Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Программирование сетевых приложений»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры экономической информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Богданова Е. А. |
|  | \_\_\_.\_\_\_\_.2021 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЗАЦИИ РАБОТЫ БРОКЕРСКОГО АГЕНТСТВА»**

БГУИР КП 1-40 01 02-08 015 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 972304  Савин Илья Дмитриевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2021  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2021

**СОДЕРЖАНИЕ**

[Введение…. 5](#_Toc90479687)

[1 Описание деятельности брокерского агентства 6](#_Toc90479688)

[1.1 Общие понятия и термины 6](#_Toc90479689)

[1.2 Описание организационной структуры брокерского агентства 7](#_Toc90479690)

[2 Постановка задачи на разработку программного средства брокерского агенстства и обзор методов её решения 11](#_Toc90479691)

[3 Функциональное моделирование на основе стандарта IDEF0 12](#_Toc90479692)

[4 Информационная модель системы и её описание 19](#_Toc90479693)

[5 Модели представления системы на основе UML 21](#_Toc90479694)

[5.1 Спецификация вариантов использования системы работы брокерского агентства 21](#_Toc90479695)

[5.2 Описание диаграммы последовательности построения графика цены услуги в различных компаниях 23](#_Toc90479696)

[5.3 Описание диаграммы состояний построения графика цены услуги различных компаний 25](#_Toc90479697)

[5.4 Описание диаграммы классов 27](#_Toc90479698)

[5.5 Описание диаграммы развертывания 29](#_Toc90479699)

[6 Обоснование оригинальных решений по использованию технических и программных средств, не включенных в требование 32](#_Toc90479700)

[7 Описание алгоритмов, реализующих бизнес-логику серверной части проектируемой системы 33](#_Toc90479701)

[7.1 Схема алгоритма решения задачи выбора компании 33](#_Toc90479702)

[7.2 Схема алгоритма построения графика цены услуги различных компаний 34](#_Toc90479703)

[7.3 Схема алгоритма клиент-серверного взаимодействия 35](#_Toc90479704)

[8 Руководство пользователя 37](#_Toc90479705)

[Заключение 38](#_Toc90479706)

[Список использованных источников 39](#_Toc90479707)

[Приложение А (обязательное) Листинг кода 40](#_Toc90479708)

Введение

Инвестирование капитала в ценные бумаги предполагает два пути ведения деятельности. Первый — самостоятельность: инвестор разрабатывает стратегию вложения средств и возлагает на себя полную ответственность за результат сделок. Второй — обращение к профессиональному управляющему. Вне зависимости от выбранного способа инвестиций понадобятся услуги лицензированного брокера. В современных рыночно-трансформационных условиях брокерская деятельность непрерывно развивается. В связи с ростом спроса на услуги брокеров растет спрос на высоко квалифицированных специалистов в этой области. Но для продуктивной работы одного персонала недостаточно, также необходимо программное обеспечение, которое позволит клиентам легко взаимодействовать со специалистами, просматривать рынок в удобной форме, безопасно хранить свои деньги на брокерском счете и выполнять транзакции, не переживая за данные своих кредитных карт.

В городах и регионах сформировался целый спектр брокерских услуг, оказывающих значительное воздействие на обеспеченность жильем и уровень жизни населения их социально-экономическое развитие. Эти факторы и определили актуальность темы данного исследования.

Целью данного курсового проекта является автоматизация процесса принятия решения в условиях риска и неопределённости с помощью статистических методов на примере брокерского агентства.

Поставленная цель потребовала решения следующих задач:

* исследовать процесс работы брокерского агентства;
* ознакомиться с особенностями организации деятельности брокерского агентства;
* проанализировать отдельные направления брокерской деятельности;
* изучить основные функции брокерского агентства;
* описать предметную область;
* выявление рациональной организации брокерской фирмы;
* спроектировать диаграмму классов;
* спроектировать объектную модель;
* протестировать разработанное программное средство;
* оценить перспективы развития программного обеспечения и определить пути его совершенствования.

Объектом исследования является организация, осуществляющая деятельность в сфере брокерских услуг.

# Описание деятельности брокерского агентства

1.1 Общие понятия и термины

Брокер – это посредник, который сводит воедино продавца и покупателя, помогая совершить сделку. В качестве него могут выступать физические или юридические лица. Брокер осуществляет свои действия согласно поручению клиентов. В процессе совершения сделок он использует деньги лиц, заключивших с ним договор. За подобные действия брокер получает вознаграждение, которое может быть фиксированным либо выражается в процентах. Деятельность брокеров подлежит лицензированию [1].

Брокерская лицензия – документ, который позволяет легально заниматься брокерской деятельностью. По сути, для брокеров лицензирование является плюсом к репутации и возможностью открыто рекламировать свои услуги, а для клиента – гарантией того, что выбранный посредник работает законно.

Виды брокеров можно выделять в зависимости от сферы их деятельности. Ниже представлены самые популярные из них:

* биржевой брокер;
* страховой брокер;
* таможенный брокер;
* кредитный брокер;
* фрахтовый брокер.

Биржевой брокер – выступает посредником на фондовой или валютной бирже, продающий и покупающий активы в интересах клиента на наиболее выгодных для него условиях. Для осуществления операций купли-продажи акций, облигаций, валюты и других финансовых инструментов необходим брокерский счет. Брокерский счет – это личный счёт клиента, открытый у одного из брокеров, на котором отображаются все его денежные средства и ценные бумаги.

Страховой брокер оказывает услуги на рынке страхования. Он выступает посредником между страховщиками и их клиентами. Деятельность такого брокера регламентируется лицензиями и договорами.

Таможенный брокер – юридическое или физическое лицо, которое выступает посредником между государственными таможенными органами и предпринимателями, и представляет интересы клиентов, заказывающих услуги по растаможке и таможенному оформлению грузов. Предоставляет услуги оформления экспорта/импорта, планирования внешнеторговой сделки или юридического сопровождения внешнеэкономической деятельности [2].

Кредитный брокер – выступает посредником между банками и заёмщиками в процессе осуществления операций кредитования физических и юридических лиц. В процессе работы кредитные брокеры взаимодействуют со многими финансовыми структурами и компаниями. Предоставляют услуги оптимальной схемы кредитования, переговоры с кредиторами, полное сопровождение проектов клиентов.

Фрахтовый брокер помогает при заключении договоров, предметом которых выступает доставка грузов. Такие компании являются посредниками между собственником груза, которого называют фрахтователем или владельцем судна.

В процессе своей деятельности брокерские компании оказывают своим клиентам посреднические услуги. Все они связаны с выполнением ряда функций. Основные из них представлены ниже:

* консультационные услуги: брокерские компании регулярно изучают изменения государственных актов, что позволяет им получать доход от консультационных услуг;
* анализ рынка и предложений: при обращении клиента к брокеру посредник в первую очередь осуществляет тщательный анализ рынка, при этом брокерские компании имеют достаточно опыта, чтобы быстро собрать информацию;
* помощь в подготовке пакета документов: брокеры уточняют, какие документы потребуются для получения услуги, консультируют, как проще и быстрее подготовить все необходимые бумаги, фактически они выполняют половину технической работы по подготовке и оформлению сделки.

Таким образом, брокеры действуют во всех сферах жизни современного общества. Они могут оказать неоценимую помощь при заключении различных сделок.

1.2 Описание организационной структуры брокерского агентства

Организационная структура брокерского агентства необходима для определения исполнителей деятельности по приведению объектов управления в целевое состояние. От организационной структуры напрямую зависит эффективность создания системы управления агентством [5].

Рассмотрим организационную структуру брокерского агентства (рис. 1.2)



Рисунок 1.2 – Организационная структура брокерского агентства

На предприятии представлена линейно-функциональная структура управления, основанная на выделении функциональных структур, т.е. подразделений управления по сферам деятельности (функциям): производство, закупки, продажи, персонал, финансы, бухучет и т.д.

Преимущества линейных структур:

* четкая система взаимных связей внутри функций и в соответствующих им подразделениях;
* четкая система единоначалия — один руководитель сосредотачивает в своих руках руководство всей совокупностью функций, составляющих деятельность;
* ясно выраженная ответственность;
* быстрая реакция исполнительных функциональных подразделений на прямые указания вышестоящих.

Главным управляющим звеном в брокерском агентстве является директор. Он имеет право первой подписи и распоряжения финансовыми и имущественными средствами общества, осуществляет общее и оперативное руководство предприятием, представляет фирму во внешних организациях, организует выполнение планов производства и реализации продукции. У главного директора есть первый заместитель, который берет некоторые функции директора на себя, а также напрямую общается с управляющими разных отделов.

Главный бухгалтер в соответствии с функциональными обязанностями, определенными Законом о бухгалтерском учете, организует бухгалтерский учет и составление отчетности на предприятии.

Начальник отдела продаж – это человек, который постоянно проводит мониторинг рынка, непосредственно работает с клиентами и устраняет их претензии. Также начальник отдела продаж занимается согласованием сроков получения и отправки партий товара.

Главный юрист обеспечивает защиту в судебных разбирательствах. Составляет разнообразные договоры между клиентами и брокерским агентством. Проводит правовые консультации по всем видам предоставляемых услуг в агентстве.

Заместитель по финансам вырабатывает экономическую политику фирмы, направленную на получение максимальной прибыли, осуществляет разработку перспективного и текущего финансового планирования, ведет постатейный анализ затрат, определяет и регулирует экономическую и финансовую устойчивость.

Главный маркетолог изучает рынок и рыночные тенденции, выбирает целевой рынок, утверждает стратегию развития товара, осуществляет тактическое управление товаром компании, а также управляет отношениями с клиентами.

Казначей компании проводит тщательный анализ общей экономической обстановки и финансового состояния агентства, регулирует и контролирует направление и силу потоков денежных средств.

Старший брокер обеспечивает проведение посреднических операций, выбирает выгодное решение из ряда решений, проводит статистический пересчет показателей для поиска решений.

Заместитель по страхованию обеспечивает разработку страховых актов агентства и подписание страховых договоров, разработку новых страховых услуг.

Главный менеджер занимается поиском потенциальных клиентов, согласованием условий работы с клиентами, оформляет и заключает договоры с клиентами, открывает счета, отслеживает сроки и суммы оплаты услуг клиентами.

Управляющий по рискам исследует бизнес-процессы, оценивает сильные и слабые стороны компании, анализирует угрозы возникновения потерь и определяет источники опасности, участвует в разработке стратегии управления рисками.

На предприятии существуют также отделы, которые занимаются вопросами контроля и управления качеством продукции, разработкой новых видов продукта, обеспечением бесперебойного технологического процесса.

# Постановка задачи на разработку программного средства брокерского агенстства и обзор методов её решения

Компьютерное приложение для брокерского агентства призвано улучшить взаимодействие между клиентами и персоналом, а также автоматизировать и повысить эффективность обработки заказов услуг брокерского агентства. Оно позволит пользователям получать полный спектр услуг компании прямо со своего персонального компьютера, а брокерскому агентству повысить продажи услуг.

Также приложение будет широко использоваться внутри брокерского агентства – оно позволит брокерам всегда быть на связи со своими клиентами, где бы они не находились, так как разработанное приложение позволит проводить анализ рынка, выступать посредником в сделках и оставаться на связи со своим клиентом из любого места.

Управляющему компании приложение призвано предоставлять всю информацию о сотрудниках и клиентах, информации о доходах, расходах компании. Разработанное приложение призвано облегчить и улучшить работу агентства.

Основными преимуществами приложения являются: минимизация потерь из-за ошибок работников, автоматизация процесса и повышения эффективности обработки заказов, возможность обработки больших объёмов данных.

Решение основной задачи данного проекта будет заключаться в выполнении ряда составных задач:

* хранение информации обо всех проведённых сделках;
* хранение информации обо всех клиентах и сотрудниках (управляющий должен иметь всю информацию о сотрудниках и выполненных ими сделках, а также обо всех клиентах, которые обслуживаются в компании);
* предоставление пользователю графической отчётности в виде круговых диаграмм;
* предоставление удобного пользовательского интерфейса как для сотрудников, так и для клиентов компании;
* обеспечения защиты и безопасности хранения данных брокерской компании;
* предоставления необходимых условий для организации решения статистическим методом.

# Функциональное моделирование на основе стандарта IDEF0

IDEF0 – нотация графического моделирования, используемая для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции. Нотация IDEF0 является одной из самых популярных нотаций моделирования бизнес-процессов. К ее особенностям можно отнести:

* контекстная диаграмма;
* поддержка декомпозиции;
* выделение 4 видов стрелок.

Контекстная диаграмма - самая верхняя диаграмма, на которой объект моделирования представлен единственным блоком с граничными стрелками. Эта диаграмма называется A-0. Стрелки на этой диаграмме отображают связи объекта моделирования с окружающей средой. Диаграмма A-0 устанавливает область моделирования и ее границу.

Поддержка декомпозиции -нотация IDEF0 поддерживает последовательную декомпозицию процесса до требуемого уровня детализации. Дочерняя диаграмма, создаваемая при декомпозиции, охватывает ту же область, что и родительский процесса, но описывает ее более подробно. При декомпозиции стрелки родительского процесса переносятся на дочернюю диаграмму в виде граничных стрелок.

Выделение 4 видов стрелок.Выделяются следующие виды стрелок: Вход, Выход, Механизм, Управление. Входы преобразуются или расходуются процессом, чтобы создать то, что появится на его выходе. Управления определяют условия, необходимые процессу, чтобы произвести правильный выход. Выходы – данные или материальные объекты, произведенные процессом. Механизмы идентифицируют средства, поддерживающие выполнение процесса. Таким образом, блок IDEF0 показывает преобразование входа в выход с помощью механизмов с учетом управляющих воздействий.

На рисунке 1.3 представлена контекстная диаграмма верхнего уровня, где:

* вход: ожидания клиентов, информация о рынке, денежные средства от клиентов;
* механизмы: консультант, брокер;
* управление: ценовая политика, устав агентства, закон о страховании и Декретом Президента от 2000-09-28г. № 20 «О совершенствовании регулирования страховой деятельности»;
* выход: выручка, выполненные услуги.

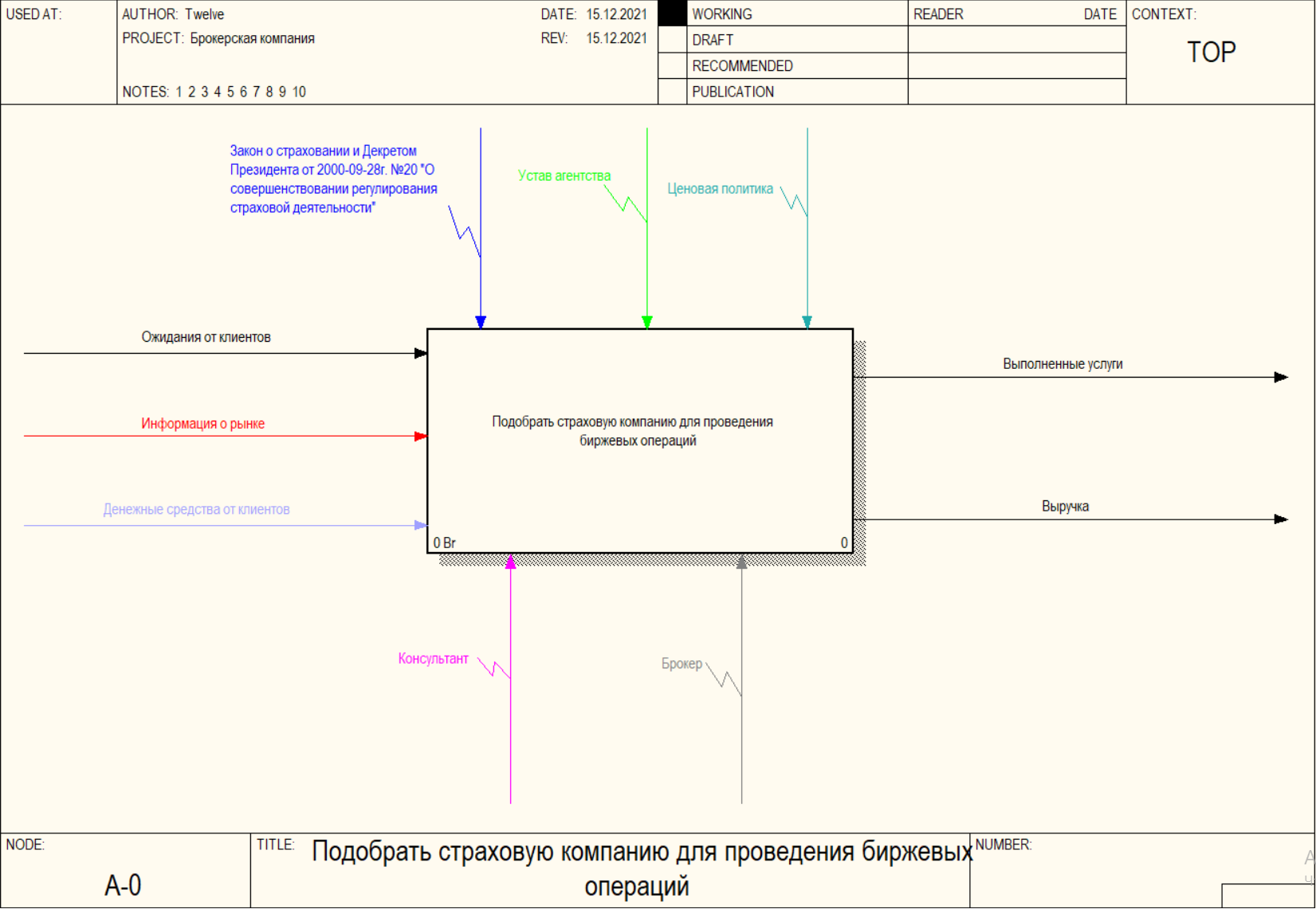


Рисунок 1.3 – Контекстная диаграмма верхнего уровня

Декомпозиция контекстной диаграммы представлена на рисунке 1.4. Она описывает разделение основного процесса на 4 процесса:

* проконсультировать клиента;
* проанализировать риски для агентства;
* определить страховую компанию соотетствующую заявленным параметрам;
* заключить сделку.

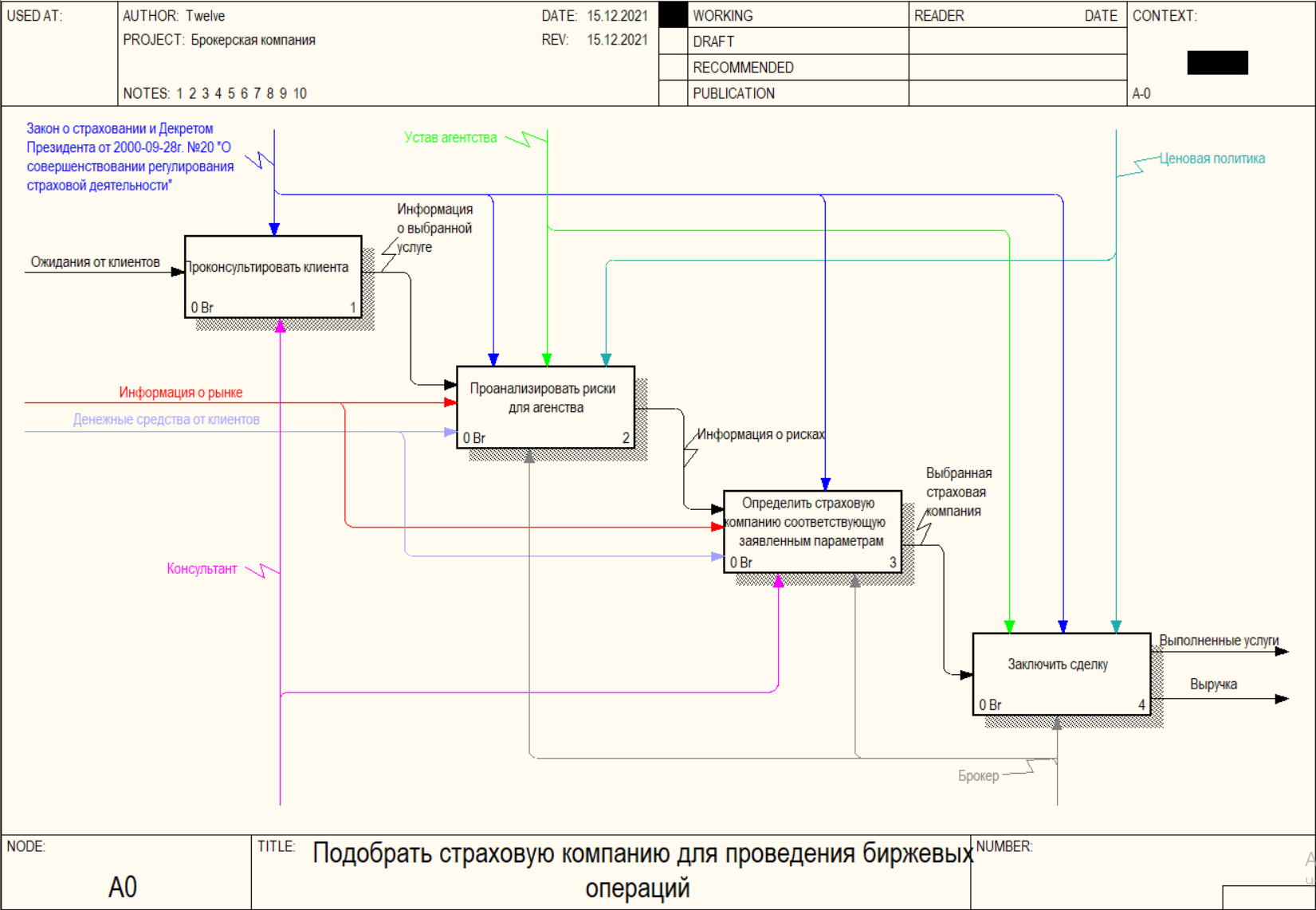


Рисунок 1.4 – Декомпозиция контекстной диаграммы верхнего уровня

Далее рассмотрим декомпозиция процесса «Консультация клиента», которая представлена на рисунке 1.5

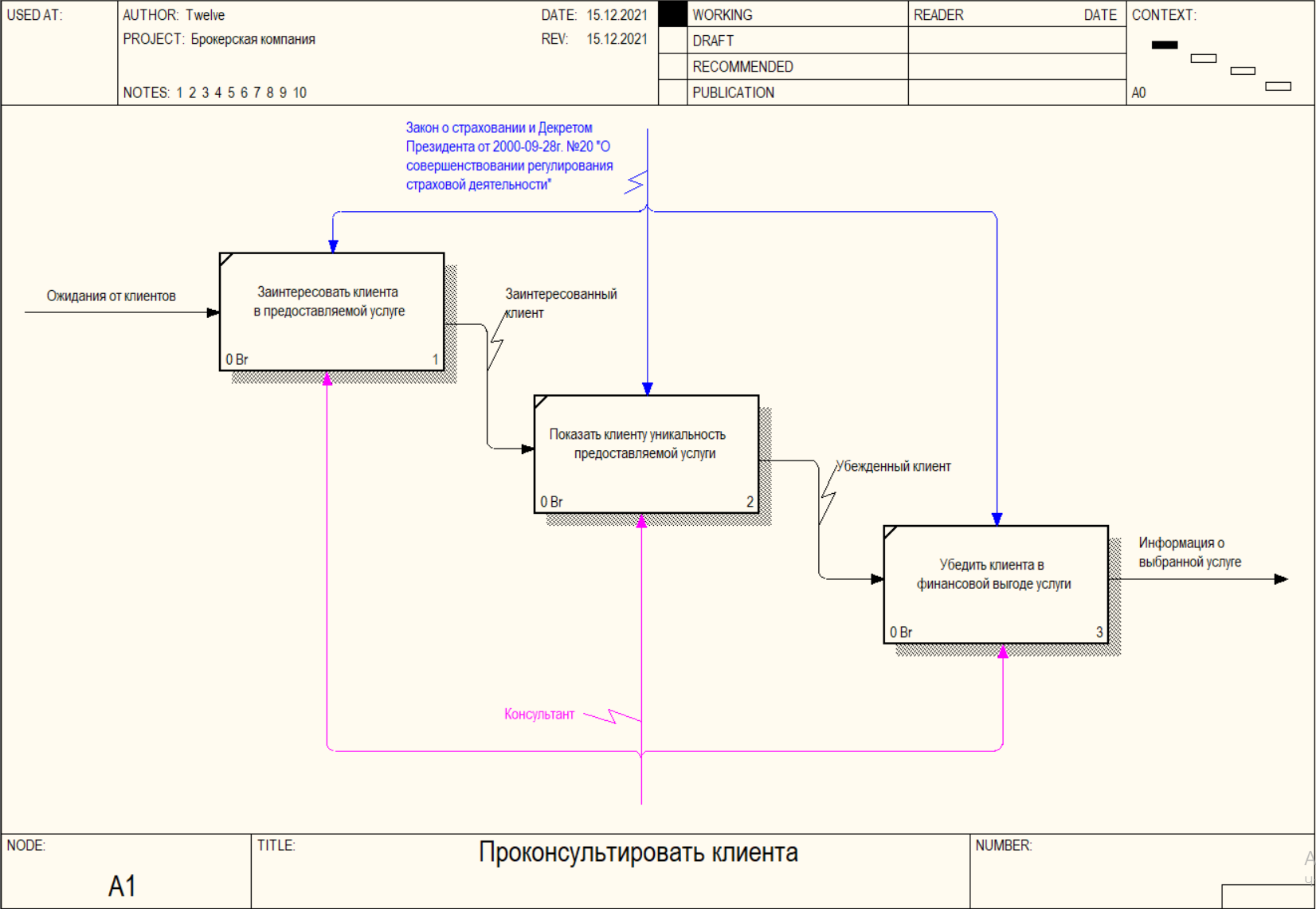


Рисунок 1.5 – Декомпозиция процесса «Консультация с клиентом»

