки выполняется в отдельном потоке, кроме случаев, если картинка находится в кэше в памяти. Тогда она просто сразу отображается. Существует отдельная очередь потоков, куда попадают задачи, если нужная картинка закэширована на файловой системе.

Если же нужной картинки нет в кэше, то задача-поток попадает в пул пото-ков. Таким образом, быстрому отображению закэшированных картинок ничего не препятствует.

Для управления загрузкой картинок, используется класс ImageLoaderService. Это singletone, поэтому чтобы получить единственный экземпляр класса нужно вызвать метод getInstance(). Перед использованием ImageLoader’а по назна-чению (для отображения картинок), необходимо проинициализировать его конфигурацией.

Самый простой вариант использования ImageLoader’а (с конфигурацией по умолчанию) представлен в листинге 3.3:

Листинг 3.3 – Определение ImageLoaderService

```
@PropertySource(value = { "classpath:imageLoader.properties" })
@Service("ImageLoaderService")
public class ImageLoaderService {

@Autowired
private Environment environment;
private static ImageLoader imageLoader;

public ImageLoader getInstance() {
if(imageLoader == null){
ImageLoader imageLoader = ImageLoader.init(getProperties());
}
return imageLoader;
}


private Properties getProperties() {
Properties properties = environment.loadConfiguration();
return properties;
}
```

```
}
```

3.3 Модуль предоставления данных

Следующий за модулем сервисов — модуль предоставляения данных. Для обработки клиентских запросов используются *spring controllers*.

Spring controllers — это классы, которые осуществляют обработку запроса от клиента (браузера). Здесь происходит валидация данных, передача данных в модуль сервисы. Завершающим этапом будет генерация и форматирование ответа в формат *JSON* [9].

JSON — это текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается. За счёт своей лаконичности по сравнению с XML, формат JSON был более подходящим для сериализации сложных структур. Если говорить о веб-приложениях, формат уместен в задачах обмена данными между браузером и сервером [10].

Бизнес логика приложения в этом модуле не присутствует, она делегируется модулю сервисов.

В качестве примера рассмотрим controller, который отвечает за регистрацию и входа пользователя в систему. Класс LoginController представлен в листинге 3.4:

Листинг 3.4 – Определение LoginController

```
@Controller
@RequestMapping(value="api/user"
public class LoginController{

@Autowired
private LoginService loginService;

@Autowired
private UserService userService;

@RequestMapping(value="/signin",method=RequestMethod.GET)
public String signIn(HttpServletRequest request, HttpServletResponse response)
    {
String username = request.getParameter("login");
String password = request.getParameter("password");

boolean isValidUser = loginService.isValidUser(username, password);
User user = null;
if(isValidUser){
user = userService.findUserByLogin(username, password);
}
```

```
return ObjectMapper.toJSON(user);
}


@RequestMapping(value="/signup",method=RequestMethod.POST)
public HttpServletResponse signUp(HttpServletRequest request,
   HttpServletResponse response) {
String username = request.getParameter("login");
String password = request.getParameter("password");

boolean isValidUser = loginService.isValidUser(username, password);
if(isValidUser){
userService.createUser(username, password)
response.setAttribute("signUp", "Success operation. You will be redirected to
   sign in page!");
}else{
response.setAttribute("signUp", "Invalid operation. Please, specify correct
   usernamge, email!");
}
return response;
}

}
```

Метод signIn нужен для входа в программное средство. Данный метод включает в себя запрос от клиента и будущий ответ. Для начала входа систему, необходимо получить имя пользователя и его пароль. Если такой пользователь существует, то система загружает пользователя и формирует ответ на клиент. Строка *ObjectMapper.toJSON* преобразует джава объект в формат JSON. Этот ответ будет обработан клиентом.

Метод signUp нужен для регистрации пользователя в программном средстве. Данный метод включает в себя запрос от клиента и будущий ответ. Для регистрации в программном средстве, необходимо получить имя пользователя и пароль, который прислал клиент в запросе. Далее проверяется, если такой пользователь не существует в системе, то пользователь будет создан, иначе происходит уведомление клиента о некорректности передаваемых данных.

3.4 Модуль действий

Рассмотрим модуль действий. Модуль находиться на клиентской части и отвечает за взаимодействие со слоем предоставления данных. Данный модуль общается посредством *http* и *ajax*.

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть

документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам). Http метод представляет собой последовательность из любых символов, кроме управляющих и разделителей, и определяет операцию, которую нужно осуществить с указанным ресурсом.

AJAX — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее. AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий. AJAX базируется на двух основных принципах:

– использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, например с использованием XMLHttpRequest (основной объект);

– использование DHTML для динамического изменения содержания страницы;

В качестве формата для обмена данных между клиентом и сервером используется JSON.

Первый пример демонструет вход в систему. LoginAction представлен в листинге 3.5:

Листинг 3.5 – Определение LoginAction

```
var LoginAction = {

signIn: function(nickname, password, callback){

      $.ajaxPost("/api/user/signIn", {"nickname": nickname, "password":
         password}, function (data) {
            callback(data);
      });

}

}
```

Рассмотри ситуацию, когда пользователь хочет оставить комментарий на недвижимость. Пользователь может добавить комментарий, удалить комментарий, редактировать комментарий, читать предыдущие комментарии. В рамках этих условий, был разработан модуль CommentsActions, который выполняет все вышеперечисленные действия. CommentsActions представлен в листинге 3.6:

Листинг 3.6 – Определение CommentsActions

```
var CommentsActions = {

addComment: function(text, callback){
      $.ajaxPost("/api/comments/addComment/", function (data) {
            if (callback) {
                  callback(data);
            }
      });
},

removeComment: function(id, callback){
      $.ajaxPost("/api/comments/removeComment/", {"id":id}, function (data) {
            if (callback) {
                  callback(data);
            }
      });
},

editComment: function(id, text, callback){
      $.ajaxPost("/api/comments/editComment/", {"id":id, "text":text},
         function (data) {
            if (callback) {
                  callback(data);
            }
      });
},

loadAllComments:function(id, callback){
      $.ajaxPost("/api/comments/loadAll/", {"id":id}, function (data) {
            if (callback) {
            callback(data);
            }
      });
}

}
```

Подобным обазом работают и другие функции. Сначала указываться url для запроса на сервер. Далее перечесляются обязательные параметры для сервиса. Следующим параметрам идет функцию в которую придет ответа от севера. Если ответ удволетворяет условию, будет вызвана *callback функция*.

Callback функция или функция обратного вызова в программировании — передача исполняемого кода в качестве одного из параметров другого кода. Обратный вызов позволяет в функции исполнять код, который задаётся в аргументах при её вызове.

В момент, когда ответит ajax запрос, будет вызва функция обратного вызова. Обработка ответа будет представлена в месте, где используется

модуль действия.

