СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc152321426)

[1 ЦЕЛИ И ЗАДАЧИ УЧЕБНОЙ ПРАКТИКИ 5](#_Toc152321427)

[1.1 Основные требования по технике безопасности во время прохождения учебной практики 5](#_Toc152321428)

[1.2 Нормативно-правовые документы 6](#_Toc152321429)

[2 ПРЕДМЕТНАЯ ОБЛАСТЬ. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ. ПОСТАНОВКА ЗАДАЧИ 8](#_Toc152321430)

[2.1 Описание предметной области 8](#_Toc152321431)

[2.2 Группы пользователей информационной системы 8](#_Toc152321432)

[3 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 11](#_Toc152321433)

[3.1 Моделирование некоторых динамических аспектов системы 12](#_Toc152321434)

[3.1.1 Создание диаграммы вариантов использования 13](#_Toc152321435)

[3.1.2 Создание диаграммы деятельности 15](#_Toc152321436)

[3.2 Разработка функциональной структуры информационной системы. Карта навигации по информационной системе 17](#_Toc152321437)

[3.3 Разработка макета информационной системы 17](#_Toc152321438)

[4 РАЗРАБОТКА ПРОТОТИПА ИНФОРМАЦИОННОЙ СИСТЕМЫ 30](#_Toc152321439)

[4.1 Моделирование и разработка базы данных 30](#_Toc152321440)

[4.2 Разработка программных модулей информационной системы 34](#_Toc152321441)

[5 РАЗРАБОТКА ТЕСТОВЫХ НАБОРОВ И ТЕСТОВЫХ СЦЕНАРИЕВ 80](#_Toc152321442)

[5.1 Назначение эксперимента. Выбор и обоснование методики проведения тестирования 80](#_Toc152321443)

[5.2 Технология тестирования 81](#_Toc152321444)

[5.3 Результаты проведения тестирования 82](#_Toc152321445)

[ЗАКЛЮЧЕНИЕ 85](#_Toc152321446)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 86](#_Toc152321447)

[ПРИЛОЖЕНИЕ А – Карты переходов пользователей 88](#_Toc152321448)

# ВВЕДЕНИЕ

В контексте динамичной сферы ломбардов, где обеспечение качественных финансовых решений играет важнейшую роль, ломбард сосредотачивается на предоставлении высококачественных услуг по залоговому кредитованию.

Широкое портфолио ломбарда включает не только стандартные категории, но и уникальные предметы, открывая возможности для разносторонних финансовых проектов.

Внедрение программного обеспечения позволит ломбардам автоматизировать множество рутинных операций, таких как оценка стоимости вещей, ведение договоров, расчет пени и сроков.

Целью практики является проектирование и разработка прикладного программного обеспечения деятельности ломбарда.

Основные задачи учебной практики:

* анализ предметной области;
* развить готовность выполнять поставленные профессиональные задачи;
* разработка программных модулей¸ оптимизированных для работы;
* тестирование разработанных программных модулей на соответствие функциональным требованиям.

Интерфейс программы обеспечит удобство для клиентов, ускорит процедуры оформления сделок, а также предоставит возможность отслеживания статуса своих договоров и заложенных вещей в режиме реального времени.

Встроенные аналитические инструменты помогут ломбардам проводить анализ эффективности работы, выявлять тренды в предоставлении займов, а также управлять финансовыми потоками.

Программное обеспечение обеспечит надежное хранение и защиту конфиденциальной информации клиентов и транзакций, соблюдая все стандарты безопасности данных.

В рамках модуля ПМ 01 Разработка модулей программного обеспечения для компьютерных систем, будет разработана и спроектирована информационная система “Продажа сортов семян”, которая будет включать в себя несколько видов профессиональных компетенций:

1. выполнять разработку спецификаций отдельных компонент;
2. осуществлять разработку кода программного продукта на основе готовых спецификаций на уровне модуля;
3. выполнять отладку программных модулей с использованием специализированных программных средств;
4. выполнять тестирование программных модулей;
5. осуществлять оптимизацию программного модуля;
6. разрабатывать компоненты проектной и технической документации с использованием графических языков спецификаций.

# 1 ЦЕЛИ И ЗАДАЧИ УЧЕБНОЙ ПРАКТИКИ

Целью практики является проектирование и разработка прикладного программного обеспечения деятельности ломбарда.

Основные задачи учебной практики:

* анализ предметной области;
* развить готовность выполнять поставленные профессиональные задачи;
* разработка программных модулей¸ оптимизированных для работы;
* тестирование разработанных программных модулей на соответствие функциональным требованиям.

## Основные требования по технике безопасности во время прохождения учебной практики

К работе в компьютерном классе допускаются лица, ознакомленные с данной инструкцией по технике безопасности и правилам поведения. Необходимо неукоснительно соблюдать правила по технике безопасности, т.к. нарушение этих правил может привести к поражению электрическим током, вызвать возгорание и навредить вашему здоровью.

- Работа обучающегося в компьютерном классе разрешается только в присутствии преподавателя (лаборанта).

- Во время занятий посторонние лица могут находиться в классе только с разрешения преподавателя.

- Во время перемен между уроками проводится обязательное проветривание компьютерного кабинета с обязательным выходом обучающегося из класса.

Обучающийся в ответе за состояние своего рабочего места и сохранность размещенного на нем оборудования.

Обучающийся, допустивший нарушение по технике безопасности и правилам поведения в компьютерном классе, может быть привлечен к дисциплинарной ответственности.

Если нарушение техники безопасности связано с причинением имущественного ущерба, обучающийся несет и материальную ответственность в установленном законом порядке.

## Нормативно-правовые документы

Оформление и содержание технического задания соответствует требованиям стандарта «ГОСТ 19.201-78. ЕСПД. Общие положения.

Техническое задание оформлено в соответствии с ГОСТ 19.106-78 на листах формата 11 и 12 по ГОСТ 2.301-68, как правило, без заполнения полей листа. Номера листов (страниц) проставлены в верхней части листа над текстом.

Лист утверждения и титульный лист оформлен в соответствии с ГОСТ 19.104-78.

Для внесения изменений или дополнений в техническое задание на последующих стадиях разработки программы или программного изделия выпускают дополнение к нему.

Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

Техническое задание должно содержать следующие разделы:

- введение;

- основания для разработки;

- назначение разработки;

- требования к программе или программному изделию;

- требования к программной документации;

- технико-экономические показатели;

- стадии и этапы разработки;

- порядок контроля и приемки;

- в техническое задание допускается включать приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

Вывод: когда были определены цели и задачи практики, путь для достижения результата стал намного проще, так же знание техники безопасности и нормативно-правовых документов создает надежную основу для прохождения практики

# ПРЕДМЕТНАЯ ОБЛАСТЬ. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ. ПОСТАНОВКА ЗАДАЧИ

Предметная область информационной системы представляет собой не только абстрактный концептуальный каркас, но и тесно связанную с реальностью материальную систему.

В этой сфере охватываются элементы физического мира, чья информация подлежит хранению и обработке. Анализируя предметную область, мы глубже погружаемся в контекст реальных объектов и их взаимосвязей.

Это, в свою очередь, становится фундаментом для разработки прикладного программного обеспечения, которое эффективно управляет и взаимодействует с этой сложной структурой информации.

В сущности, предметная область является ключевым строительным блоком информационной системы, обеспечивая ей не только логическую структуру, но и основу для реального воздействия на окружающий мир.

## Описание предметной области

Фирмой «Алмаз» был выдан заказ на создание прикладного программного обеспечения деятельности ломбарда.

В условиях современной экономики ломбарды играют важную роль в предоставлении финансовых услуг гражданам. Клиенты обращаются в ломбарды в поиске финансовой поддержки в ситуациях, требующих оперативного решения, таких как приобретение недвижимости или решение других финансовых затруднений.

Разработка прикладного программного обеспечения для ломбарда направлена на улучшение процессов залогового кредитования, оптимизацию учета заложенных вещей, а также повышение качества обслуживания клиентов.

## Группы пользователей информационной системы

Администратор - имеет полный доступ ко всей информационной системе. Может авторизоваться, просмотреть или изменить данные во всех таблицах. Также администратор может добавлять, удалять, изменять и просматривать учётные записи пользователей. Администратор занимается созданием клиентской и серверной части.

Приемщик - имеет ограниченные возможности в использовании информационной системы. Может авторизоваться, а также просматривать и изменять записи во всех таблицах кроме «User».

Клиент - ограничен в использованиях информационной системы, может только авторизоваться и просматривать данные из таблицы «Thing»

Основные требования, предъявляемые к информационной системе .

Гибкость:

* система должна обладать высокой степенью конфигурируемости, чтобы пользователи могли адаптировать её под различные потребности;
* возможность легкого масштабирования системы при увеличении объема данных или числа пользователей.

Выполнение этих условий возможно, если на этапе разработки информационной системы использовались общепринятые средства и методы документирования, так что по прошествии определенного времени сохранится возможность разобраться в структуре системы и внести в нее соответствующие изменения, даже если все разработчики или их часть по каким-либо причинам не смогут продолжить работу.

Надёжность:

* обеспечение бесперебойной работы системы с минимальным количеством сбоев и перерывов;
* регулярное создание резервных копий данных для предотвращения потери информации в случае сбоев.

Требование надежности обеспечивается созданием резервных копий хранимой информации, выполнения операций протоколирования, поддержанием качества каналов связи и физических носителей информации, использованием современных программных и аппаратных средств. Сюда же следует отнести защиту от случайных потерь информации в силу недостаточной квалификации персонала.

Эффективность:

* обеспечение высокой производительности при обработке данных, особенно в условиях одновременного использования множества пользователей;
* обеспечение высокой производительности при обработке данных, особенно в условиях одновременного использования множества пользователей.

В любом случае оценка эффективности будет производиться заказчиком, исходя из вложенных в разработку средств и соответствия представленной информационной системы его ожиданиям.

Безопасность:

* реализация надежной системы аутентификации для проверки личности пользователей и установление строгих прав доступа;
* использование шифрования для защиты передаваемой и хранимой чувствительной информации.

Защита информации от постороннего доступа обеспечивается управлением доступом к ресурсам системы, использованием современных программных средств защиты информации.

В крупных организациях целесообразно создавать подразделения, основным направлением деятельности которых было бы обеспечение информационной безопасности, в менее крупных организациях назначать сотрудника, ответственного за данный участок работы.

Вывод: основываясь на принципах гибкости, надежности, эффективности и безопасности, информационная система будет спроектирована таким образом, чтобы соответствовать ожиданиям заказчика, и обеспечевать высокий уровень обслуживания клиентов, с учетом требований всех групп пользователей, от администратора до клиента.

# ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «Ломбард»

Проектирование информационных систем представляет собой многоэтапный процесс создания или модернизации системы при помощи систематизированных методологий и инструментов. В отличие от моделирования, проектирование ориентировано на создание информационной системы, которая включает в себя:

* разработку программ, включая отчеты и экранные формы, обеспечивающие выполнение запросов к данным;
* учет функционирования конкретной среды или технологии.

Если рассматривать проектирование информационных систем как отдельный этап, его можно позиционировать между анализом и разработкой. Тем не менее, на практике четкое разграничение этапов часто затруднено или невозможно, поскольку проектирование формально начинается с определения целей проекта, но часто продолжается даже на этапах тестирования и реализации.

Процесс проектирования информационных систем включает в себя не только технические аспекты, но и стратегическое планирование, управление ресурсами и адаптацию к изменениям в окружающей среде. Он требует тщательного анализа требований пользователя, определения функциональности системы и выбора оптимальных технологических решений.

Одним из ключевых моментов является постоянное взаимодействие между различными этапами процесса проектирования. Например, изменения, внесенные на этапе разработки, могут потребовать корректировки на уровне анализа.

Важным аспектом также является обеспечение соответствия проектированной системы бизнес-целям и потребностям пользователей.

Таким образом, проектирование информационных систем – это сложный и взаимосвязанный процесс, требующий интеграции различных компонентов для достижения эффективности и успешной реализации проекта.

## Моделирование некоторых динамических аспектов системы

Диаграммы деятельности представляют собой эффективное средство моделирования динамических аспектов поведения системы. Они охватывают широкий спектр деятельности на различных уровнях абстракции в рамках системной архитектуры, включая классы (в том числе активные), интерфейсы, компоненты и узлы.

При моделировании системы диаграммы деятельности находят свое применение в различных контекстах. Вы можете использовать их для описания динамических аспектов почти любого элемента системы. Однако чаще всего они находят применение при рассмотрении всей системы, подсистемы, операции или класса.

Диаграмму деятельности можно интегрировать с вариантом использования для моделирования сценария или с кооперацией для отражения динамических взаимосвязей между объектами.

В области разработки программного обеспечения предварительное планирование и моделирование играют ключевую роль в упрощении процесса программирования. На этапе создания концептуальной модели для описания функционального назначения применяется диаграмма вариантов использования. Для детализации логики процедур и бизнес-процессов важными инструментами являются диаграммы деятельности.

Этот подход обеспечивает системный взгляд на разрабатываемое программное обеспечение и способствует более эффективному управлению проектом в целом. Таким образом, применение диаграмм деятельности в различных контекстах моделирования динамических аспектов поведения системы становится ключевым фактором в успешной разработке программного продукта.

### Создание диаграммы вариантов использования

Перед началом разработки информационной системы необходимо создать визуальное представление о вариантах использования разрабатываемой системы.

Диаграмма вариантов использования является концептуальным представлением системы в процессе ее проектирования и разработки. Диаграмма вариантов использования описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования.

Цели построения диаграммы вариантов использования:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования;
* сформулировать общие требования к функциональному проектированию системы;
* разработать исходную концептуальную модель системы для ее последующей реализации;
* подготовить документацию для взаимодействия разработчика системы с ее заказчиком и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых, вариантов использования. При этом актером или действующим лицом называется любая сущность, взаимодействующая с системой извне.

Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.

В свою очередь, вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером.

На диаграмме представлены три основных актера в информационной системе: администратор, приемщик и клиент. Каждый из них обладает определенными правами доступа, соответствующими их ролям и функциональности в системе. Диаграмма вариантов использования представлена на рисунке 1.

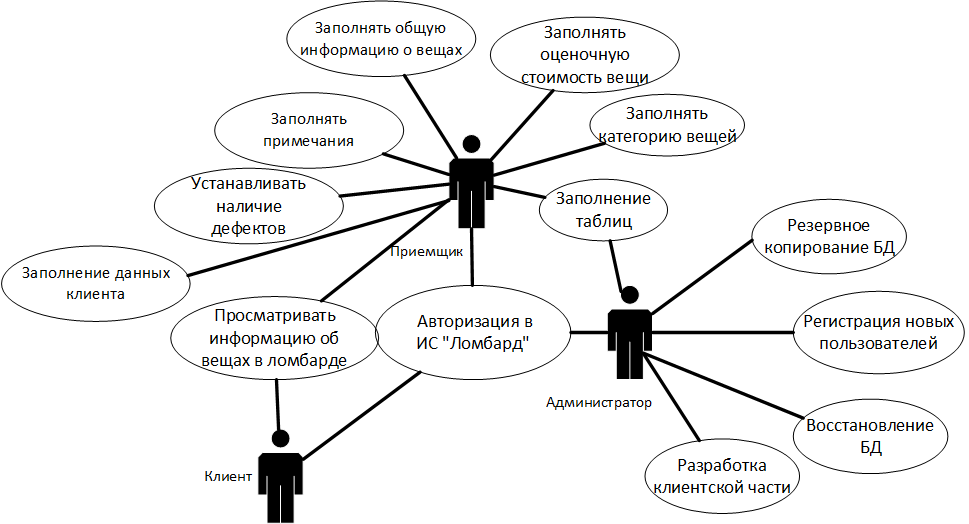


Рисунок 1 - Диаграмма вариантов использования

Администратор обладает полным доступом ко всем таблицам в системе. Его привилегии включают в себя просмотр, добавление, редактирование и удаление данных. Он также ответственен за управление пользователями, включая их добавление и удаление.

