# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

На современном уровне развития автоматизация процессов представляет собой один из подходов к управлению процессами на основе применения информационных технологий. Этот подход позволяет осуществлять управление операциями, данными, информацией и ресурсами за счет использования компьютеров и программного обеспечения, которые сокращают степень участия человека в процессе, либо полностью его исключают.

Основной целью автоматизации является повышение качества исполнения процесса. Автоматизированный процесс обладает более стабильными характеристиками, чем процесс, выполняемый в ручном режиме. Во многих случаях автоматизация процессов позволяет повысить производительность, сократить время выполнения процесса, снизить стоимость, увеличить точность и стабильность выполняемых операций.

Оценка постоянно требует большого сопутствующего аппарата. В наше время научно–технического прогресса и стремительного развития технологии почти никто не обходится без так называемой Всемирной сети и персонального компьютера.

Автоматизированный процесс оценки состоит из трех этапов:

1) сбор и анализ данных;

2) проведение расчетов;

3) создание итоговых документов

Целью данного курсового проекта является автоматизация оценки недвижимости путем расчета ее стоимости на основе введенных пользователем данных. Для визуального представления данных используются различные графики, на основе которых пользователь может быстрее и легче сделать выводы об изменении цены недвижимости в зависимости от отдаленности от центра, года постройки.

Для реализации проекта необходимо:

- изучить ситуацию на рынке недвижимости;
- изучить основные методики оценки;
- изучить, на основе каких данных производится оценка.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ К СИСТЕМЕ С ТОЧКИ ЗРЕНИЯ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Описание предметной области системы

Оценкой стоимости недвижимости называют процесс определения стоимости конкретного объекта недвижимости: например, производственного цеха, квартиры, дачи, гаража, частного дома или другого. Проведение оценки недвижимости строго регламентировано.

При осуществлении оценочной деятельности используются четыре вида стоимости объекта оценки: рыночная, ликвидационная, инвестиционная и кадастровая.

Рыночная стоимость объекта недвижимости – это цена, за которую его можно продать в условиях свободного рынка. Именно её используют, например, покупатели и продавцы при обсуждении стоимости недвижимости, либо юридические лица в качестве доказательства наличия определённых активов. Рыночную стоимость определяют, в зависимости от условий рынка: находят аналогичные объекты и рассчитывают среднерыночную цену. Она же появляется в результатах оценки.

Ликвидационная стоимость объекта недвижимости – это цена, по которой его можно продать быстро. Так, при расчете стоимости оценщики используют срок экспозиции 3-4 месяца. При расчёте ликвидационной стоимости берут срок экспозиции 1-2 месяца. Ликвидационная стоимость объекта недвижимости всегда меньше, чем рыночная. Обычно она составляет 75–80% от рыночной цены.

Инвестиционная стоимость объекта оценки – это цена для конкретного человека или группы лиц при определённых инвестиционных целях. Чаще всего ее используют для оценки эффективности отдельных проектов, например, чтобы определить, будет ли приносить доход недвижимость при сдаче в аренду. Инвестиционная стоимость всегда больше рыночной, потому что учитывает возможную прибыль с объекта в дальнейшем. Если она меньше, значит, выбран плохой план, и проще просто продать недвижимость по рыночной цене.

Кадастровая стоимость объекта недвижимости – это сумма, установленная в результате государственной кадастровой оценки. Её определяют методами массовой оценки, а если это невозможно – индивидуально для каждого объекта недвижимости. Оценщик определяет кадастровую стоимость, в том числе, для налогообложения. Реже используют

другие типы стоимости при проведении оценки, например, восстановительную. Восстановительная стоимость объекта недвижимости – это сумма, которую нужно потратить на строительство такого же объекта в текущих условиях. При её расчёте учитывают зарплату рабочих, стоимость строительных материалов, оплату проектных материалов и другие прямые, косвенные затраты.

Оценщики могут использовать три метода расчёта – сравнительный, доходный и затратный.

Основной тезис сравнительного метода в том, что покупатель не заплатит за недвижимость цену больше, чем та, по которой можно приобрести точно такой же объект. При применении этого метода оценщик должен:

1. Выбрать единицы сравнения и сравнить объект оценки с аналогичным.
2. Скорректировать значение единиц оценки, в зависимости от характеристик объекта сравнения и аналогов.
3. Согласовать результаты корректирования.

В зависимости от целей анализа, может быть использован разный период времени, в течение которого продавались объекты. В любом случае оценщик отбирает практически аналогичную недвижимость. Затем проводит сравнительный анализ отобранной недвижимости и оцениваемого объекта, корректирует цены и находит средний показатель. Этот метод требует изучения большого объема информации и расчётов, но позволяет определить стоимость как можно точнее. Например, при расчёте рыночной стоимости объекта в центре города оценщик изучает многочисленные аналоги в центре.

Суть доходного метода заключается в следующем. Стоимость недвижимости на момент оценки определяют, как источник будущих доходов. То есть оценщик определяет потенциальную прибыль объекта с момента оценки до завершения эксплуатации. При применении метода учитывают риски, характерные для имущества региона. Доходный метод используют при оценке рыночной ценности, а также в инвестиционной деятельности. Он соответствует ключевой идее инвестиций: нет смысла платить за недвижимость больше, чем она может принести прибыли.

Главная мысль проста: объект недвижимости не может стоить больше, чем сумма, которую придётся потратить на его строительство сейчас. Этот метод используют, когда нужно оценить объекты недвижимости без отрыва от земельных участков, на которых они расположены. При затратном методе оценщик учитывает в числе других параметров и стоимость надела.

При применении затратного метода эксперт действует так:

1. Оценивает рыночную стоимость земельного участка.
2. Определяет восстановительную стоимость здания, в том числе размер предпринимательской прибыли.
3. Оценивает выявленные виды износа.
4. Рассчитывает итоговую стоимость объекта. Для этого корректирует восстановительную стоимость на износ и добавляет стоимость земельного участка.

Самая точная цифра получается, когда оценщик использует сразу три метода и принимает во внимание результаты всех полученных расчетов. Если перед специалистом стоит необычная цель, он может использовать и другие методы оценки, например, ипотечно-инвестиционный анализ.

Самый очевидный и главный фактор, который влияет на цену – себестоимость недвижимости. Под себестоимостью недвижимости обычно понимают цену на строительные материалы и расходы на строительство – оплату труда рабочим, затраты на проектную документацию, транспортировку материалов и другие. Например, частный дом из кирпича высотой в 3 этажа будет стоить дороже коттеджа меньшей площади высотой 2 этажа из пеноблоков. Также на стоимость влияют множество других параметров, не зависящих от расходов на строительство:

– ситуация на рынке, например, если в последние 2–3 месяца квартиры в городе подешевели на 20 – 30%, скорее всего, не получится продать квартиру по прежней полной стоимости: тоже придётся делать скидку;

– сезонность – некоторые объекты зависят от сезона продажи: например, дачи летом имеют максимальную стоимость, а зимой она снижается на 15–25% в зависимости от региона;

– местоположение – жилье в центре всегда ценится дороже аналогичного на окраине, потому что центр города отличается более развитой инфраструктурой;

– инфраструктура района или населённого пункта – жилье в небольших посёлках с 1–2 магазинами стоит намного дешевле домов в городе, неподалёку от которых расположены детские сады, школы, магазины, поликлиники и другие важные объекты;

– степень износа – чем старше дом, тем больше денег придётся вложить в ремонт, поэтому старые объекты ценятся меньше;

– материал стен – от него зависит, тепло ли будет в помещении, поэтому жилье из кирпича или древесины ценится выше, чем дом из шлакоблока, пеноблоков – последние пропускают больше холода;

– экологическая обстановка – недвижимость рядом с озерами, лесами, парками ценится больше, чем жилье неподалёку от крупных заводов, перерабатывающих предприятий;

– наличие балкона – он создаёт дополнительное свободное пространство, поэтому повышает ценность недвижимости;

– характеристики двора, например, если речь идет о многоквартирных домах, то квартиры с закрытым двором с видеонаблюдением будут стоить дороже, чем аналогичное жилье с открытым двором.

Все эти параметры оценщик учитывает и обязательно отражает в итоговом документе – отчёте об оценке.

Оценка объектов недвижимости проходит в течение разного срока: от 1–2 дней до 2–3 недель. Все зависит от количества исходных данных, региона, числа заказов у оценщика. В целом процедура включает в себя 6 этапов:

1. Постановка задания на оценку.
2. Подписание договора.
3. Сбор информации.
4. Расчет стоимости.
5. Учёт условий.
6. Формирование отчёта.

В целом, оценка недвижимости необходима в случае:

– судебных споров;
– оформления ипотеки;
– реорганизации предприятия;
– продажи недвижимости;
– обмена жилья;
– получения залога.

Также оценку недвижимости могут использовать в других сделках с недвижимостью – например, при оформлении договора дарения, завещания.

Чтобы использовать сложные методы оценки стоимости, нужно иметь определённые знания и опыт. Лучше найти цену сравнительным методом – просто посмотреть, сколько стоят аналогичные объекты недвижимости, и найти средний показатель. В случае с оценкой квартиры, следует учитывать следующие показатели:

– расположение, инфраструктуру района;
– тип, конструкцию дома;
– класс отделки;
– количество комнат;
– состояние отделки;

– этаж, наличие балконов, лоджий;
– площадь кухни.

Провести оценку недвижимости можно самостоятельно, но результаты могут быть далеки от реальности, и эту оценку не примет банк или государственный орган. Оценка недвижимости специалистом нужна для многих типов сделок, для оформления ипотеки, кредита с залогом. Она стоит денег, но отличается более высокой точностью.

## 1.2 Требования к системе с точки зрения предметной области

Изучив предметную область, следует выделить следующие требования к системе:

– возможность регистрации и авторизации пользователей;
– возможность редактирования аккаунтов пользователей;
– возможность просмотра всех введенных объектов недвижимости пользователем;
– хранение и добавление недвижимости, подлежащей оценке, в базе данных;
– возможность графического отображения данных об объектах недвижимости в зависимости от отдаленности от центра, года постройки.

Также необходимо обеспечить пользовательский интерфейс, позволяющий быстро и легко освоить работу с программой.

# 2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

## 2.1 Задачи, решаемые системой

Основной задачей, решаемой разрабатываемой системой, является расчет стоимости недвижимости на основе введенных показателей. Описание бизнес–процесса расчета представлено на рисунке 2.1.

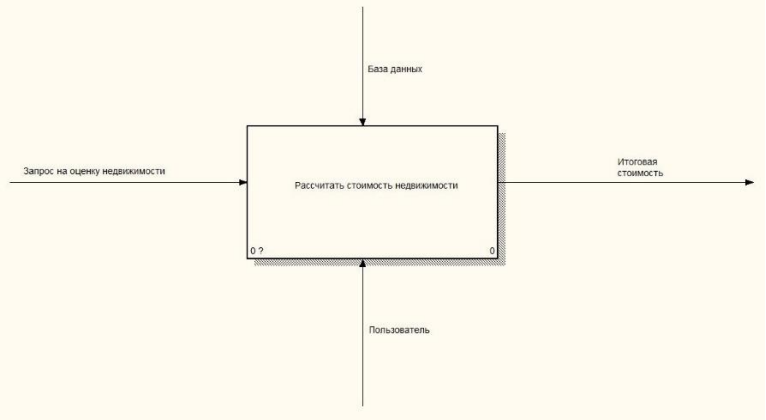


Рисунок 2.1 – Бизнес-процесс расчёта стоимости недвижимости.

На рисунке 2.2 представлена декомпозиция бизнес-процесса расчёта стоимости недвижимости. Она описывает разделение расчета стоимости, которое состоит из следующих этапов:

– Авторизацию.
– Ввод необходимой информации о недвижимости.
– Расчет стоимости недвижимости.

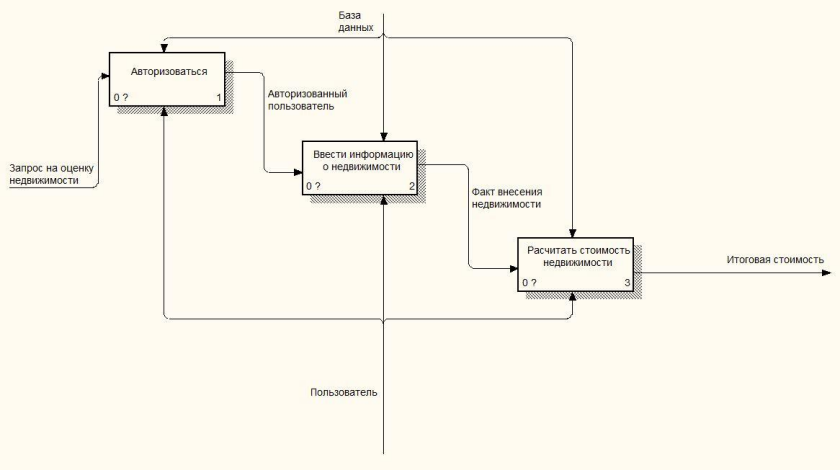


Рисунок 2.2 – Декомпозиция диаграммы верхнего уровня.