3.5 Модуль отображения

Рассмотрим модуль отображений. Модуль находиться на клиентской части и отвечает за взаимодействие со слоем действий. Данный модуль непосредственно отображает информацию для конечного пользователя. Для реализации данного модуля, была выбрана библиотека *React*.

React — уровень представления данных. React дает язык шаблонов и некоторые callback-функции для отрисовки HTML. Весь результат работы React — это готовый HTML. Компоненты react занимаются тем, что хранят свое внутреннее состояние в памяти (например: какая закладка выбрана), но в итоге отображается html.

Библиотека была выбрана из-за следующих плюсов:

– Всегда можно сказать, как компонент будет отрисован, глядя на исходный код;

– Связывание JavaScript и HTML в JSX делает компоненты простыми для понимания;

– Можно генерировать html на сервере;

Рассмотрим форму для регистрации, реализованую с помощью библиотеки React. Код представлен в листинге 3.7:

Листинг 3.7 – Форма регистрации LoginAction

```
import React, { PropTypes } from 'react';
import { Link } from 'react-router';
import { Card, CardText } from 'material-ui/Card';
import RaisedButton from 'material-ui/RaisedButton';
import TextField from 'material-ui/TextField';


const LoginForm = ({
onSubmit,
onChange,
errors,
user
}) => (
<Card className="container">
<form action="/" onSubmit={onSubmit}>
<h2 className="card-heading">Login</h2>

{errors.summary && <p className="error-message">{errors.summary}</p>}

<div className="field-line">
<TextField
```

```
floatingLabelText="Email"
name="email"
errorText={errors.email}
onChange={onChange}
value={user.email}
/>
</div>

<div className="field-line">
<TextField
floatingLabelText="Password"
type="password"
name="password"
onChange={onChange}
errorText={errors.password}
value={user.password}
/>
</div>

<div className="button-line">
<RaisedButton type="submit" label="Log in" primary />
</div>

<CardText>Don't have an account? <Link to={'/signup'}>Create one</Link>.</
   CardText>
</form>
</Card>
);

LoginForm.propTypes = {
onSubmit: PropTypes.func.isRequired,
onChange: PropTypes.func.isRequired,
errors: PropTypes.object.isRequired,
user: PropTypes.object.isRequired
};

export default LoginForm;

import React, { PropTypes } from 'react';
import { Link, IndexLink } from 'react-router';


const Base = ({ children }) => (
<div>
<div className="top-bar">
<div className="top-bar-left">
<IndexLink to="/">React App</IndexLink>
</div>

<div className="top-bar-right">
<Link to="/login">Log in</Link>
<Link to="/signup">Sign up</Link>
</div>
```

```
</div>

{children}

</div>
);

Base.propTypes = {
children: PropTypes.object.isRequired
};

export default Base;

import Base from './components/Base.jsx';
import HomePage from './components/HomePage.jsx';
import LoginPage from './containers/LoginPage.jsx';
import SignUpPage from './containers/SignUpPage.jsx';


const routes = {
// base component (wrapper for the whole application).
component: Base,
childRoutes: [

{
path: '/',
component: HomePage
},

{
path: '/login',
component: LoginPage
},

{
path: '/signup',
component: SignUpPage
}

]
};

export default routes;
```

В React имеет компоненты, состояние, рендер. Компонент включает в себя html и функции для управления элементами на странице. Рендер предоставляет конечный вариант HTML браузеру (то, что видит пользователь). В некоторых компонентах нужно сохранять внутреннее состояние, которое используется во время визуализации. Например, флажок должен помнить, что он был выбран. Для этого используется функция состояние.

В завершении, рассмотрим диаграмму компонентов, отражающая работу всего приложения. Диаграмма представлена на рисунке 3.2

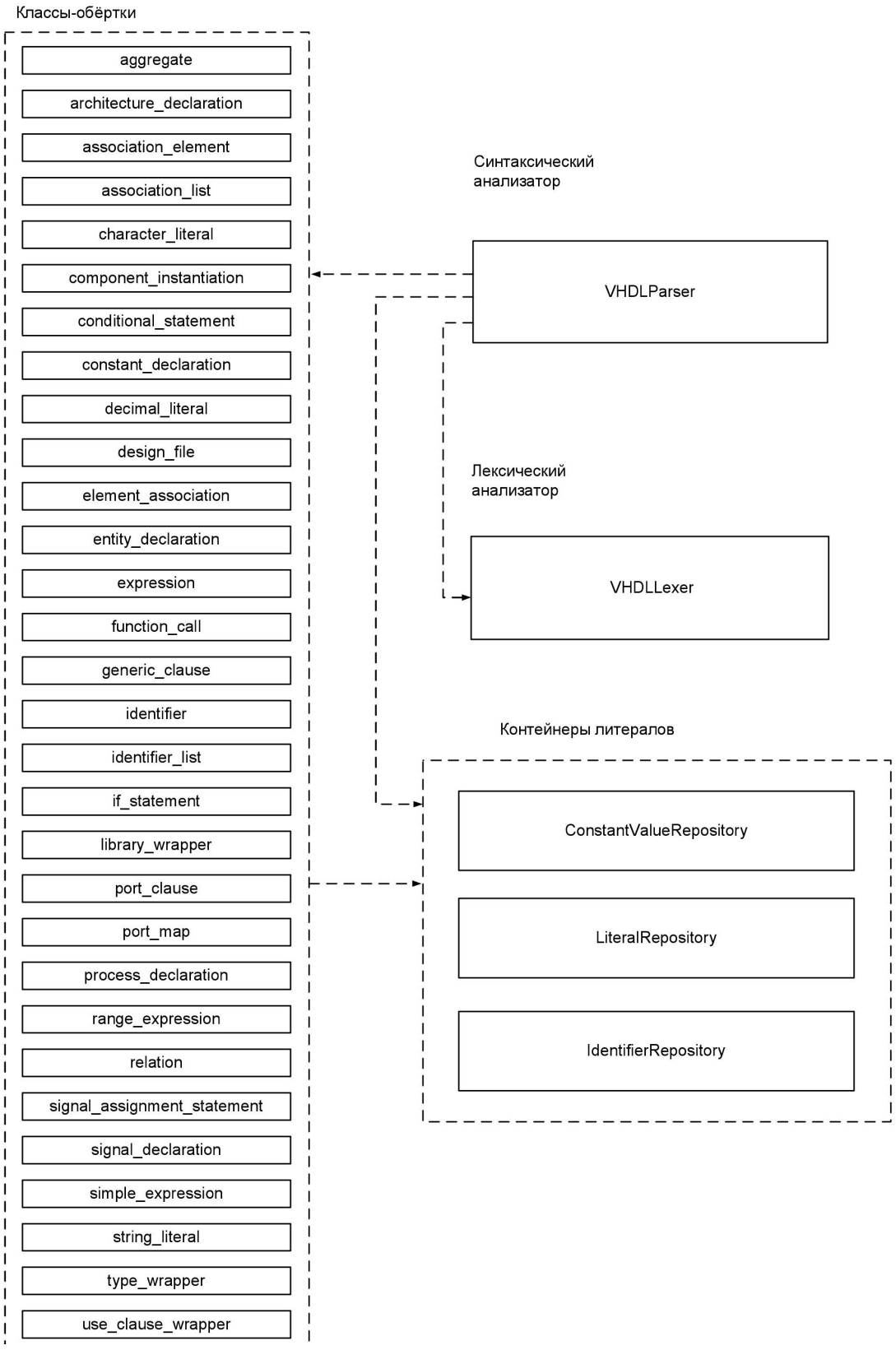

Рисунок 3.2 – Диаграмма компонентов программного средства

4 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Для оценки правильности работы программного средства было проведено тестирование. Тест-кейсы для функционального требования «Взаимодействие с пользователем» представлены в таблице 4.1