Процесс «Консультация с клиентом» состоит из 3 более простых процессов. После того как клиент был успешно проконсультирован, необходимо провести исследование рынка страховых услуг. Данный процесс представлен на рисунке 1.6

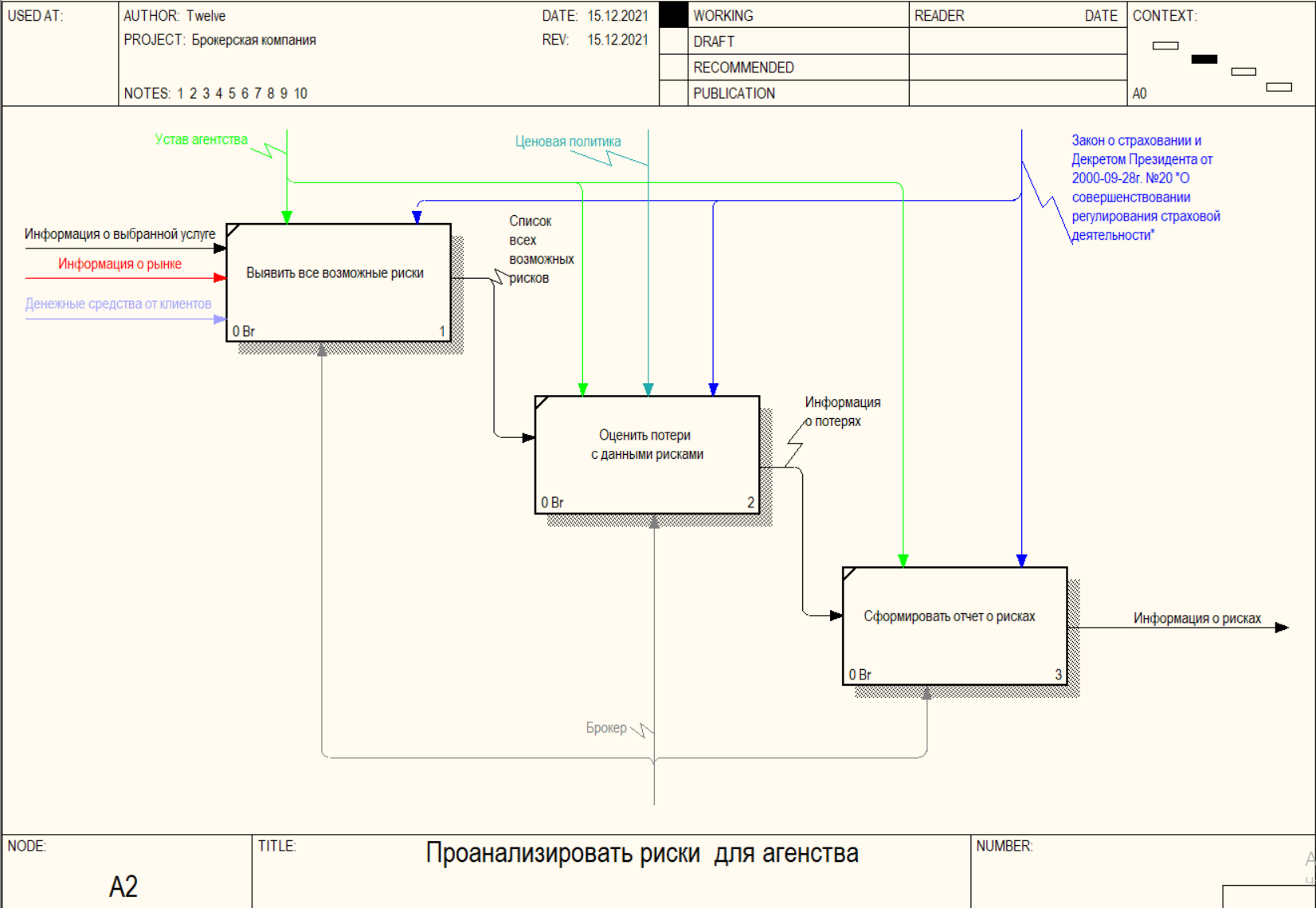


Рисунок 1.6 – Декомпозиция процесса «Анализ рынка страховых услуг»

На рисунке видно, что для выполнения процесса необходимо выявить все возможные риски, оценить потери с данными рисками и сформировать отчет о рисках, чтобы предоставить информацию о рисках в более наглядной форме. После анализа рынка необходимо заключить с клиентом соглашение. Данный процесс представлен на рисунке 1.7

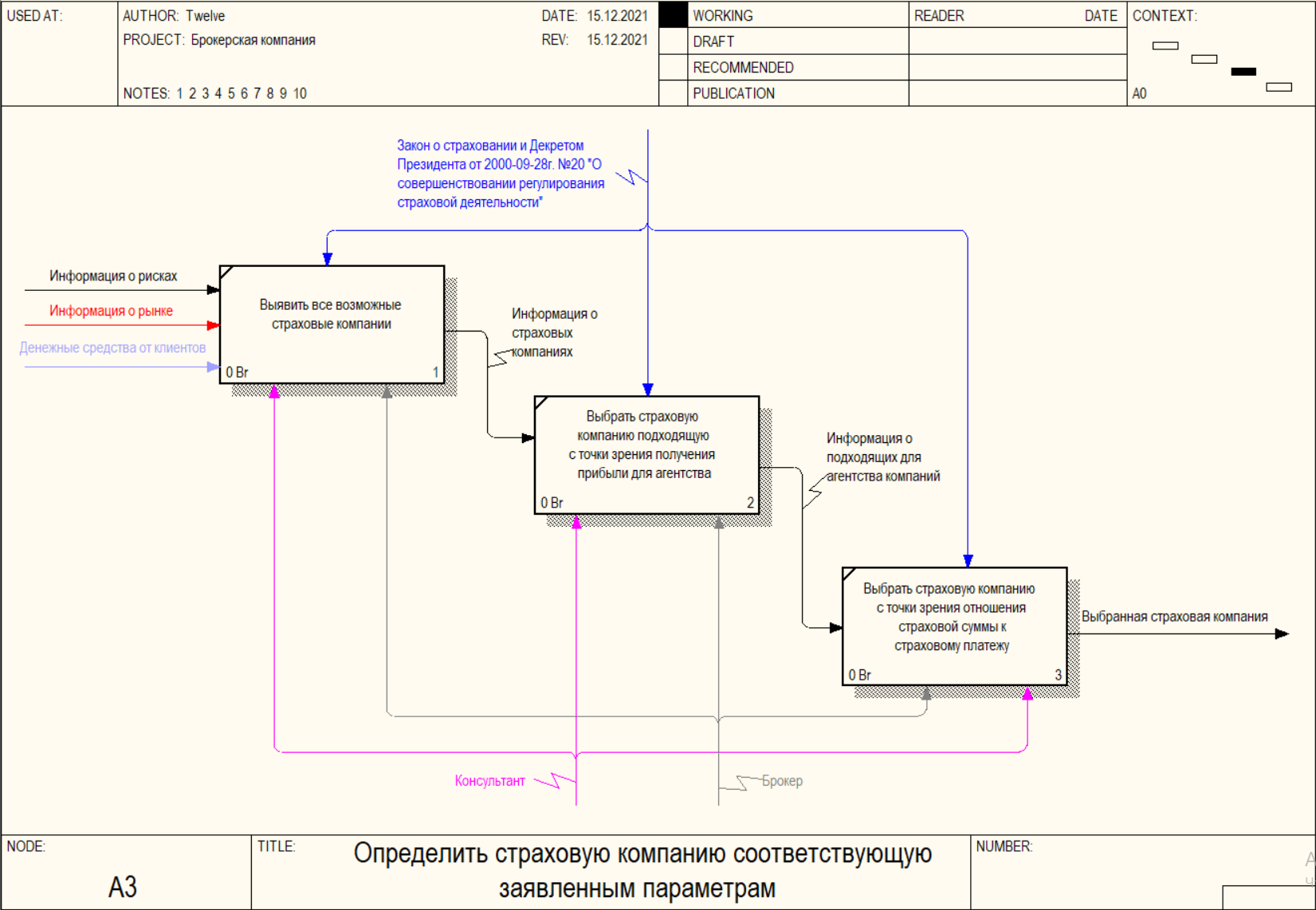


Рисунок 1.7 – Декомпозиция процесса «Совершение сделки с клиентом»

На рисунке видно, что сначала происходит выявление всех возможных страховых компаний, затем выбирается подходящая для агентства страховая компания, а только потом выбирается подходящая для клиента страховая компания. После анализа рынка необходимо заключить сделку. Процесс этого проиллюстрирован на рисунке 1.8

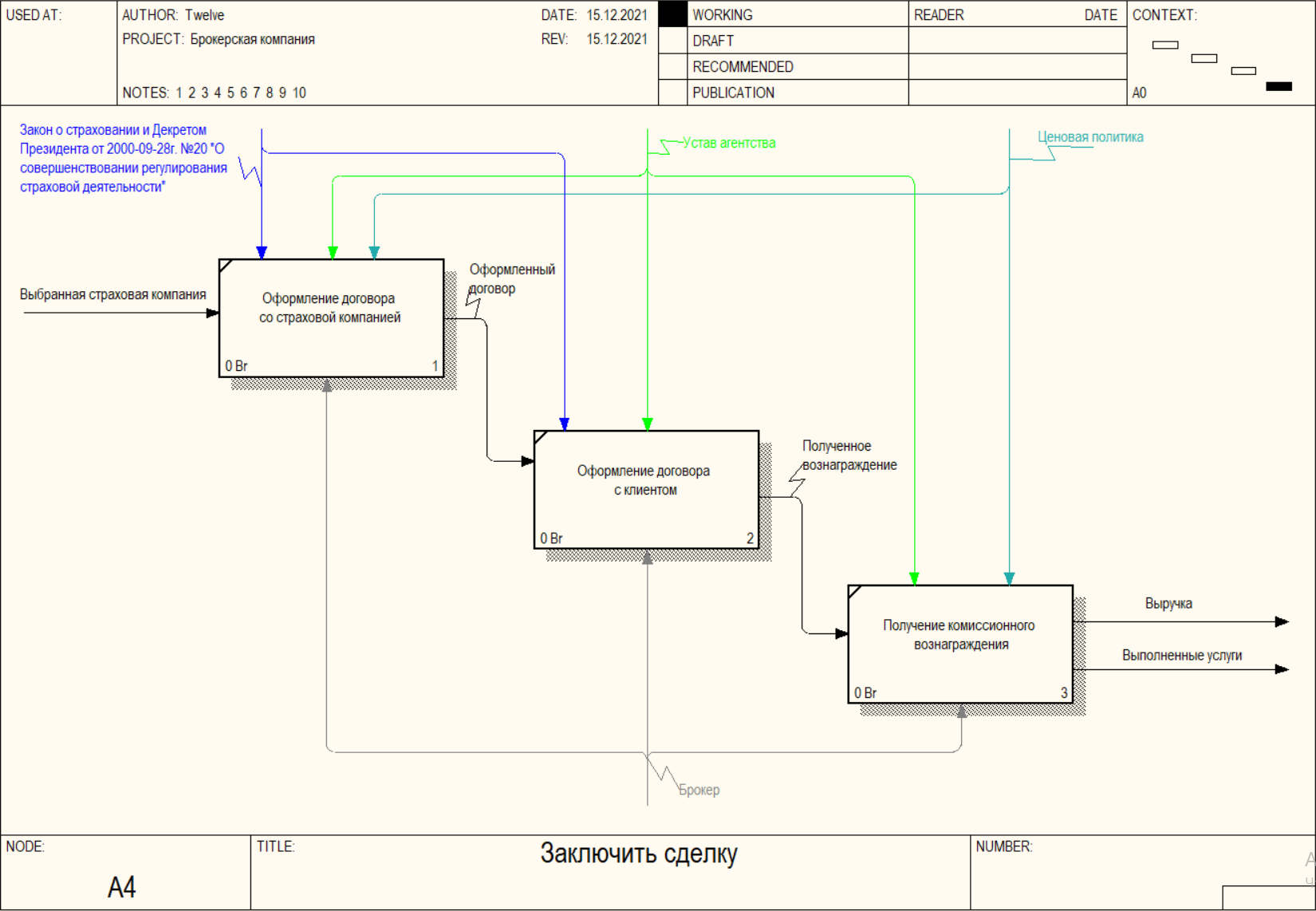


Рисунок 1.8 – Декомпозиция процесса «Оформление юридического соглашения со страховой компанией»

Все эти действия в совокупности описывают деятельность страхового агентства.

Разложение процесса на подпроцессы позволяет глубоко и в деталях понять принципы его работы и детали разрабатываемой информационной системы.

Анализ деятельности компании является неотъемлемой частью разработки планов дальнейшего развития компании, а также автоматизации её деятельности. А графическое представление различных процессов, структур компании, в свою очередь, обеспечивает беспроблемное восприятие всех процессов и структур, действующих в компании, для достижения целей.

# Информационная модель системы и её описание

Методология моделирования IDEF1X, являясь расширением стандарта IDEF1, предназначена для описания данных. В ее основе лежит язык семантического моделирования, основанного на концепции «сущность — связь», позволяющей определять данные и связи между ними. Чаще всего такая методология используется для описания данных в целях последующей автоматизации их обработки с помощью систем управления базами данных. Таким образом, можно говорить о том, что модели данных в нотации IDEF1X используются для создания баз данных [8].

Основными элементами модели IDEF1X являются сущности, атрибуты и отношения.

Как правило, в зависимости от глубины описания, выделяют три класса логических моделей данных:

* + диаграмма «Сущность — связь»;
  + модель данных, основанная на ключах;
  + полная атрибутивная модель.

Диаграмма «сущность — связь» используется для описания данных на верхнем уровне в целях визуализации данных для руководителей и экспертов, не обладающих специальными знаниями в области моделирования.

Модель стандарта IDEF1.X для разрабатываемой системы представлена на рисунке 4.4

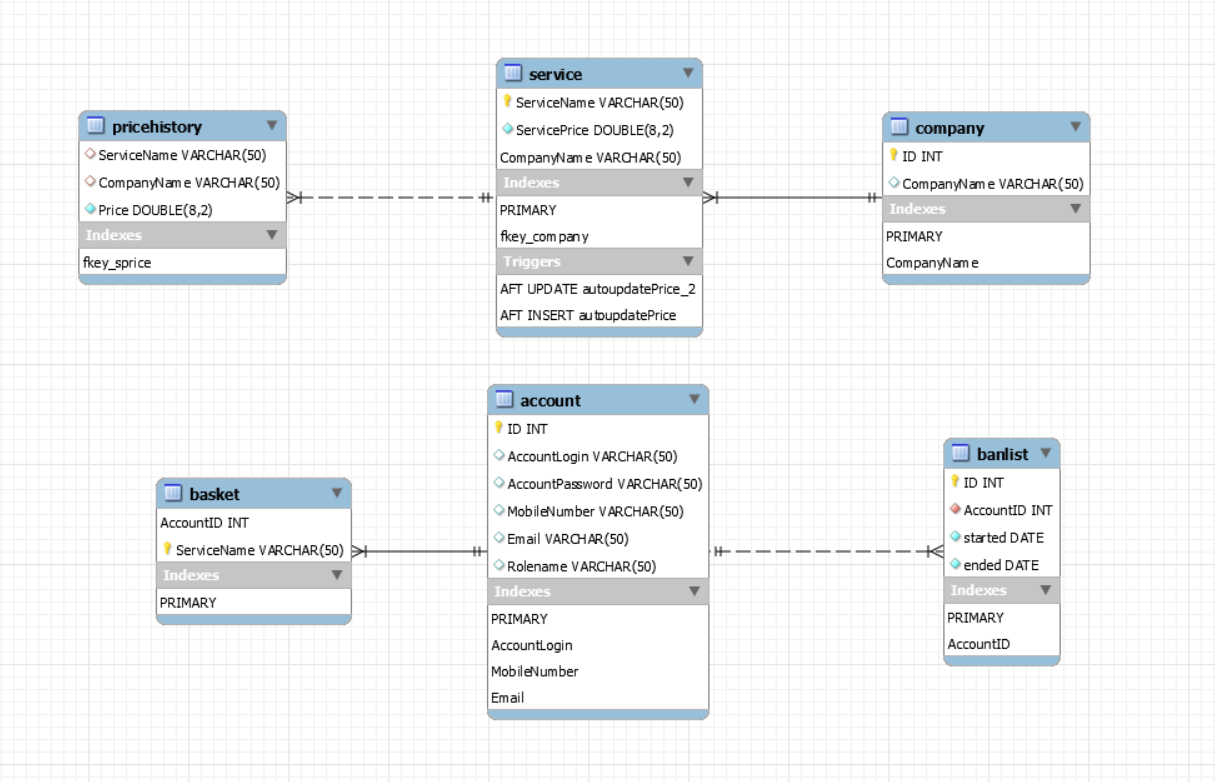


Рисунок 4.4 – Диаграмма IDEF1.X системы работы брокерского агентства

На данной диаграмме присутствуют следующие сущности: Company, Service, Account, BanList, PriceHistory, Basket.

Сущность Company» представляет собой страховую компанию, которая будет иметь услуги.

Сущность «Service» представляет собой услуги, которые может выполнить страховая компания. Они уникальны в рамках одной компании, несмотря на то, что они могут быть похожими.

Сущность «Account» представляет собой аккаунт, который может быть аккаунтом брокера, консультанта или клиента в зависимости от значения поля «Rolename». Также каждому аккаунту может быть назначен бан, суть которого отражается в сущности «BanList».

Для выбора решения необходимы статистические данные, которые отражаются в сущности «PriceHistory», где хранятся истории цен услуг компаний. При добавлении записи в таблицу «Service» или изменении отдельной услуги, срабатывает триггер, который автоматически добавляет запись о цене в таблицу «PriceHistory».

Каждый клиент может покупать услуги суть которых отражена в сущности «Service», количество услуг и информация об услуги хранится в корзине – сущность «Basket».

Для привидения базы данных к третьей нормальной форме, воспользуемся определениями первой, второй и третьей нормальных форм.

Отношение находится в первой нормальной форме, если все его атрибуты являются простыми, все используемые домены содержат только скалярные значения. Также не должно быть повторений строк в таблице.

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме и каждый не ключевой атрибут неприводимо зависит от первичного ключа. Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Отношение находится в третьей нормальной форме, когда находится во второй нормальной форме и каждый не ключевой атрибут не транзитивно зависит от первичного ключа, т.е., второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Разработанная модель соответствует вышеперечисленным требованиям, следовательно, находится в третьей нормальной форме

# Модели представления системы на основе UML

В данном разделе будет продемонстрировано моделирование информационной системы с помощью стандарта UML, который использует графические обозначения для создания абстрактной модели системы и предназначен для определения, визуализации, проектирования и документирования в основном программных систем. UML позволяет описать систему практически со всех возможных точек зрения и разные аспекты поведения системы.

5.1 Спецификация вариантов использования системы работы брокерского агентства

На диаграммах вариантов использования отображается взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему. Из диаграмм вариантов использования можно получить довольно много информации о системе. Этот тип диаграмм описывает общую функциональность системы. Пользователи, менеджеры проектов, аналитики, разработчики, специалисты по контролю качества и все, кого интересует система в целом, могут, изучая диаграммы вариантов использования, понять, что система должна делать [7].

Вариант использования применяется для спецификации общих особенностей поведения системы или другой сущности без рассмотрения ее внутренней структуры (например, оформление заказа на покупку товара, получение информации о кредитоспособности клиента, отображение графической формы на экране монитора).

Актер – это внешняя по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для решения определенных задач. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой [7].

Имя актера должно быть достаточно информативным с точки зрения семантики, например клиент банка, продавец магазина, пассажир авиарейса, водитель автомобиля, сотовый телефон. Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется. Для актера имеет значение только его внешнее представление, т.е. то, как он воспринимается со стороны системы.

На рисунке 5.1 представлена диаграмма вариантов использования системы работы брокерского агентства.

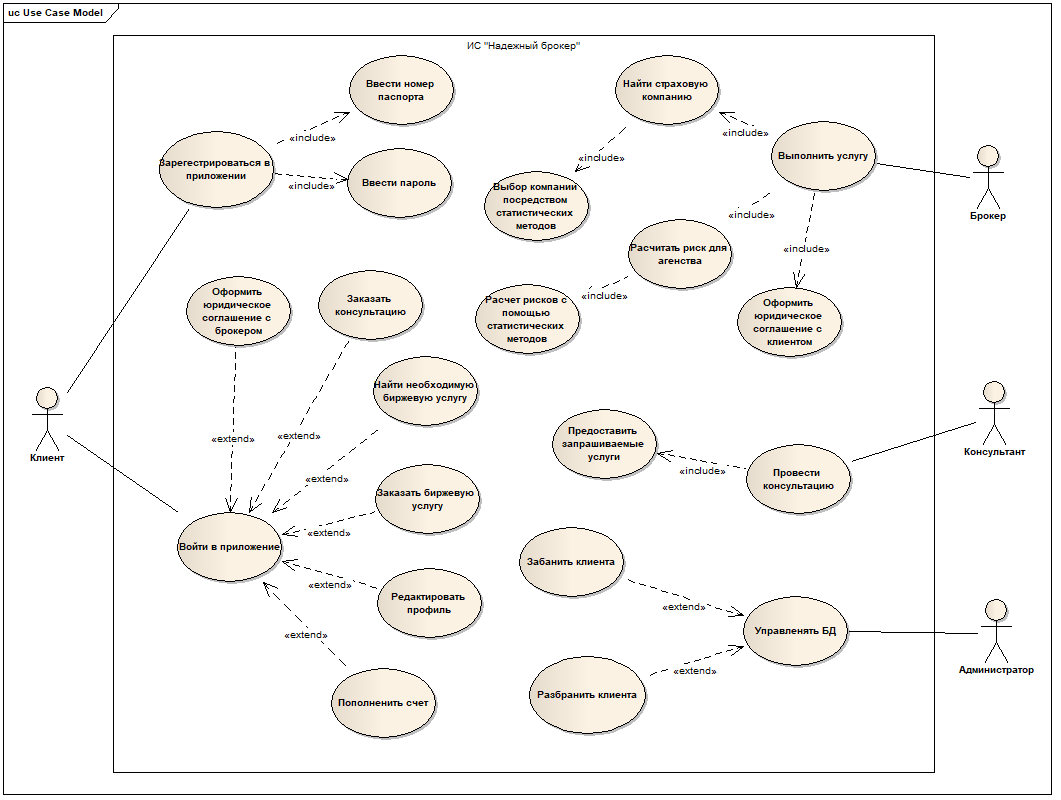


Рисунок 5.1 – Диаграмма вариантов использования системы работы брокерского агентства

Как видно, на данной диаграмме присутствуют такие актеры, как: администратор, клиент, консультант и брокер.