На рисунке 2.3 представлена декомпозиция процесса A1 «Авторизоваться». Данная диаграмма состоит из следующих процессов:

– Ввод логина.
– Ввод пароля.
– Авторизация или регистрация.

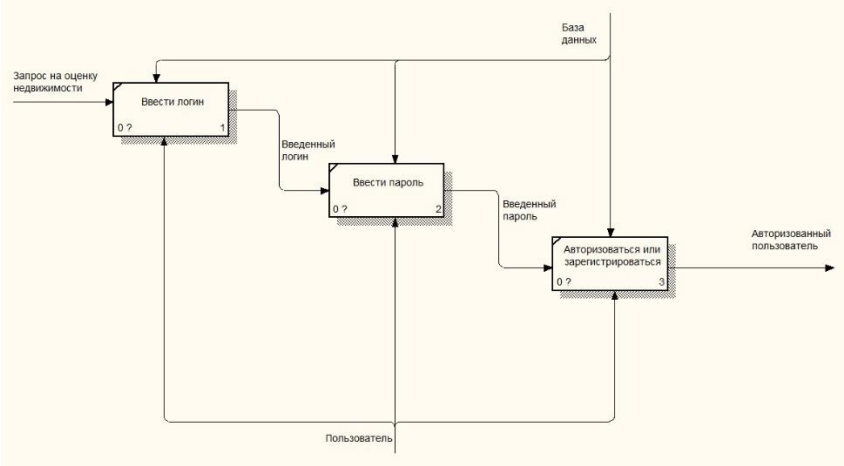


Рисунок 2.3 – Декомпозиция процесса A1.

На рисунке 2.4 представлена декомпозиция процесса A2 «Ввести информацию о недвижимости». Данная диаграмма состоит из следующих процессов:

– Ввод адреса.
– Ввод площади.
– Ввод расстояния от центра.
– Ввод года постройки.

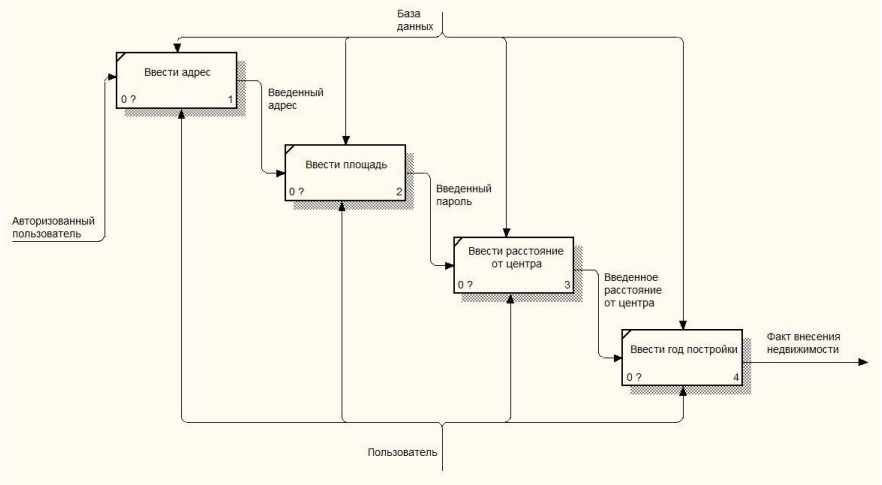


Рисунок 2.4 – Декомпозиция процесса A2

На рисунке 2.5 представлена декомпозиция процесса А3 «Рассчитать стоимость недвижимости». Данная диаграмма состоит из следующих процессов:

– Расчет стоимости сравнительным методом оценки.

– Вывод результата.

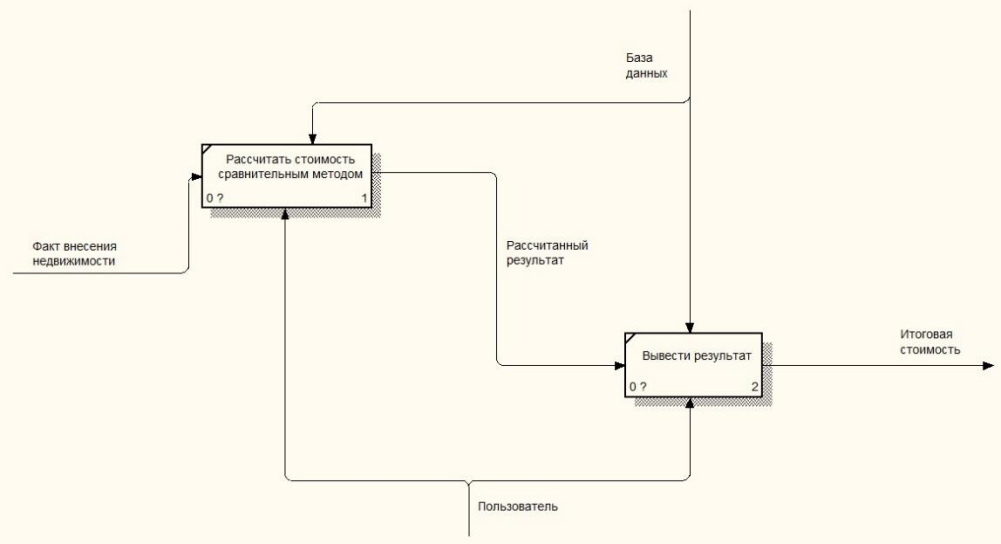


Рисунок 2.5 – Декомпозиция блока А3.

В процессе расчета стоимости недвижимости участвует непосредственно пользователь. Входными данными процесса являются: адрес недвижимости, её площадь, степень ремонта и год постройки.

На выходе необходимо получить отчет о стоимости недвижимости с возможностью записи его в базу данных. Расчет производится на основе сравнительного методом с определенными коэффициентами степени ремонта, отдаленности от центра и года постройки недвижимости.

## 2.2 Требования к системе

Необходимо придерживаться следующих требований к системе:

– Приложение должно быть выполнено в архитектуре клиент-сервер с многопоточным сервером с организацией взаимодействия с использованием языка программирования *Java*. Будут реализованы: собственная иерархия классов, паттерн проектирования *Singleton*, перегрузка методов, интерфейсы и абстрактные классы, статические методы и т.д.

– Бизнес–логика системы реализована на серверной части данного приложения.

– СУБД – *MySQL*. Доступ к данным в СУБД осуществляется через драйвер, предоставляемый производителем СУБД.

– База данных должна содержать не менее 5 таблиц и быть приведена к 3–ей нормальной форме.

– База данных должна генерироваться *SQL*-скриптом.

– Взаимодействие между серверной и клиентскими частями должно осуществляется с использованием сокетов и протокола *TCP/IP*.

– Классы и библиотеки должны размещаться в пакетах.

## 2.3 Спецификация вариантов использования системы

Диаграмма вариантов использования позволяет провести анализ использования системы, то есть провести описание ее основного предназначения.

Диаграмма вариантов использования разрабатываемой системы представлена на рисунке 2.6.

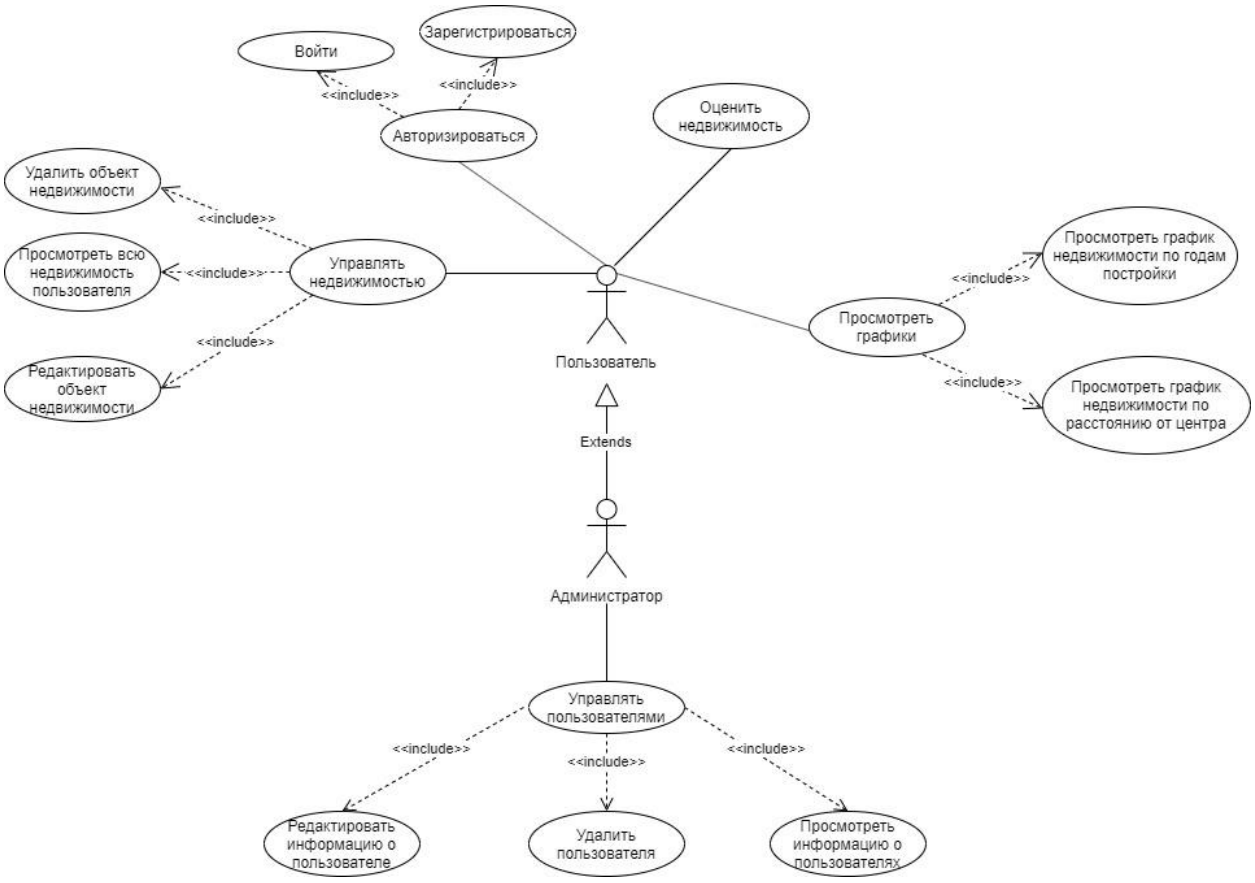


Рисунок 2.6 – Диаграмма вариантов использования.

Пользователи, работающие с данной системой, будут трёх типов:

– администратор;
– обычный пользователь;

В разрабатываемой системе будут реализованы следующие варианты использования:

1. Авторизация. Пользователи программы имеют возможность войти в систему под своим логином и паролем. В зависимости от роли, отведенной пользователю, он сможет иметь определенный функционал.
2. Пользователю будет предложена возможность управлять данными о принадлежащих ему объектах недвижимости. Он сможет их просматривать, удалять и редактировать.
3. Пользователь сможет ввести новые данные о недвижимости и сразу же получить её оценку.
4. Пользователь сможет визуализировать данные о недвижимости по годам постройки и расстоянию от центра.
5. Администратор будет обладать доступом к пользовательским данным и сможет просматривать, редактировать и удалять данные о пользователе, если в этом будет необходимость.

# 3 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ

## 3.1 Модели представления системы

### 3.1.1 Диаграммы классов системы

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

На рисунке 3.1 приведена диаграмма классов, ответственных за процесс авторизации пользователя. Данная диаграмма содержит класс *LoginWindow*, который представляет собой окно для ввода пользовательских данных и включает в себя различные компоненты библиотеки *Swing*, такие как *JTextField*, *JLabel*, *JButton*. Также на диаграмме присутствует класс *LoginController*, который непосредственно отвечает за логику авторизации.