Таблица 4.1 – Тестирование взаимодействия с пользователем

Название тест-кейса и его описание	Ожидаемый результат	Полученный результат
1	2	3
Создание учетной записи пользователя а) Ввести имя, почту, пароль. б) нажать кнопку sign up	а) Отображается сообщение об успешном создании пользователя б) Происходит перенаправление на страницу входа в приложение	Пройден
Вход в приложение используя ранее созданную учетную запись пользователя а) Ввести почту и пароль б) Нажать кнопку sign in	а) Происходит перенаправление на страницу с объявлениями б) Отображается список существующих объявлений	Пройден
Востановление учетной записи пользователя а) Нажать на кнопку forgot password б) Ввести почту при создании пользователя в) Нажать кнопку restore password	а) На указанную почту приходит письмо с новом поролем б) Пользователь успешно входит в систему используя новый пароль в) Отображается список существующих объявлений	Пройден

Продолжение таблицы 4.1

1	2	3
Добавления объявления а) Нажать кнопку create note б) Заполнить обязательные поля в) Нажать кнопку add	а) На вкладке My note отображается новое объявление б) Новое объявление должно быть доступно для поиска	Пройден
Изменить существующее объявление а) Выбрать объявление б) Нажать кнопку Edit в) Изменить несколько полей г) Нажать кнопку Save	а) Измененные данные сохранились в объявлении	Пройден
Удалить существующее объявление а) Выбрать нужное объявление б) Нажать кнопку delete	а) Объявление пропадает со вкладки My notes б) Объявление становиться недоступным для поиска	Пройден
Добавить фотографию в объявление а) Выбрать объявление б) Нажать кнопку Add Picture в) Выбрать изображение на компьютере г) Нажать кнопку Upload	а) Новая фотография добавилась к выброному объявлению	Пройден

Продолжение таблицы 4.1

1	2	3
Поиск объявлений а) Открыть страницу поиска б) Ввести необходимые критерии в) Нажать кнопку Search Notes	а) Найденные объявления отображаются в таблице, в центре экрана б) Найденные обявлнения можно просматривать в новой вкладе	Пройден
Оставить отзыв на объявление а) Выбрать необходимое объявление б) Ввести отзыв в поле Comment в) Нажать кнопку Add Comment	а) Написанный отзыв отображается снизу, под основной информацией объявления б) Отображается значок для удаления коментария	Пройден
Редактирование профиля а) Перейти в раздел Settings б) Надать кнопку Edit в) Изменить необходимые данные г) Нажать кнопку Save	а) Страница обновляется б) Новая информация отображается в профиле пользователя	Пройден

Таким образом, результат тестирования подтверждает, что программное средство web прилложение для продажи квартир функционирует в полном соответствии со спецификацией требований.

5 МЕТОДИКА ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО СРЕДСТВА

5.1 Руководство по установке и настройке

На компьютере-сервере для корректной установки серверной части должно быть установлено следующее программное обеспечение:

– операционная система семейства Windows или Linux;

– Java версии не ниже 1.7;

– MySQL сервер версии начиная с 5.1;

– веб-сервер apache tomcat.

Порядок установки серверной части:

– скачиваем последнию версию Java;

– распаковываем архив на диске;

– Необходимо прописать путь до файлов JDK в путях операционной системы. Это позволит запускать основные файлы из командной строки. В переменых среды необходимо создать JAVAHOME и прописать путь до папки с JDK. После этого необходимо добавить переменую JAVAHOME в переменую PATH;

– скачиваем последнию версию apache tomcat;

– распаковываем архив на диске;

– скопировать файл SocialNetWork.war в папку tomcat/webapps;

– скачать и установить последнию версию Mysql сервер;

– необходимо сделать импорт базы данных. Открываем консоль и прописываем следующую команду "mysql -u username -p databaseTame < socialNetwork.sql";

– следующим пунктом необходимо запусть apache tomcat. Переходим в tomcat/bin. Необходимо выполнить в консоль catalina.bat run;

Порядок установки клиетской части:

– скачиваем последнию версию Node js;

– распаковываем архив на диске;

– переходим в tomcat/webapps/SocialNetWork;

– слудующим пунктов необходимо выполнить команду "npm run server";

После установления всех настроек в браузере должна отобразиться страница, означающая, что серверная часть установлена корректно.

5.2 Руководство по использованию

Внешний вид программного средства сразу после запуска представлен на рисунке 5.1. Как видно из представленного рисунка, всю область, которую занимает программное средство можно условно поделить на три следующие части:

– главная страница;

– регистрация пользователь;

– вход в приложение;

Главная страница предоставляет краткое описание основных возможностей программного средства для пользователей. Страница регистрации позволяет создать новую учетную запись, чтобы использовать программное средство для работы. Страница входа в приложения служит точкой начала использования программного срдества.

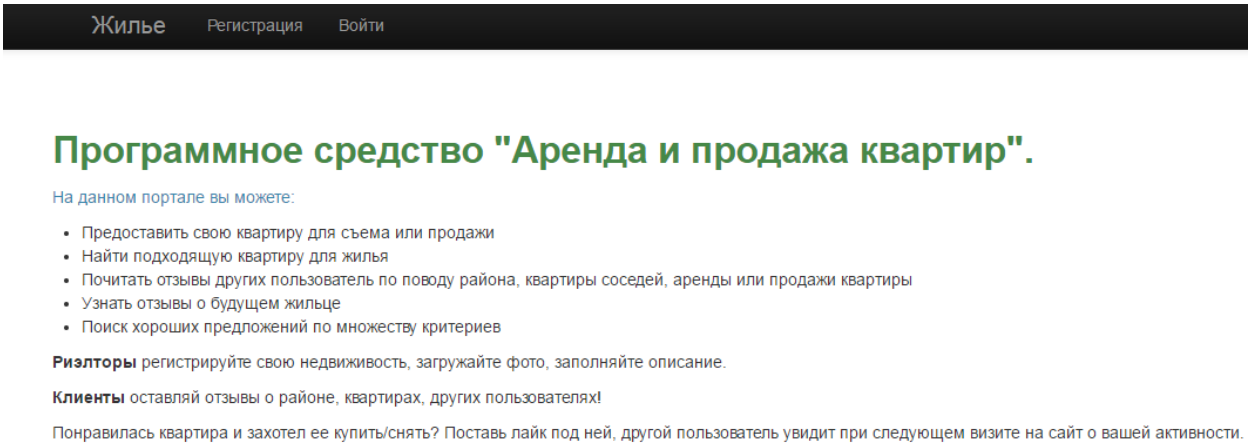

Рисунок 5.1 – Внешний вид программного средства

На рисунке 5.2 представлена страница, для создания учетной записи пользователя. Для создания новой учетной записи необходимо указать следующие обязательные поля:

– Логин. Данная информация будет использовтся в качестве входа в программное средство;

– Пароль. Данная информация будет использовтся в качестве входа в программное средство;

– Электронная почта. Данная информация будет использовтся для восстановления пароля;

– Фамилия. Данная информация будет отображена для других пользователей программного средства;

– Имя. Данная информация будет отображена для других пользователей программного средства;

– Телефон. Данная информация будет использовтся в качестве взаимодействия с другими пользователями;

– Город. Данная информация нужна для поиска;

– Роль пользователя. Исходя из этой информации, будущему пользователю будут доступны необходимые функции приложения;

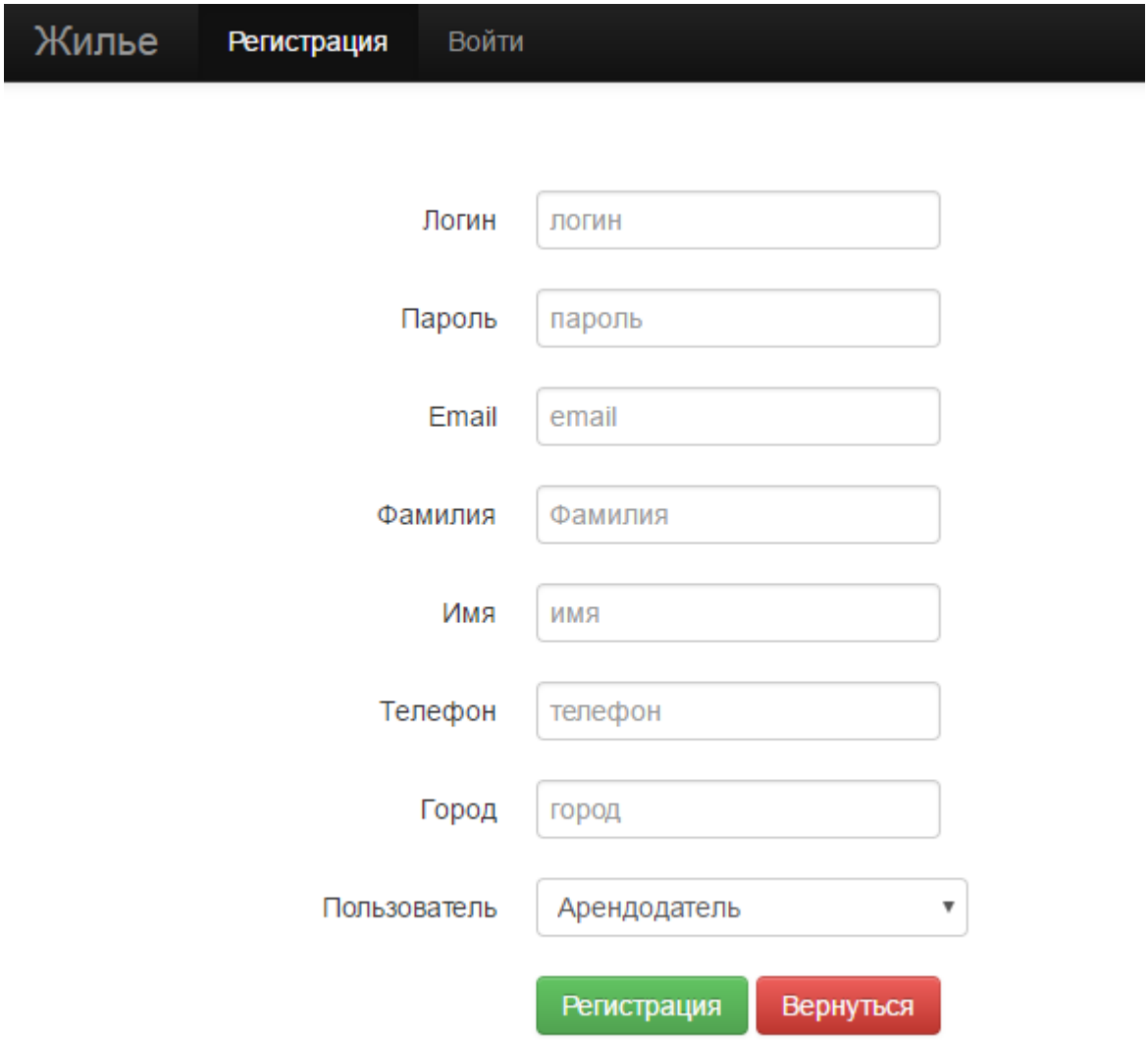

Рисунок 5.2 – Форма для создания нового пользователя

На рисунке 5.6 представлена страница, для входя в программное средство. Для входа необходимо указать обязательные поля:

– Логин;

– Пароль;

Помимо этого на данной форме есть функция для восстановления пароля, в случае, если пользователь забыл пароль от своей учетной записи. Чтобы восстановить пароль, необходимо указать электронную почту, которая была указана при регистрации.