Администратор работает с сервером, может просматривать подключение к серверу, банить пользователей по ip-адресу, а также другие административные функции.

Консультант работает с клиентами. Когда клиенту необходима консультация он отправляет заявку на консультирование. Консультанты видят данную заявку и могут ее обработать, то есть войти в чат с клиентом и провести консультацию напрямую.

Брокер занимает расчетом показателей для выбора оптимального решения, оформляет юридические договоры с клиентом, ищет страховую компанию для клиента и рассчитывает риски для агентства.

Клиент же в свою очередь может зарегистрироваться в программе, пополнить счет кошелька, заказать услугу, заказать консультацию, а также другие немало важные функции.

5.2 Описание диаграммы последовательности построения графика цены услуги в различных компаниях

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. При этом учитываются два аспекта: во-первых, взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Во-вторых, можно рассматривать структурные особенности взаимодействия объектов. Для представления структурных особенностей передачи и приема сообщений между объектами используется диаграмма кооперации [7].

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. На этих диаграммах изображаются только те объекты, которые непосредственно участвуют во взаимодействии т.к. ключевым моментом является именно динамика взаимодействия объектов во времени и не используются возможные статические ассоциации с другими объектами. При этом диаграмма последовательности имеет два измерения. Одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Второе измерение – вертикальная временная ось, направленная сверху вниз [10]. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то его линия жизни должна начинаться в верхней части диаграммы и заканчиваться в нижней части.

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы.

На рисунке 5.2 представлена диаграмма последовательности построения графика цены услуги в разных компаниях за указанные период времени

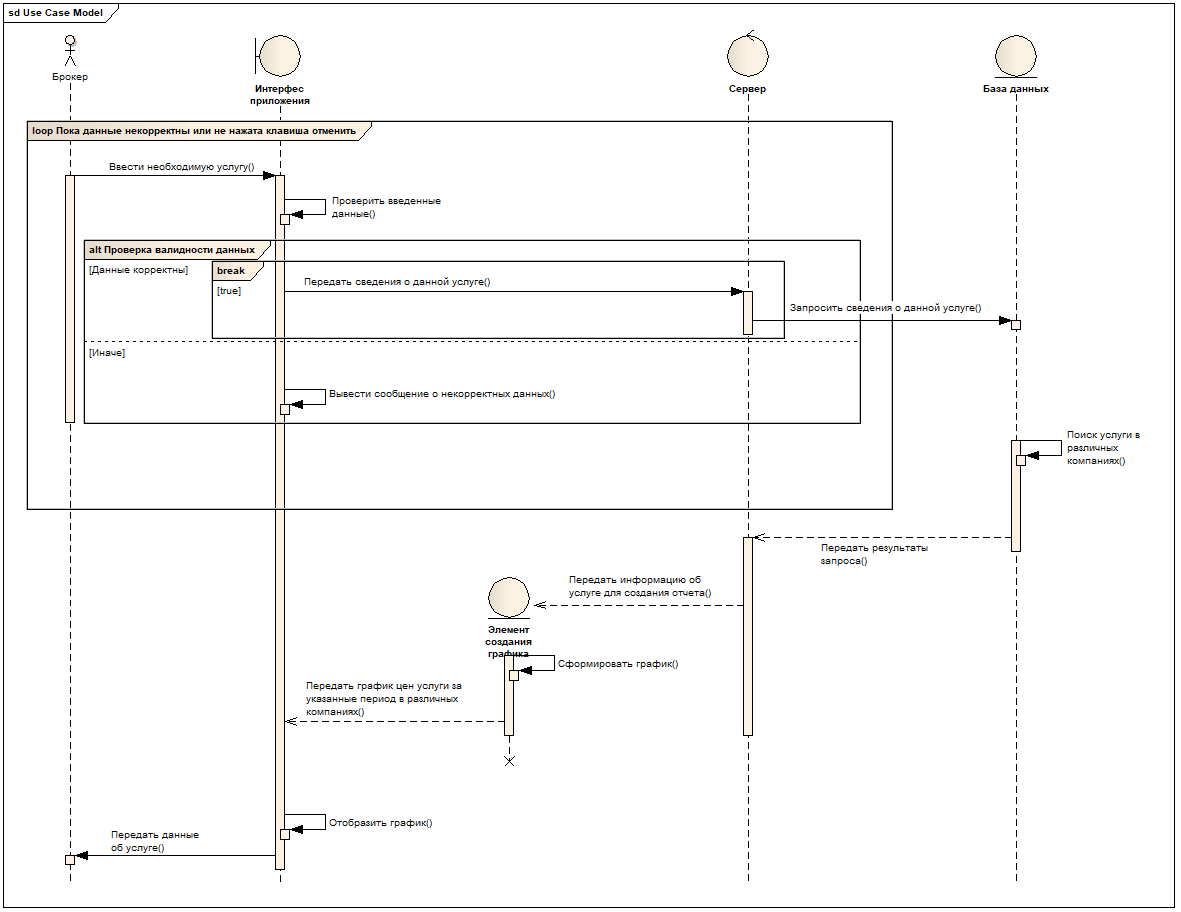


Рисунок 5.2 – Диаграмма последовательности построения графика цены услуги в различных компаниях за указанные период времени

На данной диаграмме видно, что для построения графика необходимо ввести корректные данные об услуге. В этом случае информация будет отправлена на сервер, который запросит данные об услуге в базе данных и передаст управление объекту, который сгенерирует график и уничтожится.

Если же данные будут введены некорректно, то брокер должен будет ввести данные заново или отменить действие. После того, как график будет построен, он будет выведен на экран и брокер сможет получить необходимую информацию.

5.3 Описание диаграммы состояний построения графика цены услуги различных компаний

Главное предназначение диаграммы состояний – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Чаще всего диаграммы состояний используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы [7].

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Автомат описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат [7].

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени.

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий . В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность, в которую может только входить стрелка, соответствующая переходу.

На рисунке 5.3 представлена диаграмма состояний построения графика цены услуги в разных компаниях за указанные период времени

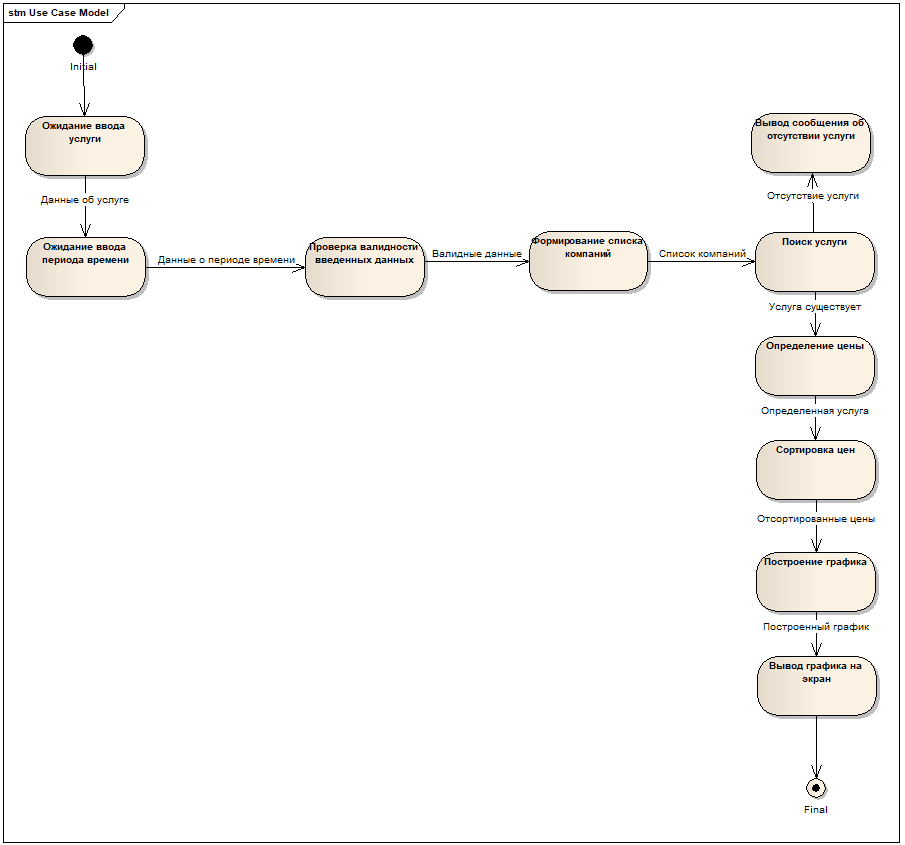


Рисунок 5.3 - Диаграмма состояний построения графика цены услуги в разных компаниях за указанные период времени

На этой диаграмме видно, что для начала в системе ожидаются входные данные, замем они проверяются на корректность. После этогоформируется список компаний и производится поиск услуги. В случае, если услуга есть, определяется ее цена, сортируется список цен этой услуги в различных компаниях и затем строится график, который отображается на экран. Если услуги не существует, тогда выводится сообщений об отстутствии услуги.

5.4 Описание диаграммы классов

Диаграммы классов при моделировании объектно-ориентированных систем встречаются чаще других. На таких диаграммах отображается множество классов, интерфейсов, коопераций и отношений между ними. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Кроме того, диаграммы классов составляют основу еще двух диаграмм – компонентов и развертывания. Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Классом называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Например, класс «Стена» описывает объекты с общими свойствами: высотой, длиной, толщиной, и т.д. При этом конкретные стены будут рассматриваться как отдельные экземпляры класса «Стена». У каждого класса есть имя, (простое или составное, к которому спереди добавлено имя пакета, в который входит класс). Имя класса в пакете должно быть уникальным. Класс реализует один или несколько интерфейсов [7].

Атрибут – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого класса. Класс может иметь любое число атрибутов или не иметь их вовсе. В языках высокого уровня, таких, как С++, Java, атрибуты соответствуют переменным, объявленным в классе. Например, у любой стены есть высота, ширина и толщина. Атрибуты представлены в разделе, расположенном под именем класса; при этом указываются их имена, и иногда начальное значение.

Операция – это некоторый сервис, который предоставляет экземпляр или объект класса по требованию своих клиентов (других объектов, в том числе и экземпляров данного класса). Класс может содержать любое число операций или не содержать их вовсе. В языках высокого уровня, таких, как С++, Java, операции соответствуют функциям, объявленным в классе. Операцию можно описать более подробно, указав имена и типы параметров, их значения, принятые по умолчанию и тип возвращаемого значения [7].

Диаграмма классов разрабатываемой системы представлена на рисунке 5.4

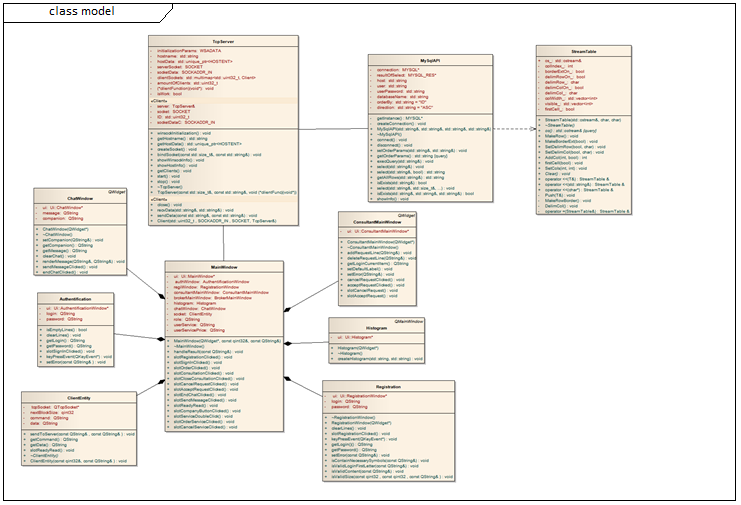


Рисунок 5.4 – Диаграмма классов системы работы брокерского агентства

На данной диаграмме представлено множество классов. Рассмотрим данные классы более подробно.

Класс TcpServer – отражает сервер. В нем содержатся поля для инициализации библиотеки socket, а также для хранения данных о подключенных клиентах и данных о хосте. Также в нем есть вложенный класс Client, который отвечат за единичную сущность, подключенную к клиенту. Объекты класса Client автоматически создаются при подключении клиентов и удаляются при их отключении. В классе клиент есть поля для сокета клиента и его идентификаторы и метода отправки и получения данных.

Класс MainWindow – отражает главное окно клиента. В него включены множество окон, которые показываются при совершении определенных действий, также в этом классе есть множество слотов, которые служат обработчиками сигналов, которые посылаются при совершении каких-либо действий.

Класс MySqlApi – отражает интерфейс для обращения к базе данных. Он включает поле, которое является непосредственно соединением к БД, а также поля для хранения результатов запросов. Среди его методов множество различных селектов, которые либо выводят данные на экран, либо извлекают строку или множество строк, а также есть метод общего назначения, которые выполняет любой запрос.

Класс Registration отражает окно регистрации. Данный класс содержит поля, которые являются введенными данными, а также множество методов для валидации.

Класс Authentification отражает окно аутентификации. Данный класс содержит поля, которые являются введенными данными, а также геттеры для этих полей, необходимые для отправки данных на сервер.

Класс StreamTable отвечает за построение таблиц в консольном приложении на сервере. Он содержит непосредственно поле потока, а также различные вспомогательные поля, отражаются дополнительные возможности, такие как отключение рамки, установление символа-разделителя.

Класс Histogram отвечает за построение и отображение гистограммы. Он содержит единственный метод, который непосредственно и строит гистограмму.

Класс ClientEntity отвечает за принятие и отправку данных на сервер. Он фактически и определеяет протокол общения между сервером и клиентом вместе с классом Client на сервере. Содержит поле сокета сервера и поле для полученных данных и команды.

Класс ChatWindow – отражает окно чата для общения консультанта с клиентом. Содержит поле адресата и поле сообщения. А также множество слотов для обработки сигналов.

5.5 Описание диаграммы развертывания

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если создается простая программа, которая может выполняться локально на компьютере пользователя, не используя никаких распределенных устройств и сетевых ресурсов, то необходимости в разработке дополнительных диаграмм нет. Однако при создании корпоративных или распределенных приложений требуется визуализировать сетевую инфраструктуру программной системы.

Сложные программные системы могут реализовываться в сетевом варианте, на различных вычислительных платформах и технологиях доступа к распределенным базам данных. Наличие локальной корпоративной сети требует решения целого комплекса дополнительных задач рационального размещения компонентов по узлам этой сети, что определяет общую производительность программной системы.

Интеграция программной системы с Интернетом определяет необходимость решения дополнительных вопросов при проектировании системы, таких как обеспечение безопасности и устойчивости доступа к информации для корпоративных клиентов. Эти аспекты в немалой степени зависят от реализации проекта в форме физически существующих узлов системы, таких как серверы, рабочие станции, брандмауэры, каналы связи и хранилища данных.

Технологии доступа и манипулирования данными в рамках общей схемы "клиент-сервер" также требуют размещения больших баз данных в различных сегментах корпоративной сети, их резервного копирования, архивирования, кэширования для обеспечения необходимой производительности системы в целом. С целью спецификации программных и технологических особенностей реализации распределенных архитектур необходимо визуальное представление этих аспектов.

Первой из диаграмм физического представления является диаграмма компонентов. Вторая форма физического представления программной системы - это диаграмма развертывания (размещения).

Диаграмма развертывания (deployment diagram) - диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений - маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения (run-time). При этом представляются только те компоненты программы, которые являются исполнимыми файлами или динамическими библиотеками. Компоненты, не используемые на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания, разрабатываемой системы представлена на рисунке 5.5

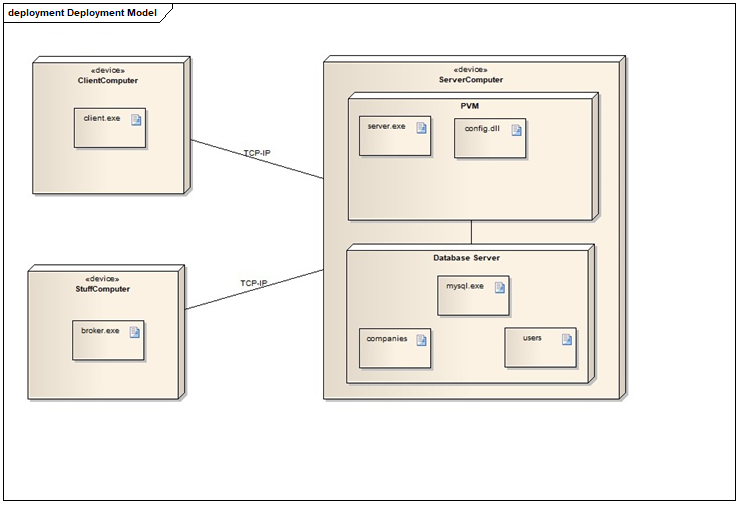


Рисунок 5.5 - Диаграмма развертывания системы работы брокерского агентства

При рассмотрении всех выше перечисленных диаграмм можно сделать вывод, что модели представления работы системы значительно упрощают понимание процесса разработки программы, они помогают четко описать сложную систему, показать взаимосвязь между требованиями, сосредоточиться на их важных аспектах, также визуализировать их.

# Обоснование оригинальных решений по использованию технических и программных средств, не включенных в требование

Для разработки данного приложения был необходим объектно- ориентированный язык программирования, поддерживающий работу с оконными приложениями и базами данных. Поэтому все проведенные работы были выполнены на языке программирования Python c использованием Tkinter. Этот язык отвечал всем необходимым для успешного выполнения проекта требования. Язык имеет множество преимуществ и является одним из наиболее востребованных и наиболее популярных на рынке на данный момент. Он является довольно легким в изучении и использовании, имеет огромное количество библиотек, которые могут быть использованы для решения множества задач.

Python легко конкурирует с другими языками программирования, так как имеет множество достоинств, это понятный и простой язык программирования.

Python, как и любой другой язык программирования, имеет свои отличительные особенности. Не составит труда найти ответ на любой возникший вопрос, так существует много бесплатной литературы, обучающих видео-пособий, готовых исходников и шаблонов для работы в открытом доступе.

Динамическая типизация – это одно из главных достоинств языка Python. Для новичков это возможность упростить написание кода и избежать множества фатальных ошибок и багов в работе. Также в Python нет операторных скобок, с расставлением которых зачастую возникают сложности. Однако как и у любого другого языка программирования у Python есть некоторые минусы. По скорости выполнения программ, когда это касается крупных полномасштабных проектов, Python, конечно же, не лидер.

Подключение библиотек, написанных на С и возможность предварительной компиляции кода в байт-код – все это позволяет улучшить быстродействие.

Таким образом становится ясно, что Python является хорошим выбором для решения поставленной нами задача. А использование встроенных в него библиотек, позволяет облегчить написание кода и ускорить выполнение данного проекта.

# Описание алгоритмов, реализующих бизнес-логику серверной части проектируемой системы

7.1 Схема алгоритма решения задачи выбора компании

Схема алгоритма решения задачи выбора компании представлена на рисунке 7.1

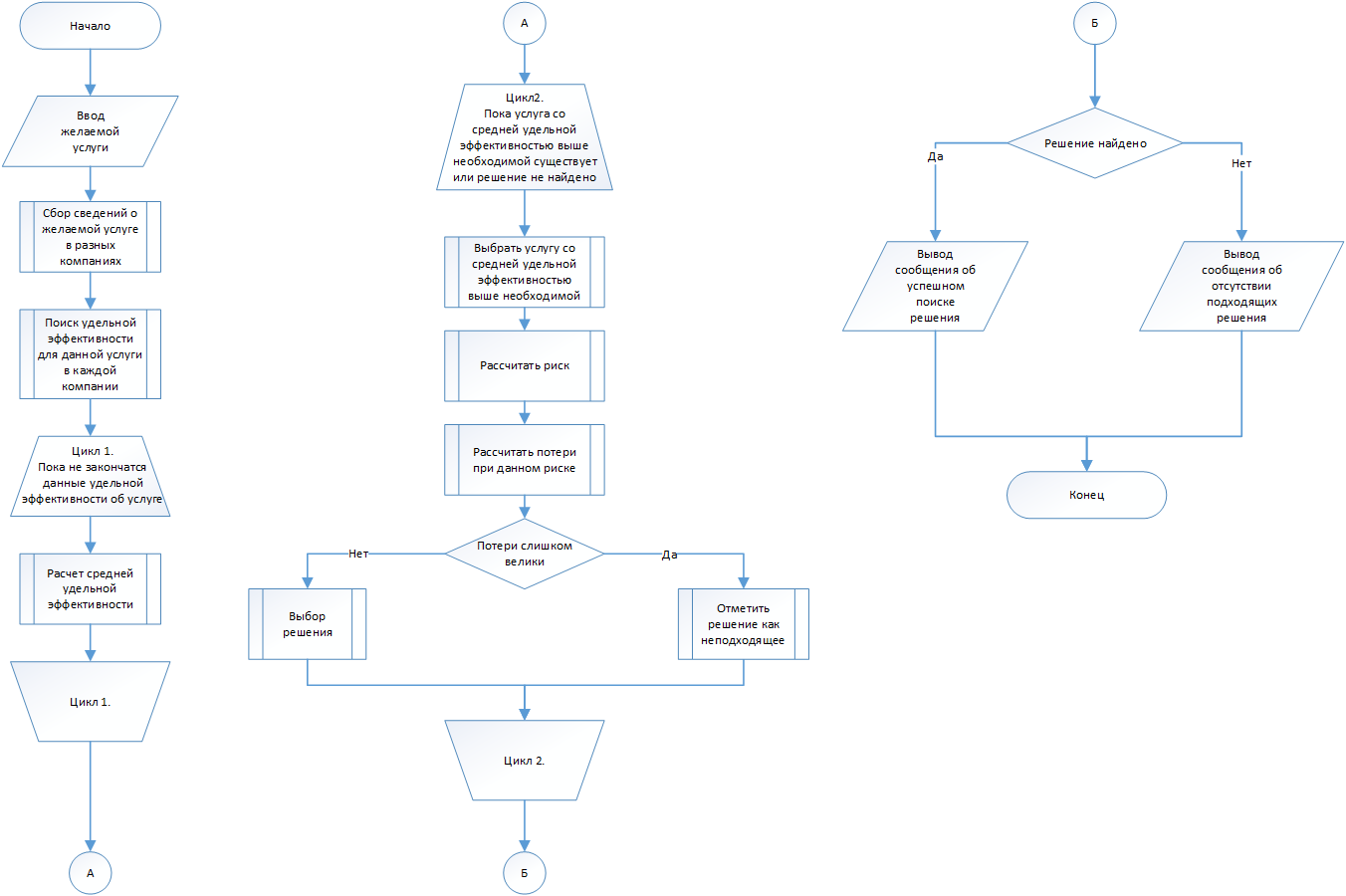


Рисунок 7.1 – Схема алгоритма решения задачи выбора компании

На данной блок-схеме видно, что принцип решения задачи поиска решения заключается в том, чтобы найти эффективное решение на основе удельной эффективности, вероятности риска и потерях при данном риске. Оптимальным решением будет выбрано то, которое соответствует ожиданиям клиента (показатель удельная эффективность), так и компании (показатель риск и потери при данном риске). Если же такого решения нет, то брокер будет уведомлен об этом.