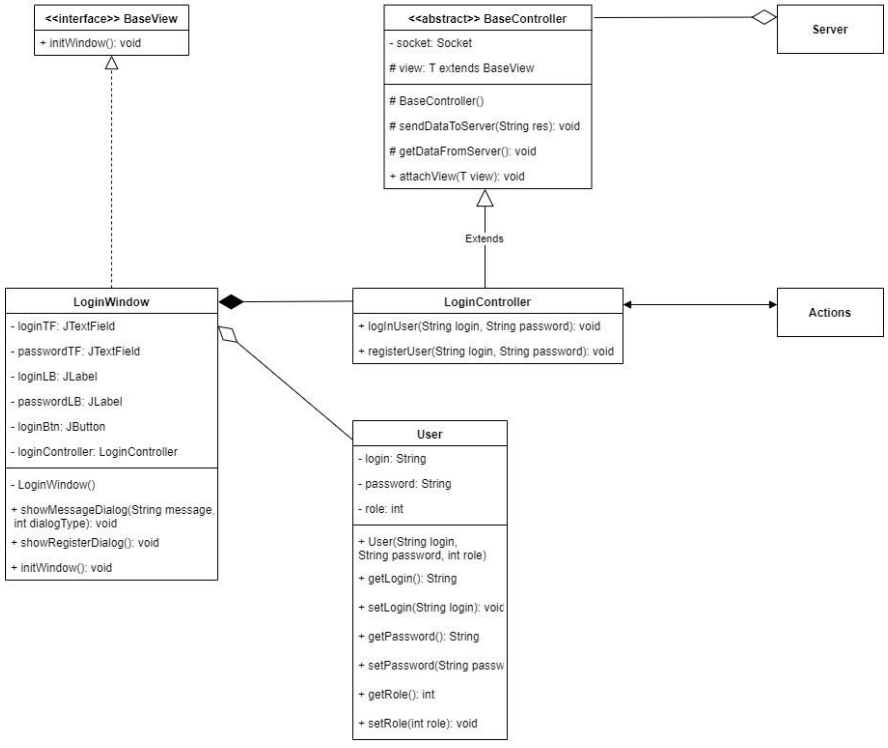


Рисунок 3.1 – Диаграмма классов процесса авторизации.

На рисунке 3.2 приведена диаграмма классов, иллюстрирующая процесс оценки недвижимости. На ней присутствуют такие классы, как *EvaluatePropertyWindow*, представляющий собой окно для ввода данных о недвижимости, *EvaluatePropertyController*, который необходим для расчета стоимости недвижимости. Также есть класс *Property*, который представляет собой сущность недвижимости.

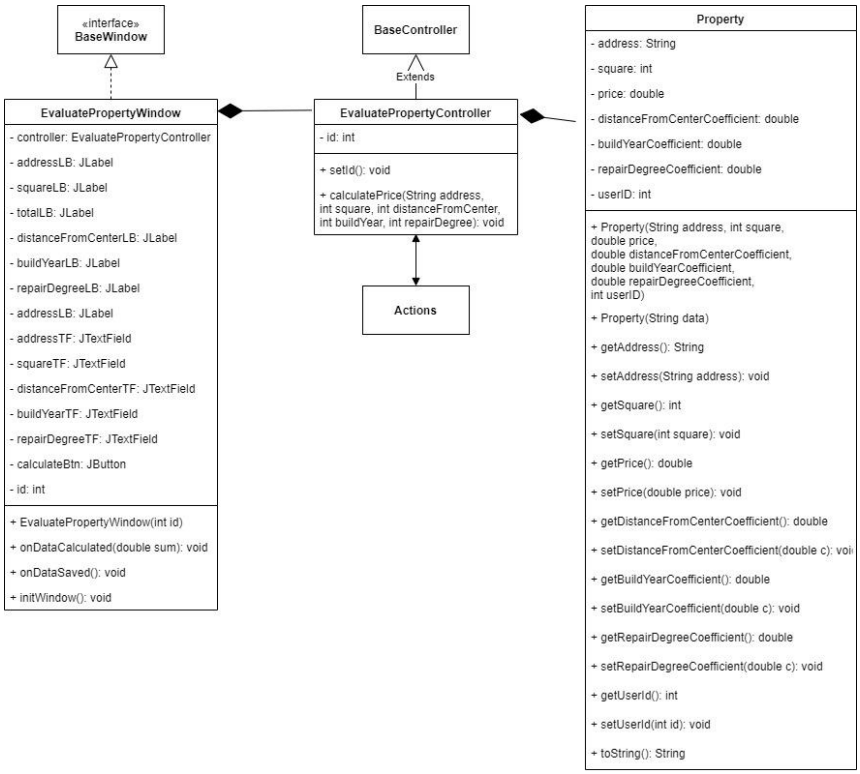


Рисунок 3.2 – Диаграмма классов оценки недвижимости.

На рисунке 3.3 представлена диаграмма классов, отвечающих за отображение всей пользовательской недвижимости. На ней присутствуют такие классы, как *ShowAllUserPropertiesWindow*, *ShowAllUserPropertiesController* и *Actions*.

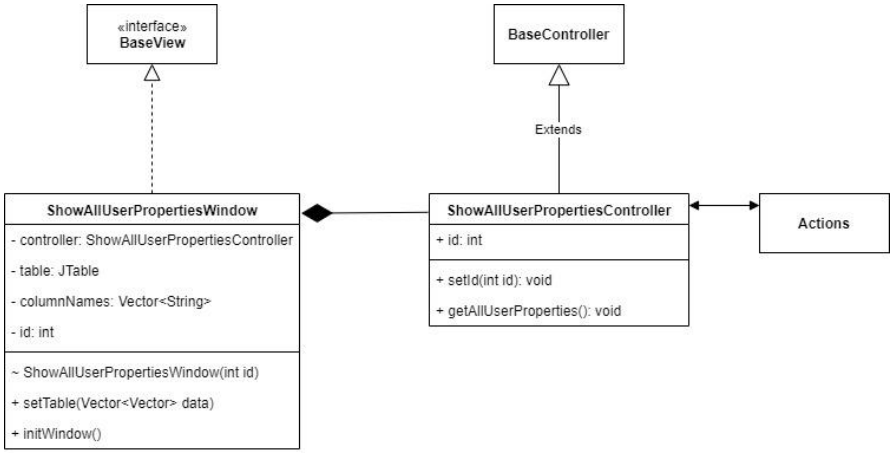

Рисунок 3.3 – Диаграмма классов отображения всей недвижимости.

Класс *ShowAllUserPropertiesWindow* является окном для отображения всей пользовательской недвижимости. В нем присутствует таблица, содержащая в себе такие столбцы, как Адрес, Площадь, Цена, Коэффициент Расстояния От Центра, Коэффициент Года Постройки, Коэффициент Степени Ремонта.

На рисунке 3.4 представлена диаграмма классов, отвечающих за показ диаграмм. На ней присутствуют классы *ChartsController*, который отвечает за получение данных для диаграмм и классы *YearPieChartWindow* и *DistanceFromCenterBarChartWindow*, представляющие собой окна для отображения графиков.

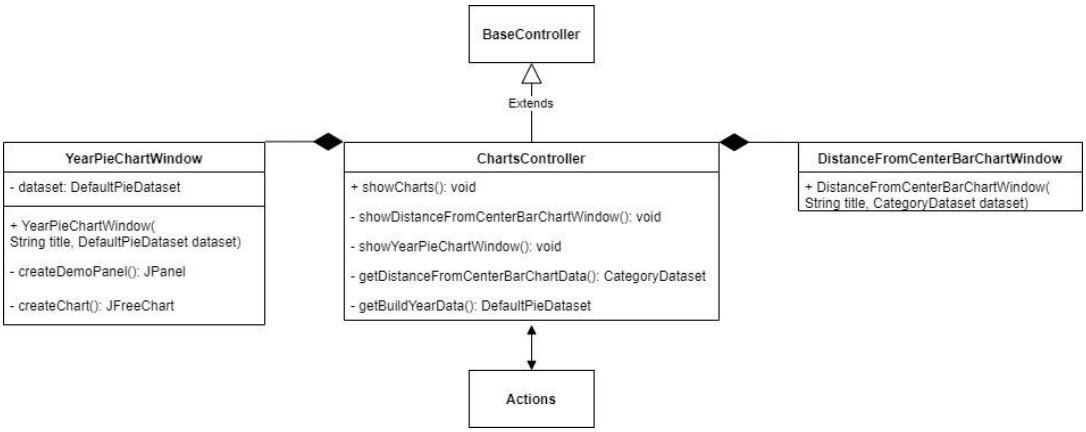


Рисунок 3.4 – Диаграмма классов отображения диаграмм.

На рисунке 3.5 представлена диаграмма классов, ответственных за отображение всех пользователей в виде таблицы. Данная возможность доступно только администратору.

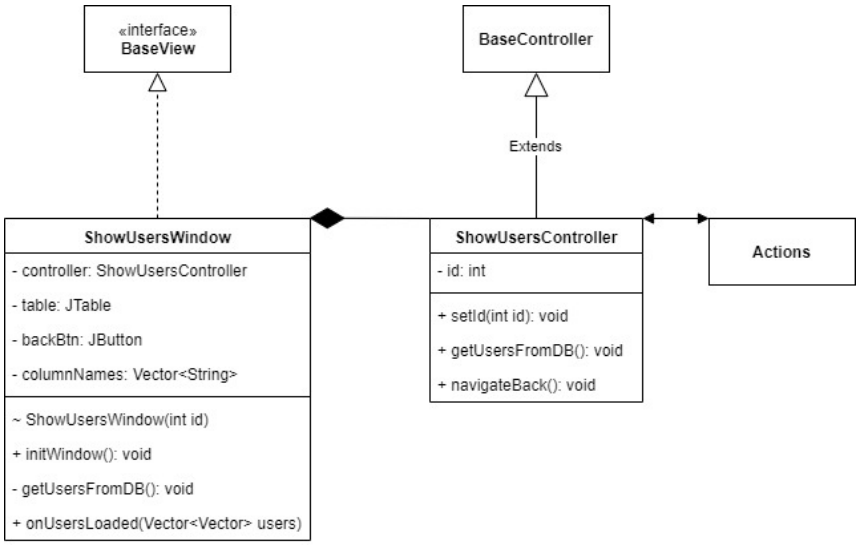


Рисунок 3.5 – Диаграмма классов отображения всех пользователей.

Класс *ShowUsersWindow*, представленный на этой диаграмме, содержит таблицу всех пользователей. Класс *ShowUsersController* подготавливает данные для отправки серверу запроса о получении информации обо всех пользователях.

На рисунке 3.6 представлен класс *Server* с его полями и методами. Сервер осуществляет обращение к базе данных и принимает запросы на получения данных от класса *BaseController*.

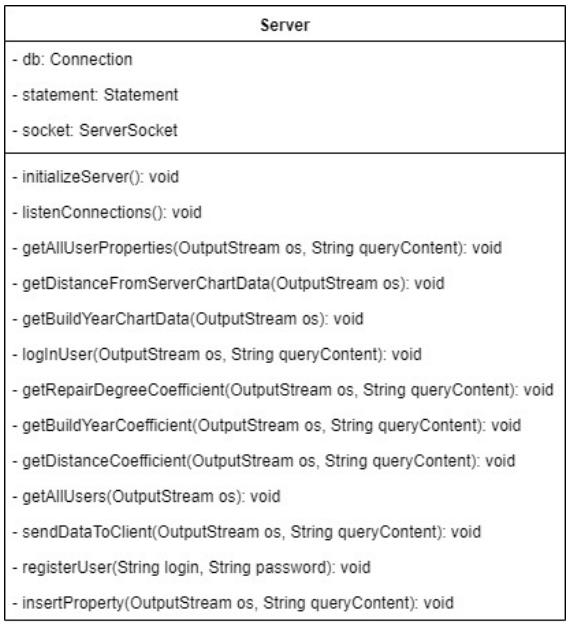


Рисунок 3.6 – поля и методы класса *Server*.

На рисунке 3.7 представлены поля класса *Actions*. Все поля являются строковыми константами и служат для определения сервером типа запроса.

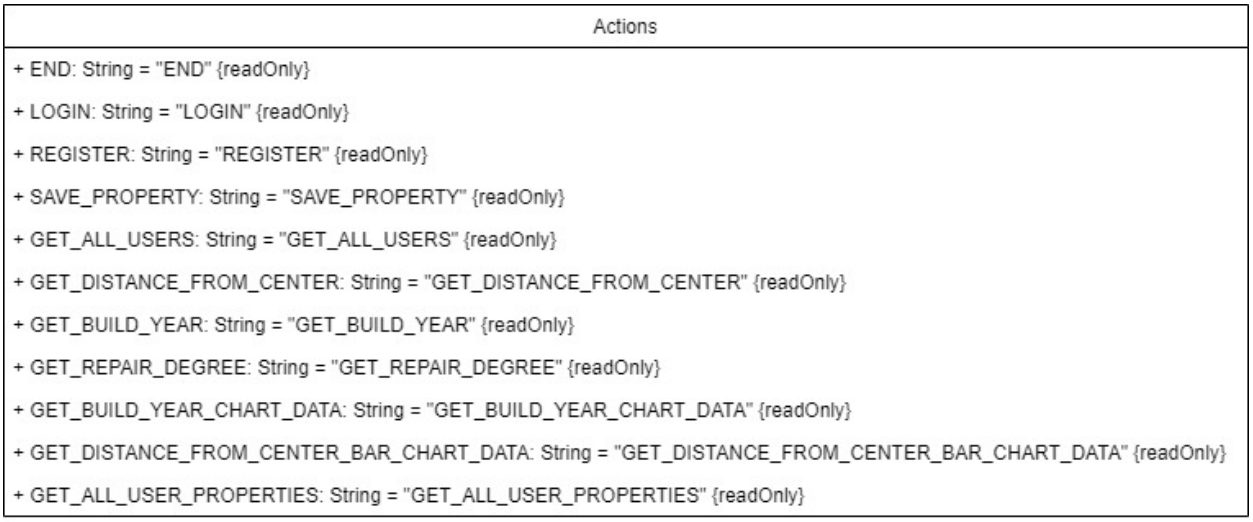


Рисунок 3.7 – поля класса *Actions*.