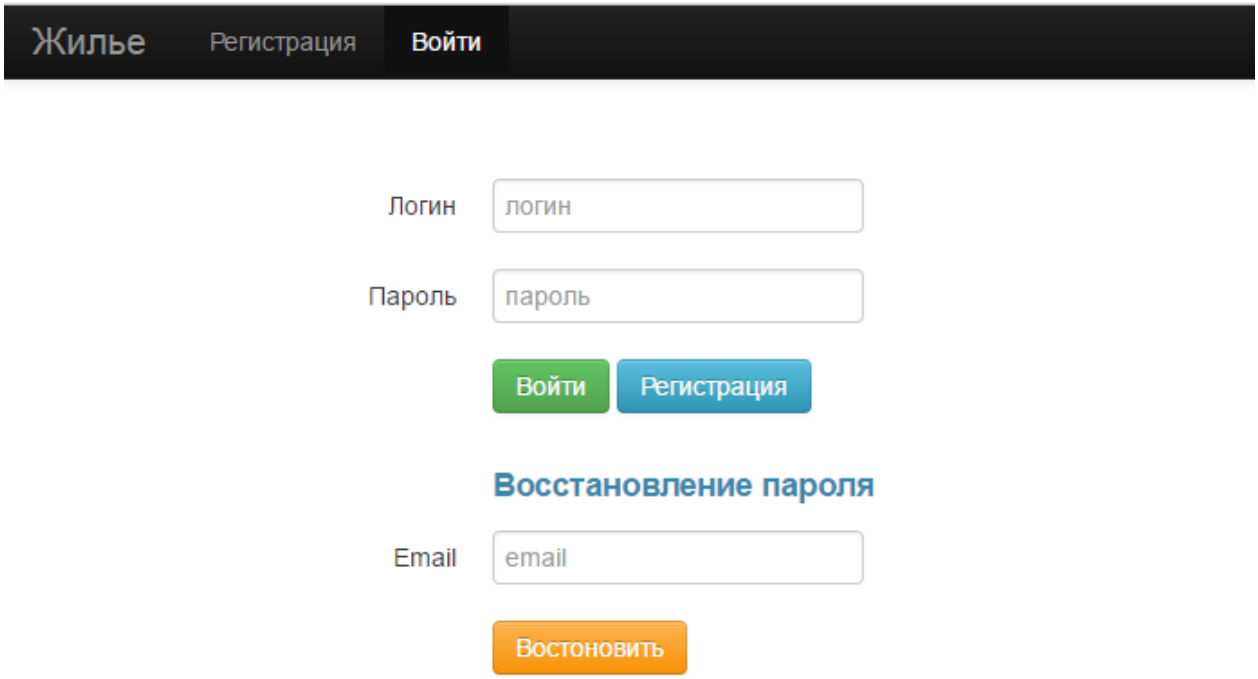

Рисунок 5.3 – Форма для восстановления пароля и входа в программное средство

После входа в приложения, пользователь увидит главный экран своей страницы. На нем отображены информация при регистрации. Помимо этого, можно посмотреть отзывы о себе, перейти на страницу поиска, изменить информацию о себе.

Для изменения информации, необходимо перейти в раздел настройки. Чтобы это сделать, нужно нажать на кнопку настройку. Для каждого из пользователь, страница выглядит с отличиями. Рассмотрим случай, когда пользователь продает свою квартиру. В разделе настройки, необходимо заполнить цену продажи/аренды. Добавить краткое описание. Помимо этого можно загрузить изображения будущей квартиры.

Чтобы загрузить изображения для своей недвижимости, необходимо перейти в раздел настройки. Далее нажимаем кнопку загрузить фотографии. В появившемся окошке выбираем фотографии, которые находятся на жестком диске, и нажимаем загрузить. После того, как изображения будут загружены, пользователь смотрит их посмотреть в своем профиле.

Рассмотрим пользователя, который хочет купить недвижимость. Для поиска необходимой недвижимости, необходимо перейти в раздел поиска. На главной форме нажимаем кнопку поиск предложений. После этого будет отображена страница для поиска предложений. На данном этапе можно использовать множество критериев, которые необходимы конечному пользователю.

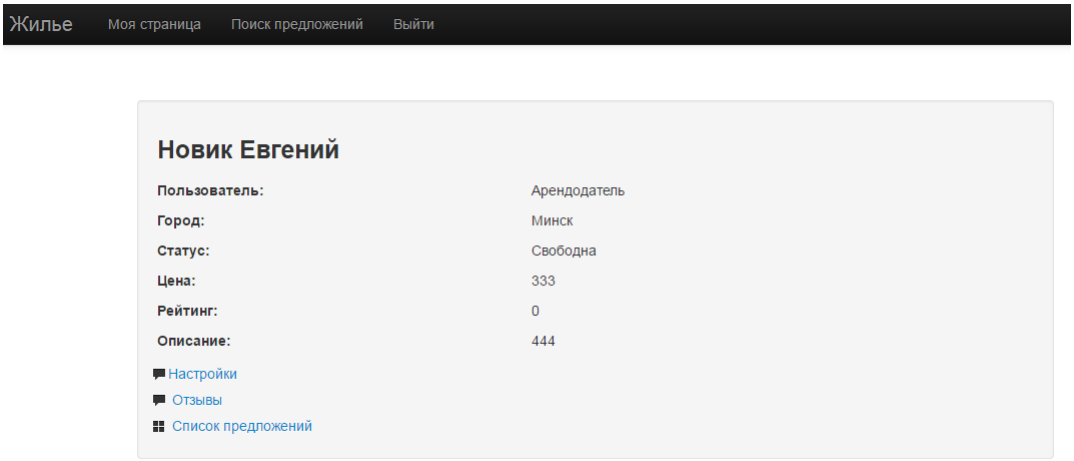

Рисунок 5.4 – Страница после входа в приложение

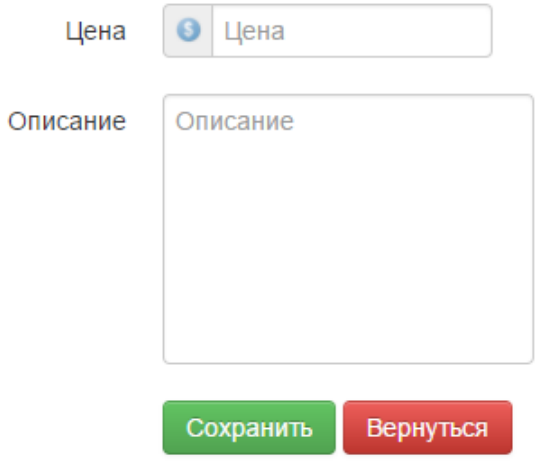

Рисунок 5.5 – Страница редактирования квартиры

После заполнений необходимых полей поиска, нажимаем на кнопку найти. После поиска, на странице появятся перечень квартир, которые можно купить/арендовать. Если пользователю понравилась квартира, ему достаточно нажать на кнопку "Нравится". Хозяину квартиры придет уве-

домление, что квартирой интересуется другой пользователь. У пользователей будет видна информация, для связи друг с другом, после двухэтапная аутентификации.