7.2 Схема алгоритма построения графика цены услуги различных компаний

Схема алгоритма построения графика цены услуги различных компаний за указанный период времени представлена на рисунке 7.2

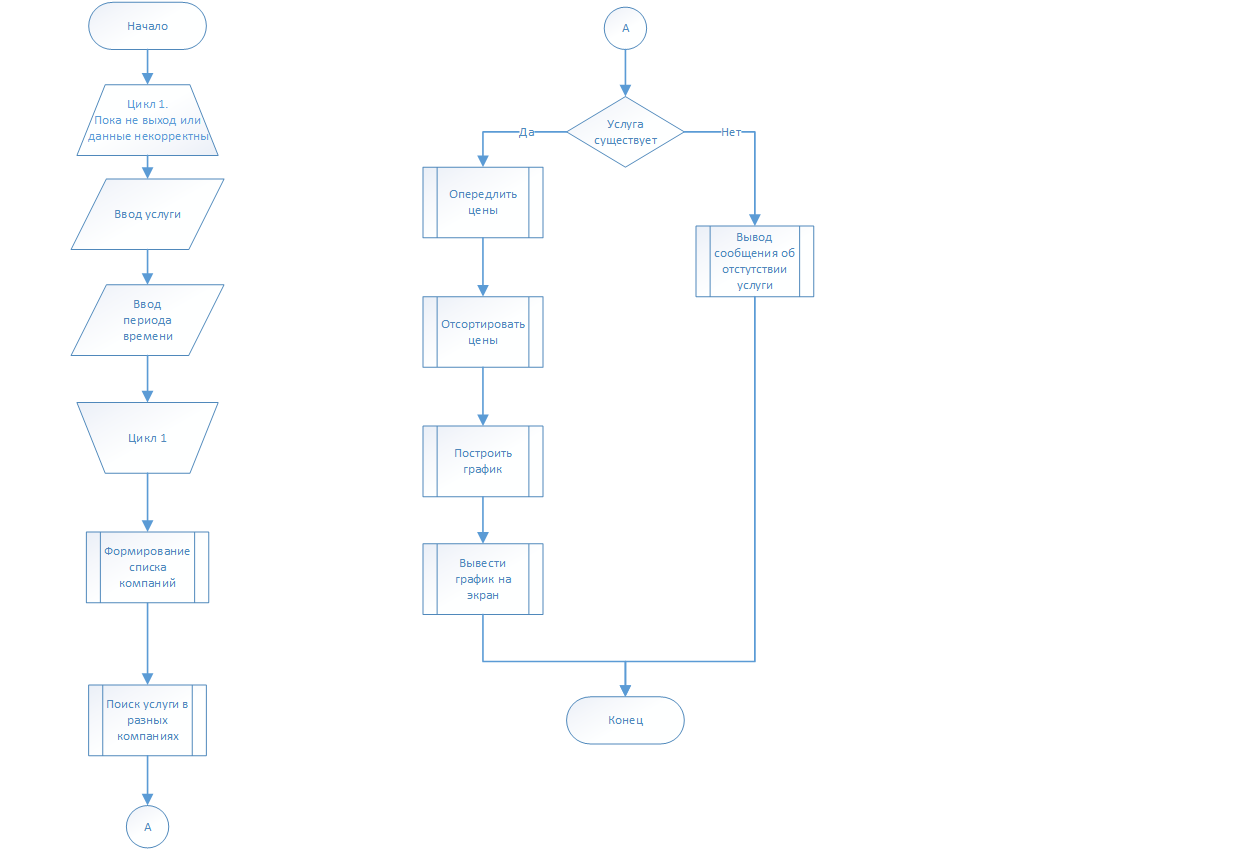


Рисунок 7.2 – Схема алгоритма построения графика цены услуги различных компаний за указанный период времени

На данной схеме видно, что для построения графика система циклично спрашивает у брокера данные об услуги. Выход из цикла будет только в том случае, если данные будут корректные или брокер не выйдет в главное меню.

В случае корректных данных информация будет отправлена на сервер, который запросит данные об услуге в базе данных. Далее следует сортировка цен услуги различных компаний и построение графика.

Если же данные будут введены некорректно, то брокер должен будет ввести данные заново или отменить действие.

После того, как график будет построен, он будет выведен на экран, и брокер сможет получить необходимую информацию.

Если необходимой услуги не существует, брокер получит соответствующее сообщение.

7.3 Схема алгоритма клиент-серверного взаимодействия

Схема алгоритма клиент-серверного взаимодействия представлена на рисунке 7.3

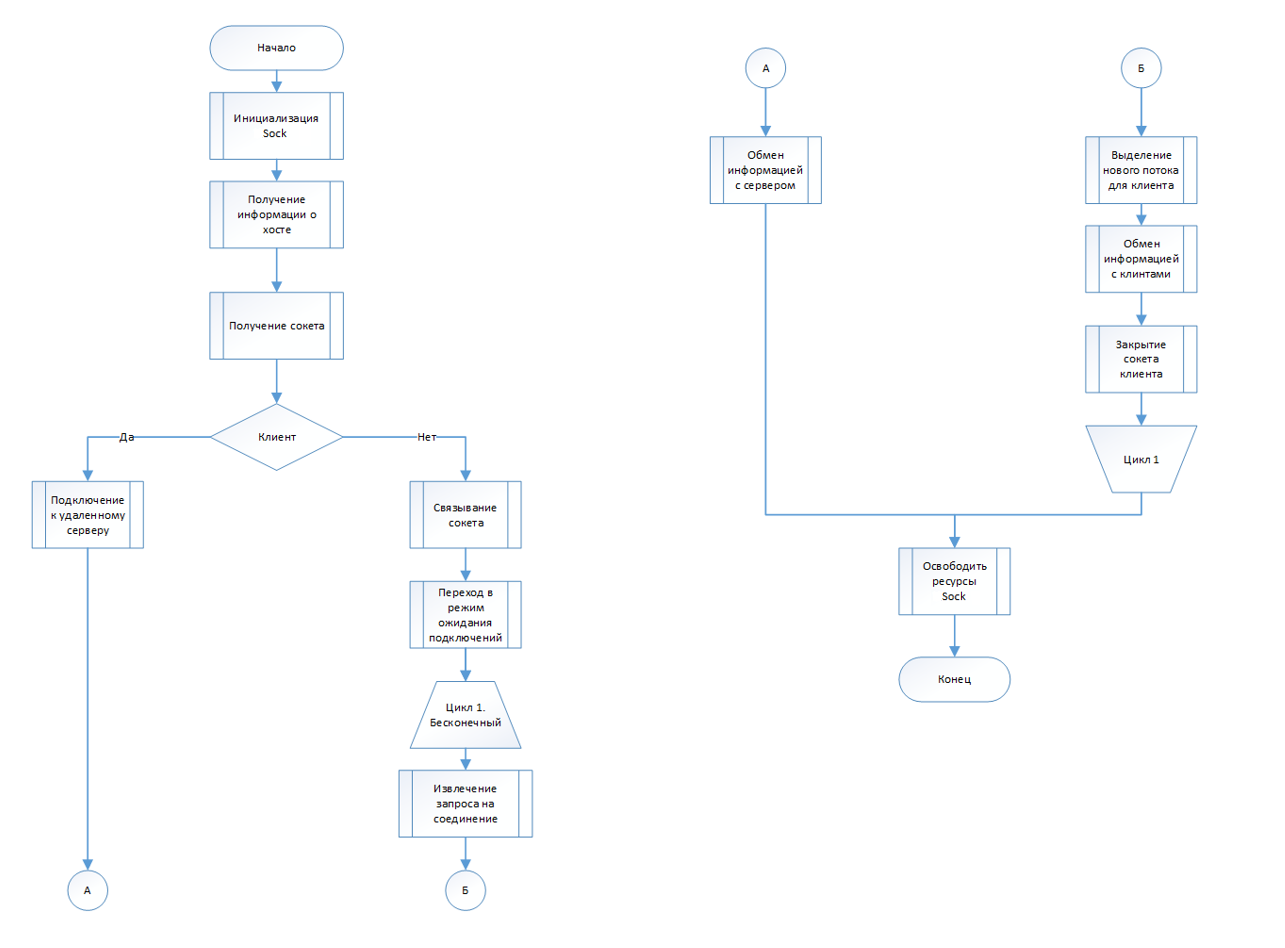


Рисунок 7.2 – Схема алгоритма клиент-серверного взаимодействия

На данной схеме видно, что клиент-серверное взаимодействие начинается с инициализации библиотеки socket, затем создается сокет. Если сокет клиента, то вызывается функций connect и происходит подключение к удаленному серверу, далее происходит обмен информацией с сервером. Если же сокет сервера, то идет связывается адреса хоста с сокетом и сервер переходит в режим ожидания подключение с помощью функции listen [9]. Далее сервер входит в цикл для ожидания запросов на подключение. При получении запроса сервер выделяет для клиента новый поток и начинается обмен информацией с клиентом. Как только клиент завершит обмен информацией с сервером, сокет этого клиента закроется. В конце завершается работа сервера и очищаются ресурсы socket.

Эта глава показывает, что блок-схема отображает принцип работы определённого алгоритма, что даёт полное понимание происходящего процесса. Если человек прежде не имел никакого отношения к программированию, для него блок-схема это что-то сродни упрощённому алгоритму.

# Руководство пользователя

Первым делом необходимо запустить сервер. Сервер можно запустить двойным кликом по иконке сервера. Кроме этого сервер можно запустить из командной строки передав параметры IP-адрес и порт (рисунок 8.1).

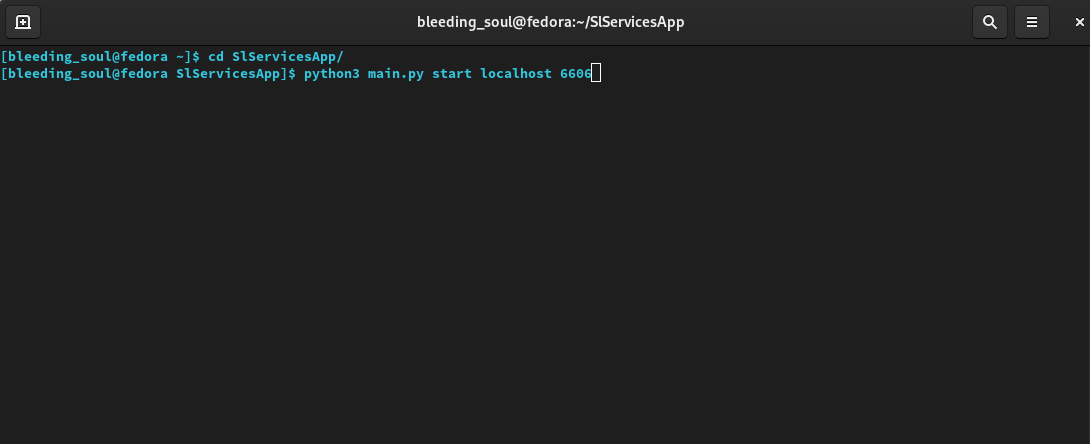


Рисунок 8.1 – Окно запуска сервера через IP-адрес и порт

Также, чтобы запустить сервер можно не указывать порт, в таком случае он будет принят по умолчанию и равен 6606 (рисунок 8.2).

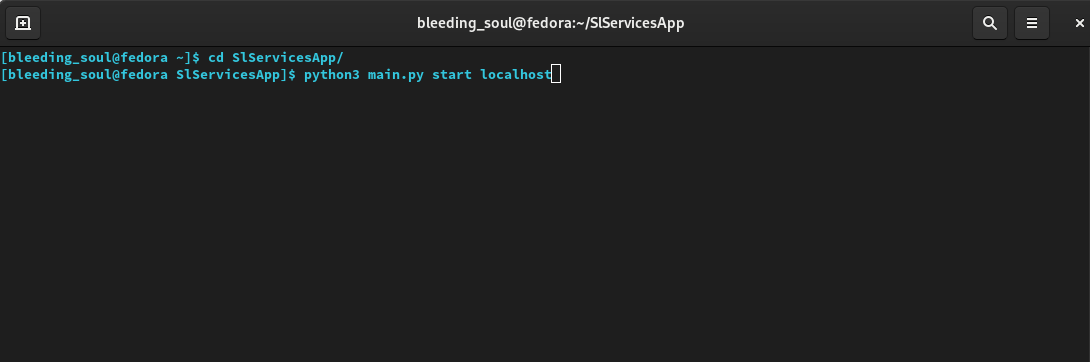


Рисунок 8.2 – Окно запуска через IP-адрес

Сервер можно запустить непосредственно в самом приложении с помощью команды start, указав через пробел IP-адрес и порт (рисунок 8.3).

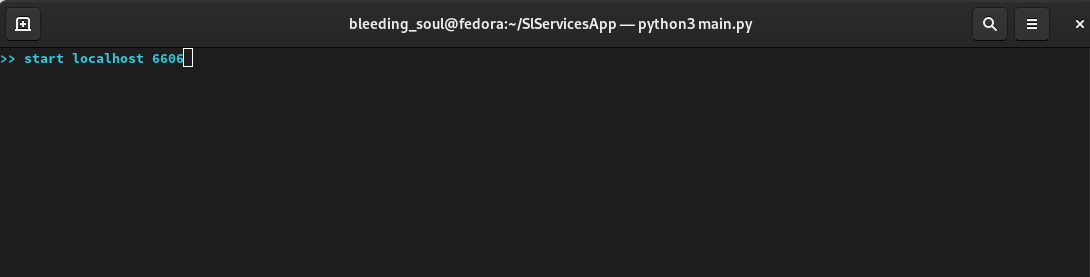


Рисунок 8.3 – Окно запуска сервера через приложение

Чтобы узнать сетевую информацию о сервере можно воспользоваться командой inetinfo (рисунок 8.4).

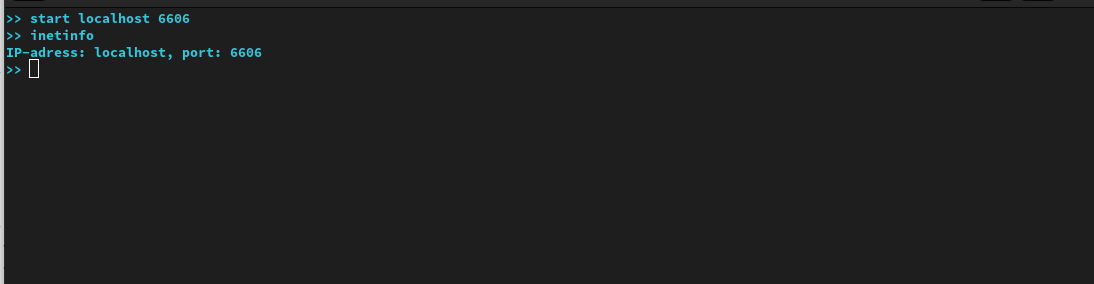


Рисунок 8.4 – Окно вывода команды inetinfo

Чтобы посмотреть количество подключенных клиентов и базовую информацию о них можно воспользоваться командой clients (рисунок 8.5).

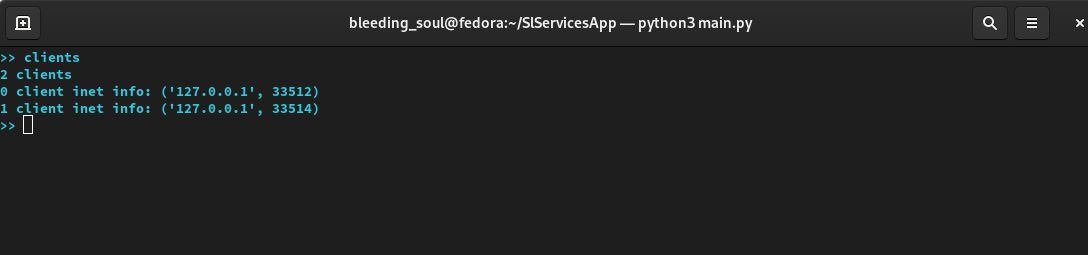


Рисунок 8.5 – Окно вывода команды clients

Чтобы посмотреть состояние потоков для каждого клиента можно воспользоваться командой threads (рисунок 8.6).

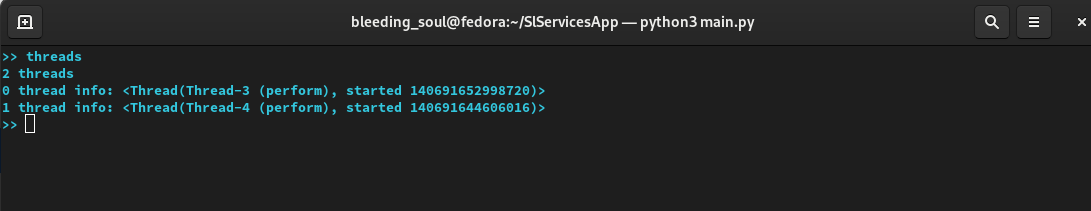


Рисунок 8.6 – Окно вывода команды threads

Чтобы получить помощь по функционалу можно воспользоваться командой help (рисунок 8.7).

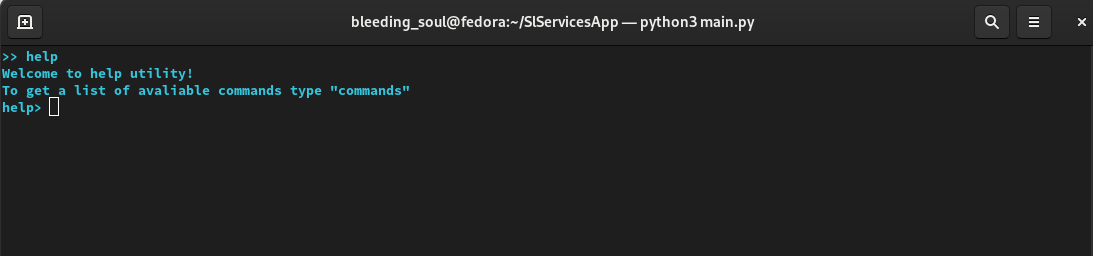


Рисунок 8.7 – Окно вывода команды help

При написании несуществующих команд будет выведено соответствущее сообщение (рисунок 8.8).

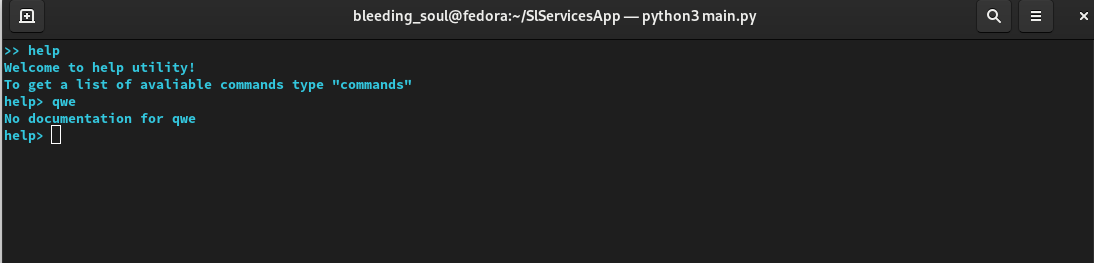


Рисунок 8.8 – Окно вывода несуществующей команды в утилите help

В случае написания существующей команды появится информационная выжимка, кратко описывающая использовании данной команды (рисунок 8.9).

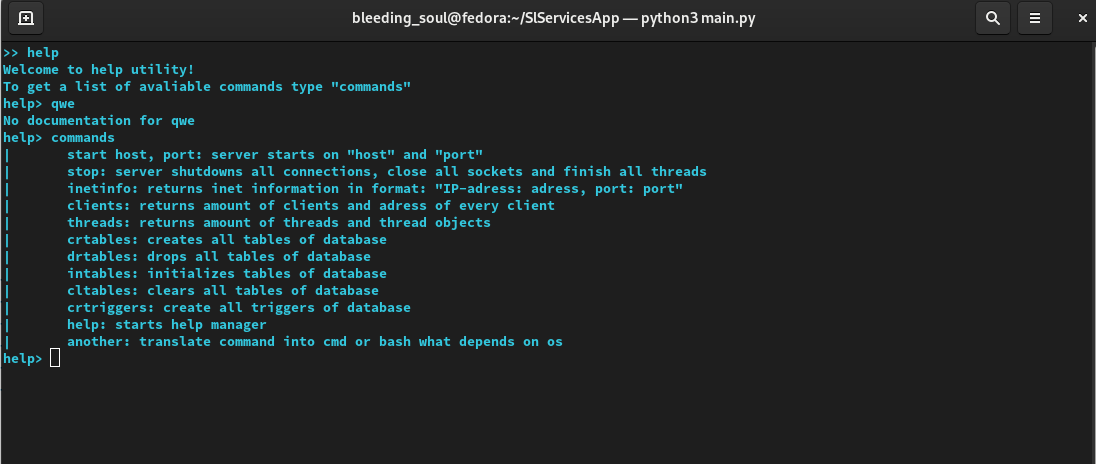


Рисунок 8.9 – Окно вывода существующей команды в улитите help

Чтобы выйти из утилиты help необходимо ввести символ q. Из программы можно создавать таблицы и триггеры. Чтобы создать таблицы необходимо ввести команду crtables (рисунок 8.10).

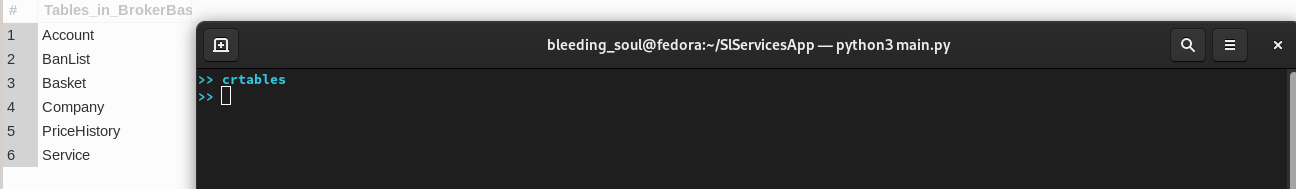


Рисунок 8.10 – Результат создания таблиц с помощью команды crtables

Для того, чтобы удалить таблицы необходимо ввести команду drtables (рисунок 8.11).

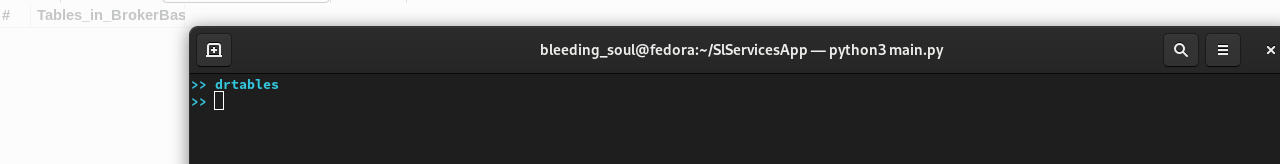


Рисунок 8.11 – Результат удаления таблиц с помощью команды drtables

Для того, чтобы инициализировать таблицы необходимо ввести команду intables (рисунок 8.12).



Рисунок 8.12 – Результат инициализации таблиц с помощью команды intables

Для того, чтобы очистить таблицы необходимо ввести команду cltables (рисунок 8.13).

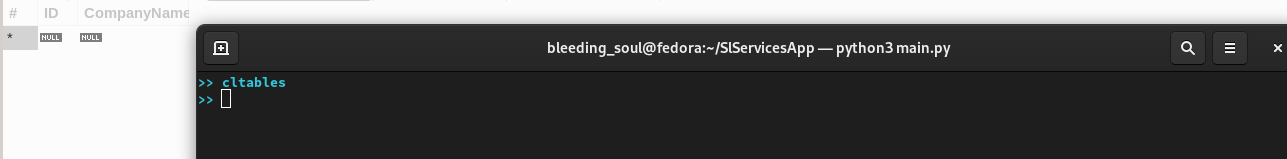


Рисунок 8.13 – Результат очистки таблиц с помощью команды cltables

Для того, чтобы создать триггеры необходимо воспользоваться командой crtriggers (рисунок 8.14).

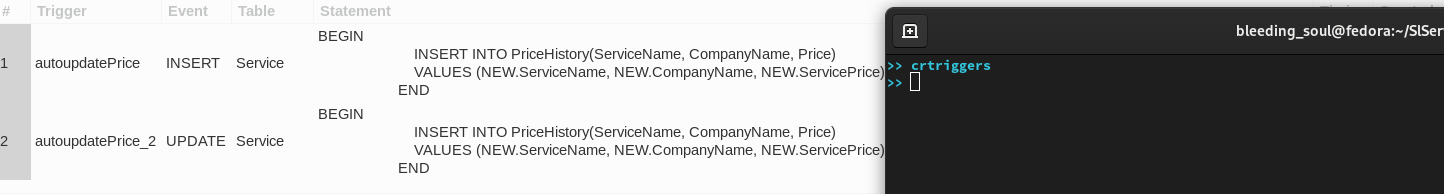


Рисунок 8.14 – Результат выполнения команды crtriggers

На сервере могут быть созданы аккаунты для сотрудников и выданы им лично. В клиенте регистрируется по умолчанию аккаунт для клиента.