### 3.1.2 Диаграмма последовательности

Диаграммы последовательности (sequence diagram) являются видом диаграмм взаимодействия языка UML, которые описывают отношения объектов в различных условиях. Условия взаимодействия задаются сценарием, полученным на этапе разработки диаграмм вариантов использования.

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. На этих диаграммах изображаются только те объекты, которые непосредственно участвуют во взаимодействии т.к. ключевым моментом является именно динамика взаимодействия объектов во времени и не используются возможные статические ассоциации с другими объектами.

При этом диаграмма последовательности имеет два измерения. Одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Второе измерение – вертикальная временная ось, направленная сверху вниз. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже.

На рисунке 3.8 представлена диаграмма последовательности оценки недвижимости.

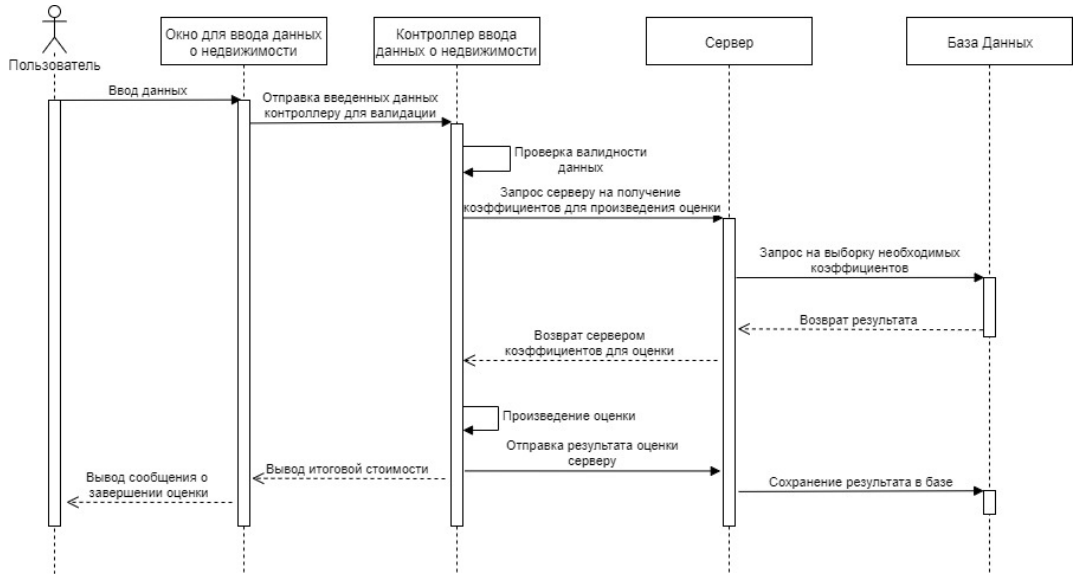


Рисунок 3.8 – Диаграмма последовательности оценка недвижимости.

### 3.1.3 Диаграммы состояния основных объектов системы

Диаграмма состояния показывает, как объект переходит из одного состояния в другое. Диаграммы состояний служат для моделирования динамических аспектов системы. Данная диаграмма полезна при моделировании жизненного цикла объекта. От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта реактивного, то есть объекта, поведение которого характеризуется его реакцией на внешние события.

На рисунке 3.9 представлена диаграмма состояния всего процесса авторизации пользователя.

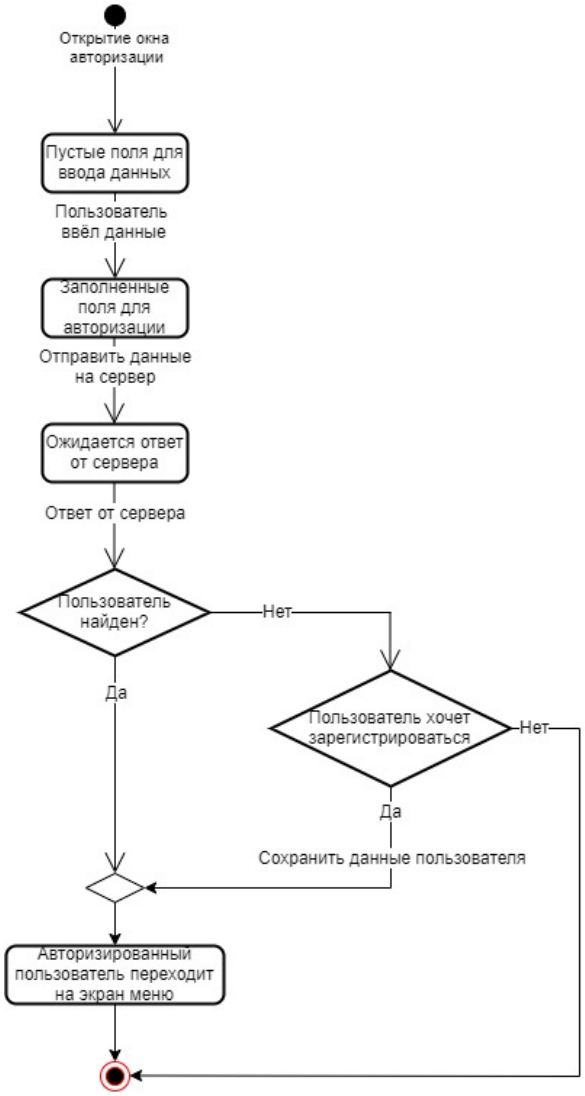


Рисунок 3.9 – Диаграмма состояния процесса авторизации пользователя.

### 3.1.4 Диаграмма компонентов системы

Диаграмма компонентов – элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты.

С помощью диаграммы компонентов представляются инкапсулированные классы вместе с их интерфейсными оболочками, портами и внутренними структурами (которые тоже могут состоять из компонентов и коннекторов).

Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом иллюстрируются отношения клиент-источник между двумя компонентами.

Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Когда диаграмма компонентов используется, чтобы показать внутреннюю структуру компонентов, предоставляемый и требуемый интерфейсы составного компонента, могут делегироваться в соответствующие интерфейсы внутренних компонентов.

Делегация показывается связь внешнего контракта компонента с внутренней реализацией этого поведения внутренними компонентами.

Диаграмма компонентов системы представлена на рисунке 3.10.

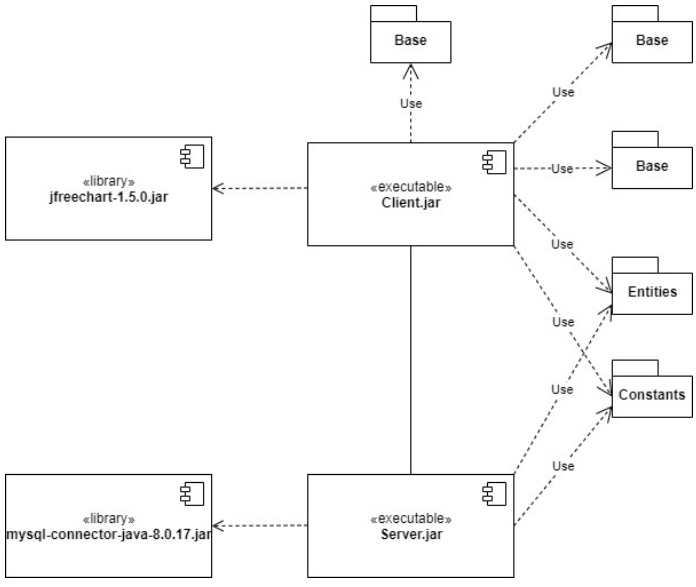


Рисунок 3.10 – Диаграмма компонентов системы.

### 3.1.5 Диаграмма развертывания системы

Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах компании, вызывают веб-сервисы, используют общие ресурсы и т. д. В таких случаях полезно иметь графическое представление инфраструктуры, на которую будет развернуто приложение. Для этого и нужны диаграммы развертывания, которые иногда называют диаграммами размещения.

Такие диаграммы есть смысл строить только для аппаратно-программных систем, тогда как UML позволяет строить модели любых систем, не обязательно компьютерных.

Диаграмма развертывания показывает топологию системы и распределение компонентов системы по ее узлам, а также соединения - маршруты передачи информации между аппаратными узлами. Это единственная диаграмма, на которой применяются “трехмерные” обозначения: узлы системы обозначаются кубиками. Все остальные обозначения в UML - плоские фигуры.

Диаграмма развёртывания представлена на рисунке 3.11.

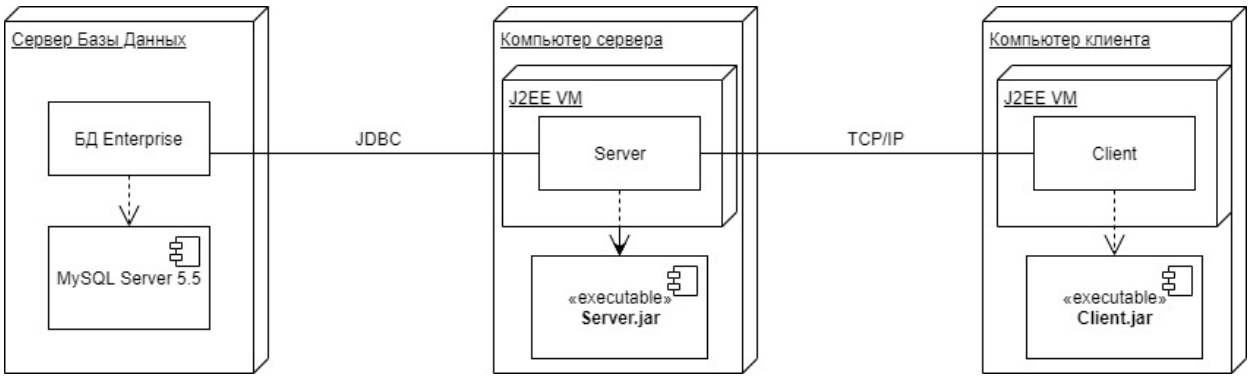


Рисунок 3.11 – Диаграмма развёртывания.

## 3.2 Применение паттернов проектирования

Паттерн проектирования – это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет подстроить под нужды программы.

Описания паттернов обычно очень формальны и чаще всего состоят из таких пунктов:

– проблема, которую решает паттерн;

– мотивации к решению проблемы способом, который предлагает паттерн;

– структуры классов, составляющих решение;

– примера на одном из языков программирования;

– особенностей реализации в различных контекстах;

– связей с другими паттернами.

Паттерны отличаются по уровню сложности, детализации и охвата проектируемой системы.

Самые низкоуровневые и простые паттерны – идиомы. Они не универсальны, поскольку применимы только в рамках одного языка программирования.

Самые универсальные – архитектурные паттерны, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных её элементов.

Основные группы паттернов:

– порождающие – необходимы для гибкого создания объектов без внесения в программу лишних зависимостей;

– структурные – показывают различные способы построения связей между объектами;

– поведенческие – необходимы для эффективной коммуникации между объектами.

В данной курсовой работе был применён паттерн *Singleton* (Одиночка). Это порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

Данный паттерн применим в следующих случаях:

– когда в программе должен быть единственный экземпляр какого-то класса;

– когда необходимо иметь больше контроля над глобальными переменными.

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ

В данном курсовом проекте все данные хранятся в базе данных. Она состоит из следующих сущностей:

– пользователь;

– собственность;

– расстояние;

– год;

– степень ремонта.

Информационная модель базы данных представлена на рисунке 4.1.

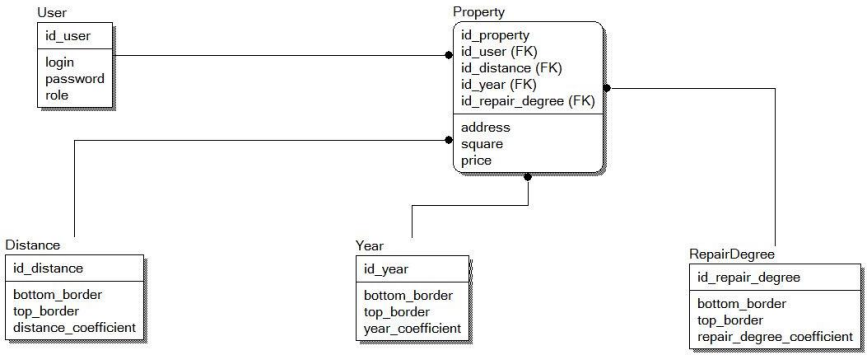


Рисунок 4.1 – Информационная модель.

Каждая сущность информационной модели имеет свой идентифицирующий номер – id.