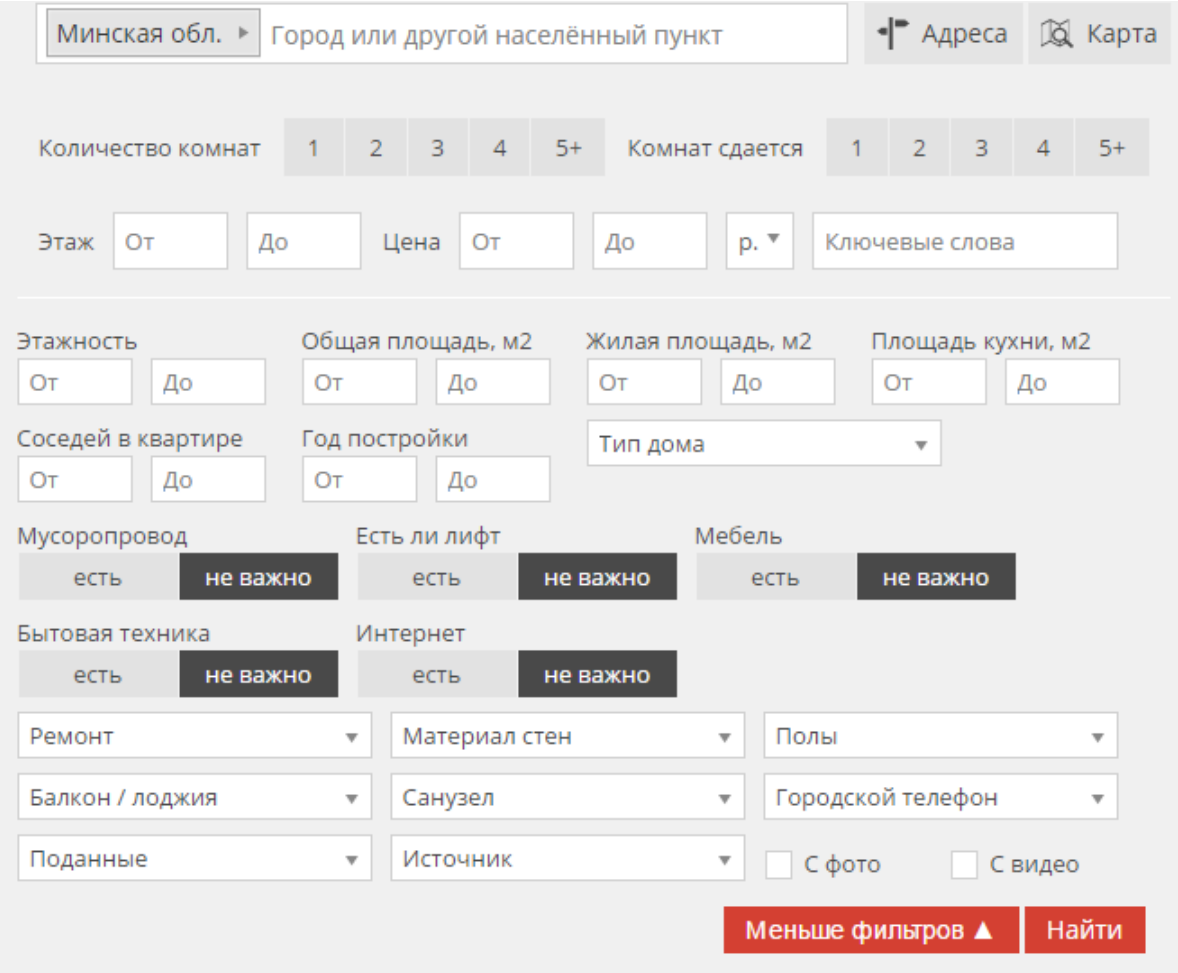

Рисунок 5.6 – Страница поиска квартир

Чтобы просмотреть отзывы об недвижимости, которую сдают в аренду, необходимо перейти на квартиру и нажать на кнопку отзывы.

На данной странице отображены отзывы других пользователей. Здесь можно узнать мнение другого человека. Отзыв состоит из текста, времени и даты, когда он был создан.

6 ОПРЕДЕЛЕНИЕ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

6.1 Расчёт экономической эффективности использования разработки веб-приложения для продажи квартир

a

a

a

a

a

a

a

a

a

a

a

a

a

a

a

a

a

a

a

ЗАКЛЮЧЕНИЕ

Недвижимость является основой материального производства и имущественного благополучия граждан. Вложение капиталов и сбережений в недвижимость – важнейший экономический процесс рыночной экономики.

В данном дипломном проекте был рассмотрен вопрос продажи и аренды недвижимости, а также была иследована бизнес область в данной тематике. В рамках дипломного проекта было разработано программное средство веб-приложение для продажи квартир.

В разработанном проекте был использован подход, который предоставляет решения с недвижимостью. Программное средство дает пользователю возможность быстрого поиска необходимой недвижимости. Быстрый способ продать или сдать в аренду свою недвижимость.

В целом получены хорошие результаты по оптимизации в данной бизнес области. Различные функции программного средства, позволяют сократить объем бумажной работы, а так же существенно сокращают время работы риэлторов.

В итоге получилось раскрыть тему дипломного проекта и создать в его рамках программное обеспечение. Но за рамками рассматриваемой темы осталось еще много других бизнес правил и функций, которые можно улучшить или реализовать.