Для регистрации специального аккуанта для сотрудника можно воспользоваться командой create\_acc (рисунок 8.15).

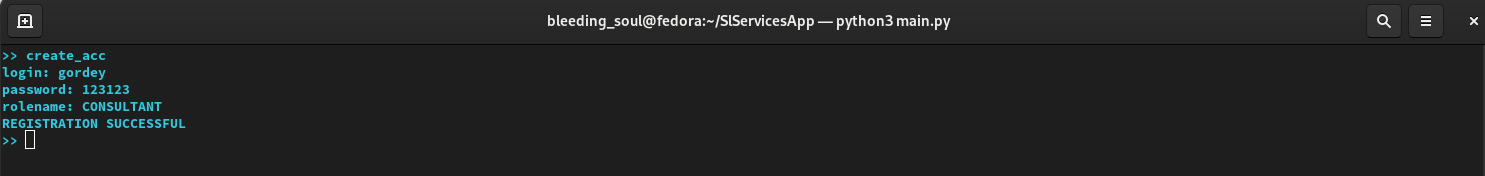


Рисунок 8.15 – Результат выполнения команды create\_acc

При попытке создать аккаунт с таким же логином будет выведено сообщение об ошибке (рисунок 8.16).

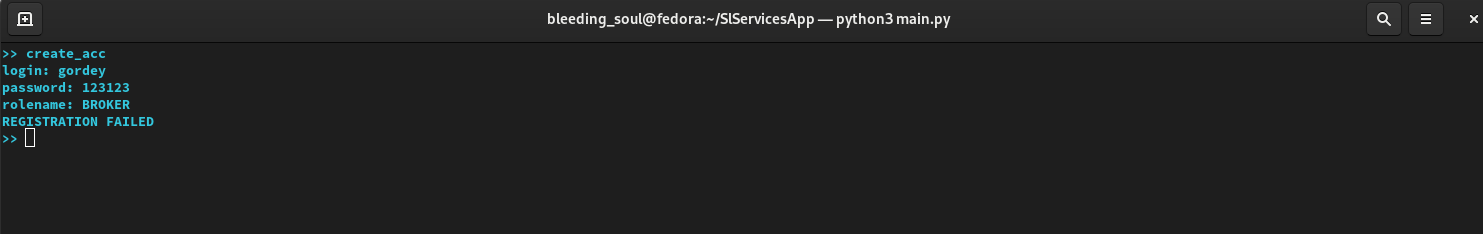


Рисунок 8.16 – Результат создания аккаунта с занятым логином

Чтобы удалить аккаунт можно воспользоваться командой delete\_acc (рисунок 8.17).

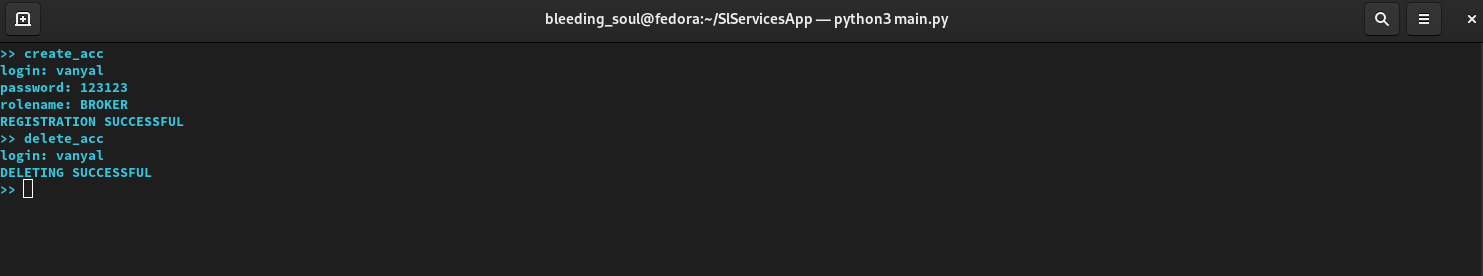


Рисунок 8.17 – Результат удаления аккаунта

Чтобы изменить данные аккаунта можно воспользоваться командой update\_acc (рисунок 8.18).

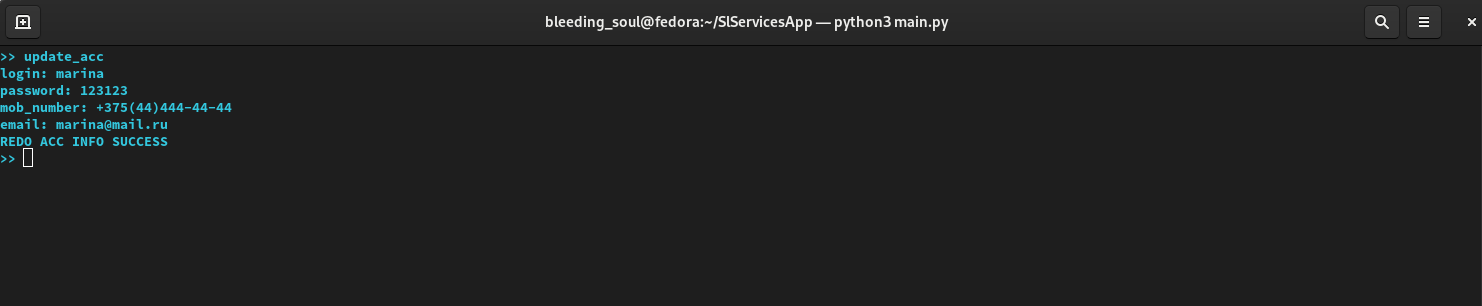


Рисунок 8.18 – Результат изменения аккаунта

Чтобы просмотреть все аккаунты необходимо ввести команду read\_accs (рисунок 8.19).

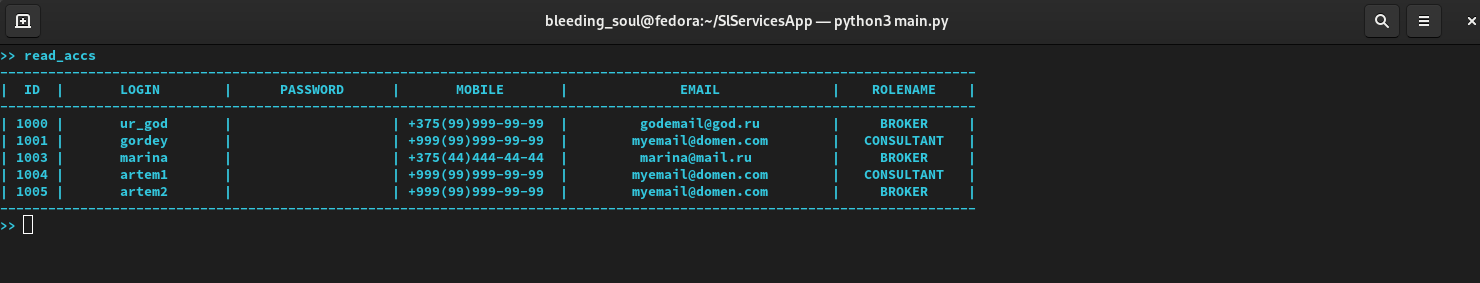


Рисунок 8.19 – Результат вывода всех аккаунтов на экран

Все пароли будут недоступны для просмотра администратором.

Чтобы забанить аккаунт необходимо воспользоваться командой ban\_acc (рисунок 8.20)

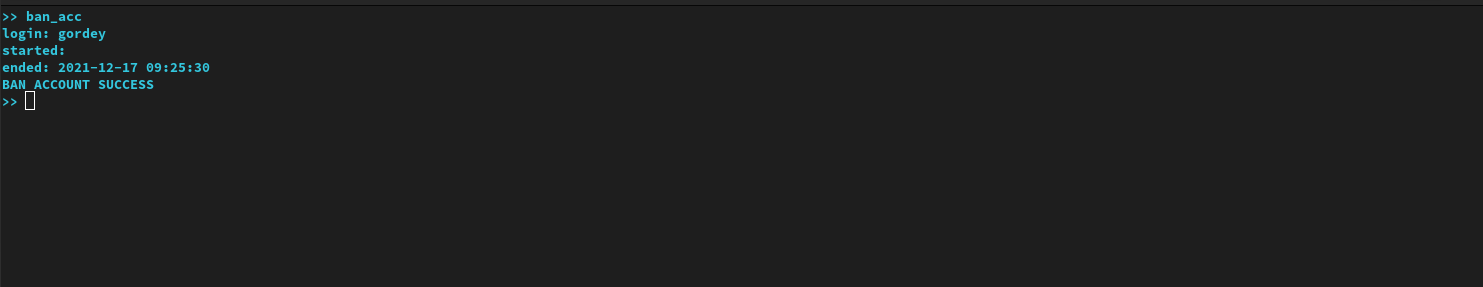


Рисунок 8.20 – Результат выполнения команды ban\_acc

В данном случае, при опускании параметра started будет принята текущая дата и время. В случае указания бан начнет действовать с указанной даты и времени. По истечению времени, бан будет снят автоматически.

Пока пользователь забанен он не сможет зайти в аккаунт (рисунок 8.21).

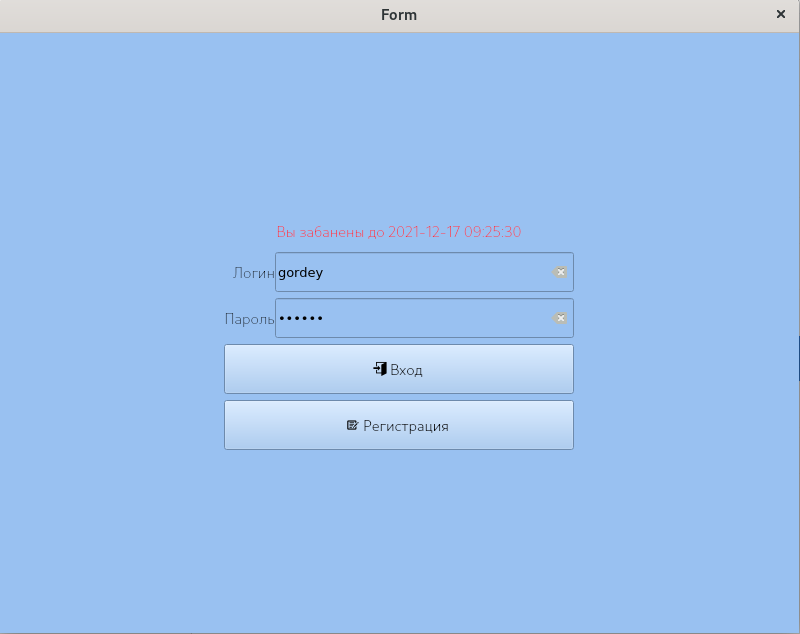


Рисунок 8.21 – Результат попытки входа в аккаунт забаненого пользователя

Чтобы вывести баны на экран необходимо воспользоваться командой view\_bans (рисунок 8.22).

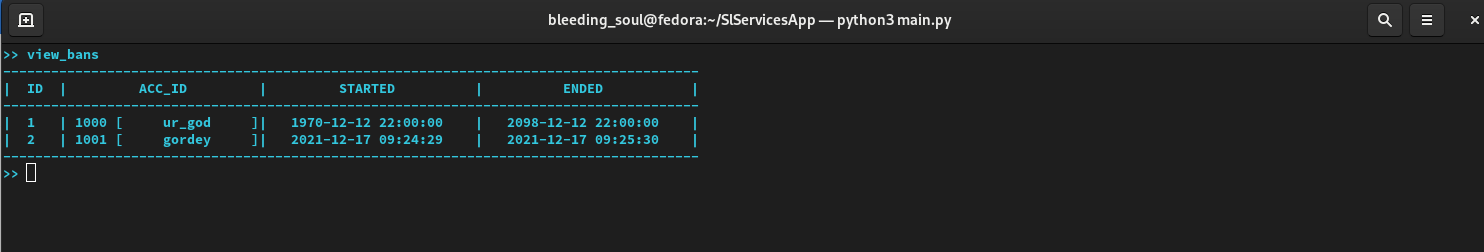


Рисунок 8.22 – Результат вывода всех блокировок на экран

Для того, чтобы снять блокировку вручную необходимо воспользоваться командой unban\_acc (рисунок 8.23).

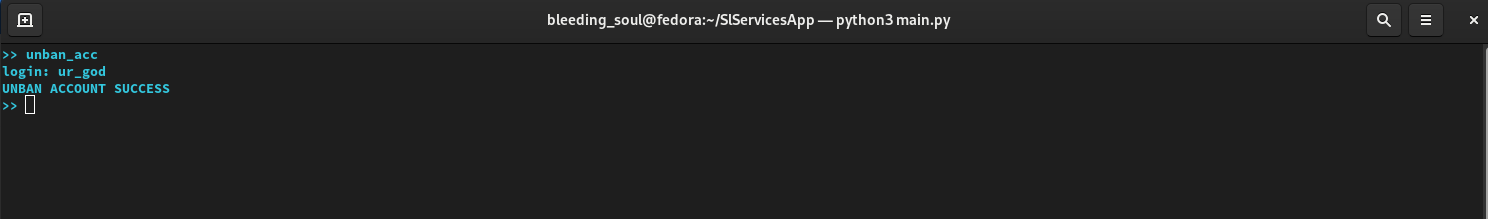


Рисунок 8.23 – Результат ручного снятия блокировки с аккаунта

Запустить клиент можно с помощью двойного щелчка по иконке приложения клиента. При запуске клиента пользователь будет видеть стартовую страницу приложения (рисунок 8.24).

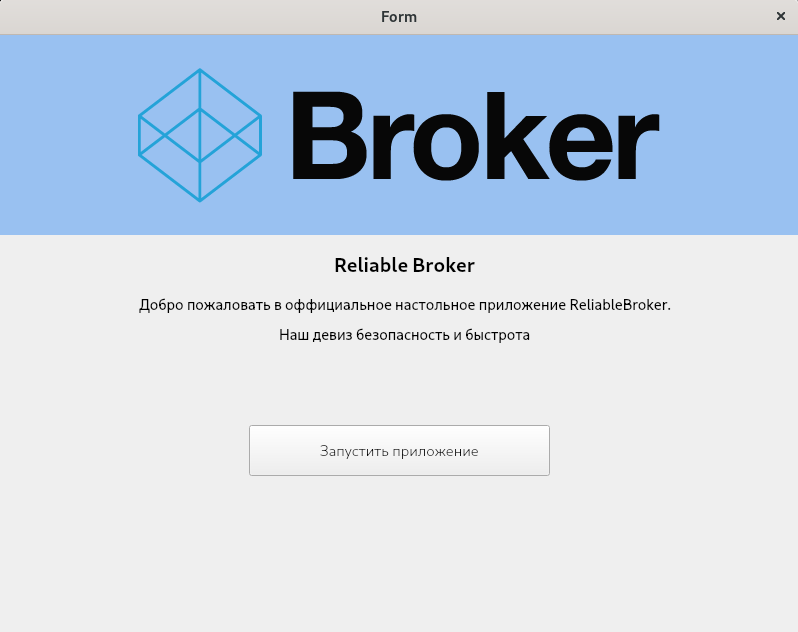


Рисунок 8.24 – Стартовая страница клиента

При нажатии на кнопку «Запустить приложение» будет открыто окно аутентификации (рисунок 8.25)

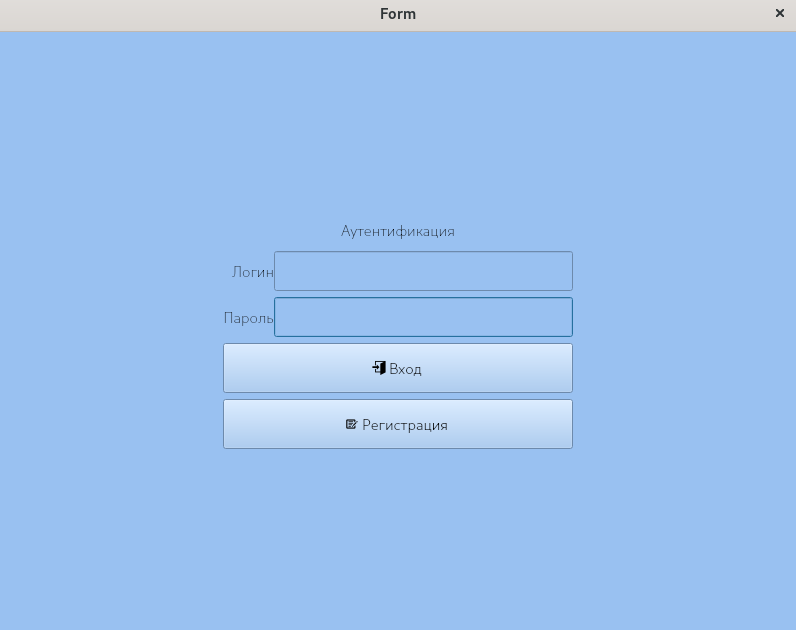


Рисунок 8.25 – Окно аутентификации

При нажатии на кнопку «Регистрация» будет открыто окно регистрации (рисунок 8.25)

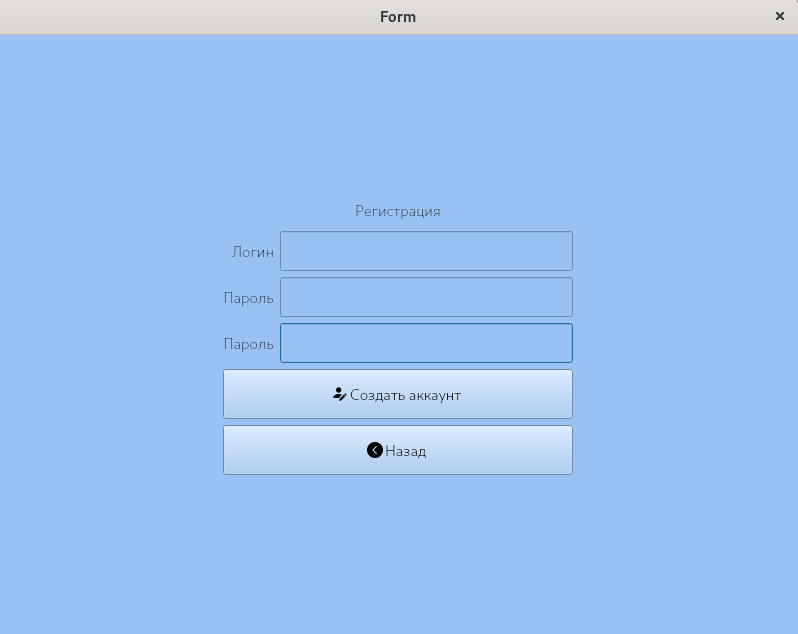


Рисунок 8.25 – Окно регистрации

При успешной регистрации будет выведено соответствующее сообщение и открыто окно аутентификации (рисунок 8.26)

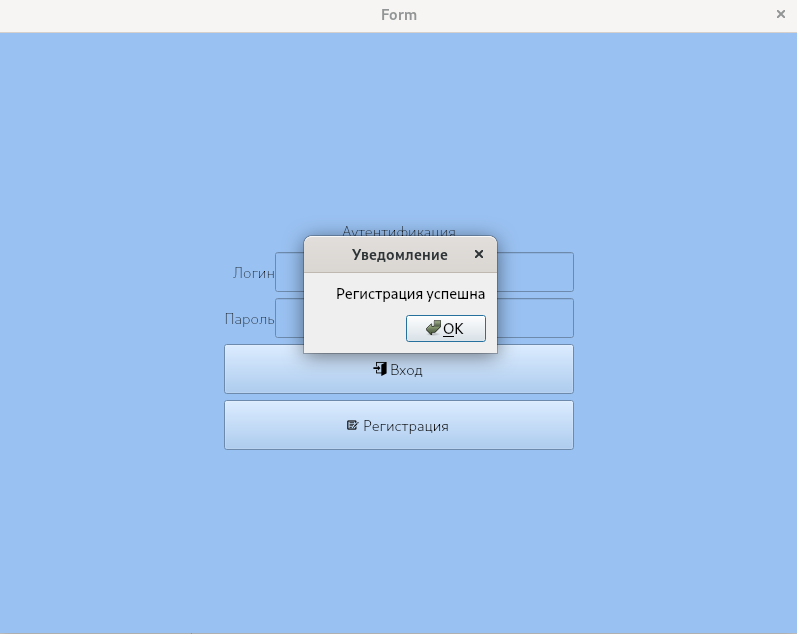


Рисунок 8.26 – Результат регистрации пользователя

При успешной аутентификации пользователь переходит в главное окно, в котором он видит список услуг и две кнопки для покупки услуги и получения консультации (рисунок 8.27)

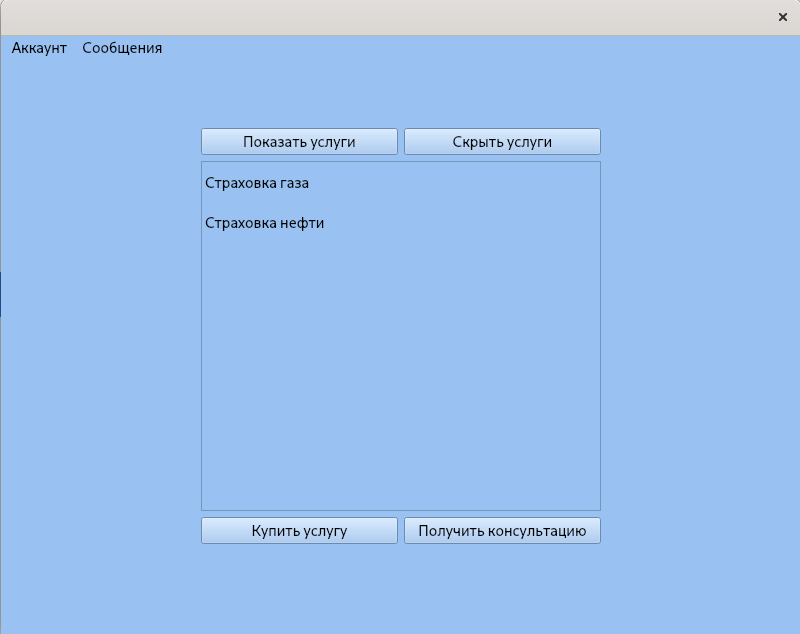


Рисунок 8.27 – Главное окно пользователя

Пользователь может зайти в личный кабинет через меню. Окно личного кабинета представлено на рисунке 8.28

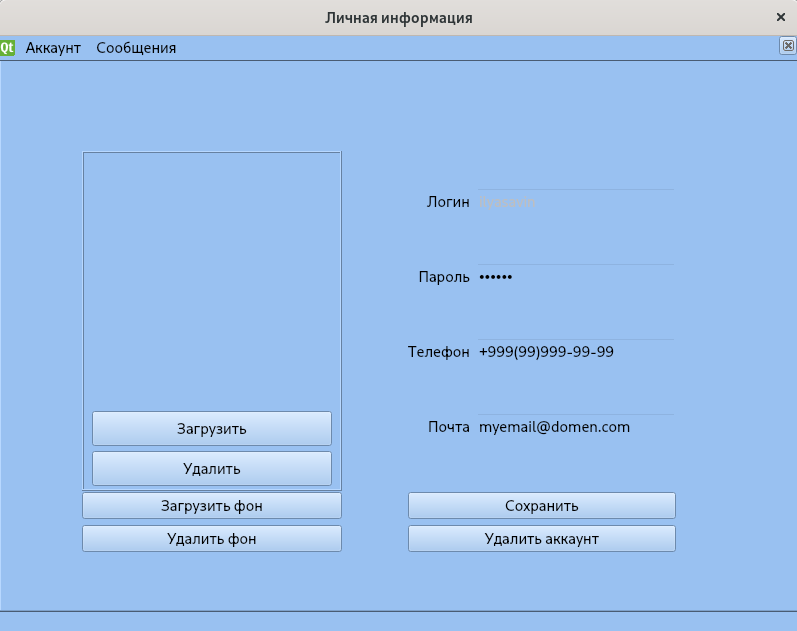


Рисунок 8.28 – Окно личного кабинета пользователя

Пользователь может добавить фон, фотографию, изменить данные (рисунок 8.29)

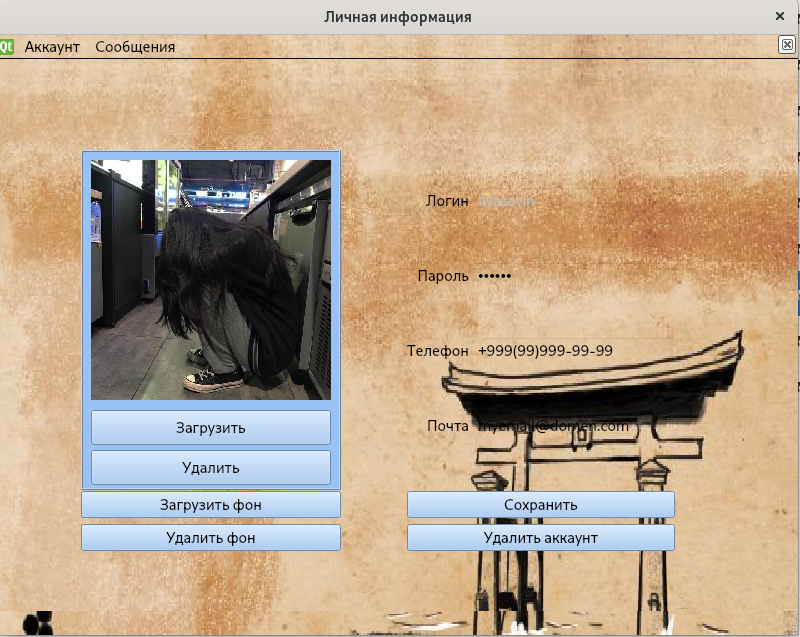


Рисунок 8.29 – Возможности в личном кабинете

Также пользователь может удалить аккаунт. Для этого потребуется ввести пароль от аккаунта (рисунок 8.30)

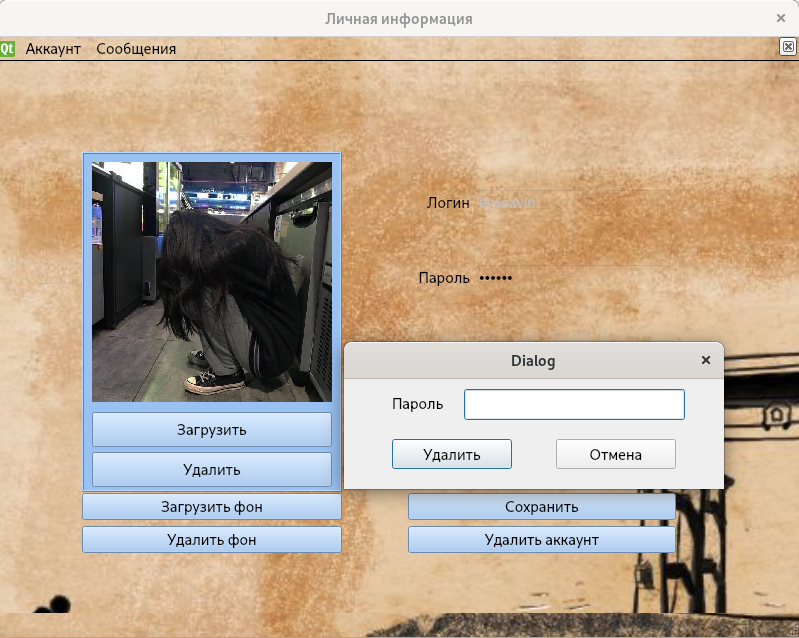


Рисунок 8.30 – Окно подтверждения удаления аккаунта

При верном пароле аккаунт будет удален, а пользователь перемещен на окно аутентификации.