Сущность Пользователь имеет следующие атрибуты:

– id_user – уникальный номер пользователя в системе. Является первичным ключом;

– login – содержит логин пользователя;

– password – содержит пароль пользователя;

– role – содержит роль пользователя в данной информационной системе. Может иметь значение 1, если пользователь является администратором. В противном случае имеет значение 0.

Данная сущность используется на этапе авторизации. Когда пользователь регистрируется, его данные добавляются в таблицу. После этого при помощи них он может авторизироваться.

Сущность Собственность состоит из следующих полей:

– id_property – уникальный номер собственности в системе;

– id_user – номер пользователя, которому принадлежит эта собственность;

– id_distance – расстояния, которому соответствует расстояние собственности от центра;

– id_year – года, которому соответствует год постройки недвижимости;

– id_repair_degree – номер степение ремонта, которому соответствует степень ремонта недвижимости;

– address – адрес, по которому расположена недвижимость;

– square – площадь недвижимости;

– price – стоимость недвижимости.

Данная таблица используется для хранения данных о недвижимости, которая подверглась оценке.

Сущность Дистанция состоит из:

– id_distance – уникальный номер расстояния от центра;

– bottom_border – минимальное расстояние, соответствующее коэффициенту;

– top_border – максимальное расстояние, соответствующее коэффициенту;

– distance_coefficient – коэффициент дистанции, участвующий в расчете стоимости недвижимости.

Сущность Год состоит из:

– id_year – уникальный номер года постройки;

– bottom_border – минимальный год постройки, соответствующий коэффициенту;

– top_border – максимальный год постройки, соответствующий коэффициенту;

– year_coefficient – коэффициент года постройки, участвующий в расчете стоимости недвижимости.

Сущность Степень Ремонта состоит из:

– id_repair_degree – уникальный номер степени ремонта;

– bottom_border – минимальная степень ремонта, соответствующая данному коэффициенту;

– top_border – максимальная степень ремонта, соответствующая данному коэффициенту;

– repair_degree_coefficient – коэффициент степени ремонта, участвующий в расчете стоимости недвижимости.

Сущности Дистанция, Год и Степень Ремонта используются при расчете стоимости недвижимости.

Данная база данных соответствует требованиям, потому что:

– все атрибуты простые и содержат только скалярные значения, а значит находится в 1НФ;

– каждый не ключевой атрибут неприводимо зависит от первичного ключа, а значит находится в 2НФ;

– каждый не ключевой атрибут нетранзитивно зависит от первичного ключа, а значит находится в 3НФ.

# 5 ОБОСНОВАНИЕ ОРИГИНАЛЬНЫХ РЕШЕНИЙ ПО ИСПОЛЬЗОВАНИЮ ТЕХНИЧЕСКИХ И ПРОГРАММНЫХ СРЕДСТВ

В данном курсовом проекте была использована сторонняя библиотека *JFreeChart*, загруженная с сайта *mvnrepository.com*.

*JFreeChart* - это бесплатная библиотека диаграмм *Java*, которая позволяет разработчикам легко отображать диаграммы профессионального качества в своих приложениях.

Обширный набор функций *JFreeChart* включает в себя:

– согласованный и хорошо документированный функционал, поддерживающий различные типы диаграмм;

– гибкий дизайн, который легко расширять, и предназначенный как для клиентских, так и для серверных приложений;

– поддержку многих типов вывода, включая компоненты *Swing* и *JavaFX*, файлы изображений (включая *PNG* и *JPEG*) и форматы графических файлов (включая *PDF, EPS* и *SVG*).

*JFreeChart* – это программное обеспечение с открытым исходным кодом, которое можно использовать в своих проектах.

# 6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СИСТЕМЫ

Главная задача данного проекта – оценка стоимости недвижимости. Для данной задачи в проекте предусмотрено отдельное окно *EvaluatePropertyWindow*.

После того, как пользователь ввел данные во все текстовые поля, он нажимает кнопку рассчитать. Код действия, происходящего по нажатию на кнопку, представлен ниже:

```
calculateBtn.addActionListener(e -> {
            try {
                String address = addressTF.getText();
                int square = Integer.parseInt(squareTF.getText());
                int distanceFromCenter =
Integer.parseInt(distanceFromCenterTF.getText());
                int buildYear =
Integer.parseInt(buildYearTF.getText());
                int repairDegree =
Integer.parseInt(repairDegreeTF.getText());
                controller.calculatePrice(address, square,
distanceFromCenter, buildYear, repairDegree);
            } catch (NullPointerException | NumberFormatException exc)
{
                JOptionPane.showMessageDialog(this, "Fill all fields
correctly", "Error!", JOptionPane.ERROR_MESSAGE);
                exc.printStackTrace();
            }
        });
```

Из окна данные передаются в контроллер. В нем происходит оценка стоимости. Ниже представлен код метода calculatePrice(String address, int square, int distanceFromCenter, int buildYear, int repairDegree):

```
public void calculatePrice(String address, int square, int
distanceFromCenter, int buildYear, int repairDegree) {
        double sum;

        sendDataToServer(Actions.GET_DISTANCE_FROM_CENTER);
        sendDataToServer(String.valueOf(distanceFromCenter));
        double distanceCoefficient = Double.parseDouble(getDataFromServer());
        int distanceID = (int) Double.parseDouble(getDataFromServer());

        sendDataToServer(Actions.GET_BUILD_YEAR);
        sendDataToServer(String.valueOf(buildYear));
        double yearCoefficient = Double.parseDouble(getDataFromServer());
        int yearID = (int) Double.parseDouble(getDataFromServer());
```

```
        sendDataToServer(Actions.GET_REPAIR_DEGREE);
        sendDataToServer(String.valueOf(repairDegree));
        double repairDegreeCoefficient =
Double.parseDouble(getDataFromServer());
        int repairID = (int) Double.parseDouble(getDataFromServer());

        sum = 1500 * square * distanceCoefficient * yearCoefficient *
repairDegreeCoefficient;

        Property property = new Property(address, square, sum, distanceID,
yearID, repairID, id);

        sendDataToServer(Actions.SAVE_PROPERTY);
        sendDataToServer(property.toString());
        String response = getDataFromServer();

        if (response.equalsIgnoreCase("Inserted")) {
            view.onDataCalculated(sum);
            view.onDataSaved();
        }
    }
```

Контроллер делает запрос серверу на получение коэффициентов расстояния от центра, года постройки и степени ремонта для расчёта стоимости. Для этого сначала посылается тип запроса (*Actions.GET_DISTANCE_FROM_CENTER*, *Actions.GET_BUILD_YEAR*, *Actions.GET_REPAIR_DEGREE*), а затем посылается значение, на основе которого возвращается необходимый коэффициент.

После того, как сервер возвращает запрошенные данные, производится расчёт стоимости недвижимости. Данные о недвижимости отправляются серверу, который сохраняет их в базе данных.

Из контроллера итоговая стоимость недвижимости передаётся назад окну и выводится в специально отведенное текстовое поле. При успешном сохранении в окне выводится диалоговое окно, уведомляющее об этом.

# 7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске программы перед пользователем появляется окно авторизации (рисунок 7.1). Оно состоит из полей для ввода логина, пароля и кнопки Войти. Для повышения безопасности пользовательских данных, символы, введенные в поле для ввода, отображаются звёздочками.

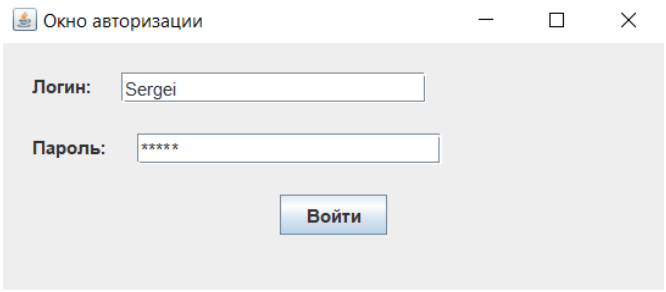


Рисунок 7.1 – Окно авторизации.

В случае, если пользователь с введенными данными не найден, появляется диалоговое окно, предлагающее пользователю зарегистрироваться (рисунок 7.2).

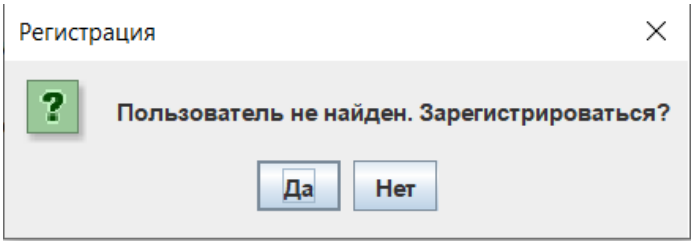


Рисунок 7.2 – Диалог регистрации.

После того, как пользователь ввел корректные данные, появляется диалоговое окно, уведомляющее пользователя об успешном входе в систему (рисунок 7.3).

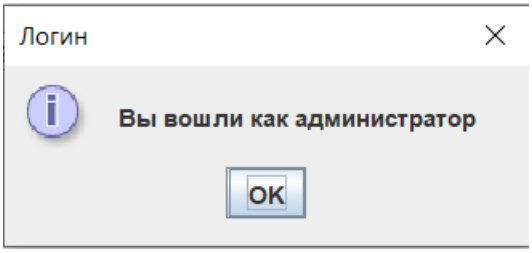


Рисунок 7.3 – Диалог успешного входа в систему.

В зависимости от того, является ли пользователь администратором системы или нет, ему предоставляется различный функционал. На рисунке 7.4 представлен функционал, которым обладает обычный пользователь. Он имеет возможность оценить недвижимость, просмотреть и редактировать информацию об уже оцененной недвижимости и просмотреть различные графики, лежащие в основе оценки недвижимости.

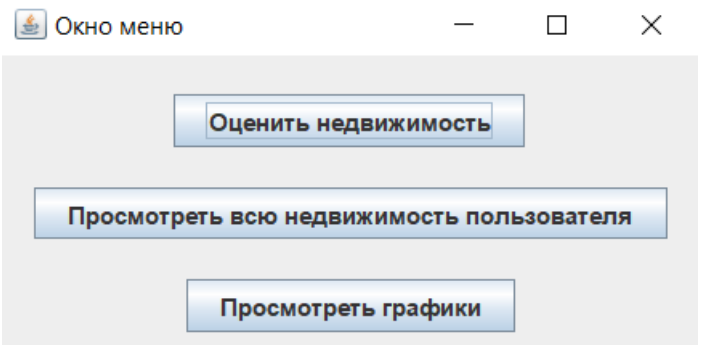


Рисунок 7.4 – Меню обычного пользователя.

Администратор обладает более расширенным функционалом, представленным на рисунке 7.5. Он дополнительно может просмотреть и редактировать информацию обо всех пользователях.

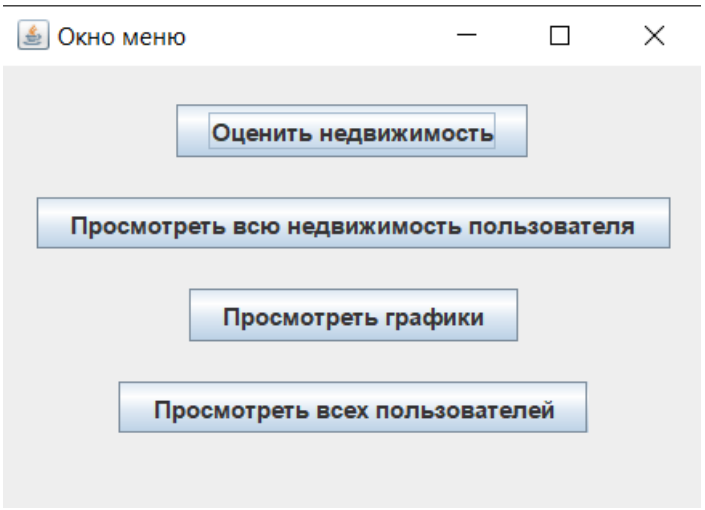


Рисунок 7.5 – Меню администратора системы.

По нажатию на кнопку Оценить недвижимость, открывается окно оценки недвижимости, представленное на рисунке 7.6.