Приемщик имеет доступ ко всем таблицам, за исключением таблицы с пользователями и приемщиков. Ему разрешено добавлять новые записи, редактировать и удалять существующие данные. Он может работать с информацией о вещах, категориях и других соответствующих таблицах.

Клиент обладает ограниченными правами доступа и может только просматривать данные в таблице «Thing». Его функциональность ограничивается просмотром информации, без возможности внесения изменений.

Эта ролевая модель обеспечивает эффективное управление доступом и соответствует требованиям безопасности для каждой категории пользователей. Такой подход к управлению правами доступа в информационной системе обеспечивает эффективное использование ресурсов и предотвращает несанкционированный доступ к чувствительной информации.

### Создание диаграммы деятельности

При моделировании поведения системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для этого применяется диаграмма деятельности.

Диаграмма деятельности — диаграмма, описывающая динамические аспекты системы. Диаграмма деятельности UML позволяет более детально визуализировать конкретный случай использования. Это поведенческая диаграмма, которая иллюстрирует поток деятельности через систему.

Диаграмма деятельности администратора представлена на рисунке 2.

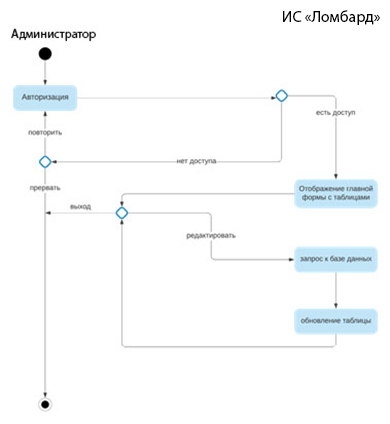


Рисунок 2 – Диаграмма деятельности администратора

Диаграмма деятельности приемщика представлена на рисунке 3.

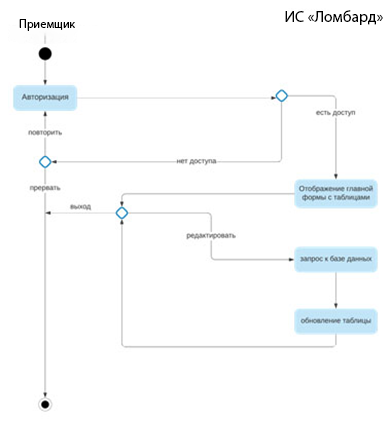


Рисунок 3 – Диаграмма деятельности приемщика

Диаграмма деятельности гостя представлена на рисунке 4.

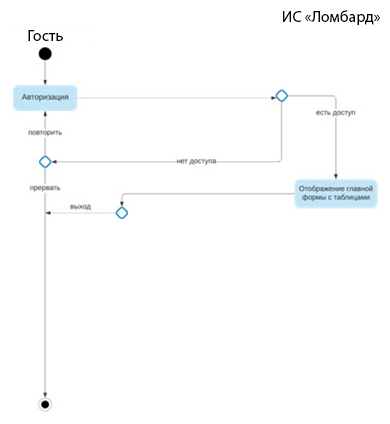


Рисунок 4 – Диаграмма деятельности гостя

Данная диаграмма деятельности показывает последовательность действий Администратора, необходимых для достижения той или иной цели.

## Разработка функциональной структуры информационной системы. Карта навигации по информационной системе

Принципы модульного программирования, аналогично нисходящему проектированию, начинаются с определения состава и подчиненности функций, после чего формируется набор программных модулей, реализующих указанные функции. Высокоуровневые функции осуществляются через главный модуль, который эффективно управляет выполнением более низкоуровневых функций, каждой из которых соответствует свой подчиненный модуль.

Эти принципы содействуют созданию структурированных, гибких и легко поддерживаемых программных систем, где модули служат строительными блоками, обеспечивающими четкость и эффективность разработки программного продукта.

Карты навигации для каждого из пользователей по информационной системе представлены в приложении A.

## Разработка макета информационной системы

Перед тем как приступить к проектированию информационной системы, важно подробно ознакомиться с элементами будущего интерфейса. Этот этап начинается с четкой постановки задачи, изучения предметной области, определения аудитории пользователей и понимания, каким образом они будут взаимодействовать с программным продуктом.

Пользовательский интерфейс представляет собой неотъемлемую часть продукта, определяющую визуальное и функциональное взаимодействие между пользователем и программой. Основная цель разработки интерфейса заключается в создании удобного, интуитивно понятного и максимально простого средства взаимодействия.

Эффективный пользовательский интерфейс требует качественного дизайна, который улучшает внешний облик информационной системы. Это включает в себя стратегическое использование шрифтов, цветовой гаммы и изображений, что способствует не только улучшению восприятия, но и создает привлекательность продукта.

Учитывая, что разрабатываемая информационная система рассчитана на широкий круг пользователей и предусматривает различные уровни доступа, обязательными компонентами являются механизмы авторизации и регистрации.

Окно авторизации представлено на рисунке 5.

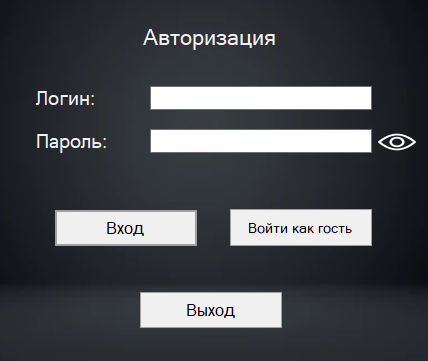


Рисунок 5 – Окно авторизации

В процессе авторизации пользователь имеет возможность ввести логин и пароль, что позволит ему продолжить взаимодействие с информационной системой. В том случае, если пользователь выступает в роли гостя, для входа ему предоставлена опция "Войти как гость". После выбора данной кнопки происходит автоматический переход на специальное окно, предназначенное для гостевого режима.

Этот механизм позволяет пользователям, не обладающим учетными данными, все равно получить доступ к определенным функциональным возможностям системы. При входе в режим гостя пользователю могут быть предложены ограниченные функции или ограниченный доступ к данным, сохраняя при этом удобство использования информационной системы.

Окно гостя представлено на рисунке 6.

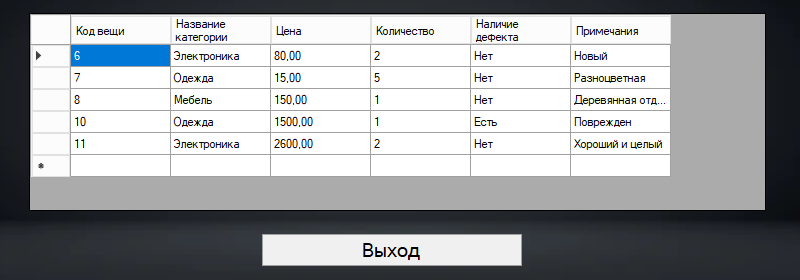


Рисунок 6 – Окно гостя

В случае, если введенные сотрудником данные не обнаружены в базе данных в качестве учетной записи сотрудника, пользователь не сможет выполнить процесс регистрации и, соответственно, не получит доступ к работе в информационной системе. Важно отметить, что данная система предназначена исключительно для сотрудников образовательной организации, что обеспечивает ее ориентированность на целевую аудиторию.

После успешной процедуры авторизации пользователь может продолжить взаимодействие с таблицами, представленными на главной форме, в зависимости от своей роли в системе. В случае, если пользователь авторизован как администратор, перед ним открывается специальная форма, предназначенная для администрирования и управления системой.

Это обеспечивает возможность осуществлять административные функции и управлять параметрами системы в соответствии с его привилегиями.

Форма администратора представлена на рисунке 7.

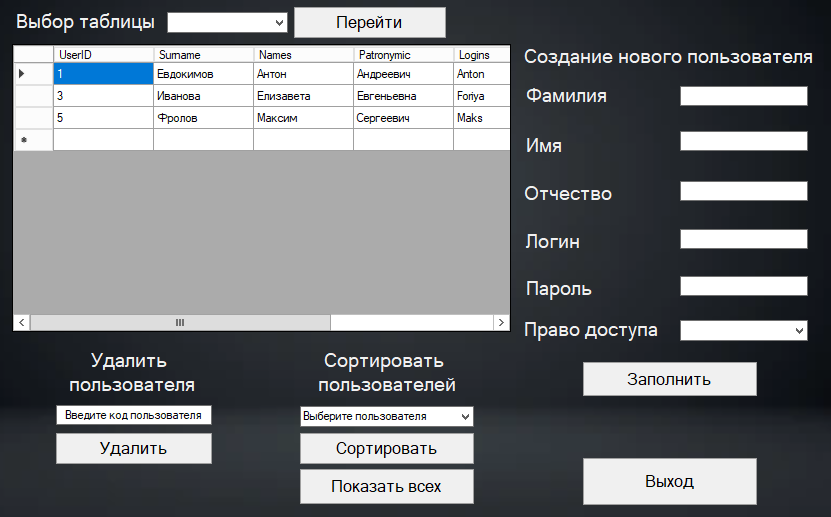


Рисунок 7 – Главная форма администратора

На представленной форме отображается таблица, содержащая информацию о существующих пользователях в базе данных. Кроме того, на форме присутствует удобный функционал, предоставляющий администратору возможность осуществлять различные действия с данными. Среди этих действий: переход по всем доступным таблицам при помощи выпадающего списка, добавление новых данных, а также редактирование и удаление уже существующих записей.

При выборе конкретной таблицы, администратор переносится в соответствующую таблицу для редактирования ее содержимого. Например, при выборе таблицы "Agreement", пользователь переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Agreement". Это обеспечивает администратору максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Таблица Agreement представлена на рисунке 8.

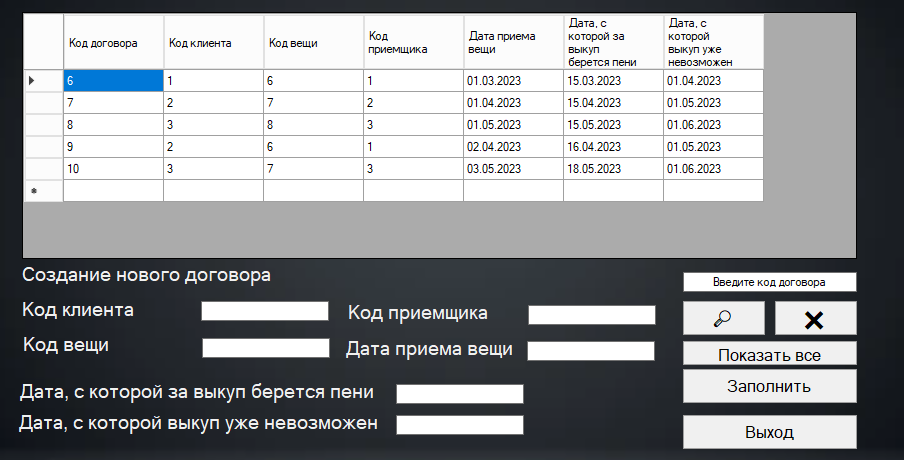


Рисунок 8 - Таблица Agreement

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду договора. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду договора или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Agreement".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

При выборе таблицы "Back", пользователь переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Back". Это обеспечивает администратору максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Таблица Back представлена на рисунке 9.

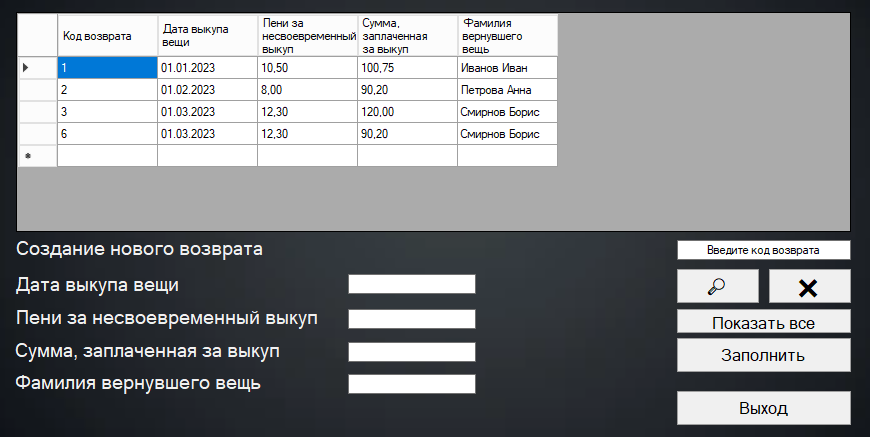


Рисунок 9 - Таблица Back

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду возврата. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду возврата или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Back".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

При выборе таблицы "Back", пользователь переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Back". Это обеспечивает администратору максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Таблица Category представлена на рисунке 10.

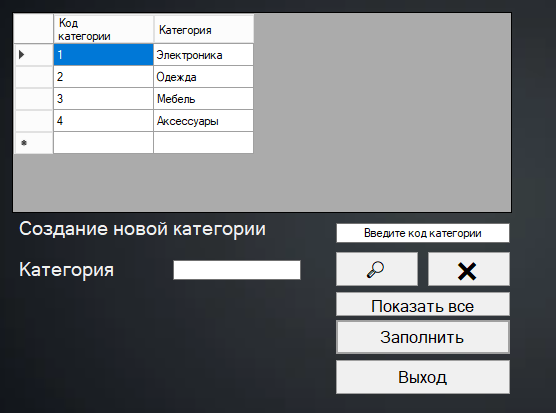


Рисунок 10 - Таблица Category

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду категории. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду категории или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Category".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

При выборе таблицы "Client", пользователь переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Client". Это обеспечивает администратору максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Таблица Client представлена на рисунке 11.

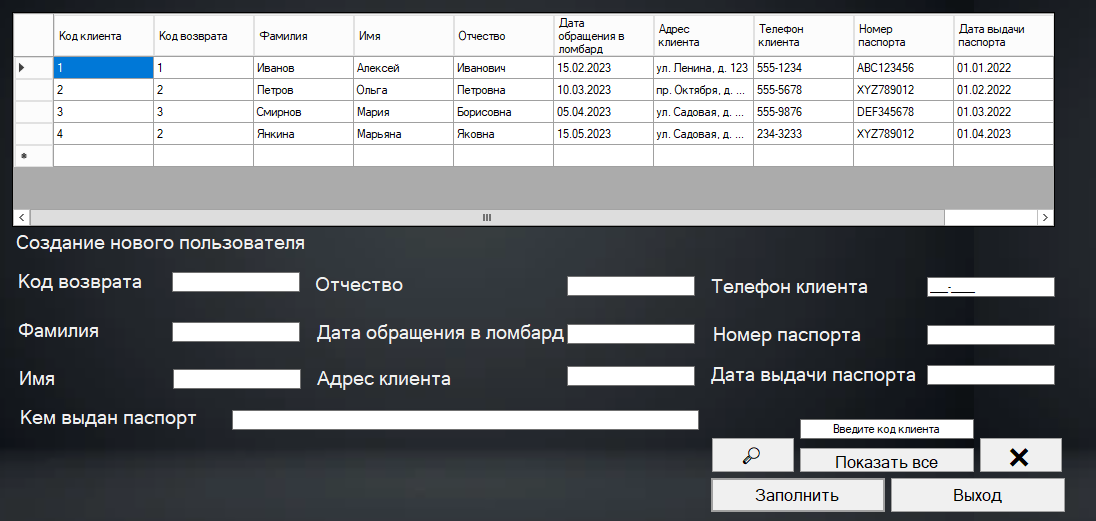


Рисунок 11 - Таблица Client

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду клиента. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду клиента или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Client".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

При выборе таблицы "Registrar", пользователь переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Registrar". Это обеспечивает администратору максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Таблица Registrar представлена на рисунке 12.