Если клиент является служебным лицом, а именно: консультант, то в таком случае у него будет другое главное окно (рисунок 8.31)

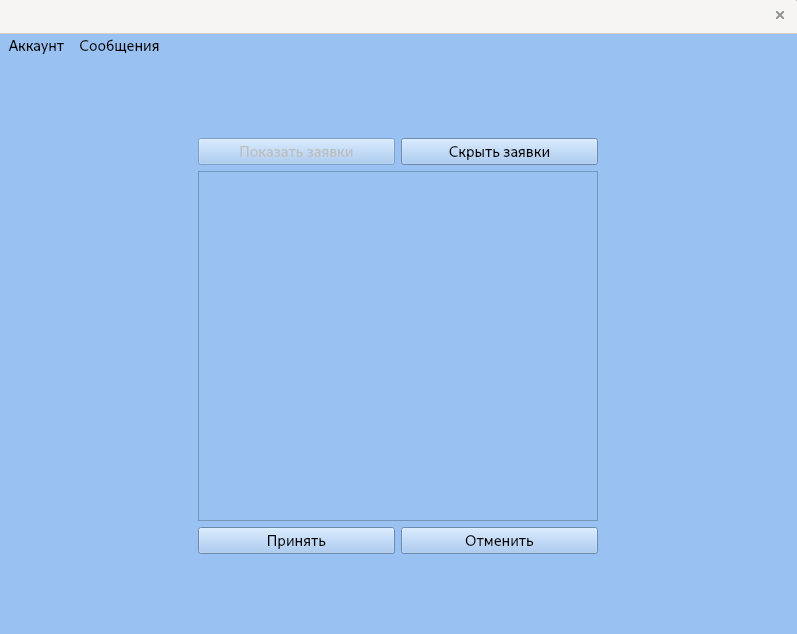


Рисунок 8.31 – Главное окно консультанта

Пользователь может заказать консультацию (рисунок 8.32)

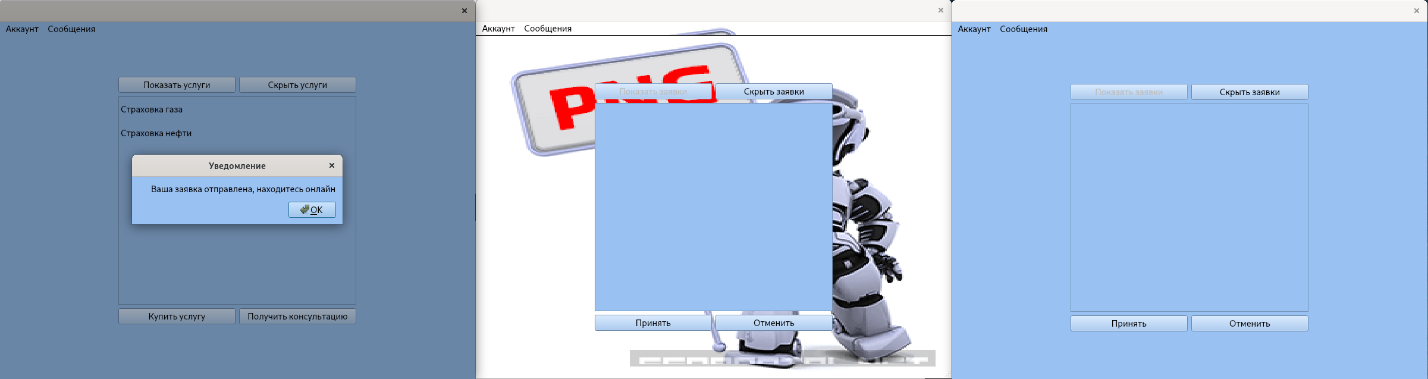


Рисунок 8.32 – Заказ консультации пользователем

Как только пользователь подтвердит заказ консультации, консультанты увидят заявку на консультацию (рисунок 8.33)

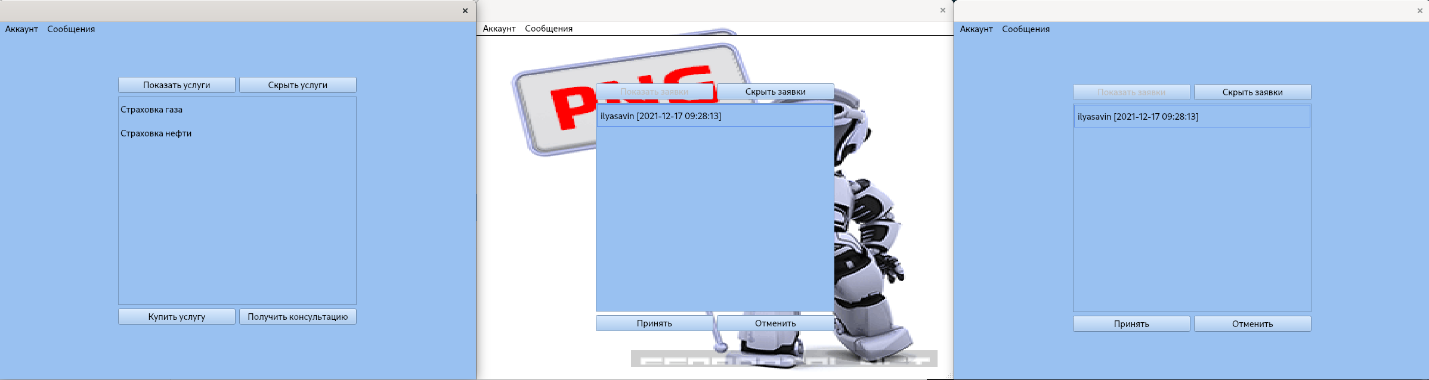


Рисунок 8.33 – Результат подачи завяки на консультацию пользователем

Консультант может принять или отклонить заявку, причем, если консультант примет завяку, то она удалится у всех других консультантах, а между принявшим консультантом и клиентом откроется чат (рисунок 8.34)

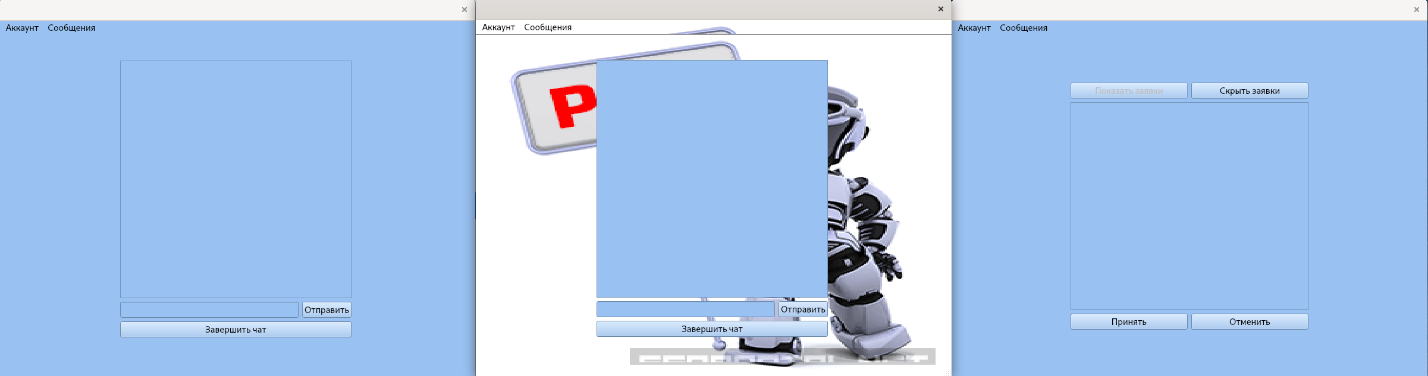


Рисунок 8.34 – Результат обработки заявки на консультацию

Консультант и клиент могут общаться в режиме онлайн (рисунок 8.35)

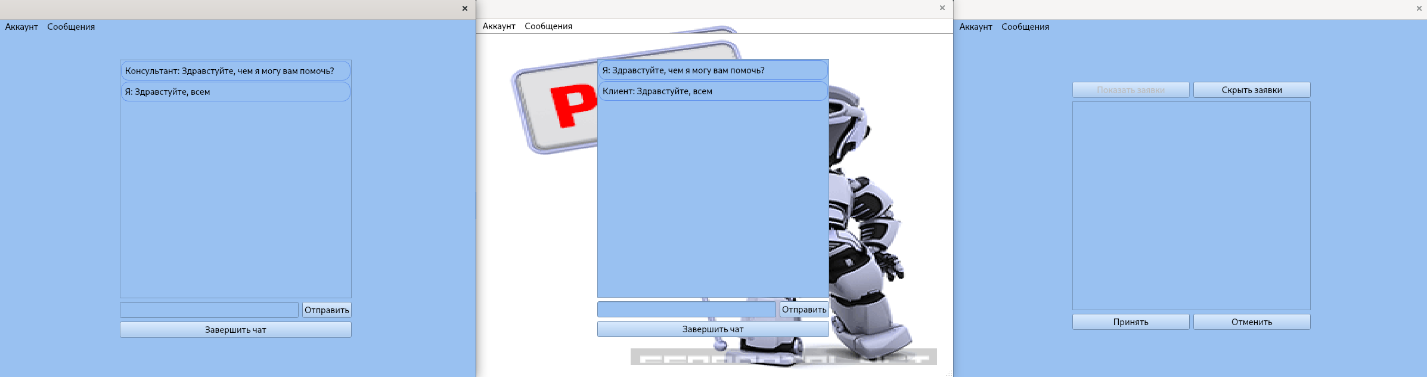


Рисунок 8.35 – Взаимодействие клиента и консультанта

При завершении чата, выхода из аккаунта или закрытии приложения консультация прекращается, причем если чат был завершен с помощью кнопки, то клиент и консультант возвращаются на главное окно. Если же приложение было закрыто или был осуществлен выход из аккаунта, то то лицо, с кем был оборван чат будет возвращано на главное окно.

Если же аккаунт является специальным, а именно: брокер, то главное окно будет другим (рисунок 8.36)

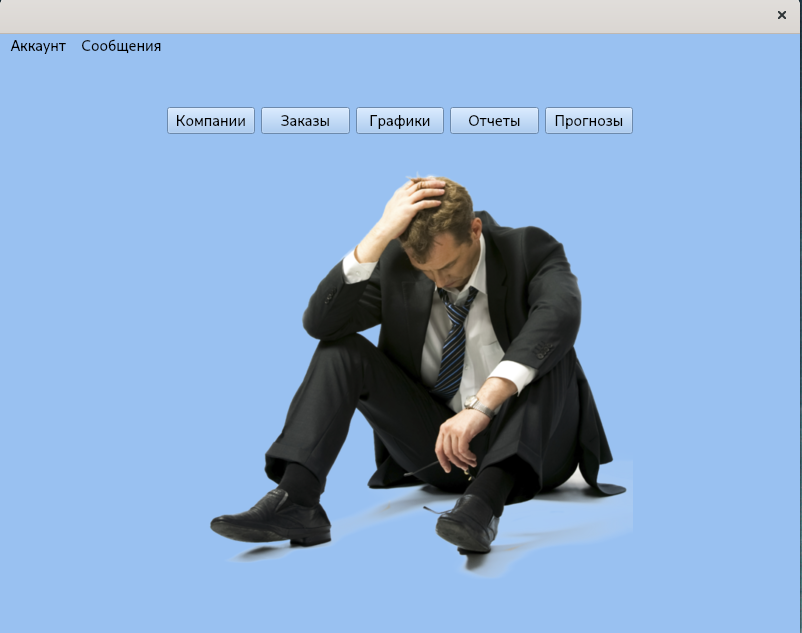


Рисунок 8.36 – Главное окно брокера

Клиент может заказать услугу (рисунок 8.37)

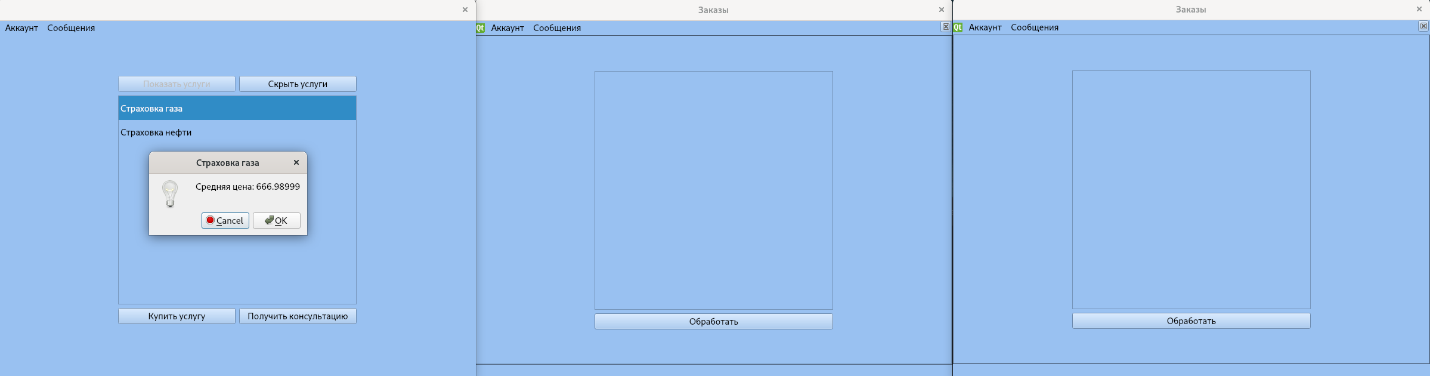


Рисунок 8.37 – Заказ услуги клиентом

При подтверждении заказа услуги, брокеры получают заявку на покупку услуги (рисунок 8.38)

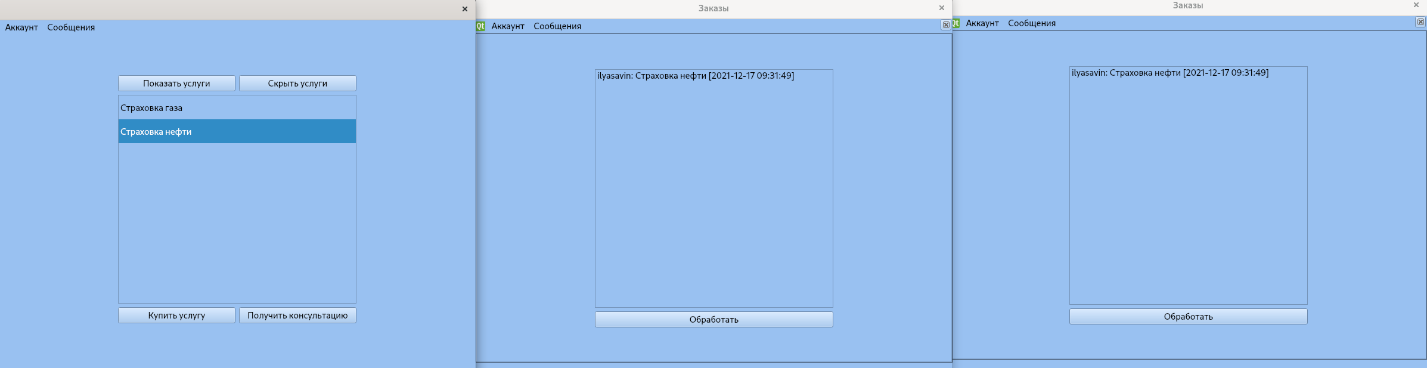


Рисунок 8.38 – Результат покупки услуги клиентом

При нажатии клавиши обработать, заявка удаляется для всех брокеров и брокер, который принял завяку берет на все все обязательства по ее обработке, что включает связывание с клиентом и обсуждение услуги и предложение клиенту лучшей компании для данной услуги.

Брокер может управлять компаниями. Для этого ему необходимо кликнуть «Компании» после чего откроется окно управления компаниями (рисунок 8.39)

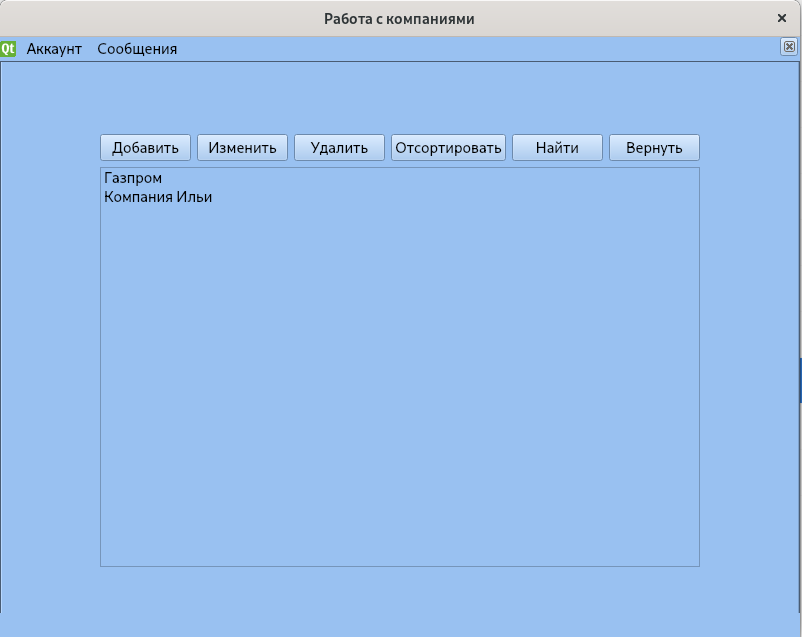


Рисунок 8.39 – Окно управления компаниями

Брокер может добавить компанию (рисунок 8.40)

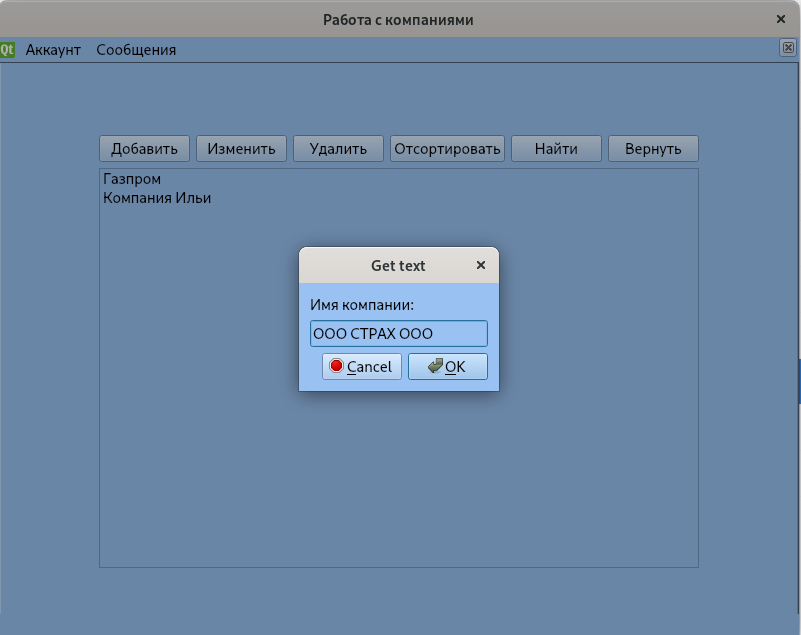


Рисунок 8.40 – Окно добавления компании

После этого новая компания будет добавлена в общий список компаний (рисунок 8.41)



Рисунок 8.41 – Результат добавления новой компании

Брокер может отсортировать названия компаний по алфавиту (рисунок 8.42)

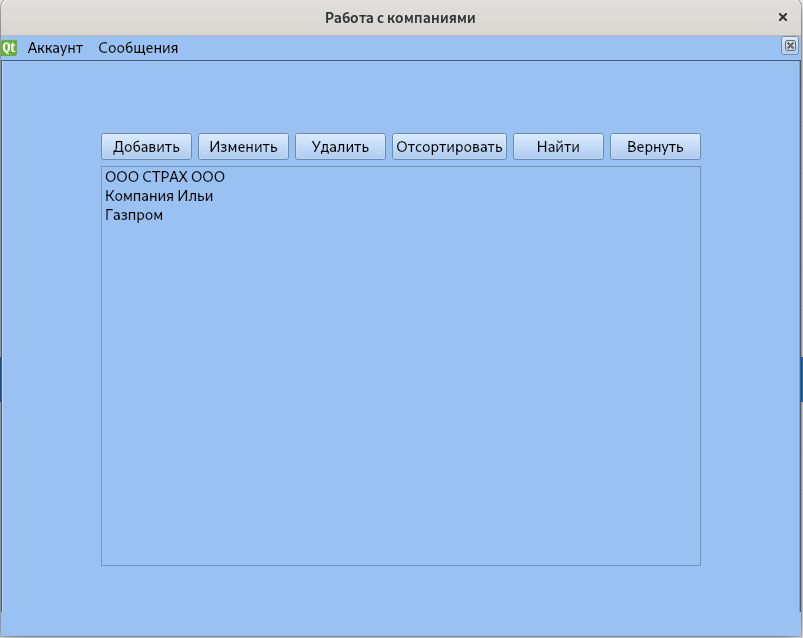


Рисунок 8.42 – Результат сортировки списка компаний

Для того, чтобы найти необходимую компанию нужно нажать на кнопку «Найти» и ввести данные (рисунок 8.43)