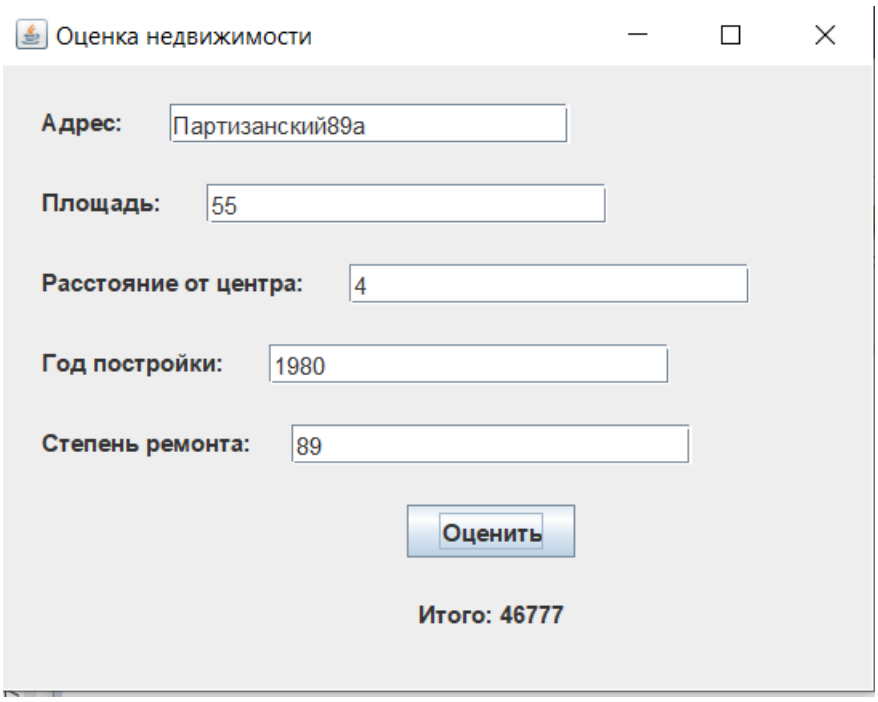


Рисунок 7.6 – Окно оценки стоимости недвижимости.

На окне оценки находятся текстовые поля для ввода данных о недвижимости, такие как адрес, площадь, расстояние от центра, год постройки, степень ремонта. По нажатию на кнопку Рассчитать на основе введённых данных рассчитывается стоимость недвижимости и данные сохраняются в базу данных, о чём свидетельствует диалоговое окно.

По нажатию на кнопку Показать всю недвижимость пользователя, открывается окно, представленное на рисунке 7.7.

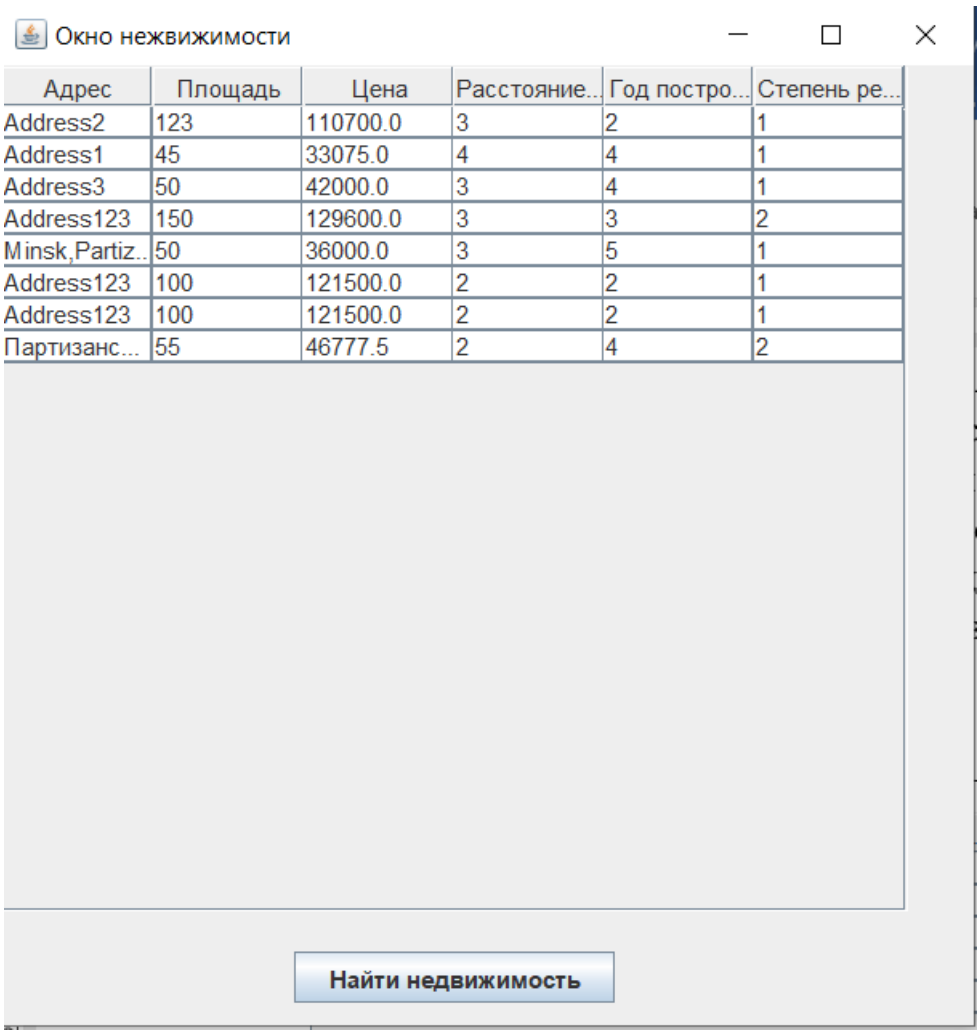


Рисунок 7.7 – Окно всей недвижимости пользователя.

Для удобной визуализации данные представлены в виде таблицы. Также в этом окне имеется возможность удалить недвижимость либо добавить и оценить новую недвижимость.

Внизу окна расположена кнопка Найти недвижимость. По нажатию на неё открывается диалог (рисунок 7.8), в который пользователю необходимо ввести необходимый адрес и на экране останется только недвижимость с введённым адресом.

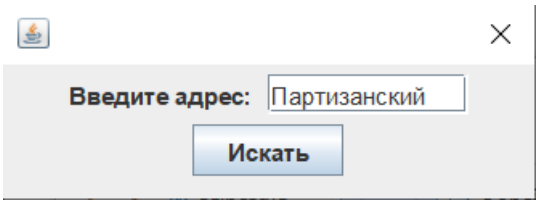


Рисунок 7.8 – Диалог поиска недвижимости по адресу.

По нажатию на кнопку Показать графики, открываются окна, содержащие диаграммы для более удобного анализа ситуации на рынке недвижимости.

На рисунке 7.9 представлена столбчатая диаграмма, отображающая количество недвижимости с различным расстоянием от центра.

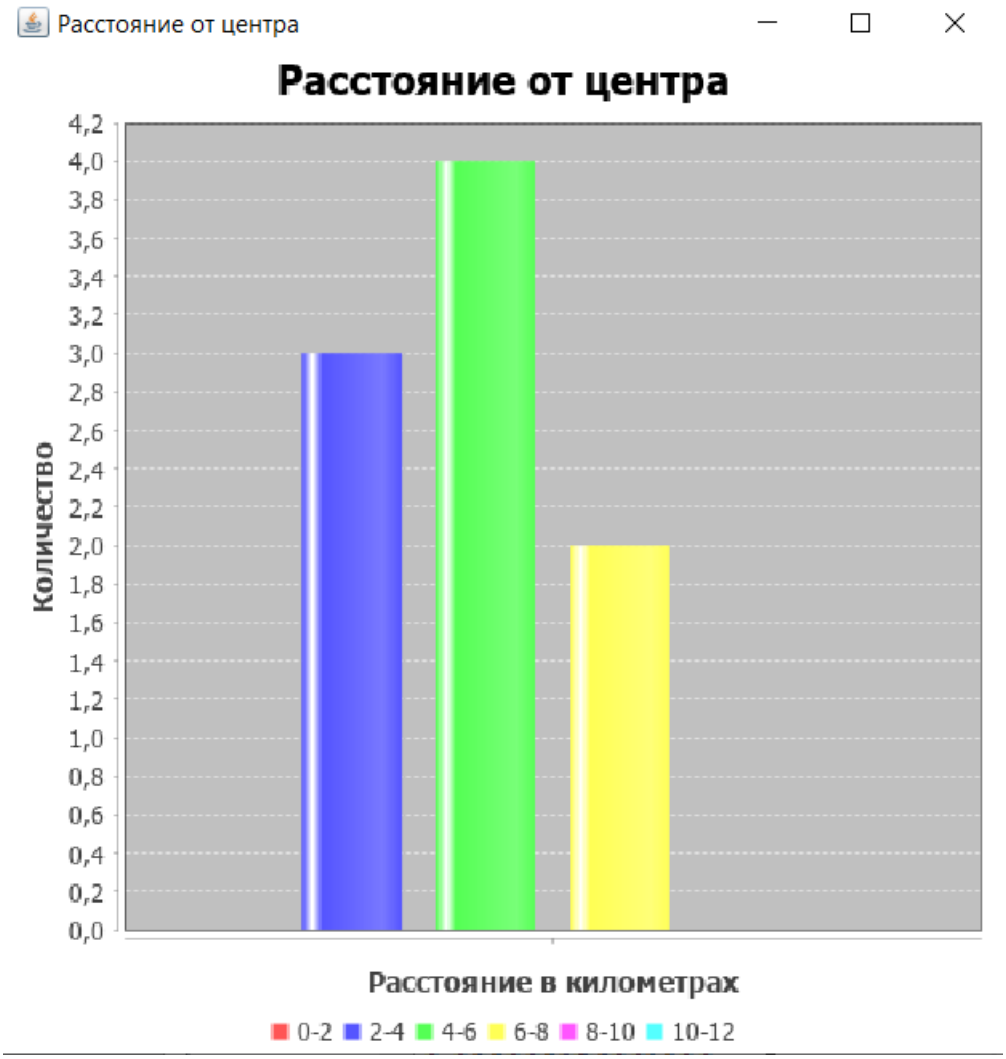


Рисунок 7.9 – Диаграмма отдалённости недвижимости от центра.

На рисунке 7.10 представлена круговая диаграмма, из которой удобно видно какая часть недвижимости, имеющаяся в базе данных, была построена в интересующий пользователя временной промежуток. На основе этого можно получить представление о ситуации на рынке недвижимости.

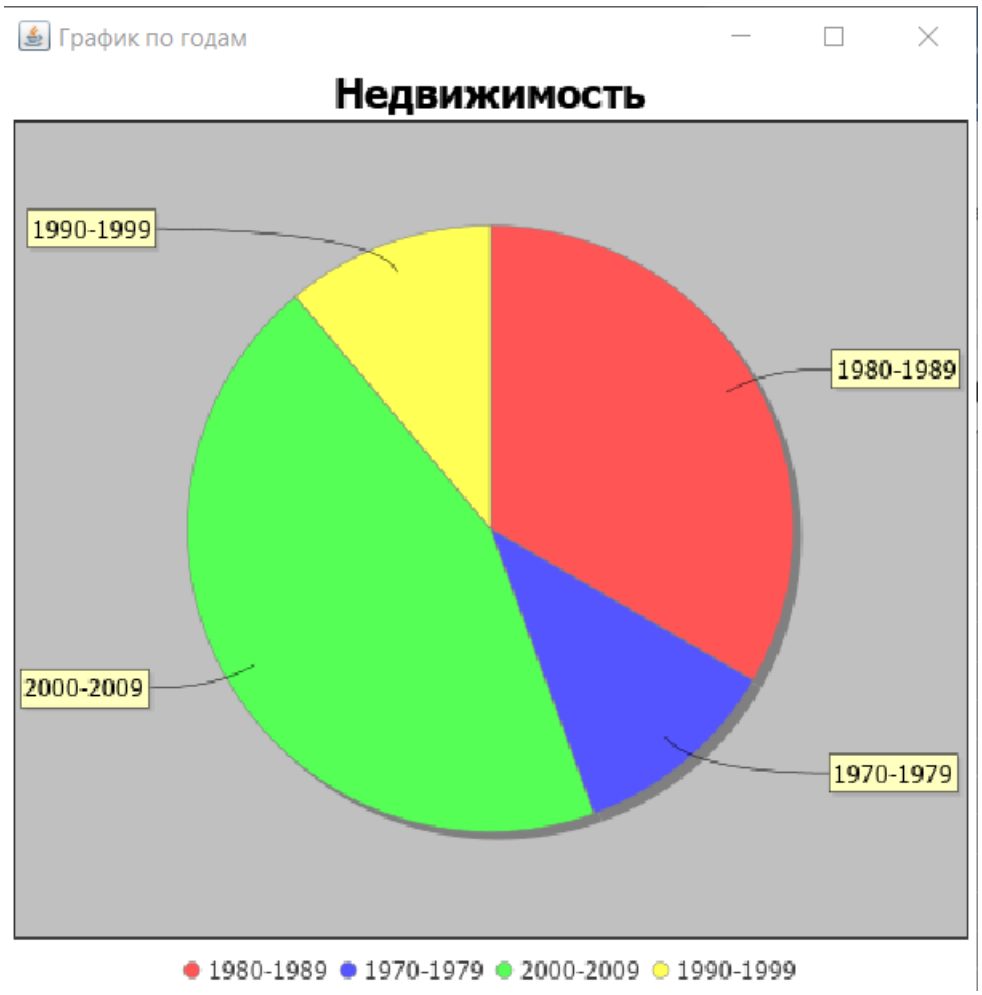


Рисунок 7.10 – Диаграмма недвижимости по году постройки.