В дальнейшем планируется развивать и довести существующее ПО до полноценного приложения, способное решать более широкий класс задач, возникающих в области недвижимости.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Хорстманн К., Корнелл Г. Java. Библиотека профессионала. Том 1. Основы / Корнелл Г. Хорстманн К. — Вильямс , 2016. — 866 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программного средства

```
package org.jazzteam.model.persistence;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class HibernateListener implements ServletContextListener {

public void contextInitialized(ServletContextEvent event) {
HibernateUtil.getSessionFactory();
}

public void contextDestroyed(ServletContextEvent event) {
HibernateUtil.getSessionFactory().close();
}
}

package org.jazzteam.model.serializer;

import org.jazzteam.model.table.Apartment;
import org.jazzteam.model.table.User;

public class ApartmentManager extends Transaction {

public ApartmentManager() {
}

public void addApartment(Apartment apartment, User user) throws Exception {
try {
begin();
apartment.setUser(user);
user.setApartment(apartment);
getSession().save(apartment);
getSession().save(user);
commit();
} catch (Exception e) {
rollback();
throw new Exception("Error");
}
}

public Apartment getApartment(User user) {
return user.getApartment();
}

public void updateApartment(Apartment apartment) throws Exception {
try {
begin();
getSession().update(apartment);
```

```
commit();
} catch (Exception e) {
rollback();
e.printStackTrace();
throw new Exception("Error");
}
}
}
package org.jazzteam.model.serializer;

import org.hibernate.Query;
import org.jazzteam.model.table.Comment;
import org.jazzteam.model.table.User;

import java.util.List;

public class CommentManager extends Transaction {

public void addComment(User landlord, User lessee, String text) throws
    Exception {
begin();
try {
Comment comment = new Comment();
comment.setUser(landlord);
comment.setCommentator(lessee);
comment.setComment(text);
landlord.getComments().add(comment);
getSession().save(comment);
getSession().save(landlord);
} catch (Exception e) {
rollback();
throw new Exception("Error");
} finally {
commit();
}
}

public List<Comment> getComments(String nickname) {
begin();
Query query = null;
try {
UserManager userManager = new UserManager();
User user = userManager.getUser(nickname);
query = getSession().createQuery("from Comment where iduser=:code");
query.setParameter("code", user.getIduser());
} catch (Exception e) {
e.printStackTrace();
} finally {
commit();
}
return (List<Comment>) query.list();
}
```

```
}
package org.jazzteam.model.serializer;

import org.jazzteam.model.table.Apartment;
import org.jazzteam.model.table.Image;

public class ImageManager extends Transaction {

public ImageManager() {

}

public void addImage(Image image, Apartment apartment) throws Exception {
begin();
try {
image.setApartment(apartment);
apartment.getImages().add(image);
getSession().save(image);
getSession().save(apartment);
} catch (Exception e) {
rollback();
throw new Exception("Error");
} finally {
commit();
}
}

public void updateImage(Image image) throws Exception {
begin();
try {
getSession().update(image);
} catch (Exception e) {
rollback();
throw new Exception("Error");
} finally {
commit();

}
}

public void removeImage(Image image) throws Exception {
begin();
try {
getSession().update(image);
} catch (Exception e) {
rollback();
throw new Exception("Error");
} finally {

commit();
}
}
```

```
}
package org.jazzteam.model.serializer;

import org.jazzteam.model.table.Apartment;
import org.jazzteam.model.table.Story;
import org.jazzteam.model.table.User;

public class StoryManager extends Transaction {

public void addStory(Apartment apartment, User user) throws Exception {
if (user.getControl().getType().equals("")) {
begin();
try {
Story story = new Story();
story.setUser(user);
story.setApartment(apartment);
apartment.getStories().add(story);
user.getStories().add(story);
getSession().save(story);
getSession().save(user);
getSession().save(apartment);
} catch (Exception e) {
rollback();
throw new Exception("Error");
} finally {
commit();
}
} else {
throw new Exception(user.getControl().getType() + "can not live in the
    apartment");
}

}
}
package org.jazzteam.model.serializer;

import org.hibernate.Session;
import org.jazzteam.model.persistence.HibernateUtil;

public class Transaction {
private static Session session;

public static Session getSession() {
if (session == null) {
session = HibernateUtil.getSessionFactory().openSession();
}
return session;
}

protected void begin() {
getSession().beginTransaction();
}
```

```
protected void commit() {
getSession().getTransaction().commit();
}

protected void rollback() {
getSession().getTransaction().rollback();
getSession().close();
Transaction.session=null;
}


}
package org.jazzteam.model.serializer;

import org.hibernate.Query;
import org.jazzteam.model.table.Control;
import org.jazzteam.model.table.User;

import java.util.List;

public class UserManager extends Transaction {

public UserManager() {
}

public User getUser(String nikname) throws Exception {
User user;
try {
begin();
Query query = getSession().createQuery("from User where nikname=:code");
query.setParameter("code", nikname);
user = (User) query.list().get(0);

} catch (Exception e) {
rollback();
throw new Exception("User "+ nikname+" not found");
}
return user;
}

public void addUser(User user, String access) throws Exception {
begin();
try {
Query query = getSession().createQuery("from Control where type=:code");
query.setParameter("code", access);
Control control = (Control) query.list().get(0);
user.setControl(control);
control.getUsers().add(user);
getSession().save(user);
commit();
} catch (Exception e) {
rollback();
```

```
throw new Exception("Come up with another nickname");
}
}

public void removeUser(String nikname) throws Exception {
try {
begin();
Query query = getSession().createQuery("from User where nikname=:code");
query.setParameter("code", nikname);
User user = (User) query.list().get(0);
getSession().delete(user);
commit();
} catch (Exception e) {
rollback();
throw new Exception("User not found");
}
}

public void update(User user) throws Exception {
try {
begin();
getSession().update(user);
commit();
} catch (Exception e) {
rollback();
throw new Exception("Error");
}
}

public List<User> getAllUser() throws Exception {
Query query=null;
try {
begin();
query = getSession().createQuery("from User");
} catch (Exception e) {
rollback();
throw new Exception("Error");
}
return (List<User>) query.list();
}

}

package org.jazzteam.model.table;

import javax.persistence.*;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import static javax.persistence.GenerationType.*;
```

```
@Entity
@Table(name = "user")
public class User implements Serializable {
private Integer idUser;
private String nikname;
private String password;
private String email;
private String surname;
private String name;
private String city;
private int mobile;
private Control control;
private Apartment apartment;
private List<Story> stories = new ArrayList<Story>();
private List<Comment> comments = new ArrayList<Comment>();
private List<LikeHouse> proposals = new ArrayList<LikeHouse>();
private List<LikeHouse> bids = new ArrayList<LikeHouse>();

public User() {

}

public User(String nikname, String password, String email, String surname,
    String name, String city, int mobile) {
this.nikname = nikname;
this.password = password;
this.email = email;
this.surname = surname;
this.name = name;
this.city = city;
this.mobile = mobile;
}

public User(String nikname, String password, String email, String surname,
    String name, String city, int mobile, Control control) {
this.nikname = nikname;
this.password = password;
this.email = email;
this.surname = surname;
this.name = name;
this.city = city;
this.mobile = mobile;
this.control = control;
}

@Id
@GeneratedValue(strategy = IDENTITY)
@Column(name = "iduser")
public Integer getIduser() {
return idUser;
}

public void setIduser(Integer idUser) {
```

```
this.idUser = idUser;
}

@Column(unique = true)
public String getNikname() {
return nikname;
}

public void setNikname(String nikname) {
this.nikname = nikname;
}

public String getPassword() {
return password;
}

public void setPassword(String password) {
this.password = password;
}

public String getEmail() {
return email;
}

public void setEmail(String email) {
this.email = email;
}

public String getSurname() {
return surname;
}

public void setSurname(String surname) {
this.surname = surname;
}

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public String getCity() {
return city;
}

public void setCity(String city) {
this.city = city;
}

public int getMobile() {
```

```
return mobile;
}

public void setMobile(int mobile) {
this.mobile = mobile;
}

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "idcontrol")