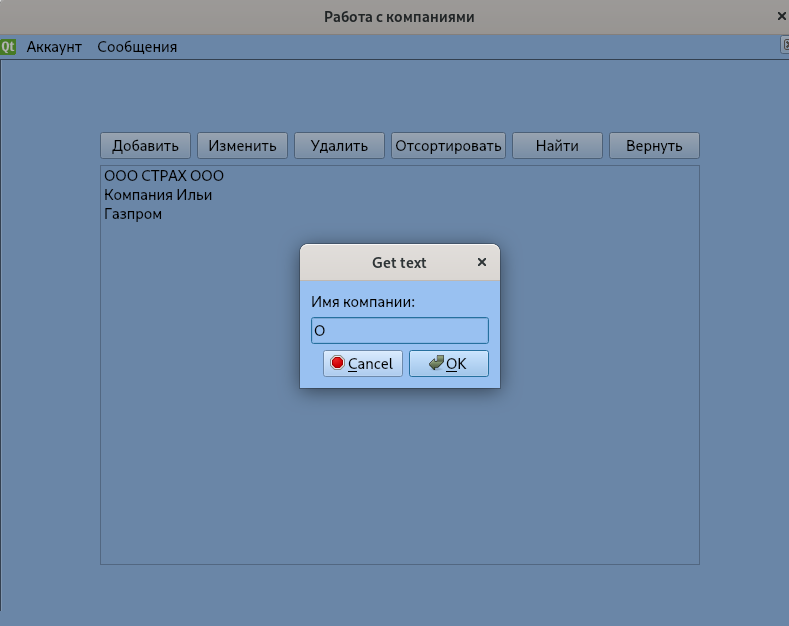


Рисунок 8.43 – Окно поиска компании

После этого на экране будут отображены найденные компании (рисунок 8.44)

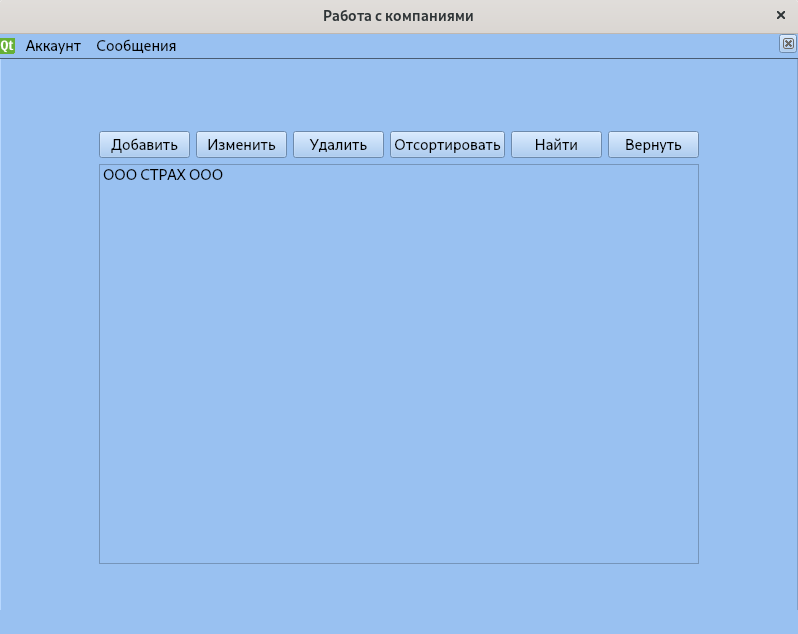


Рисунок 8.44 – Результат поиска компании

Чтобы изменить имя компании необходимо нажать на кнопку «Изменить» и ввести новые данные (рисунок 8.45)



Рисунок 8.45 – Окно изменения данных о компании

После этого на экране будет отображена новая информация (рисунок 8.46)



Рисунок 8.46 – Результаты изменения данных о компании

Для того, чтобы перейти к работе с услугами необходимо совершить двойной клик на компанию (рисунок 8.47)

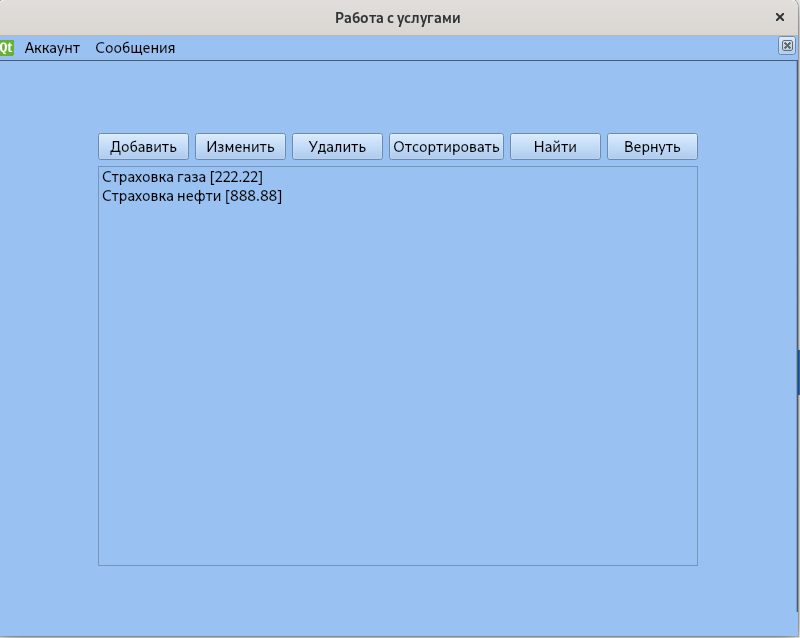


Рисунок 8.47 – Окно работы с услугами

Обработка услуг поддерживает абсолютно те же возможности, что и обработка компаний.

Брокер также может отобразить необходимую информацию в виде графиков (рисунок 8.48)

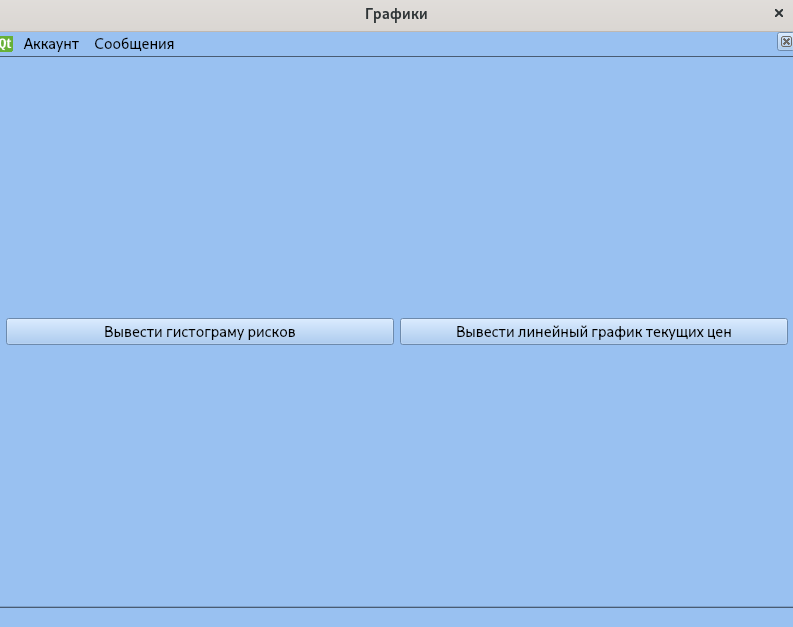


Рисунок 8.48 – Окно выбора графического отображения информации

При выборе линейного графика текущих цен, будет выведен график текущих цен о каждой услуге в каждой компании. Услугу можно выбрать в выпадающем списке (рисунок 8.49)

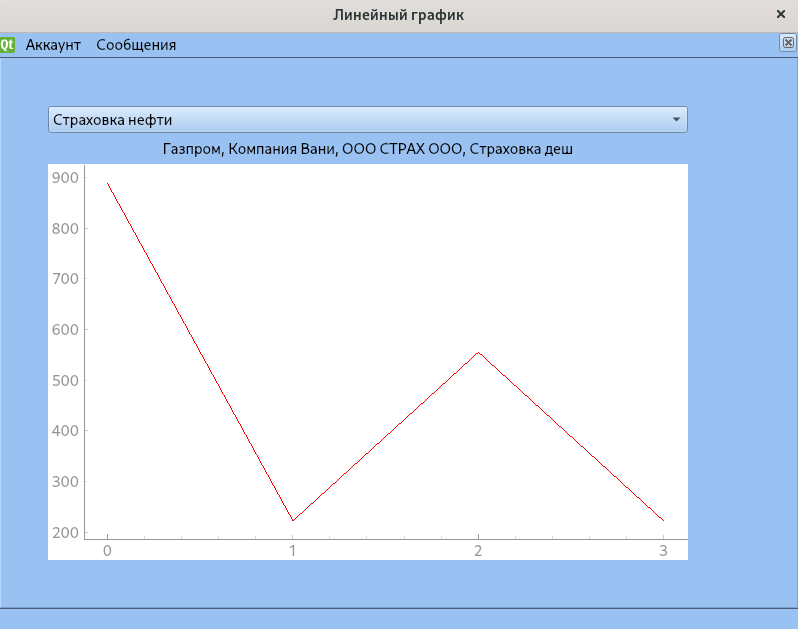


Рисунок 8.49 – График текущих цен на услугу в компаниях

При выборе гистограммы рисков будет выведены гистограммы со средними ценами за всю историю изменений цены и дисперсия, показывающая насколько сильно могут быть изменены цены (рисунок 8.50)

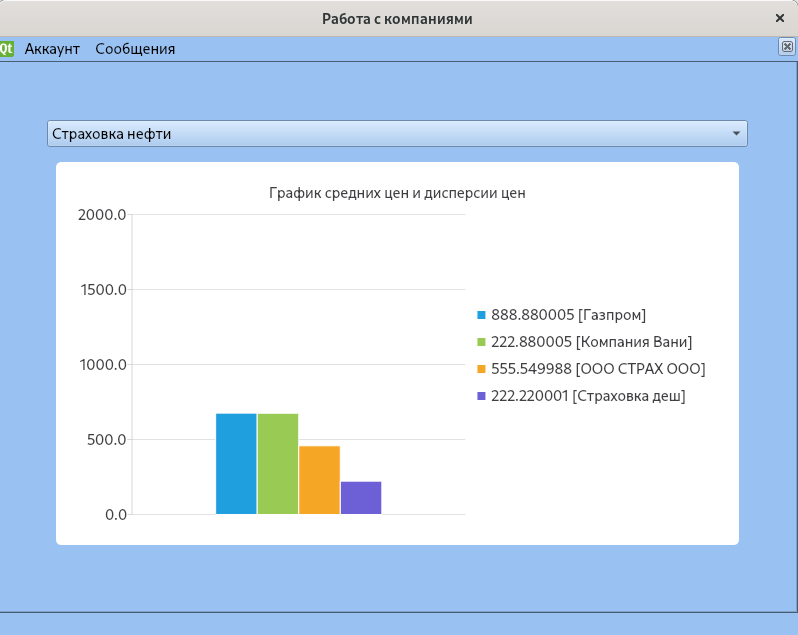


Рисунок 8.50 – Гистограмма рисков

Также брокер может вывести историю цен, чтобы посмотреть почему такие расчеты на графике (рисунок 8.51)

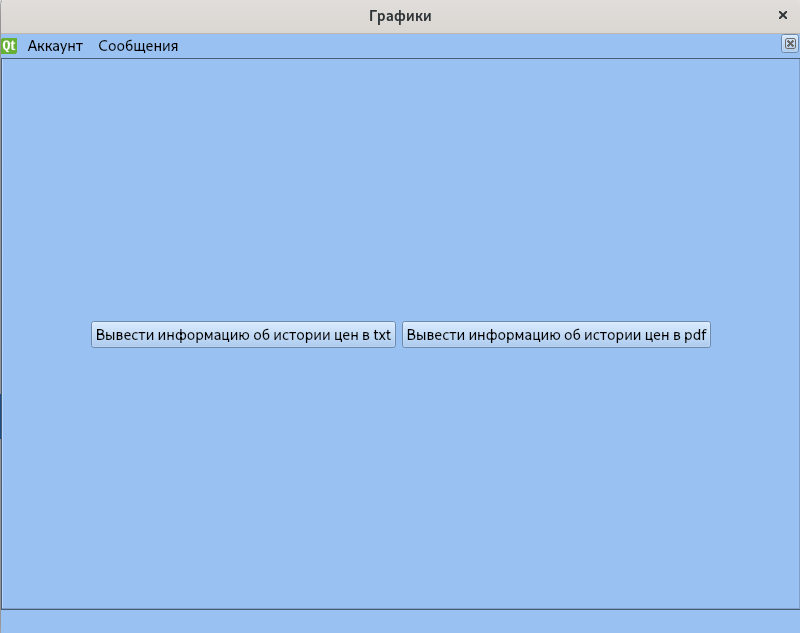


Рисунок 8.51 – Окно выбора отчета

Результат вывода отчета в txt представлен на рисунке 8.52

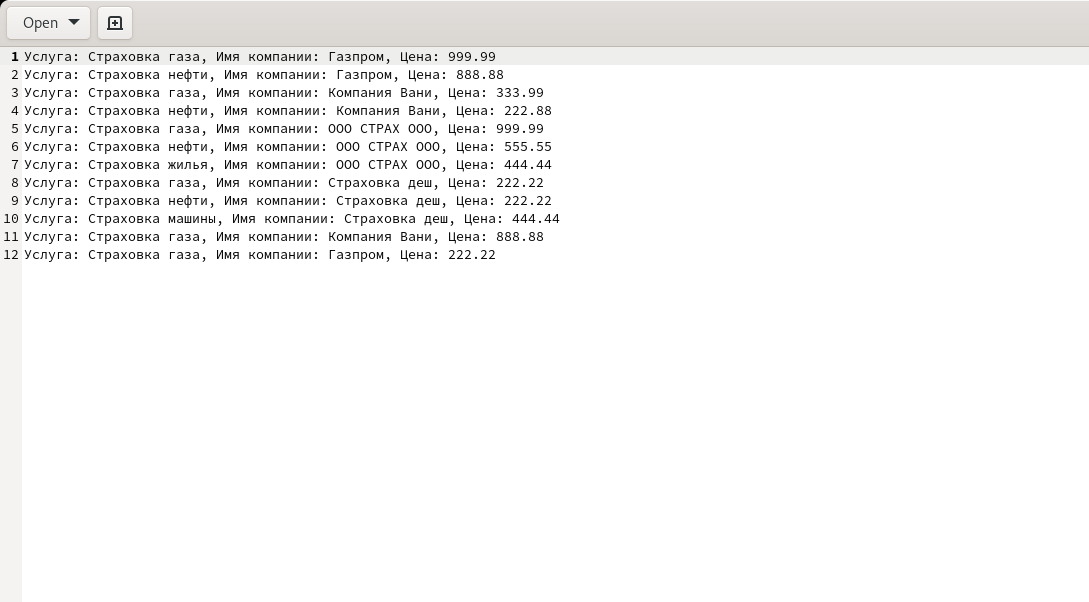


Рисунок 8.52 – Результат вывода отчета в txt

Результат вывода отчета в pdf представлен на рисунке 8.53

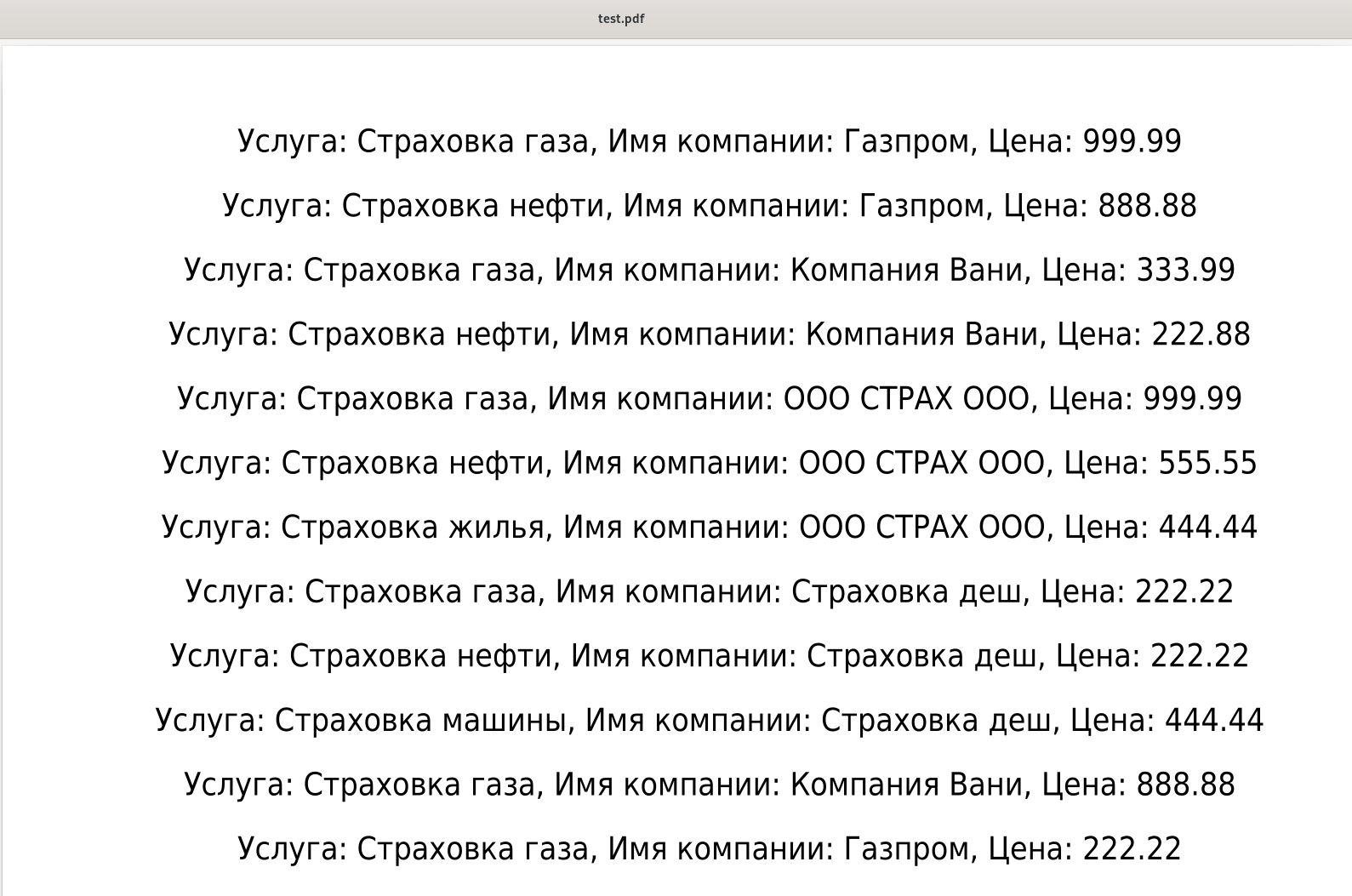


Рисунок 8.53 – Результат вывода отчета в pdf

# Заключение

В ходе выполнения курсового проекта был изучен процесс и разработано программное средство для автоматизации деятельности брокерского агентства. Программа разрабатывалась преимущественно для использования клиентами и сотрудниками брокерского агентства.

Было проведено исследование предметной области проекта (брокерского агентства), а также созданы структуры организации хранимых данных, диаграммы классов, диаграмма usecase, диаграмма развертывания, схема функционального моделирования на основе стандарта IDEF0, диаграмма состояний, диаграмма последовательности, диаграмма развёртывания и разработана схема базы данных для более детального понимания работы программы.

Приложение было разработано для удобной обработки заказов и услуг брокерского агентства, было добавлено описание функциональности всей программы. Был разработан красивый и удобный графический интерфейс для взаимодействия с пользователем. Программа может обслуживать множество клиентов одновременно, также было предусмотрено возникновение различных ошибок, которые могут возникнуть при работе с программой. Таким образом, программа выполняет обработку исключительных ситуаций и сообщает пользователю о его ошибочных действиях.

Практическая значимость заключается в том, был достигнут определенный уровень автоматизации, который позволил сократить издержки, непроизводственные трудозатраты за счёт повышения функциональности и увеличения скорости взаимодействия между клиентами и сотрудниками агентства.

Благодаря хорошей проектировки данной программы в дальнейшем она может легко быть усовершенствована, расширена другим функционалом и дополнена новыми деталями.

В заключении проведенного исследования можно сказать, что поставленные цели и задачи были реализованы в полной мере. Безусловно, разработанное программное обеспечение позволило значительно повысить эффективность обработки заказов услуг и взаимодействия клиентов с персоналом брокерского агентства.

Список использованных источников

[1] Бизне$менам – Интернет журнал о бизнесе, недвижимости и финансах [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://biznesmenam.com/informatsiya>

[2] Transinfo – База знаний [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://wiki.transinfo.by/kto-takoi-tamojennii-broker>

[3] Metanit – Руководство по языку Python [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://metanit.com/python/tutorial>

[4] Professor Web – Протоколы TCP и UDP [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://professorweb.ru>

[5] Современные технологии программирования. Лабораторный практикум: учеб.-метод. пособие / А. П. Пашкевич, О.А. Чумаков, – Мн.: БГУИР, 2007. – 106с.