Администратор системы обладает возможностью просматривать данные о пользователе. Для этого в меню необходимо нажать кнопку Показать пользователей и откроется окно, представленное на рисунке 7.11. Администратор имеет возможность удалить пользователя, если тот нарушает правила пользования системой.

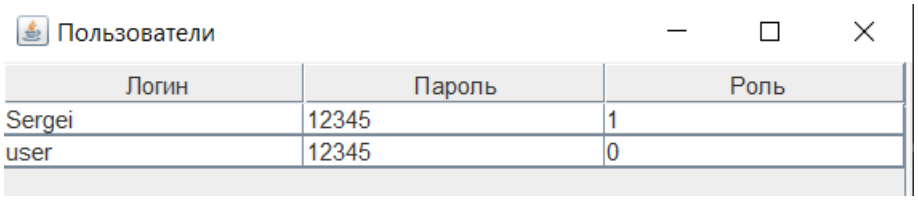


Рисунок 7.11 – Окно, отображающее всех пользователей системы.

# ЗАКЛЮЧЕНИЕ

В ходе работы над данным курсовым проектом была создана программа, позволяющая автоматизировать процесс оценки недвижимости. Все основные требования к программе были соблюдены, а именно:

– реализована возможность авторизации пользователя в системе;

– реализован особый функционал как для обычного пользователя, так и для администратора;

– реализована возможность оценки недвижимости и сохранения данных о ней в базе данных;

– реализована возможность редактирования уже имеющейся в базе недвижимости;

– реализована возможность графического отображения данных о недвижимости для сравнительного анализа;

– реализована возможность редактирования данных уже имеющихся пользователей в системе.

Данный курсовой проект имеет легко расширяемую клиент-серверную архитектуру, поэтому в будущем можно легко добавить новые возможности. Также можно сделать данное приложение веб-ориентированным, чтобы пользователь имел возможность произвести оценку в любой удобный для него момент. Помимо веб-приложения, можно сделать его приложением для мобильных устройств и загрузить в рынки приложений.

# СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

[1] Автоматизация процессов. – Электронные данные. – Режим доступа: https://www.kpms.ru/Automatization/Process_automation.htm.

[2] Оценка недвижимости – что это такое, для чего нужна оценка квартиры при ипотеке, как оценить рыночную стоимость. – Электронные данные. – Режим доступа: https://j.etagi.com/ps/ocenka-nedvizhimosti/.

[3] Построение диаграммы классов. – Электронные данные. – Режим доступа: https://flexberry.github.io/ru/gpg_class-diagram.html.

[4] Основы UML. Диаграммы последовательности – Блог программиста. – Электронные данные. – Режим доступа: https://pro-prof.com/archives/2769.

[5] Диаграмма компонентов – Википедия. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Диаграмма_компонентов.

[6] Диаграмма развёртывания. – Электронные данные. – Режим доступа: https://flexberry.github.io/ru/fd_deployment-diagram.html.

[7] Паттерны/шаблоны проектирования. – Электронные данные. – Режим доступа: https://refactoring.guru/ru/design-patterns.

[8] Одиночка (шаблон проектирования) – Википедия. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Одиночка.

# ПРИЛОЖЕНИЕ А
# (ОБЯЗАТЕЛЬНОЕ)
## Функциональная модель (IDEF0)

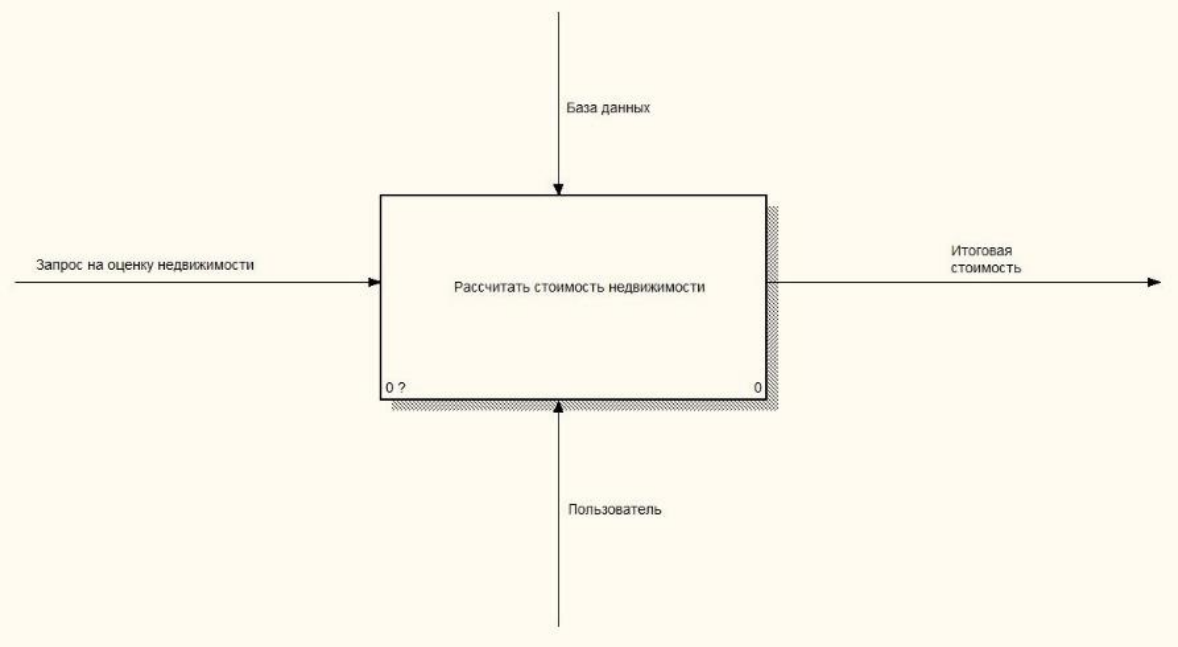


Рисунок А.1 – Бизнес-процесс расчёта стоимости недвижимости.

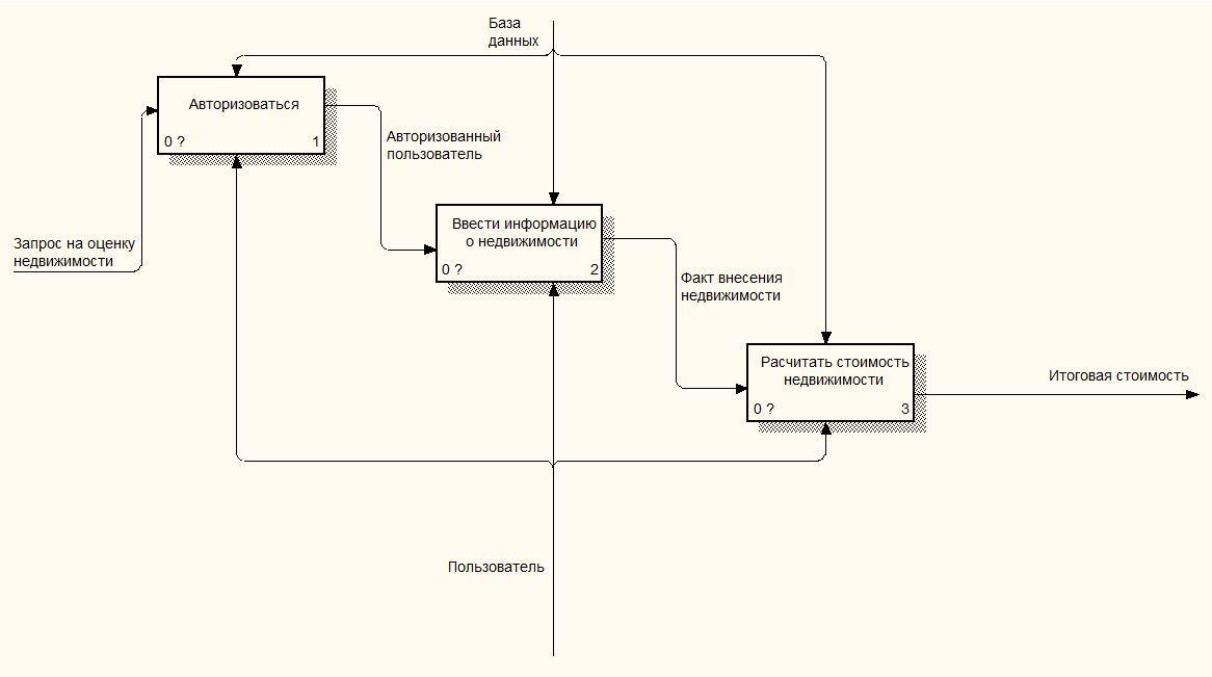


Рисунок А.2 – Декомпозиция бизнес-процесса расчета стоимости недвижимости.

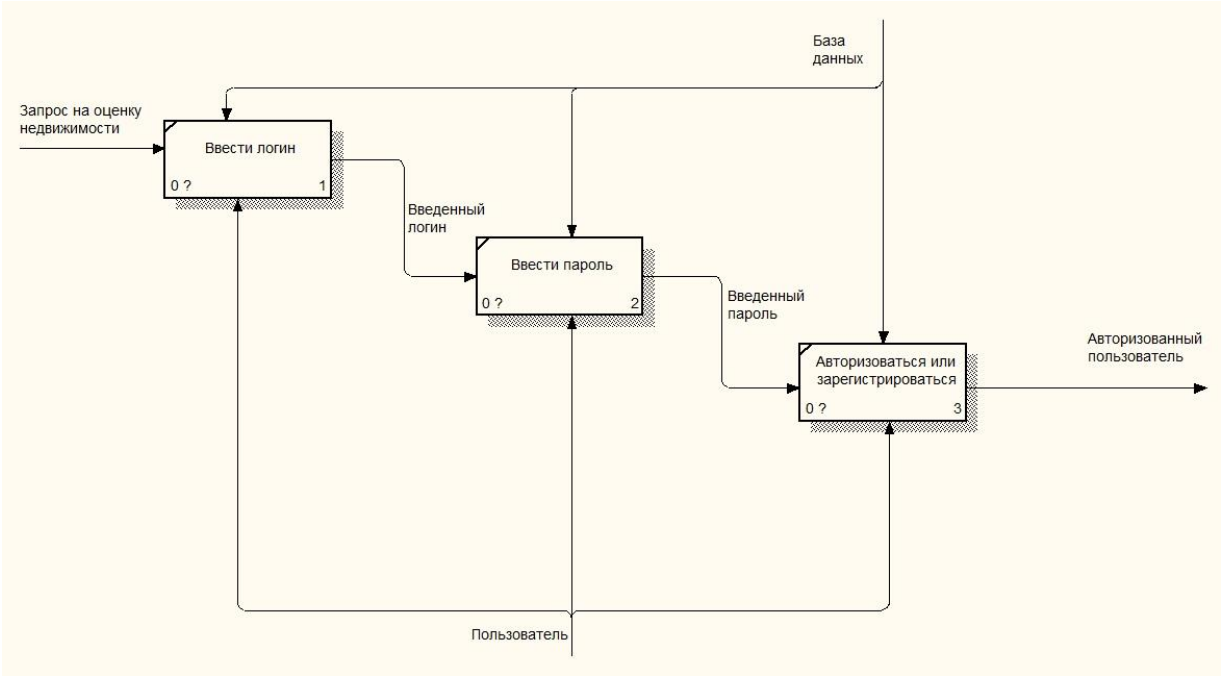


Рисунок А.3 – Декомпозиция блока «Авторизоваться».

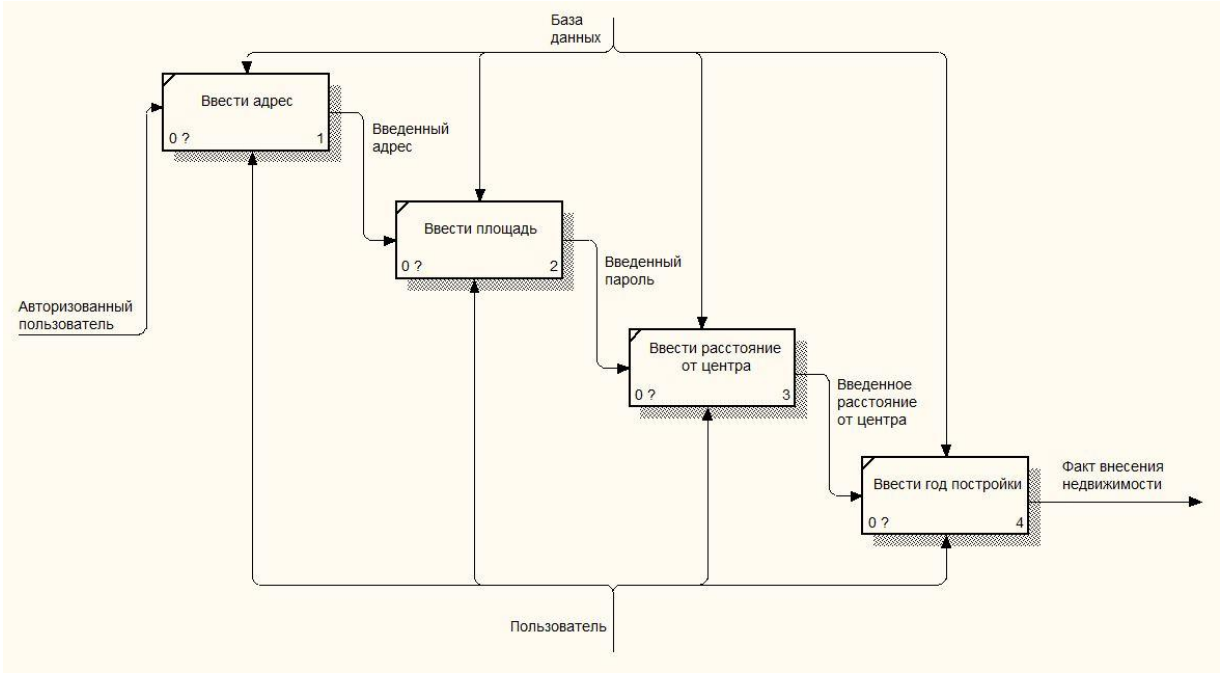


Рисунок А.4 – Декомпозиция блока «Ввести информацию о недвижимости».