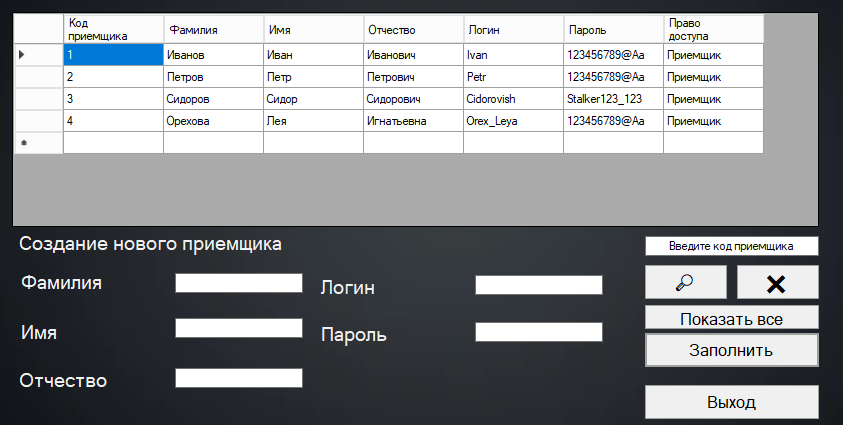


Рисунок 12 - Таблица Registrar

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду приемщика. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду приемщика или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Registrar".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

При выборе таблицы "Sale", пользователь переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Sale". Это обеспечивает администратору максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Таблица Sale представлена на рисунке 13.

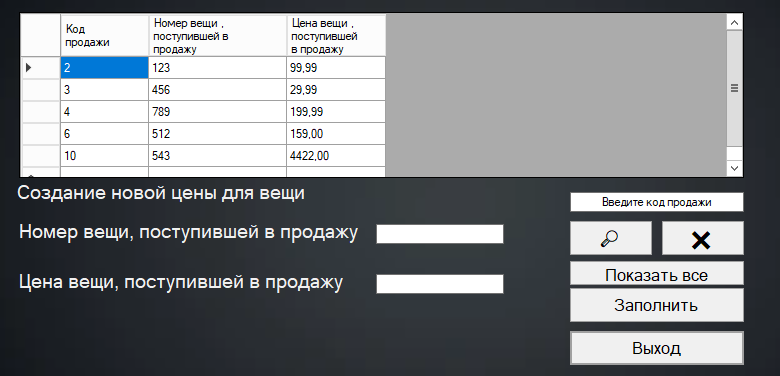


Рисунок 13 - Таблица Sale

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду продажи. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду продажи или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Sale".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

При выборе таблицы "Thing", пользователь переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Thing". Это обеспечивает администратору максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Таблица Thing представлена на рисунке 14.

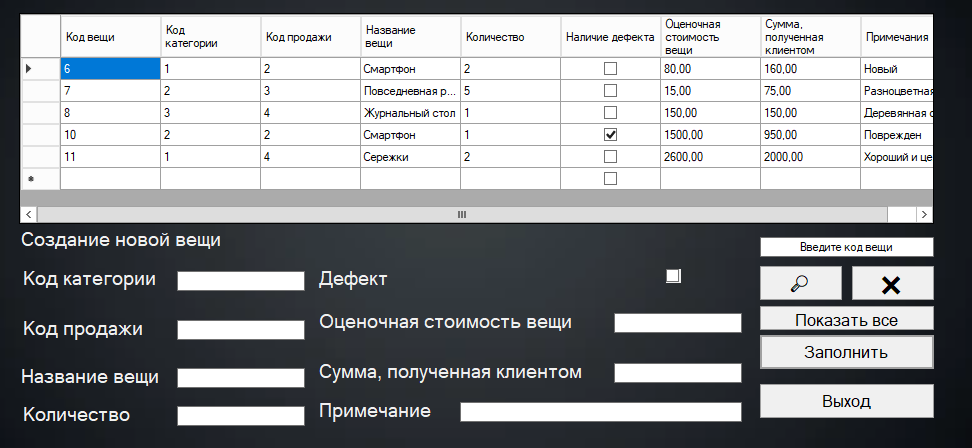


Рисунок 14- Таблица Thing

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду вещи. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду вещи или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Thing".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

Если пользователь производит авторизацию как приемщик, то он переходит на специальную форму с панелью управления, предназначенную для редактирования содержимого таблицы "Thing". Это обеспечивает приемщику максимальную гибкость и удобство в управлении данными в рамках различных таблиц базы данных.

Форма приемщика представлена на рисунке 15.

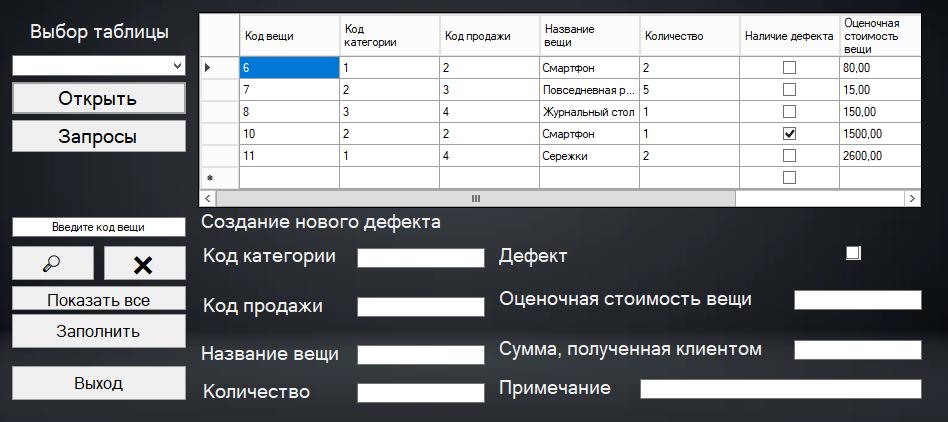


Рисунок 15 - Форма приемщика

На данной форме предоставлены удобные кнопки и текстовые поля, которые позволяют легко заполнить таблицу и выполнять необходимые операции. Пользователь может добавлять новые записи в таблицу, а также выполнять действия, такие как удаление и сортировка данных по коду вещи. Интуитивно понятный интерфейс формы обеспечивает эффективное взаимодействие с данными.

При необходимости сортировки данных по коду вещи или выполнения других операций, пользователями могут быть использованы соответствующие элементы управления, предоставленные на форме. Это позволяет администратору или другим пользователям системы легко управлять содержимым таблицы "Thing".

Кроме того, форма предусматривает кнопку "Выход", при нажатии на которую пользователь перемещается обратно на первоначальную форму. Этот механизм обеспечивает удобство в навигации и позволяет пользователям легко переключаться между различными разделами информационной системы.

Если пользователь нажимает на кнопку «Запросы» то его перемещает на форму с запросами, на которой он с помощью кнопок может выбрать необходимый запрос, и увидеть его на экране.

Форма с запросами представлена на рисунке 16.

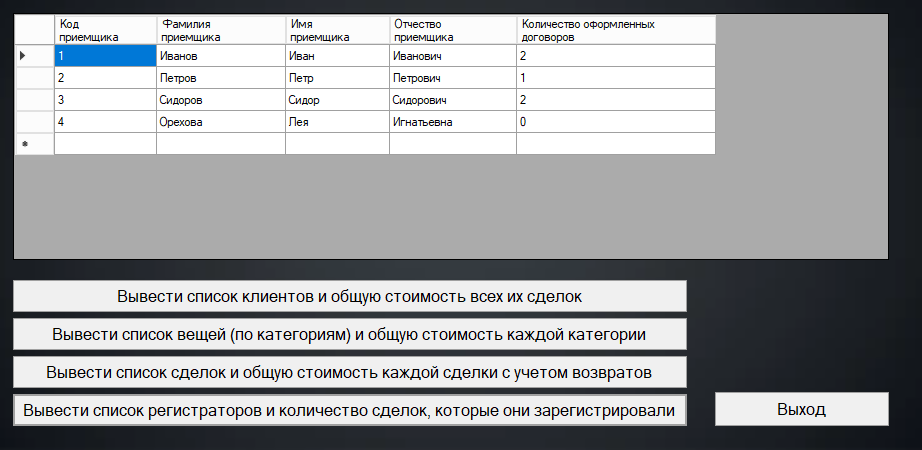


Рисунок 16 - Форма с запросами

Дизайн оформлен в минималистичном стиле с целью обеспечить удобство работы с программным обеспечением для пользователей всех уровней компьютерной грамотности.

Благодаря тщательному моделированию и изучению всех необходимых аспектов удалось разработать интерфейс информационной системы, который не только соответствует высоким стандартам, но и обеспечивает приятное и эффективное взаимодействие пользователя с системой.

Полученный результат свидетельствует о успешном совмещении функциональности с дизайном, создавая удобное и интуитивно понятное средство работы с информацией.

# РАЗРАБОТКА ПРОТОТИПА ИНФОРМАЦИОННОЙ СИСТЕМЫ «Ломбард»

Прототипирование представляет собой важный этап разработки, где акцент делается на структуре и расположении ключевых элементов интерфейса.

Прототип является визуальным моделированием конечного продукта, представляя собой интерактивный макет с разной степенью детализации.

Основная цель этого этапа — проверить последовательность действий пользователя и выявить возможные препятствия во взаимодействии с продуктом.

Прототипы не только предоставляют возможность оценить удобство использования разрабатываемого продукта до начала кодирования, но также способствуют обнаружению неожиданных моментов и вдохновляют новыми идеями.

Эти открытия могут поднять проектируемый продукт на новый уровень, создавая более инновационное и удовлетворяющее пользовательские потребности решение.

## Моделирование и разработка базы данных

Этап разработки базы данных начинается с анализа предметной области и моделирования логической схемы данных.

Этап проектирования логической модели данных включат в себя:

* определение состава и структуры таблиц;
* назначение ключевых полей;
* нормализация таблиц;
* установка связей между таблицами.

Исходя из анализа предметной области, можно выделить семь сущностей: «Back», «Agreement», «Category», «Client», «Registrar» , «Sale», «Thing» и «User».

На рисунке 17 представлена логическая модель данных.

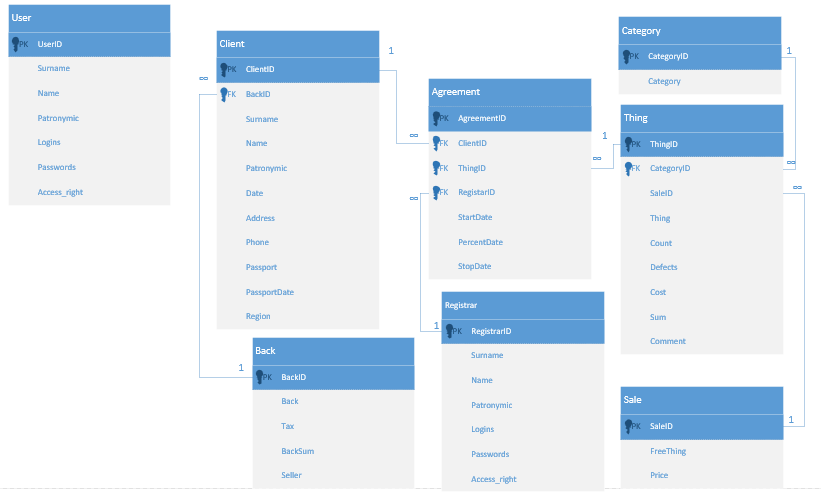


Рисунок 17 – Логическая модель данных

С помощью кода были созданы таблицы и успешно соединены между собой.

Названия полей и их типы данных представлены в таблицах 1-8.

Таблица 1 – Поля таблицы «User»

|  |  |
| --- | --- |
| Название поля | Тип данных |
| UserID | Счетчик |
| Surname | Текстовый |
| Names | Текстовый |
| Patronymic | Текстовый |
| Logins | Текстовый |
| Passwords | Текстовый |
| Access\_right | Текстовый |

Таблица 2 – Поля таблицы «Client»

| Название поля | Тип данных | |
| --- | --- | --- |
| ClientID | Счетчик |
| BackID | Числовой |
| Surname | Текстовый |
| Names | Текстовый |
| Patronymic | Текстовый |
| Dates | Дата/Время |
| Addres | Текстовый |
| Phone | Текстовый |
| Passport | Текстовый |
| PassportDate | Дата/Время |
| Region | Текстовый |

Таблица 3 – Поля таблицы «Back»

|  |  |
| --- | --- |
| Название поля | Тип данных |
| BackID | Счетчик |
| Back | Дата/Время |
| Tax | Денежный |
| BackSum | Денежный |
| Seller | Текстовый |

Таблица 4 – Поля таблицы «Registrar»

|  |  |
| --- | --- |
| Название поля | Тип данных |
| RegistrarID | Счетчик |
| Surname | Текстовый |
| Names | Текстовый |
| Patronymic | Текстовый |
| Logins | Текстовый |
| Passwords | Текстовый |
| Access\_right | Текстовый |

Таблица 5 – Поля таблицы «Category»

|  |  |
| --- | --- |
| Название поля | Тип данных |
| CategoryID | Счетчик |
| Category | Текстовый |

Таблица 6 – Поля таблицы «Sale»

|  |  |
| --- | --- |
| Название поля | Тип данных |
| SaleID | Счетчик |
| FreeThing | Текстовый |
| Price | Денежный |

Таблица 7 – Поля таблицы «Thing»

| Название поля | Тип данных |
| --- | --- |
| ThingID | Счетчик |
| CategoryID | Числовой |
| SaleID | Числовой |
| Thing | Текстовый |
| Counts | Текстовый |
| Defects | Логический |
| Cost | Денежный |
| Summ | Денежный |
| Comment | Текстовый |

Таблица 8 – Поля таблицы «Agreement»

|  |  |
| --- | --- |
| Название поля | Тип данных |
| AgreementID | Счетчик |
| ClientID | Числовой |
| ThingID | Числовой |
| RegistrarID | Числовой |
| StartDate | Дата/Время |
| PercentDate | Дата/Время |
| StopDate | Дата/Время |

Все созданные таблицы представлены на рисунке 18.

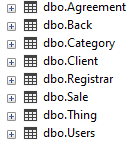


Рисунок 18 – Созданные таблицы

После того, как таблицы созданы, необходимо создать между ними связи.

Физическая модель данных отражающая все таблицы и их поля, а также связи между таблицами.

Физическая модель данных представлена на рисунке 19.

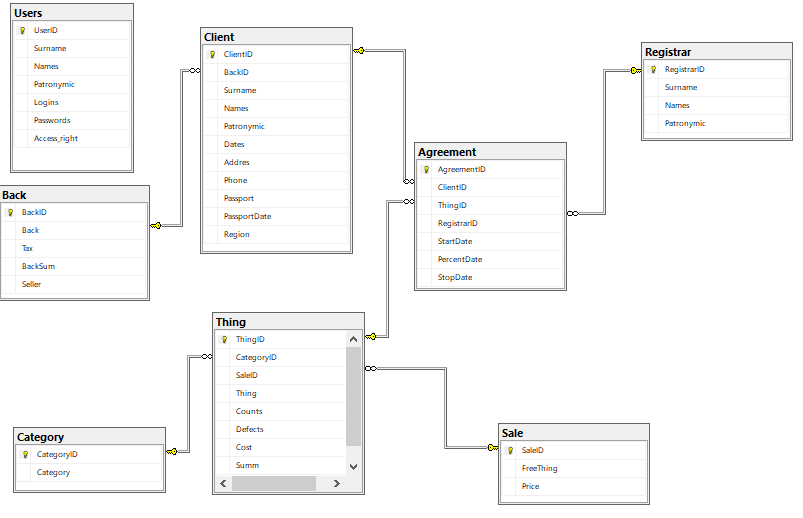


Рисунок 19 – Физическая модель базы данных

## Разработка программных модулей информационной системы

Модульное программирование — это организация программы как совокупности небольших блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок.

Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Принцип модульности является средством упрощения задачи проектирования программного обеспечения и распределения процесса разработки. При разбиении ПО на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями.

Удобство использования модульной архитектуры заключается в возможности обновления или замены модуля, без необходимости изменения остальной системы.

Окно авторизации представлено на рисунке 20.