[6] Studme.org – Модилирование бизнес-процессов [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://studme.org/87187/ekonomika/metodologiya_idef1x>

[7] Lucidchart – Блок-схемы [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.lucidchart.com/pages/ru/блок-схема>

[8] Informicus – Диаграммы классов UML. Логическое моделирование [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=3>

# Приложение А

**(обязательное)**

**Листинг кода**

# This Python file uses the following encoding: utf-8

import sys

import time

from PyQt5 import QtCore, uic

from PyQt5.QtWidgets import QMainWindow, QApplication, QMessageBox, \

QMdiArea, QWidget, QLineEdit, QFileDialog

from PyQt5.QtCore import pyqtSignal, Qt, QFile, QDir, QSize

from PyQt5.QtGui import QPixmap, QPalette, QImage, QBrush

from client\_socket import ClientSocket

from stack\_of\_widgets import StackOfWidgets

from initial\_widget import InitialWidget

from authentification\_widget import AuthentificationWidget

from registration\_widget import RegistrationWidget

from account\_info\_widget import AccountInfoWidget

from user\_menu import UserMenu

from consultant\_menu import ConsultationMenu

from consultation\_chat import ConsultationChat

from broker\_menu import BrokerMenu

from company\_widget import CompanyWidget

from service\_widget import ServiceWidget

from plot\_choice import PlotChoice

from histogram import Histogram

from linear\_plot import LinearPlot

from report\_widget import ReportWidget

from broker\_client import BrokerClient

from fpdf import FPDF

class MainWindow(QMainWindow):

def \_\_init\_\_(self, client\_socket):

super(MainWindow, self).\_\_init\_\_()

uic.loadUi('Form/main\_window\_form.ui', self)

self.client\_socket = client\_socket

self.stack\_of\_widgets = StackOfWidgets()

self.client\_socket.onReadyRead.connect(self.handleRespond)

self.chat\_wnd = None

self.chat = None

Продолжение приложения А

#---------------CREATION WIDGETS BELOW------------------------------------------

self.init\_wdg = InitialWidget()

self.auth\_wdg = AuthentificationWidget()

self.regi\_wdg = RegistrationWidget()

#--------------SIGNAL-SLOT WIDGETS CONNECTIONS BELOW----------------------------

self.init\_wdg.startButtonClicked.connect(

self.slotInitStartButtonClicked

)

self.init\_wdg.close.connect(

self.childClosed

)

self.auth\_wdg.authentificationButtonClicked.connect(

self.slotAuthentificationButtonClicked

)

self.auth\_wdg.registrationButtonClicked.connect(

self.slotRegistrationButtonClicked

)

self.auth\_wdg.close.connect(

self.childClosed

)

self.regi\_wdg.createButtonClicked.connect(

self.slotCreateAccountButtonClicked

)

self.regi\_wdg.backButtonClicked.connect(

self.slotBackFromRegistrationButtonClicked

)

self.regi\_wdg.close.connect(

self.childClosed

)

#------------SIGNAL-SLOT MAINWINDOW MENUBAR CONNECTIONS BELOW-------------------

self.accInfo.triggered.connect(self.slotAccInfoButtonClicked)

self.quitAccount.triggered.connect(self.slotQuitAccountClicked)

#----------------------------INIT ACTIONS---------------------------------------

self.mdiArea = QMdiArea()

self.setCentralWidget(self.mdiArea)

self.setWindowTitle('')

self.stack\_of\_widgets.push(self.init\_wdg)

self.sourcePalette = self.palette()

#------------------------INITIAL WIDGET SLOTS BELOW---------------------------------

def slotInitStartButtonClicked(self):

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self.auth\_wdg)

Продолжение приложения А

#----------------AUTHENTIFICATION WIDGET SLOTS BELOW--------------------------------

def slotAuthentificationButtonClicked(self):

self.client\_socket.sendToServer('SIGN IN ACCOUNT~!#$~login:' + \

self.auth\_wdg.login + '~!#$~password:' + self.auth\_wdg.passw

)

def slotRegistrationButtonClicked(self):

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self.regi\_wdg)

#---------------REGISTRATION WIDGET SLOTS BELOW-------------------------------------

def slotCreateAccountButtonClicked(self):

self.client\_socket.sendToServer('REGISTRATION ACCOUNT~!#$~login:' + \

self.regi\_wdg.login + '~!#$~password:' + self.regi\_wdg.passw

)

def slotBackFromRegistrationButtonClicked(self):

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self.auth\_wdg)

#-----------------MAINWINDOW MENUBAR SLOTS BELOW------------------------------------

def slotAccInfoButtonClicked(self):

self.info\_wdg = AccountInfoWidget()

self.info\_wdg.saveButtonClicked.connect(

self.slotSaveAccountInfoButtonClicked

)

self.info\_wdg.deleteAccountButtonClicked.connect(

self.slotDeleteAccountButtonClicked

)

self.info\_wdg.uploadThemeButtonClicked.connect(

self.slotUploadThemeButtonClicked

)

self.info\_wdg.deleteThemeButtonClicked.connect(

self.slotDeleteThemeButtonClicked

)

self.client\_socket.sendToServer('GET ACCOUNT INFO~!#$~login:' + \

self.auth\_wdg.login

)

wnd = self.mdiArea.addSubWindow(self.info\_wdg)

wnd.setWindowTitle('Личная информация')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

def slotQuitAccountClicked(self):

if self.mdiArea.activeSubWindow() == self.chat:

self.client\_socket.sendToServer("FINISH CHAT~!#$~" + \

self.chat\_wnd.login + "~!#$~" + self.chat\_wnd.companion

Продолжение приложения А

)

self.auth\_wdg.clearLines()

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self.auth\_wdg)

self.mdiArea = QMdiArea()

self.setCentralWidget(self.mdiArea)

#--------------INFO WINDGET SLOTS BELOW--------------------------------------------

def slotUploadThemeButtonClicked(self):

login = self.auth\_wdg.loginLineEdit.text()

filename = QFileDialog.getOpenFileName(

self,

"Выберите файл",

QDir.homePath()

)

profilesPath = QDir.currentPath() + '/Profiles/'

profilesDir = QDir(profilesPath)

if not QDir(profilesPath + login).exists():

profilesDir.mkdir(login)

if QFile(profilesPath + login + '/background.jpg').exists():

QFile.remove(profilesPath + login + '/background.jpg')

QFile.copy(filename[0], profilesPath + login + '/background.jpg')

self.setStyleSheet("");

originalImage = QImage(profilesPath + login + '/background.jpg')

scaledImage = originalImage.scaled(

QSize(self.frameGeometry().width(), self.frameGeometry().height())

)

palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(scaledImage))

self.setPalette(palette)

def slotDeleteThemeButtonClicked(self):

login = self.auth\_wdg.loginLineEdit.text()

profilesPath = QDir.currentPath() + '/Profiles/'

if QFile(profilesPath + login + '/background.jpg').exists():

QFile.remove(profilesPath + login + '/background.jpg')

self.setStyleSheet("background-color: rgb(153, 193, 241);");

def slotSaveAccountInfoButtonClicked(self):

self.client\_socket.sendToServer('REDO ACCOUNT INFO~!#$~login:' + \

self.info\_wdg.login + '~!#$~password:' + self.info\_wdg.passw + \

'~!#$~mob\_num:' + self.info\_wdg.mobnum + '~!#$~email:' + self.info\_wdg.email

)

Продолжение приложения А

def slotDeleteAccountButtonClicked(self):

if self.info\_wdg.passw == self.info\_wdg.passw\_:

Продолжение приложения А

profilesPath = QDir.currentPath() + '/Profiles/'

currentDir = QDir(profilesPath + self.info\_wdg.login)

currentDir.removeRecursively()

self.client\_socket.sendToServer('DELETE ACCOUNT~!#$~login:' + \

self.auth\_wdg.login

)

else:

QMessageBox.about(self, "Уведомление", "Неверный пароль")

#----------------------------------------MAINWINDOW SLOTS----------------------------------

def closeEvent(self, event):

if self.chat\_wnd:

self.client\_socket.sendToServer("FINISH CHAT~!#$~" + \

self.chat\_wnd.login + "~!#$~" + self.chat\_wnd.companion

)

elif self.auth\_wdg.rolename == 'USER':

self.client\_socket.sendToServer("DELETE BASKET~!#$~login:" + \

self.auth\_wdg.login + "~!#$~name:" + \

self.auth\_wdg.login)

self.client\_socket.sendToServer('EXIT')

def resizeEvent(self, event):

login = self.auth\_wdg.loginLineEdit.text()

profilesPath = QDir.currentPath() + '/Profiles/'

if QFile(profilesPath + login + '/background.jpg').exists():

palette = QPalette()

originalImage = QImage(profilesPath + login + '/background.jpg')

scaledImage = originalImage.scaled(

self.size(), Qt.KeepAspectRatioByExpanding, transformMode = Qt.SmoothTransformation

)

palette.setBrush(QPalette.Window, QBrush(scaledImage))

self.setPalette(palette)

#-----------------MAINWINDOW FUNCTION BELOW------------------------------------------------

def childClosed(self):

if self.chat\_wnd:

self.client\_socket.sendToServer("FINISH CHAT~!#$~" + \

self.chat\_wnd.login + "~!#$~" + self.chat\_wnd.companion

)

elif self.auth\_wdg.rolename == 'USER':

Продолжение приложения А

self.client\_socket.sendToServer("DELETE BASKET~!#$~login:" + \

self.auth\_wdg.login + "~!#$~name:" + \

self.auth\_wdg.login)

self.client\_socket.sendToServer('EXIT')

def slotCreateConsultationClicked(self):

self.chat\_wnd = ConsultationChat()

self.chat\_wnd.sendMessage.connect(lambda: self.client\_socket.sendToServer("MESSAGE~!#$~" + \

self.chat\_wnd.login + "~!#$~" + self.chat\_wnd.companion + "~!#$~" + self.chat\_wnd.message

)

)

self.chat\_wnd.finishChat.connect(

self.slotFinishChatClicked

)

self.chat\_wnd.setParticipants(self.auth\_wdg.login, self.cons\_wdg.currentUser)

self.client\_socket.sendToServer("START CHAT~!#$~" + self.auth\_wdg.login + "~!#$~" + self.cons\_wdg.currentUser)

self.client\_socket.sendToServer("DELETE BASKET~!#$~login:" + \

self.chat\_wnd.companion + "~!#$~name:" + \

self.chat\_wnd.companion

)

self.chat = self.mdiArea.addSubWindow(self.chat\_wnd)

self.chat.setWindowTitle('Чат')

self.chat.setWindowFlags(QtCore.Qt.FramelessWindowHint);

self.chat.showMaximized()

def slotFinishChatClicked(self):

self.chat.close()

self.client\_socket.sendToServer("FINISH CHAT~!#$~" + \

self.chat\_wnd.login + "~!#$~" + self.chat\_wnd.companion

)

def slotCompanyClicked(self):

self.comp\_wdg = CompanyWidget()

self.comp\_wdg.addButtonClicked.connect(lambda: self.client\_socket.sendToServer('ADD COMPANY~!#$~company\_name:' + \

self.comp\_wdg.selected\_company

)

)

self.comp\_wdg.changeButtonClicked.connect(lambda: self.client\_socket.sendToServer('CHANGE COMPANY~!#$~old\_name:' + \

self.comp\_wdg.selected\_company + '~!#$~company\_name:' + \

self.comp\_wdg.new\_name

Продолжение приложения А

)

)

self.comp\_wdg.deleteButtonClicked.connect(lambda: self.client\_socket.sendToServer('DELETE COMPANY~!#$~company\_name:' + \

self.comp\_wdg.selected\_company

)

)

self.comp\_wdg.itemDoubleClicked.connect(

self.slotItemDoubleClicked

)

self.client\_socket.sendToServer('GET COMPANY')

wnd = self.mdiArea.addSubWindow(self.comp\_wdg)

wnd.setWindowTitle('Работа с компаниями')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

def slotItemDoubleClicked(self):

self.serv\_wdg = ServiceWidget()

self.serv\_wdg.addButtonClicked.connect(lambda: self.client\_socket.sendToServer('ADD SERVICE~!#$~service\_name:' + \

self.serv\_wdg.service + '~!#$~price:' + self.serv\_wdg.price + '~!#$~company\_name:' + self.comp\_wdg.selected\_company

)

)

self.serv\_wdg.changeButtonClicked.connect(lambda: self.client\_socket.sendToServer('CHANGE SERVICE~!#$~service\_name:' + \

self.serv\_wdg.service + '~!#$~price:' + self.serv\_wdg.price + '~!#$~company\_name:' + self.comp\_wdg.selected\_company

)

)

self.serv\_wdg.deleteButtonClicked.connect(lambda: self.client\_socket.sendToServer('DELETE SERVICE~!#$~service\_name:' + \

self.serv\_wdg.service + '~!#$~company\_name:' + self.comp\_wdg.selected\_company

)

)

self.client\_socket.sendToServer('GET SERVICE~!#$~company\_name:' + self.comp\_wdg.selected\_company)

wnd = self.mdiArea.addSubWindow(self.serv\_wdg)

wnd.setWindowTitle('Работа с услугами')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

def uploadTheme(self):

login = self.auth\_wdg.loginLineEdit.text()

profilesPath = QDir.currentPath() + '/Profiles/'

if QFile(profilesPath + login + '/background.jpg').exists():

Продолжение приложения А

self.setStyleSheet("");

originalImage = QImage(profilesPath + login + '/background.jpg')

scaledImage = originalImage.scaled(

QSize(self.frameGeometry().width(), self.frameGeometry().height())

)

palette = QPalette()

palette.setBrush(QPalette.Window, QBrush(scaledImage))

self.setPalette(palette)

else:

self.setStyleSheet("background-color: rgb(153, 193, 241);");

def slotPlotClicked(self):

self.plot\_choice = PlotChoice()

self.plot\_choice.histogramClicked.connect(self.slotHistogramClicked)

self.plot\_choice.linearClicked.connect(self.slotLinearClicked)

wnd = self.mdiArea.addSubWindow(self.plot\_choice)

wnd.setWindowTitle('Графики')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

def slotHistogramClicked(self):

self.client\_socket.sendToServer('GET DATA FOR HISTOGRAM')

def slotLinearClicked(self):

self.client\_socket.sendToServer('GET SERVICES ONLY')

def slotReportClicked(self):

self.report\_wid = ReportWidget()

self.report\_wid.txtButtonClicked.connect(lambda: self.client\_socket.sendToServer('GET PRICE HISTORY'))

self.report\_wid.pdfButtonClicked.connect(lambda: self.client\_socket.sendToServer('GET PRICE HISTORY'))

wnd = self.mdiArea.addSubWindow(self.report\_wid)

wnd.setWindowTitle('Графики')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

def slotOrderClicked(self):

self.brcl\_wgd = BrokerClient()

self.brcl\_wgd.orderClicked.connect(self.slotBrokerClientOrderClicked)

self.client\_socket.sendToServer("GET ORDERS~!#$~type:SERVICE")

wnd = self.mdiArea.addSubWindow(self.brcl\_wgd)

wnd.setWindowTitle('Заказы')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

Продолжение приложения А

def slotBrokerClientOrderClicked(self):

self.client\_socket.sendToServer("DELETE BASKET~!#$~login:" + \

self.brcl\_wgd.login + "~!#$~name:" + \

self.brcl\_wgd.order

)

def handleRespond(self):

respond = self.client\_socket.get\_data()

print(respond)

respond\_list = respond.split('~!#$~')

command, args = "", []

match len(respond\_list):

case 1: command = respond\_list[0]

case \_: command, \*args = respond\_list

if command == 'REGISTRATION SUCCESSFUL':

self.regi\_wdg.clearLines()

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self.auth\_wdg)

QMessageBox.about(self, "Уведомление", "Регистрация успешна")

elif command == 'REGISTRATION FAILED':

self.regi\_wdg.setError('Ошибка регистрации')

elif command == 'AUTHENTIFICATION SUCCESSFUL' and args[0] == 'USER':

self.auth\_wdg.rolename = args[0]

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self)

self.user\_wdg = UserMenu()

self.user\_wdg.close.connect(

self.childClosed

)

self.user\_wdg.consultationButtonClicked.connect(

lambda: self.client\_socket.sendToServer("ADD BASKET~!#$~login:" + \

self.auth\_wdg.login + "~!#$~name:" + \

self.auth\_wdg.login + "~!#$~type:CONSULTATION"

)

)

self.user\_wdg.orderButtonClicked.connect(lambda: self.client\_socket.sendToServer("ADD BASKET~!#$~login:" + \

self.auth\_wdg.login + "~!#$~name:" + self.user\_wdg.currentService + "~!#$~type:SERVICE"

)

)

Продолжение приложения А

self.client\_socket.sendToServer("GET AVG PRICE AND SERVICE")

wnd = self.mdiArea.addSubWindow(self.user\_wdg)

wnd.setWindowTitle('Меню')

wnd.setWindowFlags(QtCore.Qt.FramelessWindowHint);

wnd.showMaximized()

self.uploadTheme()

elif command == 'AUTHENTIFICATION SUCCESSFUL' and args[0] == 'CONSULTANT':

self.auth\_wdg.rolename = args[0]

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self)

self.cons\_wdg = ConsultationMenu()

self.cons\_wdg.close.connect(

self.childClosed

)

self.cons\_wdg.cancelConsultation.connect(

lambda: self.client\_socket.sendToServer("DELETE BASKET~!#$~login:" + \

self.cons\_wdg.currentUser + "~!#$~name:" + \

self.cons\_wdg.currentUser

)

)

self.cons\_wdg.createConsultation.connect(

self.slotCreateConsultationClicked

)

self.client\_socket.sendToServer("GET BASKET~!#$~type:CONSULTATION")

self.cons\_wdg.showRequests.setEnabled(False)

wnd = self.mdiArea.addSubWindow(self.cons\_wdg)

wnd.setWindowTitle('Меню')

wnd.setWindowFlags(QtCore.Qt.FramelessWindowHint);

wnd.showMaximized()

self.uploadTheme()

elif command == 'AUTHENTIFICATION SUCCESSFUL' and args[0] == 'BROKER':

self.auth\_wdg.rolename = args[0]

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self)

self.brok\_wdg = BrokerMenu()

self.brok\_wdg.close.connect(

self.childClosed

)

self.brok\_wdg.companyClicked.connect(

self.slotCompanyClicked

)

self.brok\_wdg.plotClicked.connect(

self.slotPlotClicked

)

self.brok\_wdg.reportClicked.connect(

self.slotReportClicked

Продолжение приложения А

)

self.brok\_wdg.orderClicked.connect(

self.slotOrderClicked

)

wnd = self.mdiArea.addSubWindow(self.brok\_wdg)

wnd.setWindowTitle('Меню')

wnd.setWindowFlags(QtCore.Qt.FramelessWindowHint);

wnd.showMaximized()

self.uploadTheme()

elif command == 'AUTHENTIFICATION FAILED':

self.auth\_wdg.setError(args[0])

elif command == 'REDO ACC INFO SUCCESS':

QMessageBox.about(self, "Уведомление", "Данные изменены успешно")

self.info\_wdg.setError('')

elif command == 'REDO ACC INFO FAILED':

self.info\_wdg.setError('Ошибка. Некорректные данные')

elif command == 'DELETING SUCCESSFUL':

QMessageBox.about(self, "Уведомление", "Аккаунт удален")

self.auth\_wdg.clearLines()

self.stack\_of\_widgets.pop()

self.stack\_of\_widgets.push(self.auth\_wdg)

elif command == 'DELETING FAILED':

QMessageBox.about(self, "Уведомление", "Ошибка удаления аккаунта")

elif command == "GET INFORMATION SUCCESS":

self.info\_wdg.setInfo(\*args)

elif command == "GETDATA FAILED":

self.info\_wdg.setError("Ошибка при соединении с сервером")

elif command == "GET BASKET SUCCESS" and self.auth\_wdg.rolename == 'CONSULTANT':

if args:

self.cons\_wdg.setConsultation(args)

else:

self.cons\_wdg.clearList()

elif command == "GET BASKET SUCCES" and self.auth\_wdg.rolename == 'BROKER':

pass

elif command == "GET BASKET FAILED" and self.auth\_wdg.rolename == 'CONSULTANT':

self.cons\_wdg.setError('Ошибка при соединении с сервером')

elif command == "GET BASKET FAILED" and self.auth\_wdg.rolename == 'BROKER':

pass

elif command == 'REQUEST AGAIN' and self.auth\_wdg.rolename == 'CONSULTANT':

self.client\_socket.sendToServer("GET BASKET~!#$~type:CONSULTATION")

elif command == "REQUEST AGAIN" and self.auth\_wdg.rolename == 'BROKER':

self.client\_socket.sendToServer("GET ORDERS~!#$~type:SERVICE")

elif command == "DELETE BASKET FAILED" and self.auth\_wdg.rolename == "CONSULTANT":

Продолжение приложения А

pass

elif command == "START CHAT" and self.auth\_wdg.login == args[1]:

companion, login = args

self.chat\_wnd = ConsultationChat()

self.chat\_wnd.sendMessage.connect(lambda: self.client\_socket.sendToServer("MESSAGE~!#$~" + \

self.chat\_wnd.login + "~!#$~" + self.chat\_wnd.companion + "~!#$~" + self.chat\_wnd.message

)

)

self.chat\_wnd.finishChat.connect(

self.slotFinishChatClicked

)

self.chat\_wnd.setParticipants(login, companion)

self.chat = self.mdiArea.addSubWindow(self.chat\_wnd)

self.chat.setWindowTitle('Чат')

self.chat.setWindowFlags(QtCore.Qt.FramelessWindowHint);

self.chat.showMaximized()

elif command == "MESSAGE" and self.auth\_wdg.login == args[1]:

if self.auth\_wdg.rolename == 'USER':

companion = 'Консультант'

else:

companion = 'Клиент'

self.chat\_wnd.setMessage(companion, args[2])

elif command == "FINISH CHAT" and self.auth\_wdg.login == args[1]:

if self.chat\_wnd:

self.chat.close()

self.chat\_wnd = None

elif command == "GET COMPANY SUCCESS" and self.auth\_wdg.rolename == 'BROKER':

self.comp\_wdg.clearError()

self.comp\_wdg.setCompanies(args)

elif command == "GET COMPANY FAILED" and self.auth\_wdg.rolename == 'BROKER':

self.comp\_wdg.setError('Ошибка при соединении с сервером')

elif command == 'DELETE COMPANY FAILED' and self.auth\_wdg.rolename == 'BROKER':

self.comp\_wdg.setError('Ошибка при соединении с сервером')

elif command == 'ADD COMPANY FAILED' and self.auth\_wdg.rolename == 'BROKER':

self.comp\_wdg.setError('Ошибка! Такая компания уже существует')

elif command == 'CHANGE COMPANY FAILED' and self.auth\_wdg.rolename == 'BROKER':

self.comp\_wdg.setError('Ошибка! Новое имя компании уже существует')

elif command == 'REQUEST COMPANIES' and self.auth\_wdg.rolename == 'BROKER':

self.client\_socket.sendToServer('GET COMPANY')

elif command == "GET SERVICE SUCCESS" and self.auth\_wdg.rolename == 'BROKER':

self.serv\_wdg.clearError()

Продолжение приложения А

self.serv\_wdg.setServices(args)

elif command == "GET SERVICE FAILED" and self.auth\_wdg.rolename == 'BROKER':

self.serv\_wdg.setError('Ошибка при соединении с сервером')

elif command == 'DELETE SERVICE FAILED' and self.auth\_wdg.rolename == 'BROKER':

self.serv\_wdg.setError('Ошибка при соединении с сервером')

elif command == 'ADD SERVICE FAILED' and self.auth\_wdg.rolename == 'BROKER':

self.serv\_wdg.setError('Ошибка! Такая услуга уже существует')

elif command == 'CHANGE SERVICE FAILED' and self.auth\_wdg.rolename == 'BROKER':

self.serv\_wdg.setError('Ошибка! Новое имя услуги уже существует')

elif command == 'REQUEST SERVICES' and self.auth\_wdg.rolename == 'BROKER':

self.client\_socket.sendToServer('GET SERVICE~!#$~company\_name:' + self.comp\_wdg.selected\_company)

elif command == "CREATE LINEAR" and self.auth\_wdg.rolename == 'BROKER':

companies, prices = [], []

for x in args:

com, pr = x.split(':')

companies.append(com)

prices.append(float(pr))

self.linear.createLinearPlot(companies, prices)

elif command == 'GET SERVICE ONLY SUCCESS' and self.auth\_wdg.rolename == 'BROKER':

self.linear = LinearPlot()

self.linear.setServices(args)

self.linear.selOption.connect(lambda: self.client\_socket.sendToServer('CREATE LINEAR~!#$~service\_name:' + self.linear.currName()))

wnd = self.mdiArea.addSubWindow(self.linear)

wnd.setWindowTitle('Линейный график')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

self.client\_socket.sendToServer('CREATE LINEAR~!#$~service\_name:' + self.linear.currName())

elif command == 'CREATE HISTOGRAM' and self.auth\_wdg.rolename == 'BROKER':

companies\_prices, std\_list = [], []

for x in args:

std, com\_pr = x.split(':')

companies\_prices.append(com\_pr)

std\_list.append(std)

self.hist.createHistogram(companies\_prices, std\_list)

elif command == 'GET DATA FOR HISTOGRAM SUCCESS' and self.auth\_wdg.rolename == 'BROKER':

self.hist = Histogram()

self.hist.setServices(args)

self.hist.selOption.connect(lambda: self.client\_socket.sendToServer('CREATE HISTOGRAM~!#$~service\_name:' + self.hist.currName()))

wnd = self.mdiArea.addSubWindow(self.hist)

Продолжение приложения А

wnd.setWindowTitle('Работа с компаниями')

wnd.setWindowFlags(QtCore.Qt.Dialog);

wnd.showMaximized()

elif command == 'GET PRICE HISTORY SUCCESS' and self.auth\_wdg.rolename == 'BROKER':

if self.report\_wid.format:

file = open(self.report\_wid.filename + '.txt', 'w')

for x in args:

file.write(x + '\n')

file.close()

else:

pdf = FPDF()

pdf.add\_page()

pdf.add\_font('DejaVu', '', 'font/DejaVuSansCondensed.ttf', uni=True)

pdf.set\_font('DejaVu', '', 14)

for x in args:

pdf.cell(200, 10, txt=x, ln=1, align="C")

pdf.output(self.report\_wid.filename + '.pdf')

elif command == "GET AVG PRICE AND SERVICE SUCCESS" and self.auth\_wdg.rolename == "USER":

avgs, services = [], []

for x in args:

price, service = x.split(':')

avgs.append(price)

services.append(service)

data = dict(zip(services, avgs))

self.user\_wdg.setServices(data)

elif command == "GET ORDERS SUCCESS" and self.auth\_wdg.rolename == "BROKER":

self.brcl\_wgd.setData(args)

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication([])

client\_socket = ClientSocket('localhost', 6606)

main\_window = MainWindow(client\_socket)

sys.exit(app.exec\_())