public Control getControl() {
return control;
}

public void setControl(Control control) {
this.control = control;
}

@OneToOne(fetch = FetchType.LAZY, mappedBy = "user")
public Apartment getApartment() {
return apartment;
}

public void setApartment(Apartment apartment) {
this.apartment = apartment;
}

@OneToMany(fetch = FetchType.LAZY, mappedBy = "user")
public List<Story> getStories() {
return stories;
}

public void setStories(List<Story> stories) {
this.stories = stories;
}

@OneToMany(fetch = FetchType.LAZY, mappedBy = "user")
public List<Comment> getComments() {
return comments;
}

public void setComments(List<Comment> comments) {
this.comments = comments;
}

@OneToMany(fetch = FetchType.LAZY, mappedBy = "landlord")
public List<LikeHouse> getProposals() {
return proposals;
}

public void setProposals(List<LikeHouse> proposals) {
this.proposals = proposals;
}
```

```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "lessee")
public List<LikeHouse> getBids() {
return bids;
}

public void setBids(List<LikeHouse> bids) {
this.bids = bids;
}
}
package org.jazzteam.servlets;


import org.jazzteam.model.serializer.CommentManager;
import org.jazzteam.model.table.User;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class AddComment extends HttpServlet {
private CommentManager commentManager;

@Override
public void init() throws ServletException {
commentManager = new CommentManager();
}

@Override
protected void service(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
User user = (User) request.getSession().getAttribute("show");
User commentator = (User) request.getSession().getAttribute("user");
String text = request.getParameter("text");
response.sendRedirect("reviewShow.jsp");
try {
commentManager.addComment(user, commentator, text);
} catch (Exception e) {
e.printStackTrace();

}
}
}
package org.jazzteam.servlets;

import org.jazzteam.model.serializer.UserManager;
import org.jazzteam.model.table.User;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import java.io.IOException;

public class LogIn extends HttpServlet {
private UserManager userManager;

@Override
public void init() {
userManager = new UserManager();
}


@Override
protected void service(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
final String nickname = request.getParameter("nickname");
final String password = request.getParameter("password");
User user = null;
try {
user = userManager.getUser(nickname);
if (user.getPassword().equals(password)) {
if (user.getControl().getType().equals("")) {
response.getWriter().print("Landlord");
} else {
response.getWriter().print("Lessee");
}
request.getSession().setAttribute("user", user);
} else {
response.getWriter().print("ERROR");
}
} catch (Exception e) {
response.getWriter().print("ERROR");
}

}

}
package org.jazzteam.servlets;


import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;

public class LogOut extends HttpServlet {


@Override
protected void service(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
HttpSession session = request.getSession();
```

```
session.removeAttribute("user");
session.invalidate();
response.sendRedirect("logIn.jsp");
}
}
package org.jazzteam.servlets;

import org.jazzteam.model.table.User;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;

public class Navigation {

public static void redirectToIndexPage(final HttpServletResponse response,
    final String redirect) {
try {
response.sendRedirect(redirect);
} catch (IOException e) {
e.printStackTrace();
}
}

public static boolean objectInSession(HttpSession session, String name) {
if (getSessionUser(session,name) != null) {
return true;
}
return false;
}

public static User getSessionUser(HttpSession session, String name) {
final User user = (User) session.getAttribute(name);
return user;
}




}
package org.jazzteam.servlets;

import org.jazzteam.model.serializer.UserManager;
import org.jazzteam.model.table.User;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class Registration extends HttpServlet {
private UserManager userManager;
```

```
@Override
public void init() {
userManager = new UserManager();
}


protected void doPost(HttpServletRequest request, HttpServletResponse response
    ) throws IOException {
User user = new User();
user.setNikname(request.getParameter("nickname"));
user.setPassword(request.getParameter("password"));
user.setEmail(request.getParameter("email"));
user.setName(request.getParameter("name"));
user.setSurname(request.getParameter("surname"));
user.setMobile(Integer.parseInt(request.getParameter("mobile")));
user.setCity(request.getParameter("city"));
try {
userManager.addUser(user, request.getParameter("type"));
response.sendRedirect("logIn.jsp");
System.out.println("Ok");
} catch (Exception e) {
response.sendRedirect("registration.jsp");
response.getWriter().print("ERROR");
System.out.println("Error");
}

}
}
package org.jazzteam.servlets;

import org.jazzteam.model.table.User;
import org.json.JSONException;
import org.json.JSONObject;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jazzteam.model.table.Comment;

import java.io.IOException;

public class Review extends HttpServlet {

@Override
protected void service(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
request.setCharacterEncoding("UTF-8");
response.setCharacterEncoding("UTF-8");
User user = (User) request.getSession().getAttribute("user");
String allComment = "[";
JSONObject comment = new JSONObject();
```

```
if (user.getComments() != null) {
for (Comment item : user.getComments()) {
try {
comment.put("comment", item.getComment());
comment.put("commentator", item.getCommentator().getNikname());
comment.put("user", item.getUser().getNikname());
allComment += comment + ",";

} catch (JSONException e) {
e.printStackTrace();
}
}
allComment = allComment.substring(0, allComment.length() - 1);
allComment += "]";
response.getWriter().print(allComment);
} else {
response.getWriter().print("ERROR");
}
}
}
package org.jazzteam.servlets;

import org.jazzteam.model.table.User;
import org.json.JSONException;
import org.json.JSONObject;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jazzteam.model.table.Comment;

import java.io.IOException;

public class ReviewShow extends HttpServlet {

@Override
protected void service(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
request.setCharacterEncoding("UTF-8");
response.setCharacterEncoding("UTF-8");
User user = (User) request.getSession().getAttribute("show");
String allComment = "[";
JSONObject comment = new JSONObject();
if (user.getComments() != null) {
for (Comment item : user.getComments()) {
try {
comment.put("comment", item.getComment());
comment.put("commentator", item.getCommentator().getNikname());
comment.put("user", item.getUser().getNikname());
allComment += comment + ",";
```

```
} catch (JSONException e) {
e.printStackTrace();
}
}
allComment = allComment.substring(0, allComment.length() - 1);
allComment += "]";
response.getWriter().print(allComment);
} else {
response.getWriter().print("ERROR");
}
}
}
package org.jazzteam.servlets;

import org.jazzteam.model.serializer.ApartmentManager;
import org.jazzteam.model.table.Apartment;
import org.jazzteam.model.table.User;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class Settings extends HttpServlet {

private ApartmentManager apartmentManager;

@Override
public void init() {
apartmentManager = new ApartmentManager();

}

@Override
protected void service(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
User user = (User) request.getSession().getAttribute("user");
try {
if (user.getApartment() != null) {
user.getApartment().setStatus(request.getParameter("status"));
user.getApartment().setDescription((request.getParameter("description")));
user.getApartment().setPrice(Integer.parseInt(request.getParameter("price")));
apartmentManager.updateApartment(user.getApartment());
} else {
Apartment apartment = new Apartment();
apartment.setStatus(request.getParameter("status"));
apartment.setPrice(Integer.parseInt(request.getParameter("price")));
apartment.setDescription(request.getParameter("description"));
apartmentManager.addApartment(apartment, user);
}
} catch (Exception e) {
response.getWriter().print("Error");
```

```
}finally {
request.getSession().setAttribute("user", user);
response.sendRedirect("landlord.jsp");
}
}
}
package org.jazzteam.servlets;

import org.jazzteam.model.serializer.UserManager;
import org.jazzteam.model.table.User;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class Show extends HttpServlet {
private UserManager userManager;

@Override
public void init() {
userManager = new UserManager();
}

@Override
protected void service(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
try {
System.out.println(request.getParameter("nickname"));
User user=userManager.getUser(request.getParameter("nickname"));
request.getSession().setAttribute("show",user);
} catch (Exception e) {
e.printStackTrace();
}
}
}
```