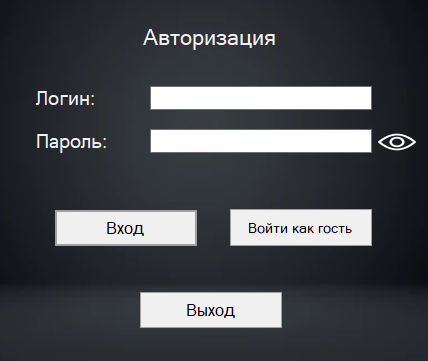


Рисунок 20 – Окно авторизации

Таблица 9 – Свойства элементов формы «Авторизация»

|  |  |  |
| --- | --- | --- |
| Объект | Свойство | Значение |
| authentication\_form | Name | Authentication\_form |
| BackColor | 144; 128; 239 |
| FormBorderStyle | None |
| Size | 428; 361 |
| authentication\_label | Font | Microsoft Sans Serif; 16pt |
| Text | Авторизация |
| ForeColor | ControlText |
| login\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Логин |
| ForeColor | ControlText |
| password\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Пароль |
| ForeColor | ControlText |
| login\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 222; 24 |
| password\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 222; 24 |
| login\_button | Size | 142; 36 |
| Text | Вход |
| Font | Microsoft Sans Serif; 12pt |
| exit\_button | Size | 142; 37 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| login\_guest\_button | Size | 142; 36 |
| Text | Войти как гость |
| Font | Microsoft Sans Serif; 12pt |
| open\_eye\_picturebox | Location | 378; 129 |
| SizeMode | StretchImage |
| open\_eye\_picturebox | Location | 378; 129 |
| SizeMode | StretchImage |

Таблица 10 – Событийно-управляемые процедуры в форме «Авторизация»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| login\_button | // Код для проверки символов, а также авторизация в систему  using System;  using System.Data;  using System.Data.SqlClient;  using System.Windows.Forms;  using System.Text.RegularExpressions;  namespace Серверподключ  {  public partial class Authentication\_form : Form  {  private int осталосьПопыток = 5;  public Authentication\_form()  {  InitializeComponent();  }  private void label3\_Click(object sender, EventArgs e)  {  }  private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)  {  password\_textbox.PasswordChar = '\*';  }  // Код для проверки символов , а так же авторизация в систему  private void login\_button\_Click(object sender, EventArgs e)  {  login\_button.FlatStyle = FlatStyle.Flat;  login\_button.FlatAppearance.BorderSize = 0;  System.Drawing.Drawing2D.GraphicsPath path = new System.Drawing.Drawing2D.GraphicsPath();  path.AddEllipse(0, 0, login\_button.Width, login\_button.Height);  if (осталосьПопыток <= 0)  {  MessageBox.Show("Вы исчерпали максимальное количество попыток входа. Пожалуйста, свяжитесь с администратором.");  this.Close();  return;  }  string введенныйЛогин = login\_textbox.Text.Trim();  string введенныйПароль = password\_textbox.Text;  if (string.IsNullOrWhiteSpace(введенныйЛогин) || string.IsNullOrWhiteSpace(введенныйПароль))  {  MessageBox.Show("Пожалуйста, заполните оба поля - имя пользователя и пароль.");  return;  }  using (SqlConnection sqlConnect = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  sqlConnect.Open();  SqlDataAdapter da = new SqlDataAdapter("select \* from Users", sqlConnect);  SqlDataAdapter dab = new SqlDataAdapter("select \* from Registrar", sqlConnect);  DataTable dt = new DataTable();  da.Fill(dt);  dab.Fill(dt);  Boolean flag = false;  string password = password\_textbox.Text;  // Проверка длины пароля (не менее 9 символов)  if (password.Length <= 9)  {  MessageBox.Show("Пароль должен содержать минимум 9 символов.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  if (!Regex.IsMatch(password, @"[a-zA-Z]"))  {  MessageBox.Show("Пароль должен содержать хотя бы одну латинскую букву.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  if (!Regex.IsMatch(password, @"[!@#\$]"))  {  MessageBox.Show("Пароль должен содержать хотя бы один специальный символ (!, @, #, $ и т. д.).", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  for (int i = 0; i < dt.Rows.Count; i++)  {  if ((dt.Rows[i]["Logins"].ToString() == login\_textbox.Text) &&  (dt.Rows[i]["Passwords"].ToString() == password\_textbox.Text))  {  string роль = dt.Rows[i]["Access\_right"].ToString();  switch (роль)  {  case "Администратор":  this.Hide();  admin\_form AdmFrm = new admin\_form();  AdmFrm.Show();  break;  case "Приемщик":  this.Hide();  Operator OperatorFrm = new Operator();  OperatorFrm.Show();  break;  default:  MessageBox.Show("Неправильный логин или пароль.");  break;  }  flag = true;  break;  }  }  if (!flag)  {  MessageBox.Show("Неправильный логин или пароль.");  осталосьПопыток--;  }  }  } |
| exit\_button | // выход из приложения  private void exit\_button\_Click(object sender, EventArgs e)  {  Application.Exit();  } |
| login\_guest\_button | // авторизация как гость  private void login\_guest\_button\_Click(object sender, EventArgs e)  {  this.Hide();  Guest GuestFrm = new Guest();  GuestFrm.Show();  } |
| Open\_eye\_picturebox | // картинка для показа пароля  private void open\_eye\_picturebox\_Click(object sender, EventArgs e)  {  if (password\_textbox.UseSystemPasswordChar)  {  password\_textbox.UseSystemPasswordChar = false;  open\_eye\_picturebox.Visible = true;  close\_eye\_picturebox.Visible = false;  password\_textbox.PasswordChar = '\*';  }  else  {  password\_textbox.UseSystemPasswordChar = true;  open\_eye\_picturebox.Visible = false;  password\_textbox.PasswordChar = '\0';  close\_eye\_picturebox.Visible = true;  }  } |
| close\_eye\_picturebox | // картинка для закрытия пароля  private void close\_eye\_picturebox\_Click(object sender, EventArgs e)  {  if (password\_textbox.UseSystemPasswordChar)  {  password\_textbox.UseSystemPasswordChar = false;  open\_eye\_picturebox.Visible = true;  close\_eye\_picturebox.Visible = false;  password\_textbox.PasswordChar = '\*';  }  else  {  password\_textbox.UseSystemPasswordChar = true;  open\_eye\_picturebox.Visible = false;  close\_eye\_picturebox.Visible = true;  password\_textbox.PasswordChar = '\0';  }  } |

Окно гостя представлено на рисунке 21.

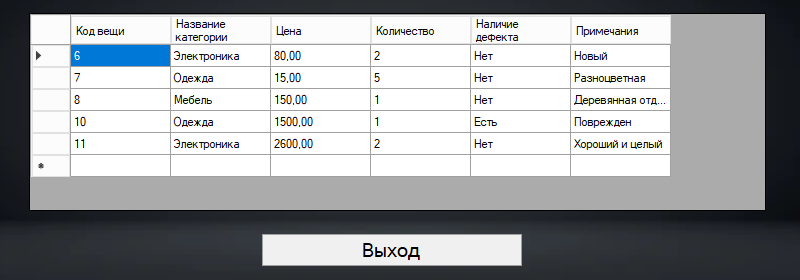


Рисунок 21 – Окно гостя

Таблица 11 – Свойства элементов формы «Guest»

|  |  |  |
| --- | --- | --- |
| Объект | Свойство | Значение |
| Guest | Name | Guest |
| FormBorderStyle | None |
| Size | 800; 280 |
| Datagridview1 | Size | 737; 198 |
| Borderstyle | FixedSingle |
| exit\_button | Size | 260; 32 |
| Text | Выход |
| Font | Microsoft Sans Serif; 14pt |

Таблица 12 – Событийно-управляемые процедуры в форме «Guest»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // выход из приложения  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  Authentication\_form frm = new Authentication\_form();  frm.Show();  } |
| Formload | // Подключаемся к базе и выводим таблицу при загрузке формы  private void Guest\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |

Главная форма администратора представлена на рисунке 22.

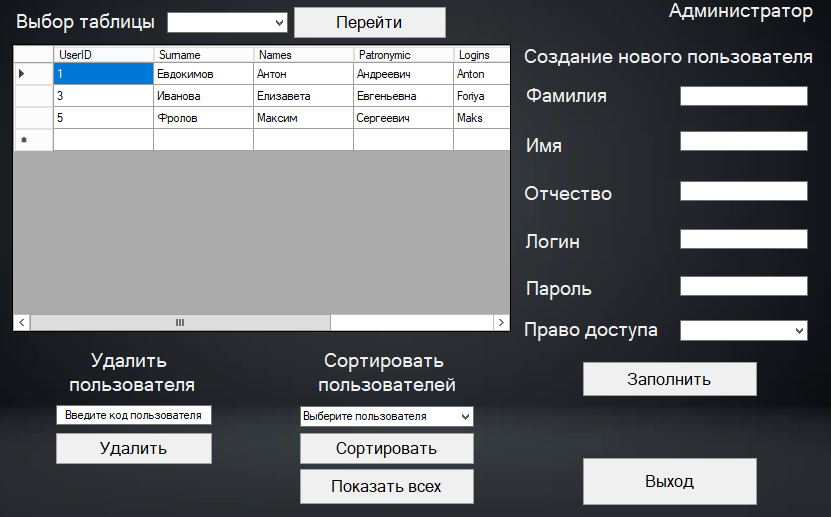


Рисунок 22 – Главная форма администратора

Таблица 13 – Свойства элементов формы «Admin»

| Объект | Свойство | Значение |
| --- | --- | --- |
| admin\_form | Name | admin\_form |
| FormBorderStyle | None |
| Size | 831; 517 |
| Datagridview1 | Size | 737; 198 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 156; 31 |
| Text | Удалить |
| Font | Microsoft Sans Serif; 12pt |
| exit\_button | Size | 174; 47 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| go\_button | Size | 152; 31 |
| Text | Перейти |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 174; 31 |
| Text | Сортировать |
| Font | Microsoft Sans Serif; 12pt |
| show\_all\_button | Size | 174; 35 |
| Text | Показать всех |
| Font | Microsoft Sans Serif; 12pt |
| surname\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| name\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| patronymic\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| login\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| password\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| delete\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| Text | Введите код пользователя |
| right\_access\_combobox | Font | Microsoft Sans Serif; 8,25pt |
| Items | Администратор  Пользователь  Приемщик |
| Text |  |
| choice\_user\_combobox | Font | Microsoft Sans Serif; 8,25pt |
| Items | Администратор  Приемщик  Пользователь |
| Text | Выберите пользователя |
| choice\_table\_combobox | Font | Microsoft Sans Serif; 8,25pt |
| Items | Agreement  Back  Category  Client  Registrar  Sale  Thing |
| Text |  |
| choice\_table\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Выбор таблицы |
| ForeColor | ControlText |
| name\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Имя |
| ForeColor | ControlText |
| patronymic\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Отчество |
| ForeColor | ControlText |
| login\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Логин |
| ForeColor | ControlText |
| password\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Пароль |
| ForeColor | ControlText |
| surname\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Фамилия |
| ForeColor | ControlText |
| right\_access\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Права доступа |
| ForeColor | ControlText |
| delete\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Удалить пользователя |
| ForeColor | ControlText |
| sort\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Сортировать пользователей |
| ForeColor | ControlText |
| create\_new\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание нового пользователя |
| ForeColor | ControlText |
| admin\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Администратор |
| ForeColor | ControlText |

Таблица 14 – Событийно-управляемые процедуры в форме «Admin»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // выход на форму авторизации  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  Authentication\_form frm = new Authentication\_form();  frm.Show();  } |
| Formload | // загружает метод который показывает таблицу пользователи в datagridview  private void Admin\_Load(object sender, EventArgs e)  {  LoadUsersData();  } |
| fill\_button | // Проверка поля пароля , так же создание подключения и передача запроса в базу , для создания нового пользователя  private void fill\_button\_Click(object sender, EventArgs e)  {  string password1 = password\_textbox.Text;  if (password1.Length <= 9)  {  MessageBox.Show("Пароль должен содержать минимум 9 символов.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  if (!Regex.IsMatch(password1, @"[a-zA-Z]"))  {  MessageBox.Show("Пароль должен содержать хотя бы одну латинскую букву.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  if (!Regex.IsMatch(password1, @"[!@#\$]"))  {  MessageBox.Show("Пароль должен содержать хотя бы один специальный символ (!, @, #, $ и т. д.).", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  txtSurname = surname\_textbox;  txtFirstName = name\_textbox;  txtPatronymic = patronymic\_textbox;  txtUsername = login\_textbox;  txtPassword = password\_textbox;  txtPravo = right\_access\_combobox;  string surname = txtSurname.Text;  string firstName = txtFirstName.Text;  string patronymic = txtPatronymic.Text;  string username = txtUsername.Text;  string password = txtPassword.Text;  string accessRights = txtPravo.Text;  if (string.IsNullOrEmpty(surname) || string.IsNullOrEmpty(firstName) ||  string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))  {  MessageBox.Show("Пожалуйста, заполните все поля.");  return;  }  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  string query = "INSERT INTO Users (Surname, Names, Patronymic, Logins, Passwords, Access\_right) " +  "VALUES (@Surname, @FirstName, @Patronymic, @Username, @Password, @AccessRights)";  using (SqlCommand command = new SqlCommand(query, connection))  {  command.Parameters.AddWithValue("@Surname", surname);  command.Parameters.AddWithValue("@FirstName", firstName);  command.Parameters.AddWithValue("@Patronymic", patronymic);  command.Parameters.AddWithValue("@Username", username);  command.Parameters.AddWithValue("@Password", password);  command.Parameters.AddWithValue("@AccessRights", accessRights);  try  {  connection.Open();  command.ExecuteNonQuery();  MessageBox.Show("Пользователь успешно зарегистрирован.");  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при регистрации пользователя: " + ex.Message);  }  }  }  surname\_textbox.Text = "";  name\_textbox.Text = "";  patronymic\_textbox.Text = "";  login\_textbox.Text = "";  password\_textbox.Text = "";  delete\_textbox.Text = "";  right\_access\_combobox.Text = "";  LoadUsersData();  } |
| sort\_button | // с помощью запроса сортирует пользователь по коду, и выводит в datagrid  private void sort\_button\_Click(object sender, EventArgs e)  {  string accessRights = choice\_user\_combobox.Text;  if (string.IsNullOrEmpty(accessRights))  {  MessageBox.Show("Введите права доступа для поиска пользователей.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  string query = "SELECT \* FROM Users WHERE Access\_right = @AccessRights";  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  using (SqlCommand command = new SqlCommand(query, connection))  {  command.Parameters.AddWithValue("@AccessRights", accessRights);  try  {  connection.Open();  SqlDataAdapter adapter = new SqlDataAdapter(command);  DataTable resultTable = new DataTable();  adapter.Fill(resultTable);  if (resultTable.Rows.Count > 0)  {  dataGridView1.DataSource = resultTable;  MessageBox.Show("Найдены пользователи с правами доступа: " + accessRights, "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);  }  else  {  dataGridView1.DataSource = null;  MessageBox.Show("Пользователи с правами доступа: " + accessRights + " не найдены.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при поиске пользователей: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  }  } |
| Show\_all\_button | // вызывет метод, который обновляет datagrid  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Delete\_button | // с помощью запроса удаляет пользователя по коду, а так же сразу обновляет datagrid  private void delete\_button\_Click(object sender, EventArgs e)  {  string userCode = delete\_textbox.Text;  if (string.IsNullOrEmpty(userCode))  {  MessageBox.Show("Введите код пользователя для удаления.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }  string query = "SELECT \* FROM Users WHERE UserID = @UserCode";  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  using (SqlCommand command = new SqlCommand(query, connection))  {  command.Parameters.AddWithValue("@UserCode", userCode);  try  {  connection.Open();  SqlDataReader reader = command.ExecuteReader();  if (reader.Read())  {  reader.Close();  string deleteQuery = "DELETE FROM Users WHERE UserID = @UserCode";  using (SqlCommand deleteCommand = new SqlCommand(deleteQuery, connection))  {  deleteCommand.Parameters.AddWithValue("@UserCode", userCode);  deleteCommand.ExecuteNonQuery();  }  LoadUsersData();  MessageBox.Show("Пользователь с кодом " + userCode + " был удален.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);  }  else  {  MessageBox.Show("Пользователь с кодом " + userCode + " не найден.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при удалении пользователя: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  }  } |
| Delete\_textbox | // очищает текст когда пользователь нажимает на поле  private void delete\_textbox\_Click(object sender, EventArgs e)  {  delete\_textbox.Text = "";  } |
| loaduserdata | // метод обновления datagridview  public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT \* FROM Users";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |
| Admin\_form | public admin\_form()  {  InitializeComponent();  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  } |

Форма таблицы «Agreement» представлена на рисунке 23.