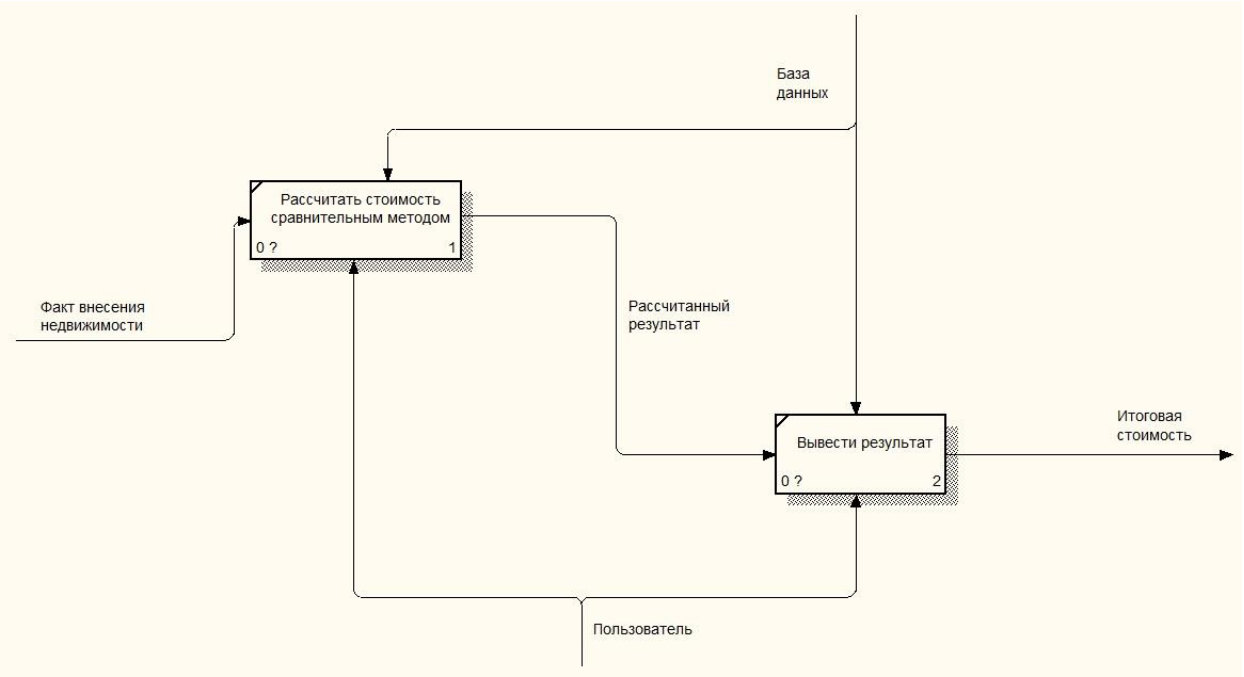


Рисунок А.5 – Декомпозиция блока «Рассчитать стоимость сравнительным методом».

# ПРИЛОЖЕНИЕ Б
# (ОБЯЗАТЕЛЬНОЕ)
## Листинг кода

Класс base.BaseController:

```
package base;

import client.ClientSocket;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public abstract class BaseController<T extends BaseView> {
    private Socket socket;
    protected T view;

    protected BaseController() {
        socket = ClientSocket.getSocket();
    }

    protected void sendDataToServer(String res) {
        try {
            OutputStream os = socket.getOutputStream();
            os.write(res.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    protected String getDataFromServer() {
        byte[] bytes = new byte[100];
        String str = null;
        try {
            InputStream is = socket.getInputStream();
            //noinspection ResultOfMethodCallIgnored
            is.read(bytes);
            str = new String(bytes, StandardCharsets.UTF_8);
            str = str.trim();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return str;
    }
```

```
    public void attachView(T view) {
        this.view = view;
    }
}
```

Класс controller.ChartsController:

```
package contoller;

import base.BaseController;
import constants.Actions;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import view.DistanceFromCenterBarChartWindow;
import view.YearPieChartWindow;

import java.util.HashMap;
import java.util.Map;

public class ChartsController extends BaseController {

    public void showCharts() {
        showYearPieChartWindow();
        showDistanceFromCenterBarChartWindow();
    }

    private void showDistanceFromCenterBarChartWindow() {
        new DistanceFromCenterBarChartWindow("Расстояние от центра",
getDistanceFromCenterBarChartData()).setVisible(true);
    }

    private void showYearPieChartWindow() {
        new YearPieChartWindow("График по годам",
getBuildYearData()).setVisible(true);
    }

    private CategoryDataset getDistanceFromCenterBarChartData() {
        final DefaultCategoryDataset dataset = new
DefaultCategoryDataset();
        final String distanceFromCenter = "";


sendDataToServer(Actions.GET_DISTANCE_FROM_SERVER_BAR_CHART_DATA);
        sendDataToServer(" ");
        int rows = Integer.parseInt(getDataFromServer());

        for (int i = 0; i < rows; i++) {
            String res = getDataFromServer();
```

```
            String[] arr = res.split(" ");
            dataset.addValue(Integer.parseInt(arr[0]),          arr[1],
distanceFromCenter);
        }

        return dataset;
    }

    private DefaultPieDataset getBuildYearData() {
        Map<String, Integer> data = new HashMap<>();
        DefaultPieDataset dataset = new DefaultPieDataset();
        int sum = 0;

        sendDataToServer(Actions.GET_BUILD_YEAR_CHART_DATA);
        sendDataToServer(" ");
        int rows = Integer.parseInt(getDataFromServer());

        for (int i = 0; i < rows; i++) {
            String res = getDataFromServer();
            String[] arr = res.split(" ");
            data.put(arr[1], Integer.parseInt(arr[0]));
            sum += Integer.parseInt(arr[0]);
        }

        for (Map.Entry<String, Integer> item : data.entrySet()) {
            if (item.getValue() != 0) {
                dataset.setValue(item.getKey(),                (double)
item.getValue() * 100 / sum);
            }
        }

        return dataset;
    }

}
```

Класс entities.Property:

```
package entities;

public class Property {
    private String address;
    private int square;
    private double price;
    private double distanceFromCenterCoefficient;
    private double buildYearCoefficient;
    private double repairDegreeCoefficient;
    private int userID;
```

```
    public Property(String address, int square, double price, double
distanceFromCenterCoefficient,
                    double         buildYearCoefficient,         double
repairDegreeCoefficient, int userID) {
        this.address = address;
        this.square = square;
        this.price = price;
        this.distanceFromCenterCoefficient                                =
distanceFromCenterCoefficient;
        this.buildYearCoefficient = buildYearCoefficient;
        this.repairDegreeCoefficient = repairDegreeCoefficient;
        this.userID = userID;
    }

    public Property(String data) {
        String[] arr = data.split(" ");
        address = arr[0];
        square = Integer.parseInt(arr[1]);
        price = Double.parseDouble(arr[2]);
        distanceFromCenterCoefficient = Double.parseDouble(arr[3]);
        buildYearCoefficient = Double.parseDouble(arr[4]);
        repairDegreeCoefficient = Double.parseDouble(arr[5]);
        userID = Integer.parseInt(arr[6]);
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getSquare() {
        return square;
    }

    public void setSquare(int square) {
        this.square = square;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public double getDistanceFromCenterCoefficient() {
```

```
        return distanceFromCenterCoefficient;
    }

    public          void          setDistanceFromCenterCoefficient(int
distanceFromCenterCoefficient) {
        this.distanceFromCenterCoefficient                                =
distanceFromCenterCoefficient;
    }

    public double getBuildYearCoefficient() {
        return buildYearCoefficient;
    }

    public void setBuildYearCoefficient(int buildYearCoefficient) {
        this.buildYearCoefficient = buildYearCoefficient;
    }

    public double getRepairDegreeCoefficient() {
        return repairDegreeCoefficient;
    }

    public void setRepairDegreeCoefficient(int repairDegreeCoefficient)
{
        this.repairDegreeCoefficient = repairDegreeCoefficient;
    }

    public int getUserID() {
        return userID;
    }

    public void setUserID(int userID) {
        this.userID = userID;
    }

    @Override
    public String toString() {
        return  address  +  "  "  +  square  +  "  "  +  price  +  "  "  +
distanceFromCenterCoefficient  +  "  "  +  buildYearCoefficient  +  "  "  +
repairDegreeCoefficient + " " + userID;
    }
}
```

Класс controllers.EvaluatePropertyController:

```
package contoller;

import base.BaseController;
import constants.Actions;
import entities.Property;
```

```
import view.EvaluatePropertyWindow;

public class EvaluatePropertyController extends
BaseController<EvaluatePropertyWindow> {
    private int id;

    public void setId(int id) {
        this.id = id;
    }

    public void calculatePrice(String address, int square, int
distanceFromCenter, int buildYear, int repairDegree) {
        double sum;

        sendDataToServer(Actions.GET_DISTANCE_FROM_CENTER);
        sendDataToServer(String.valueOf(distanceFromCenter));
        double distanceCoefficient =
Double.parseDouble(getDataFromServer());
        int distanceID = (int) Double.parseDouble(getDataFromServer());

        sendDataToServer(Actions.GET_BUILD_YEAR);
        sendDataToServer(String.valueOf(buildYear));
        double yearCoefficient =
Double.parseDouble(getDataFromServer());
        int yearID = (int) Double.parseDouble(getDataFromServer());

        sendDataToServer(Actions.GET_REPAIR_DEGREE);
        sendDataToServer(String.valueOf(repairDegree));
        double repairDegreeCoefficient =
Double.parseDouble(getDataFromServer());
        int repairID = (int) Double.parseDouble(getDataFromServer());

        sum = 1500 * square * distanceCoefficient * yearCoefficient *
repairDegreeCoefficient;

        Property property = new Property(address, square, sum,
distanceID, yearID, repairID, id);

        sendDataToServer(Actions.SAVE_PROPERTY);
        sendDataToServer(property.toString());
        String response = getDataFromServer();

        if (response.equalsIgnoreCase("Inserted")) {
            view.onDataCalculated(sum);
            view.onDataSaved();
        }
    }

}
```

# ПРИЛОЖЕНИЕ В
# (ОБЯЗАТЕЛЬНОЕ)
## Листинг скрипта генерации базы данных

```
create table buildyear
(
    id              int auto_increment
        primary key,
    bottomBorder    int    not null,
    topBorder       int    not null,
    yearCoefficient double not null
);

create table distancefromcenter
(
    id                  int auto_increment
        primary key,
    bottomBorder        int    not null,
    topBorder           int    not null,
    distanceCoefficient double not null
);

create table property
(
    id                   int auto_increment
        primary key,
    address              varchar(40) not null,
    square               int         not null,
    price                double      not null,
    distanceFromCenterID int         not null,
    buildYearID          int         not null,
    repairDegreeID       int         not null,
    userId               int         not null
);

create table repairdegree
(
    id                int auto_increment
        primary key,
    bottomBorder      int    not null,
    topBorder         int    not null,
    repairCoefficient double not null
);

create table user
(
    id       int auto_increment
        primary key,
```

```
    login    varchar(20) not null,
    password varchar(20) not null,
    role     int         not null
);
```

---

```
ALTER TABLE Property ADD CONSTRAINT fk_property_user ADD FOREIGN KEY
(userId) REFERENCES user(id)
ALTER TABLE Property ADD CONSTRAINT fk_property_repair_degree ADD
FOREIGN KEY (repairDegreeID) REFERENCES repairDegree(id)
ALTER TABLE Property ADD CONSTRAINT fk_build_year ADD FOREIGN KEY
(buildYearID) REFERENCES buildyear(id)
ALTER TABLE Property ADD CONSTRAINT fk_distance_from_center ADD FOREIGN
KEY (distanceFromCenterID) REFERENCES distancefromcenter(id)
```