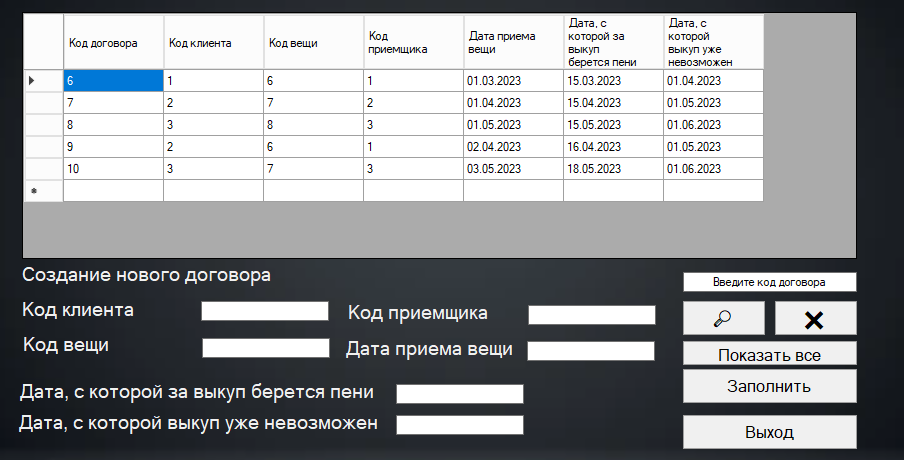


Рисунок 23 – Форма таблицы «Agreement»

Таблица 15 – Свойства элементов формы «Agreement»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Agreement | Name | Agreement |
| FormBorderStyle | None |
| Size | 904; 460 |
| Datagridview1 | Size | 835; 247 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 82; 34 |
| Text | ✖ |
| Font | Microsoft Sans Serif; 18pt |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 82; 34 |
| Text | 🔎 |
| Font | Microsoft Sans Serif; 14pt |
| show\_all\_button | Size | 174; 24 |
| Text | Показать все |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| clientID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| thingID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| registrarID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| startDate\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| percentDate\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| stopDate\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| agreementID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код договора |
| registrarID\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Код приемщика |
| ForeColor | ControlText |
| thingID\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Код вещи |
| ForeColor | ControlText |
| percentDate\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Дата, с которой за выкуп берется пени |
| ForeColor | ControlText |
| stopDate\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Дата, с которой выкуп уже невозможен |
| ForeColor | ControlText |
| startDate\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Дата приема вещи |
| ForeColor | ControlText |
| clienID\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Код клиента |
| ForeColor | ControlText |
| createAgreement\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание нового договора |
| ForeColor | ControlText |

Таблица 16 – Событийно-управляемые процедуры в форме «Admin»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрытие формы  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void Agreement\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| fill\_button | // читает textbox, и с помощью запроса заполняет их в базу данных  private void fill\_button\_Click(object sender, EventArgs e)  {  // Получите значения из TextBox'ов  int clientID = int.Parse(clientID\_textbox.Text);  int thingID = int.Parse(thingID\_textbox.Text);  int registrarID = int.Parse(registrarID\_textbox.Text);  DateTime startDate = DateTime.Parse(startDate\_textbox.Text);  DateTime percentDate = DateTime.Parse(percentDate\_textbox.Text);  DateTime stopDate = DateTime.Parse(stopDate\_textbox.Text);  string insertQuery = "INSERT INTO Agreement (ClientID, ThingID, RegistrarID, StartDate, PercentDate, StopDate) " +  $"VALUES ({clientID}, {thingID}, {registrarID}, '{startDate}', '{percentDate}', '{stopDate}')";  using (SqlConnection connection = new SqlConnection(ConnectionString))  {  connection.Open();  using (SqlCommand command = new SqlCommand(insertQuery, connection))  {  command.ExecuteNonQuery();  }  }  LoadUsersData();  clientID\_textbox.Clear();  thingID\_textbox.Clear();  registrarID\_textbox.Clear();  startDate\_textbox.Clear();  percentDate\_textbox.Clear();  stopDate\_textbox.Clear();  } |
| Show\_all\_button | // запускает метод который обновляет datagridview  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Sort\_button | // с помощью запроса сортирует договора по коду договора  private void sort\_button\_Click(object sender, EventArgs e)  {  int AgreementID = int.Parse(agreementID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT AgreementID as [Код договора], ClientID as [Код клиента], ThingID as [Код вещи], RegistrarID as [Код приемщика], StartDate as [Дата приема вещи],PercentDate as [Дата, с которой за выкуп берется пени],StopDate as [Дата, с которой выкуп уже невозможен] FROM Agreement WHERE AgreementID = '{AgreementID}'";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  } |
| Delete\_button | // с помощью запроса удаляет договор ко коду договора, и обновляет datagridbiew  private void delete\_button\_Click(object sender, EventArgs e)  {  int AgreementID = int.Parse(agreementID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"DELETE FROM Agreement WHERE AgreementID = '{AgreementID}'";  SqlCommand command = new SqlCommand(deleteQuery, connection);  command.ExecuteNonQuery();  dataGridView1.DataSource = dataTable;  agreementID\_textbox.Text = "";  LoadUsersData();  }  } |
| agreementID\_textbox | // очищает textbox по нажатию на него  private void agreementID\_textbox\_Click(object sender, EventArgs e)  {  agreementID\_textbox.Text = "";  } |
| Loaduserdata | // метод обновления datagridview  public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT AgreementID as [Код договора], ClientID as [Код клиента], ThingID as [Код вещи], RegistrarID as [Код приемщика], StartDate as [Дата приема вещи],PercentDate as [Дата, с которой за выкуп берется пени],StopDate as [Дата, с которой выкуп уже невозможен] FROM Agreement";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |

Форма таблицы «Back» представлена на рисунке 24.

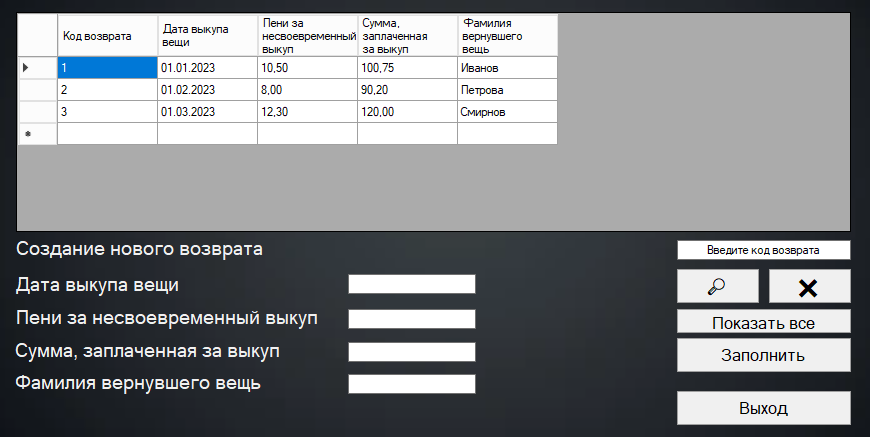


Рисунок 24 – Форма таблицы «Back»

Таблица 17 – Свойства элементов формы «Back»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Back | Name | Back |
| FormBorderStyle | None |
| Size | 870; 437 |
| Datagridview1 | Size | 835; 220 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 82; 34 |
| Text | ✖ |
| Font | Microsoft Sans Serif; 18pt |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 82; 34 |
| Text | 🔎 |
| Font | Microsoft Sans Serif; 14pt |
| show\_all\_button | Size | 174; 24 |
| Text | Показать все |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| back\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| tax\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| backSum\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| seller\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| backID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код возврата |
| tax\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Пени за несвоевременный выкуп |
| ForeColor | ControlText |
| backSum\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Сумма, заплаченная за выкуп |
| ForeColor | ControlText |
| seller\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Фамилия вернувшего вещь |
| ForeColor | ControlText |
| back\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Дата выкупа вещи |
| ForeColor | ControlText |
| createBack\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание нового возврата |
| ForeColor | ControlText |

Таблица 18 – Событийно-управляемые процедуры в форме «Admin»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрытие формы  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void Back\_Load(object sender, EventArgs e)  {  this.backTableAdapter.Fill(this.pawnshopDataSet2.Back);  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| fill\_button | // читает textbox, и с помощью запроса заполняет их в базу данных  private void fill\_button\_Click(object sender, EventArgs e)  {  // Получите значения из TextBox'ов  DateTime Back = DateTime.Parse(back\_textbox.Text);  decimal Tax = decimal.Parse(tax\_textbox.Text);  decimal BackSum = decimal.Parse(backSum\_textbox.Text);  string Seller = seller\_textbox.Text;  string insertQuery = "INSERT INTO Back (Back, Tax, BackSum, Seller) VALUES (@Back, @Tax, @BackSum, @Seller)";  using (SqlConnection connection = new SqlConnection(ConnectionString))  {  connection.Open();  using (SqlCommand command = new SqlCommand(insertQuery, connection))  {  command.Parameters.AddWithValue("@Back", Back);  command.Parameters.AddWithValue("@Tax", Tax);  command.Parameters.AddWithValue("@BackSum", BackSum);  command.Parameters.AddWithValue("@Seller", Seller);  command.ExecuteNonQuery();  }  }  LoadUsersData();  back\_textbox.Clear();  tax\_textbox.Clear();  backSum\_textbox.Clear();  seller\_textbox.Clear();  } |
| show\_all\_button | // запускает метод который обновляет datagridview  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Sort\_button | // с помощью запроса сортирует возвраты по коду возврата  private void sort\_button\_Click(object sender, EventArgs e)  {  int BackID = int.Parse(backID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT BackID as [Код возврата], Back as [Дата выкупа вещи] , Tax as [Пени за несвоевременный выкуп], BackSum as [Сумма, заплаченная за выкуп], Seller as [Фамилия вернувшего вещь] FROM Back WHERE BackID = '{BackID}'";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  } |
| Delete\_button | // с помощью запроса удаляет возврат по коду возврата, и обновляет datagridbiew  private void delete\_button\_Click(object sender, EventArgs e)  {  int BackID = int.Parse(backID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"DELETE FROM Back WHERE BackID = '{BackID}'";  SqlCommand command = new SqlCommand(deleteQuery, connection);  command.ExecuteNonQuery();  dataGridView1.DataSource = dataTable;  backID\_textbox.Text = "";  LoadUsersData();  }  } |
| backID\_textbox | // очищает textbox по нажатию на него  private void backID\_textbox\_Click(object sender, EventArgs e)  {  backID\_textbox.Text = "";  } |
| Loaduserdata | // метод обновления datagridview  public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT BackID as [Код возврата], Back as [Дата выкупа вещи] , Tax as [Пени за несвоевременный выкуп], BackSum as [Сумма, заплаченная за выкуп], Seller as [Фамилия вернувшего вещь] FROM Back";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |

Форма таблицы «Category» представлена на рисунке 25.

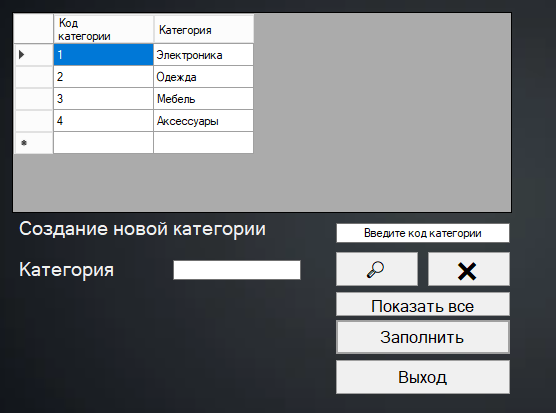


Рисунок 25 – Форма таблицы «Category»

Таблица 20 – Свойства элементов формы «Category»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Category | Name | Category |
| FormBorderStyle | None |
| Size | 556; 413 |
| Datagridview1 | Size | 500; 201 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 82; 34 |
| Text | ✖ |
| Font | Microsoft Sans Serif; 18pt |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 82; 34 |
| Text | 🔎 |
| Font | Microsoft Sans Serif; 14pt |
| show\_all\_button | Size | 174; 24 |
| Text | Показать все |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| category\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| categoryID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код категории |
| category\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Категория |
| ForeColor | ControlText |
| createCategory\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание новой категории |
| ForeColor | ControlText |

Таблица 21 – Событийно-управляемые процедуры в форме «Category»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрытие формы  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void Category\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog= pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| fill\_button | // читает textbox, и с помощью запроса заполняет их в базу данных  private void fill\_button\_Click(object sender, EventArgs e)  {  // Получите значения из TextBox'ов  string Categorys = category\_textbox.Text;  string insertQuery = "INSERT INTO Category (Category) " +  $"VALUES ('{Categorys}')";  using (SqlConnection connection = new SqlConnection(ConnectionString))  {  connection.Open();  using (SqlCommand command = new SqlCommand(insertQuery, connection))  {  command.ExecuteNonQuery();  }  }  LoadUsersData();  category\_textbox.Clear();  } |
| show\_all\_button | // запускает метод который обновляет datagridview  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Sort\_button | // с помощью запроса сортирует категории по коду категории  private void sort\_button\_Click(object sender, EventArgs e)  {  int CategoryID = int.Parse(categoryID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT CategoryID as [Код категории], Category as [Категория] FROM Category WHERE CategoryID = '{CategoryID}'";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  } |
| Delete\_button | // с помощью запроса удаляет возврат по коду категории, и обновляет datagridbiew  private void delete\_button\_Click(object sender, EventArgs e)  {  int CategoryID = int.Parse(categoryID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"DELETE FROM Category WHERE CategoryID = '{CategoryID}'";  SqlCommand command = new SqlCommand(deleteQuery, connection);  command.ExecuteNonQuery();  dataGridView1.DataSource = dataTable;  categoryID\_textbox.Text = "";  LoadUsersData();  }  } |
| categoryID\_textbox | // очищает textbox по нажатию на него  private void categoryID\_textbox\_Click(object sender, EventArgs e)  {  categoryID\_textbox.Text = "";  } |
| Loaduserdata | // метод обновления datagridview  public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT CategoryID as [Код категории], Category as [Категория] FROM Category";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |

Форма таблицы «Client» представлена на рисунке 26.

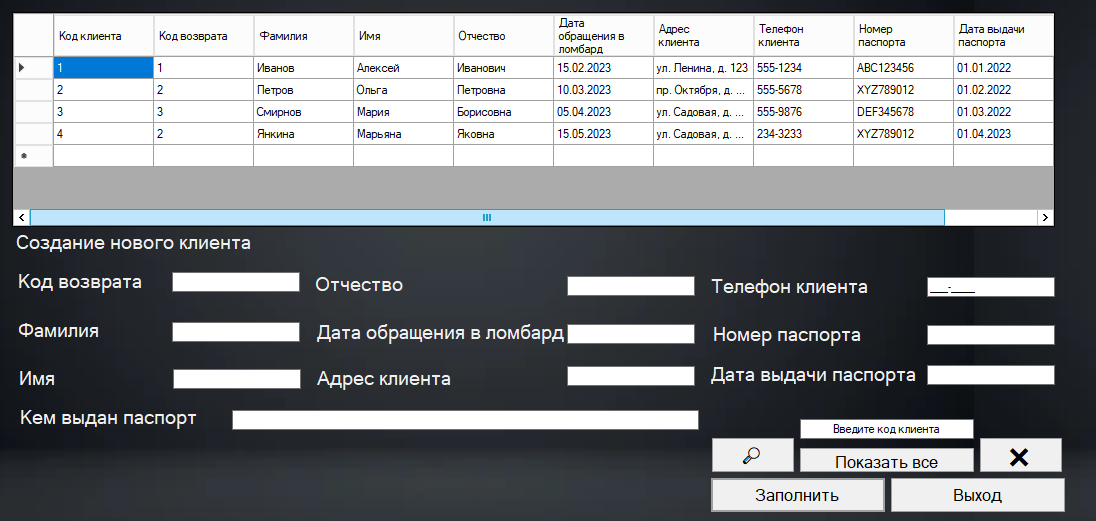


Рисунок 26 – Форма таблицы «Client»

Таблица 22 – Свойства элементов формы «Client»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Client | Name | Client |
| FormBorderStyle | None |
| Size | 1096; 521 |
| Datagridview1 | Size | 1043; 215 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 82; 34 |
| Text | ✖ |
| Font | Microsoft Sans Serif; 18pt |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 82; 34 |
| Text | 🔎 |
| Font | Microsoft Sans Serif; 14pt |
| show\_all\_button | Size | 174; 24 |
| Text | Показать все |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| backID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| surname\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| name\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| patronymic\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| date\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| address\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| phoneClient\_maskedtextbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код договора |
| Mask | 000-0000 |
| region\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 467; 20 |
| date\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| clientID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код клиента |
| passport\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| passportDate\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| backID\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Код возврата |
| ForeColor | ControlText |
| surname\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Фамилия |
| ForeColor | ControlText |
| name\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Имя |
| ForeColor | ControlText |
| patronymic\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Отчество |
| ForeColor | ControlText |
| date\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Дата обращения в ломбард |
| ForeColor | ControlText |
| Address\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Код клиента |
| ForeColor | ControlText |
| phoneClient\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Телефон клиента |
| ForeColor | ControlText |
| passport\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Номер паспорта |
| ForeColor | ControlText |
| passportDate\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Дата выдачи паспорта |
| ForeColor | ControlText |
| region\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Кем выдан паспорт |
| ForeColor | ControlText |
| createClient\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание нового клиента |
| ForeColor | ControlText |

Таблица 23 – Событийно-управляемые процедуры в форме «Admin»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрытие формы  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void Client\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| fill\_button | // читает textbox, и с помощью запроса заполняет их в базу данных  private void fill\_button\_Click(object sender, EventArgs e)  {  // Получите значения из TextBox'ов  int BackID = int.Parse(backID\_textbox.Text);  string Surname = surname\_textbox.Text;  string Names = name\_textbox.Text;  string Patronymic = patronymic\_textbox.Text;  DateTime Dates = DateTime.Parse(date\_textbox.Text);  string Addres = address\_textbox.Text;  string Phone = phoneClient\_maskedtextbox.Text; //textBox7.Text;  string Passport = passport\_textbox.Text;  DateTime PassportDate = DateTime.Parse(passportDate\_textbox.Text);  string Region = region\_textbox.Text;  string insertQuery = "INSERT INTO Client (BackID, Surname, Names, Patronymic, Dates, Addres, Phone, Passport, PassportDate, Region) " +  "VALUES (@BackID, @Surname, @Names, @Patronymic, @Dates, @Addres, @Phone, @Passport, @PassportDate, @Region)";  using (SqlConnection connection = new SqlConnection(ConnectionString))  {  connection.Open();  using (SqlCommand command = new SqlCommand(insertQuery, connection))  {  command.Parameters.AddWithValue("@BackID", BackID);  command.Parameters.AddWithValue("@Surname", Surname);  command.Parameters.AddWithValue("@Names", Names);  command.Parameters.AddWithValue("@Patronymic", Patronymic);  command.Parameters.AddWithValue("@Dates", Dates);  command.Parameters.AddWithValue("@Addres", Addres);  command.Parameters.AddWithValue("@Phone", Phone);  command.Parameters.AddWithValue("@Passport", Passport);  command.Parameters.AddWithValue("@PassportDate", PassportDate);  command.Parameters.AddWithValue("@Region", Region);  command.ExecuteNonQuery();  }  }  LoadUsersData();  backID\_textbox.Clear();  name\_textbox.Clear();  patronymic\_textbox.Clear();  date\_textbox.Clear();  address\_textbox.Clear();  surname\_textbox.Clear();  passportDate\_textbox.Clear();  passport\_textbox.Clear();  region\_textbox.Clear();  clientID\_textbox.Clear();  } |
| Show\_all\_button | // запускает метод который обновляет datagridview  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Sort\_button | // с помощью запроса сортирует клиентов по коду клиента  private void sort\_button\_Click(object sender, EventArgs e)  {  int ClientID = int.Parse(clientID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT ClientID as [Код клиента], BackID as [Код возврата], Surname as [Фамилия] , Names as [Имя] , Patronymic as [Отчество],Dates as [Дата обращения в ломбард], Addres as [Адрес клиента], Phone as [Телефон клиента], Passport as [Номер паспорта] , PassportDate as [Дата выдачи паспорта], Region as [Кем выдан паспорт] FROM Client WHERE ClientID = '{ClientID}'";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  } |
| Delete\_button | // с помощью запроса удаляет клиента ко коду клиента, и обновляет datagridbiew  private void delete\_button\_Click(object sender, EventArgs e)  {  int ClientID = int.Parse(clientID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"DELETE FROM Client WHERE ClientID = '{ClientID}'";  SqlCommand command = new SqlCommand(deleteQuery, connection);  command.ExecuteNonQuery();  dataGridView1.DataSource = dataTable;  clientID\_textbox.Text = "";  LoadUsersData();  }  } |
| clientID\_textbox | // очищает textbox по нажатию на него  private void clientID\_textbox\_Click(object sender, EventArgs e)  {  clientID\_textbox.Text = "";  } |
| Loaduserdata | // метод обновления datagridview  public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT ClientID as [Код клиента], BackID as [Код возврата], Surname as [Фамилия] , Names as [Имя] , Patronymic as [Отчество],Dates as [Дата обращения в ломбард], Addres as [Адрес клиента], Phone as [Телефон клиента], Passport as [Номер паспорта] , PassportDate as [Дата выдачи паспорта], Region as [Кем выдан паспорт] FROM Client";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  foreach (DataGridViewColumn column in dataGridView1.Columns)  {  column.AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;  }  dataGridView1.DataSource = dataTable;  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |

Форма таблицы «Registrar» представлена на рисунке 27.

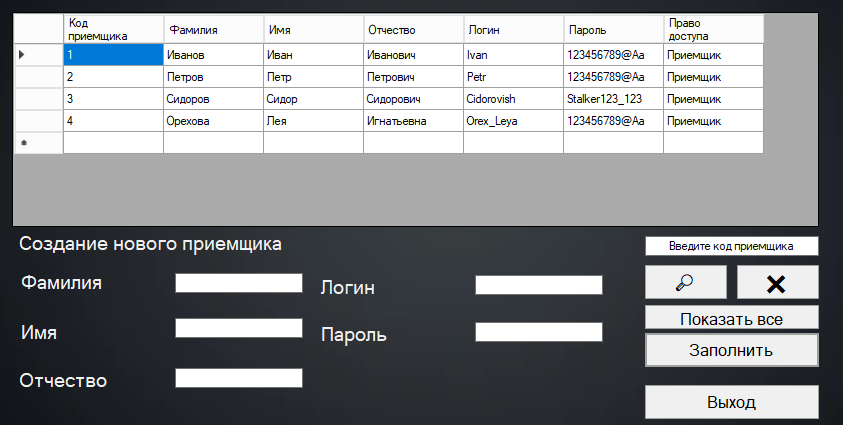


Рисунок 27 – Форма таблицы «Registrar»

Таблица 24 – Свойства элементов формы «Registrar»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Registrar | Name | Registrar |
| FormBorderStyle | None |
| Size | 791; 523 |
| Datagridview1 | Size | 741; 265 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 82; 34 |
| Text | ✖ |
| Font | Microsoft Sans Serif; 18pt |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 82; 34 |
| Text | 🔎 |
| Font | Microsoft Sans Serif; 14pt |
| show\_all\_button | Size | 174; 24 |
| Text | Показать все |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| surname\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| name\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| patronymic\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| seller\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| login\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| password\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| registrarID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код приемщика |
| surname\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Фамилия |
| ForeColor | ControlText |
| name\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Имя |
| ForeColor | ControlText |
| patronymic\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Отчество |
| ForeColor | ControlText |
| login\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Логин |
| ForeColor | ControlText |
| password\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Пароль |
| ForeColor | ControlText |
| createRegistrar\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание нового приемщика |
| ForeColor | ControlText |

Таблица 25 – Событийно-управляемые процедуры в форме «Admin»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрытие формы  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void Registrar\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  // Инициализация адаптера данных и таблицы данных  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  // Привяжите DataGridView к таблице данных  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| fill\_button | // читает textbox, и с помощью запроса заполняет их в базу данных  private void fill\_button\_click(object sender, EventArgs e)  {  // Получите значения из TextBox'ов  string Surname = surname\_textbox.Text;  string Names = name\_textbox.Text;  string Patronymic = patronymic\_textbox.Text;  string username = login\_textbox.Text;  string password = password\_textbox.Text;  string accessRights = "Приемщик";  string insertQuery = "INSERT INTO Registrar (Surname, Names, Patronymic,Logins,Passwords,Access\_right) " +  $"VALUES ('{Surname}', '{Names}', '{Patronymic}', '{username}','{password}','{accessRights}')";  using (SqlConnection connection = new SqlConnection(ConnectionString))  {  connection.Open();  using (SqlCommand command = new SqlCommand(insertQuery, connection))  {  command.ExecuteNonQuery();  }  }  LoadUsersData();  surname\_textbox.Clear();  name\_textbox.Clear();  patronymic\_textbox.Clear();  login\_textbox.Clear();  password\_textbox.Clear();  } |
| show\_all\_button | // запускает метод который обновляет datagridview  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Sort\_button | // с помощью запроса сортирует приемщиков по коду приемщика  private void sort\_button\_Click(object sender, EventArgs e)  {  int RegistrarID = int.Parse(registrarID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT RegistrarID as [Код приемщика] , Surname as [Фамилия] , Names as [Имя] , Patronymic as [Отчество], Logins as [Логин],Passwords as [Пароль],Access\_right as [Право доступа] FROM Registrar WHERE RegistrarID = '{RegistrarID}'";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }} |
| Delete\_button | // с помощью запроса удаляет приемщика по коду приемщика, и обновляет datagridbiew  private void delete\_button\_Click(object sender, EventArgs e)  {  int RegistrarID = int.Parse(registrarID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"DELETE FROM Registrar WHERE RegistrarID = '{RegistrarID}'";  SqlCommand command = new SqlCommand(deleteQuery, connection);  command.ExecuteNonQuery();  dataGridView1.DataSource = dataTable;  registrarID\_textbox.Text = "";  LoadUsersData();  }  } |
| registrarID\_textbox | // очищает textbox по нажатию на него  private void registrarID\_textbox\_Click(object sender, EventArgs e)  {  registrarID\_textbox.Text = "";  } |
| Loaduserdata | public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT RegistrarID as [Код приемщика] , Surname as [Фамилия] , Names as [Имя] , Patronymic as [Отчество],Logins as [Логин],Passwords as [Пароль],Access\_right as [Право доступа] FROM Registrar";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  catch (Exception ex) {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |

Форма таблицы «Sale» представлена на рисунке 28.

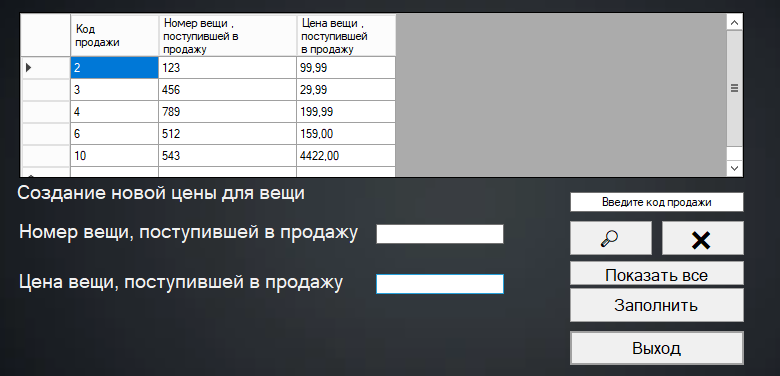


Рисунок 28 – Форма таблицы «Sale»

Таблица 26 – Свойства элементов формы «Sale»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Sale | Name | Sale |
| FormBorderStyle | None |
| Size | 1040; 463 |
| Datagridview1 | Size | 967; 204 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 82; 34 |
| Text | ✖ |
| Font | Microsoft Sans Serif; 18pt |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 82; 34 |
| Text | 🔎 |
| Font | Microsoft Sans Serif; 14pt |
| show\_all\_button | Size | 174; 24 |
| Text | Показать все |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| freeThing\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| price\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| saleID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код продажи |
| freeThing \_label | Font | Microsoft Sans Serif; 14pt |
| Text | Номер вещи, поступившей в продажу |
| ForeColor | ControlText |
| price \_label | Font | Microsoft Sans Serif; 14pt |
| Text | Цена вещи, поступившей в продажу |
| ForeColor | ControlText |
| createSale\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание новой цены для вещи |
| ForeColor | ControlText |

Таблица 27 – Событийно-управляемые процедуры в форме «Sale»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрытие формы  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void Sale\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| fill\_button | // читает textbox, и с помощью запроса заполняет их в базу данных  private void fill\_button\_Click(object sender, EventArgs e)  {  // Получите значения из TextBox'ов  float FreeThing = int.Parse(freeThing\_textbox.Text);  decimal Price = decimal.Parse(price\_textbox.Text);  string insertQuery = "INSERT INTO Sale (FreeThing , Price ) VALUES (@FreeThing, @Price)";  using (SqlConnection connection = new SqlConnection(ConnectionString))  {  connection.Open();  using (SqlCommand command = new SqlCommand(insertQuery, connection))  {  command.Parameters.AddWithValue("@FreeThing", FreeThing);  command.Parameters.AddWithValue("@Price", Price);  command.ExecuteNonQuery();  }  }  LoadUsersData();  freeThing\_textbox.Clear();  price\_textbox.Clear();  } |
| show\_all\_button | // запускает метод который обновляет datagridview  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Sort\_button | // с помощью запроса сортирует продажи по коду продажи  private void sort\_button\_Click(object sender, EventArgs e)  {  int SaleID = int.Parse(saleID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT SaleID AS [Код продажи] , FreeThing AS [Номер вещи , поступившей в продажу], Price AS [Цена вещи , поступившей в продажу] FROM Sale WHERE SaleID = '{SaleID}'";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  } |
| Delete\_button | // с помощью запроса удаляет продажи по коду продажи, и обновляет datagridbiew  private void delete\_button\_Click(object sender, EventArgs e)  {  int RegistrarID = int.Parse(registrarID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"DELETE FROM Registrar WHERE RegistrarID = '{RegistrarID}'";  SqlCommand command = new SqlCommand(deleteQuery, connection);  command.ExecuteNonQuery();  dataGridView1.DataSource = dataTable;  registrarID\_textbox.Text = "";  LoadUsersData();  }  } |
| saleID\_textbox | // очищает textbox по нажатию на него  private void saleID\_textbox\_Click(object sender, EventArgs e)  {  saleID\_textbox.Text = "";  } |
| Loaduserdata | public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT SaleID AS [Код продажи] , FreeThing AS [Номер вещи , поступившей в продажу], Price AS [Цена вещи , поступившей в продажу] FROM Sale";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  foreach (DataGridViewColumn column in dataGridView1.Columns)  {  column.AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |

Форма таблицы «Thing» представлена на рисунке 29.

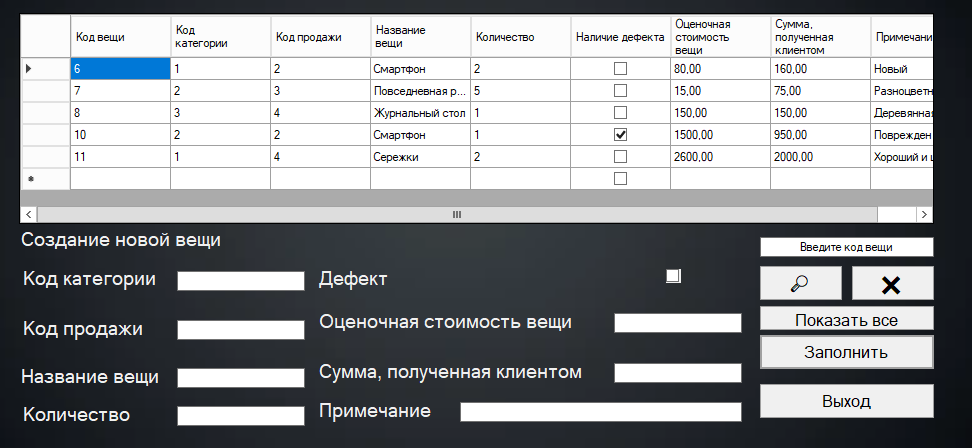


Рисунок 29 – Форма таблицы «Thing»

Таблица 28 – Свойства элементов формы «Thing»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Thing | Name | Thing |
| FormBorderStyle | None |
| Size | 1296; 551 |
| Datagridview1 | Size | 1220; 260 |
| Borderstyle | FixedSingle |
| delete\_button | Size | 82; 34 |
| Text | ✖ |
| Font | Microsoft Sans Serif; 18pt |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| sort\_button | Size | 82; 34 |
| Text | 🔎 |
| Font | Microsoft Sans Serif; 14pt |
| show\_all\_button | Size | 174; 24 |
| Text | Показать все |
| Font | Microsoft Sans Serif; 12pt |
| fill\_button | Size | 174; 34 |
| Text | Заполнить |
| Font | Microsoft Sans Serif; 12pt |
| categoryID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| thing\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| count\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| defects\_checkbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| cost\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| sum\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| comment\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 128; 20 |
| thingID\_textbox | Font | Microsoft Sans Serif; 8,25pt |
| Size | 174; 20 |
| Text | Введите код вещи |
| categoryID\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Код категории |
| ForeColor | ControlText |
| saleID\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Код продажи |
| ForeColor | ControlText |
| thing\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Название вещи |
| ForeColor | ControlText |
| count\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Количество |
| ForeColor | ControlText |
| defects\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Дефект |
| ForeColor | ControlText |
| cost\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Оценочная стоимость вещи |
| ForeColor | ControlText |
| sum\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Сумма, полученная клиентом |
| ForeColor | ControlText |
| comment\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Примечание |
| ForeColor | ControlText |
| createThing\_label | Font | Microsoft Sans Serif; 14pt |
| Text | Создание новой вещи |
| ForeColor | ControlText |

Таблица 29 – Событийно-управляемые процедуры в форме «Thing»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрытие формы  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void Thing\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| fill\_button | // читает textbox, и с помощью запроса заполняет их в базу данных  private void fill\_button\_Click(object sender, EventArgs e)  {  try  {  int CategoryID = int.Parse(categoryID\_textbox.Text);  int SaleID = int.Parse(saleID\_button.Text);  string Thing = thing\_textbox.Text;  float Counts = float.Parse(count\_textbox.Text);  bool Defects;  if (defects\_checkbox.Checked)  {  Defects = true;  }  else  {  Defects = false;  }  decimal Cost = decimal.Parse(cost\_textbox.Text);  decimal Summ = decimal.Parse(sum\_textbox.Text);  string Comment = comment\_textbox.Text;  string insertQuery = "INSERT INTO Thing (CategoryID, SaleID, Thing, Counts, Defects, Cost, Summ, Comment) " +  $"VALUES ('{CategoryID}','{SaleID}','{Thing}', '{Counts}', '{Defects}', '{Cost}','{Summ}','{Comment}')";  using (SqlConnection connection = new SqlConnection(ConnectionString))  {  connection.Open();  using (SqlCommand command = new SqlCommand(insertQuery, connection))  {  command.ExecuteNonQuery();  }  }  }  catch (FormatException ex)  {  // Обработка исключения при неверном формате ввода чисел  MessageBox.Show("Ошибка ввода числовых данных. Пожалуйста, проверьте правильность ввода.");  }  catch (SqlException ex)  {  // Обработка исключения при ошибке SQL запроса  MessageBox.Show($"Ошибка SQL: {ex.Message}");  }  catch (Exception ex)  {  // Обработка других исключений  MessageBox.Show($"Произошла ошибка: {ex.Message}");  }  LoadUsersData();  categoryID\_textbox.Clear();  saleID\_button.Clear();  thing\_textbox.Clear();  count\_textbox.Clear();  comment\_textbox.Clear();  sum\_textbox.Clear();  cost\_textbox.Clear();  } |
| show\_all\_button | // запускает метод который обновляет datagridview  private void show\_all\_button\_Click(object sender, EventArgs e)  {  LoadUsersData();  } |
| Sort\_button | // с помощью запроса сортирует вещи по коду вещи  private void sort\_button\_Click(object sender, EventArgs e)  {  int ThingID = int.Parse(thingID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT ThingID AS [Код вещи],CategoryID AS[Код категории],SaleID AS[Код продажи],Thing AS[Название вещи],Counts AS[Количество],Defects AS[Наличие дефекта],Cost AS[Оценочная стоимость вещи],Summ AS[Сумма, полученная клиентом],Comment AS[Примечания] FROM Thing WHERE ThingID = '{ThingID}'";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  } |
| Delete\_button | // с помощью запроса удаляет вещи по коду вещи, и обновляет datagridbiew  private void delete\_button\_Click(object sender, EventArgs e)  {  int ThingID = int.Parse(thingID\_textbox.Text);  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"DELETE FROM Thing WHERE ThingID = '{ThingID}'";  SqlCommand command = new SqlCommand(deleteQuery, connection);  command.ExecuteNonQuery();  dataGridView1.DataSource = dataTable;  thingID\_textbox.Text = "";  LoadUsersData();  }  } |
| thingID\_textbox | // очищает textbox по нажатию на него  private void thingID\_textbox\_Click(object sender, EventArgs e)  {  thingID\_textbox.Text = "";  } |
| Loaduserdata | public void LoadUsersData()  {  try  {  dataTable.Clear();  dataGridView1.DataSource = null;  string query = "SELECT ThingID AS [Код вещи],CategoryID AS[Код категории],SaleID AS[Код продажи],Thing AS[Название вещи],Counts AS[Количество],Defects AS[Наличие дефекта],Cost AS[Оценочная стоимость вещи],Summ AS[Сумма, полученная клиентом],Comment AS[Примечания] FROM Thing";  dataAdapter.SelectCommand = new SqlCommand(query, connection);  dataAdapter.Fill(dataTable);  dataGridView1.DataSource = dataTable;  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при загрузке данных: " + ex.Message);  }  } |

Форма таблицы «Request» представлена на рисунке 30.

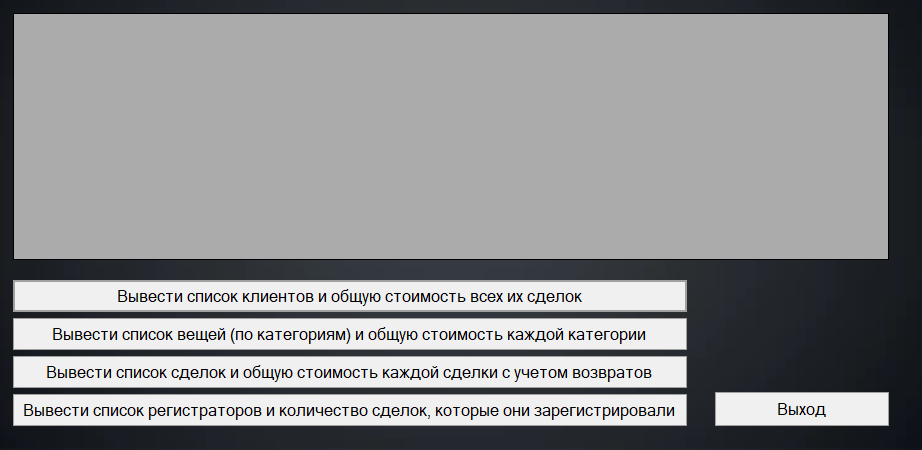


Рисунок 30 – Форма таблицы «Request»

Таблица 30 – Свойства элементов формы «Request»

| Объект | Свойство | Значение |
| --- | --- | --- |
| Request | Name | Request |
| FormBorderStyle | None |
| Size | 1229; 554 |
| Datagridview1 | Size | 1168; 304 |
| Borderstyle | FixedSingle |
| exit\_button | Size | 174; 34 |
| Text | Выход |
| Font | Microsoft Sans Serif; 12pt |
| allClient\_button | Size | 899; 39 |
| Text | Вывести список клиентов и общую стоимость всех их сделок |
| Font | Microsoft Sans Serif; 14pt |
| allThingSale\_button | Size | 899; 39 |
| Text | Вывести список вещей (по категориям) и общую стоимость каждой категории |
| Font | Microsoft Sans Serif; 14pt |
| allAgreemntBack\_button | Size | 899; 39 |
| Text | Вывести список сделок и общую стоимость каждой сделки с учетом возвратов |
| Font | Microsoft Sans Serif; 14pt |
| allRegistrarAgreemnt\_button | Size | 899; 39 |
| Text | Вывести список регистраторов и количество сделок, которые они зарегистрировали |
| Font | Microsoft Sans Serif; 14pt |

Таблица 31 – Событийно-управляемые процедуры в форме «Sale»

| Объект | Событийно-управляемые процедуры |
| --- | --- |
| exit\_button | // закрываем форму и открываем форму оператора  private void exit\_button\_Click(object sender, EventArgs e)  {  this.Close();  Operator OperatorFrm = new Operator();  OperatorFrm.Show();  } |
| Formload | // подключается к базе данных и обновляет datagridview  private void request\_Load(object sender, EventArgs e)  {  connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True");  dataAdapter = new SqlDataAdapter();  dataTable = new DataTable();  dataGridView1.DataSource = dataTable;  LoadUsersData();  } |
| allClient\_button | //Вывести список клиентов и общую стоимость всех их сдел  private void allClient\_button\_Click(object sender, EventArgs e)  {  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT Client.ClientID,Surname AS [Фамилия],Names AS [Имя],Patronymic AS [Отчество],COUNT(AgreementID) AS [Количество договоров],SUM(Cost) AS [Общая стоимость]FROM Client LEFT JOIN Agreement ON Client.ClientID = Agreement.ClientID LEFT JOIN Thing ON Thing.ThingID = Agreement.ThingID GROUP BY Client.ClientID, Surname, Names, Patronymic";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;  dataGridView1.DataSource = dataTable;  }  } |
| allThingSale\_button | //Вывести список вещей (по категориям) и общую стоимость каждой категории  private void allThingSale\_button\_Click(object sender, EventArgs e)  {  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT cat.Category as [Категория],COUNT(t.ThingID) AS [Количество вещей],SUM(t.Cost) AS [Общая стоимость] FROM Category cat LEFT JOIN Thing t ON cat.CategoryID = t.CategoryID GROUP BY cat.Category";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;  dataGridView1.DataSource = dataTable;  }  } |
| allAgreemntBack\_button | //Вывести список сделок и общую стоимость каждой сделки с учетом возвратов  private void allAgreemntBack\_button\_Click(object sender, EventArgs e)  {  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT a.AgreementID as [Код договора], c.Surname AS[Фамилия клиента], c.Names AS[Имя клиента], c.Patronymic AS[Отчество клиента], SUM(t.Cost - COALESCE(b.BackSum, 0)) AS[Общая стоимость с учетом возвратов] FROM Agreement a JOIN Client c ON a.ClientID = c.ClientID JOIN Thing t ON a.ThingID = t.ThingID LEFT JOIN Back b ON c.ClientID = a.ClientID GROUP BY a.AgreementID, c.Surname, c.Names, c.Patronymic";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;  dataGridView1.DataSource = dataTable;  }  } |
| allRegistrarAgreemnt\_button | //Вывести список регистраторов и количество сделок, которые они зарегистрировали  private void allRegistrarAgreemnt\_button\_Click(object sender, EventArgs e)  {  using (SqlConnection connection = new SqlConnection("Data Source=sql;Initial Catalog=pawnshop;Integrated Security=True"))  {  connection.Open();  string deleteQuery = $"SELECT r.RegistrarID as [Код приемщика], r.Surname as [Фамилия приемщика], r.Names as [Имя приемщика], r.Patronymic as [Отчество приемщика], COUNT(a.AgreementID) AS [Количество оформленных договоров] FROM Registrar r LEFT JOIN Agreement a ON r.RegistrarID = a.RegistrarID GROUP BY r.RegistrarID, r.Surname, r.Names, r.Patronymic";  SqlDataAdapter adapter = new SqlDataAdapter(deleteQuery, connection);  DataTable dataTable = new DataTable();  adapter.Fill(dataTable);  dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;  dataGridView1.DataSource = dataTable;  }  } |

# РАЗРАБОТКА ТЕСТОВЫХ НАБОРОВ И ТЕСТОВЫХ СЦЕНАРИЕВ

Тестирование программного обеспечения – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

Тестовый сценарий – это неавтоматизированный или автоматизированный сценарий, содержащий инструкции по реализации тестового набора. Тестовый сценарий может быть написан вручную (для выполнения человеком) либо полностью или частично автоматизирован.

## Назначение эксперимента. Выбор и обоснование методики проведения тестирования

Можно определить такие основные цели тестирования программного обеспечения:

* предоставление информации о качестве ПО конечному заказчику;
* повышение качества ПО;
* предотвращение появления дефектов.

Цели тестирования могут отличаться, в зависимости от этапа разработки программного обеспечения, на котором оно проводится. К примеру, на этапе кодирования целью тестирования будет вызов как можно большего количества сбоев в работе программы, что позволит локализовать и исправить дефекты.

В то же время, при приемочном тестировании необходимо показать, что система работает правильно. В период сопровождения, тестирование в основном необходимо для того, чтобы убедится в отсутствии новых багов, появившихся во время внесения изменений.

Главная же задача тестирования – поиск дефектов.

В данном разделе проведены два вида тестирования. Провести анализ тестирования и выбрать, как будет проходить тестирование ИС.

Модульное тестирование Цель: проверить, что код работает именно так, как должен (при заданных входных параметрах выдает предполагаемый результат).

Анализ покрытия кода тестами. Цель: проверить, что весь наш код отрабатывает при модульном тестировании, что нет не участвующих в тестировании участков кода.

Анализ стилистики кода. Цель: проверка кода на соблюдение стандартов разработки (отраслевых, корпоративных, принятых на проекте).

Анализ производительности. Цель: проанализировать производительность ключевых операций, потребление памяти, утечку памяти.

## Технология тестирования

Рассмотрим, какие технологии можно применить для модульного тестирования и покрытия кода тестами.

Как в ходе выполнения проекта, так и при развитии системы, изменения в ее алгоритмы вносятся в соответствии с техническими заданиями, хотя данный документ может иметь различные названия.

В технических заданиях, как правило, описываются критерии приемки: как мы и заказчик убедимся в том, что заказанный функционал работает верно.

В качестве примера можно привести такой упрощенный критерий:

* Исходные данные: в систему внесена вещь, которую в дальнейшем приобретёт покупатель.
* Действия пользователя: пользователь ищет в таблице необходимую вещь с определённой категорией, и сколько он будет стоить и т п.
* В результате в результате отображена вещь с ценой
* Вручную такое тестирование, как правило, выполняется единожды, поскольку выполнять каждый раз полный комплект тестов очень трудоемко.
* Нет гарантии того, что тесты, описанные в техническом задании, покрывают функциональность полностью. Т.е. даже если тесты описаны и проводятся регулярно, вероятно наличие не протестированного функционала.

Автоматизированное тестирование лишено данных недостатков. Конечно, возникает отдельная задача кодирования данного теста на основании критериев приемки, но это скорее дисциплинирующий фактор…

Прежде чем начинать юзабилити-тестирование необходимо собрать группу пользователей, которые будут тестировать данную информационную систему.

Количество привлеченных пользователей должно быть не менее пяти человек. Пользователям будет представлен список некоторых вопросов, относящихся к графическому интерфейсу.

В таблице 32 представлены сведения о пользователях.

Таблица 32 – Информация о пользователях

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО | Пол | Возраст | Образование |
| Шишова Оксана Леонтьевна | Женский | 37 | Высшее |
| Калугин Ефим Савванович | Мужской | 44 | Среднее специальное |
| Явленская Ева Васильевна | Женский | 35 | Среднее специальное |
| Язькова София Денисовна | Женский | 24 | Среднее специальное |
| Наполов Павел Петрович | Мужской | 42 | Среднее специальное |

## Результаты проведения тестирования

В ходе тестирования был применен метод нагрузочного тестирования и юзабилити-тестирования, которые проверяют работу при нагрузке системы и сделать простым и удобным в пользовании информационной системы.

Суть тестирования заключается в проверке работы всех обязательных функций.

Для проведения нагрузочного тестирования составлена таблица 22, в которой указано количество запущенных копий программного продукта, нагрузка на процесс и оперативную память.

В таблице 33 представлены результаты тестирования.

Таблица 33 – Результаты тестирования.

|  |  |  |
| --- | --- | --- |
| Число копий | Нагрузка на процессор (%) | Нагрузка на память (%) |
| 1 | 0,1 | 0,2 |
| 5 | 0,2 | 2 |
| 10 | 2,5 | 3 |
| 15 | 3,3 | 4 |
| 25 | 4,6 | 4 |

Исходя из результатов тестирования, можно прийти к выводу, что нагрузка, оказываемая на центральный процессор и память, не имеет критичных значений.

Это значит, что показатель производительности информационной системы на приемлемом уровне, то есть сбои и длительная загрузка не должны помешать пользователю при работе с данным программным продуктом.

Таблица 25 – Результаты проведения юзабилити-тестирования.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Вопрос | | Шишова Оксана Леонтьевна | Калугин Ефим Савванович | Явленская Ева Васильевна | Язькова София Денисовна | Наполов Павел Петрович |
|  | Присутствуют ли такие дизайн-элементы, которые неприятны для зрения? | нет | нет | нет | нет | нет |
|  | Присутствует ли отвлекающая информация? | нет | нет | нет | нет | нет |
|  | Понятна ли структура приложения на интуитивном уровне? | да | да | да | да | да |
|  | Смогли ли вы найти интересующую вас информацию без затруднений? | да | да | да | нет | да |
|  | Понятен ли смысл приложения? | да | да | да | да | да |
| **Общая оценка** | | **9/10** | **7/10** | **9/10** | **7/10** | **8/10** |

Исходя из средней оценки информационной системы, рассчитанной на основе общих оценок всех пользователей с учетом ответов на все поставленные вопросы, можно сделать вывод о том, что внешние характеристики графического интерфейса находятся на довольно высоком уровне. Также в информационной системе нет помех для пользователя в виде отвлекающей информации, так как все отображаемые данные находятся в рамках предметной области

Вывод по главе: были проведены тесты путём проведения модульного тестирования, анализа покрытия кода тестами, анализа стилистики кода и анализом производительности.

Также было проведено нагрузочное тестирование, в результате чего нагрузка на процессор и память не имела критичных значений.

# ЗАКЛЮЧЕНИЕ

В результате учебной практики был проведен анализ предметной области, спроектирована и разработана информационная система «Ломбард», серверная часть которой была реализована в MS SQL, а клиентская часть в MS Visual Studio.

В MS SQL была создана база данных для хранения информации о вещах, категориях, клиентах, премщиков, договорах, продажи и возврата.

Клиентская часть, реализованная в MS Visual Studio, отражает все хранящиеся в базе данных таблицы. Также разработаны некоторые запросы, позволяющие редактировать и удалять существующие данные, добавлять новые и осуществлять поиск по таблицам.

Цели учебной практики были достигнуты путем проведения анализа предметной области, проектирования и прототипирования информационной системы.

Это позволяет сделать вывод, что поставленные цели и задачи выполнены в полном объеме.

В дальнейшем данная информационная система может быть модернизирована путем добавления формы личного кабинета, создания версии для слабовидящих или редизайна.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Казанский, А. А. Программирование на Visual C# : учебное пособие для среднего профессионального образования / А. А. Казанский. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2023. — 192 с. — (Профессиональное образование). — ISBN 978-5-534-14130-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/513400 (дата обращения: 02.12.2023).
2. Маркин, А. В. Программирование на SQL : учебник и практикум для вузов / А. В. Маркин. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2024. — 805 с. — (Высшее образование). — ISBN 978-5-534-18371-9. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/534873 (дата обращения: 02.12.2023).
3. Что такое ломбард и когда он выгоднее банка [Электронный ресурс]: URL: <https://finuslugi.ru/navigator/kredity/stat_chto_takoe_lombard_i_kogda_on_vygodnee_banka> (дата обращения: 25.05.2023).
4. Как работает ломбард [Электронный ресурс]: URL: <https://journal.tinkoff.ru/lombard/> (дата обращения: 25.05.2023).
5. Ломбард — как это работает и в каких случаях пригодится [Электронный ресурс]: URL: <https://fincult.info/article/lombard-kak-eto-rabotaet-i-v-kakikh-sluchayakh-prigoditsya/> (дата обращения: 25.05.2023).
6. ГОСТ 19.201-78. ЕСПД. Техническое задание. [Электронный ресурс]: URL: <https://www.swrit.ru/doc/espd/19.201-78.pdf> (дата обращения: 25.05.2023).
7. Что такое ГОСТ и зачем он нужен? [Электронный ресурс]: URL: <https://aif.ru/dontknows/eternal/chto_takoe_gost_i_zachem_on_nuzhen> (дата обращения: 25.05.2023).
8. Чтение из textbox, выполнение SQL запроса и выдача результата в textbox [Электронный ресурс]: URL: <https://www.cyberforum.ru/windows-forms/thread2783322.html> (дата обращения: 25.05.2023).
9. ГОСТ 2.301-68 Единая система конструкторской документации [Электронный ресурс]: URL: <https://files.stroyinf.ru/Data/9/912.pdf> (дата обращения: 25.05.2023).
10. Юзабилити-тестирование [Электронный ресурс]: URL: <https://ru.wikipedia.org/wiki/Юзабилити-тестирование> (дата обращения: 25.05.2023).
11. C# и SQL-запросы: что нужно знать программисту [Электронный ресурс]: URL: <https://otus.ru/journal/c-i-sql-zaprosy-chto-nuzhno-znat-programmistu/> (дата обращения: 25.05.2023).
12. Как провести юзабилити-тестирование с респондентом и не провалить его [Электронный ресурс]: URL: <https://habr.com/ru/companies/lamoda/articles/673884/> (дата обращения: 25.05.2023).

# **ПРИЛОЖЕНИЕ А** - **Карты переходов пользователей**

На рисунках 1-3 представлены карты переходов пользователей, таких как «Гость», «Администратор» и «Приемщик».

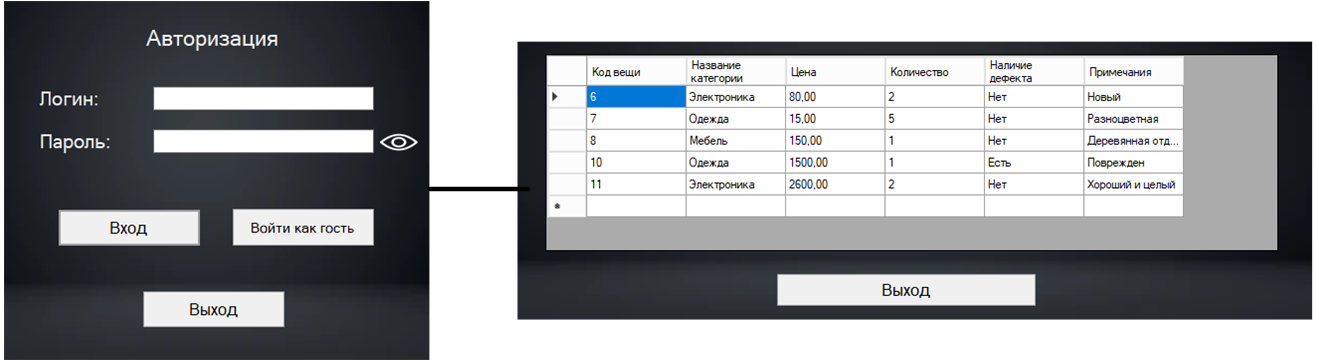


Рисунок 1 - Карта перехода пользователя «Гость»

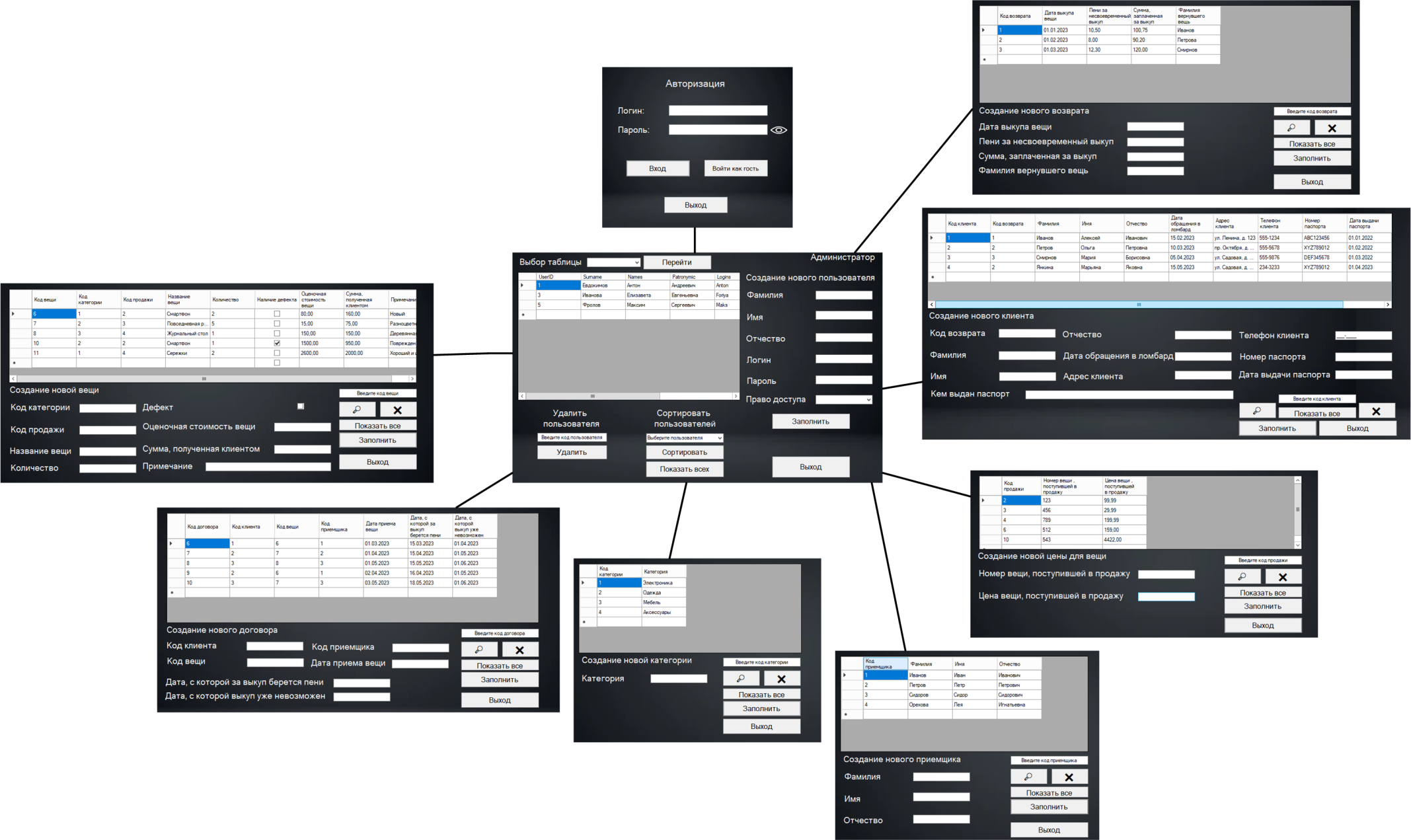


Рисунок 2 - Карта перехода пользователя «Администратор»

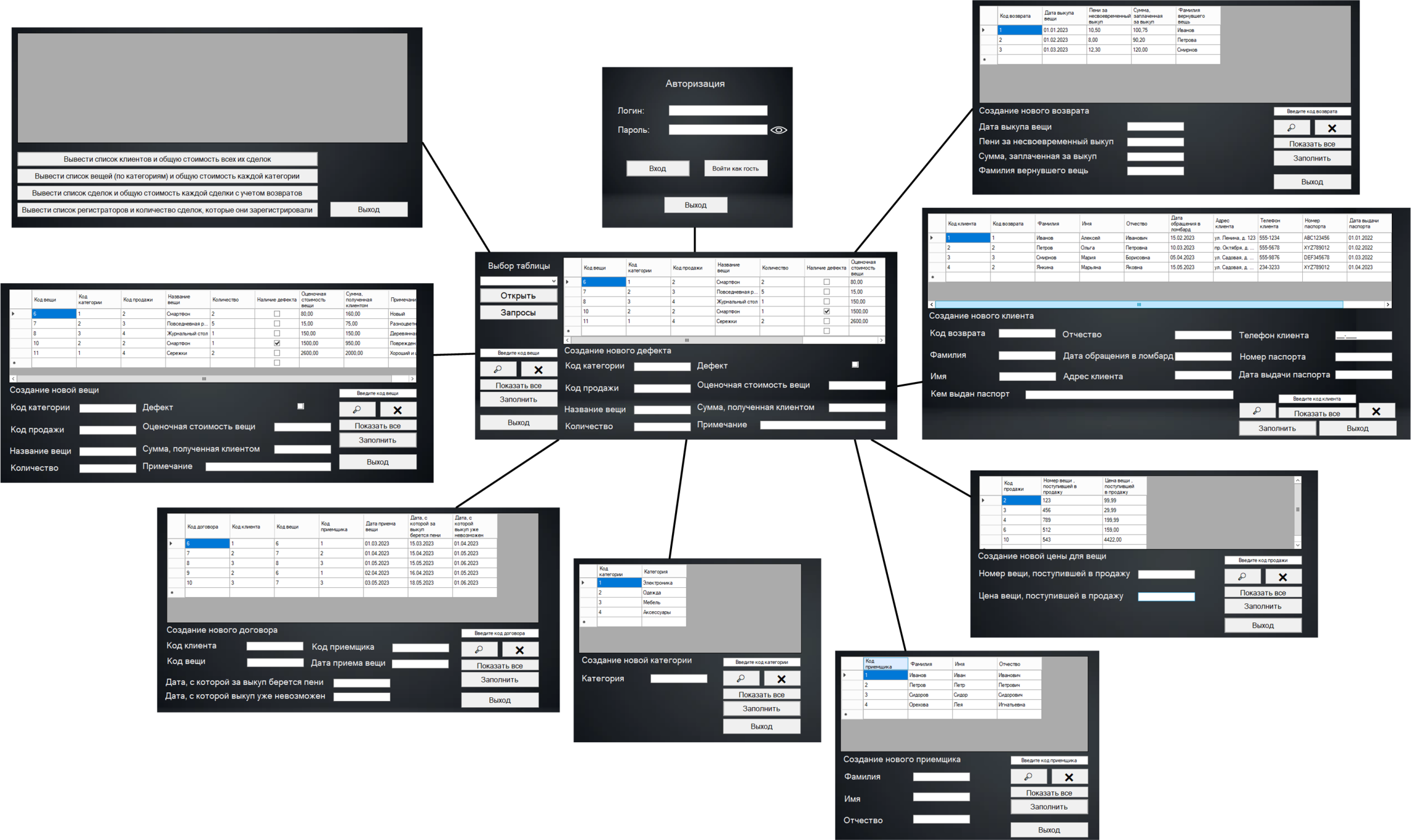


Рисунок 3 - Карта перехода пользователя «Приемщик»