

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования
Кафедра экономической информатики
Дисциплина «Программирование сетевых приложений»

«К ЗАЩИТЕ ДОПУСТИТЬ»

Руководитель курсового проекта
магистр.эконом.наук, старший
преподаватель

_____. Д. А. Сторожев
_____.2022

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**«КРЕДИТНАЯ ПОЛИТИКА БАНКА И АВТОМАТИЗАЦИЯ ВЫДАЧИ
КРЕДИТОВ ФИЗИЧЕСКИМ ЛИЦАМ»**

БГУИР КР 1-40 05 01-10 017 ПЗ

Выполнил студент группы 014301
НЕВЕЙКОВ Андрей Сергеевич

(подпись студента)

Курсовой проект представлен на
проверку _____.12.2022

(подпись студента)

Минск 2022

СОДЕРЖАНИЕ

Введение	4
1 Описание системы работы банка.....	6
1.1 Описание принципов работы	6
1.2 Процесс выдачи кредита	7
1.3 Автоматизация необходимых процессов	7
2 Постановка задачи и обзор методов ее решения	8
3 Функциональное моделирование на основе стандарта IDEF0.....	10
4 Информационная модель системы и ее описание	16
5 Обоснование оригинальных решений по использованию технических и программных средств, не включенных в требования	19
5.1 Диаграмма вариантов использования.....	19
5.2 Диаграмма развертывания и компонентов системы.....	20
5.3 Диаграмма классов	22
5.4 Диаграмма последовательности.....	24
5.5 Диаграмма состояний.....	25
6 Описание алгоритмов, реализующих бизнес-логику серверной части проектируемой системы	27
6.1 Алгоритм авторизации в программе.....	27
6.2 Алгоритм работы программы.....	28
6.3 Алгоритм создания нового кредитного продукта	30
7 Руководство пользователя.....	32
7.1 Запуск приложения.....	32
7.2 Функции продавца-консультанта	34
7.3 Функции менеджера	36
7.4 Функции администратора	40
8 Результаты тестирования разработанной системы.....	41
Заключение	44
Список использованных источников	45
Приложение А (обязательное) Листинг кода.....	46
Приложение Б (обязательное) Листинг скрипта генерации базы данных	58
Приложение В (обязательное) Отчет о проверке на заимствования в системе «Антиплагиат».....	60

ВВЕДЕНИЕ

На сегодняшний день развитие интернет-технологий позволяет оптимизировать большинство процессов, связанных с оптимизацией повторяющихся операций, требующих математических вычислений. Одной из таких операций является процесс выдачи кредитов физическим лицам.

Так как выдача кредитов является одной из самых больших статей дохода банков, применение информационных технологий в основе кредитной политике банка может ускорить процесс выдачи кредита, сделать доходность кредитования стабильнее[1], путём исключения человеческого фактора при принятии решений и следования алгоритму в скоринговой модели.

Также внедрение системы автоматизации выдачи кредитов поможет банку сократить штат сотрудников, которые изучали кредитную историю человека, рассчитывали платежеспособность клиента, предлагали кредитные продукты и занимались обучением персонала, способного выполнять эту работу.

Еще одним достоинством разработанной информационной системы является то, что при увеличении числа потребителей, скорость принятия решения останется постоянной. Одинаковое число сотрудников физически не сможет обрабатывать вручную в 3, 5 или 10 раз больше заявок, без увеличения времени ожидания для клиентов в "конце очереди" или снижения качества проверок, а программное средство может.

Объектами исследования являются процесс выдачи кредитов физическим лицам и кредитная политика банка. Предмет исследования – создание системы принятия решений о возможности выдачи кредита физическому лицу.

Таким образом, целью данного курсового проекта является повышение экономической эффективности кредитной политики банка и автоматизация процесса выдачи кредитов.

Поставленная цель требует решения следующих задач:

- исследовать и описать предметную область;
- разработать и описать постановку задачи на разработку программного средства;
- выполнить проектирование программного средства, используя UML-диаграммы, IDEF0, IDEF1X;
- разработать и описать алгоритм работы программного средства;
- разработать и описать скоринговую модель;
- реализовать клиент-серверное приложение;
- реализовать взаимодействие с базой данных;

- создать визуальный интерфейс для удобной работы пользователя;
- разработать и описать руководство пользователя.

1 ОПИСАНИЕ СИСТЕМЫ РАБОТЫ БАНКА

1.1 Описание принципов работы

В условиях развития потребительского капитализма, увеличения числа потребительских кредитов и тенденции на переход к обществу потребления банкам необходимо обрабатывать всё большее количество заявок и принимать решения о выдаче кредита, связанные с финансовыми рисками для банка.

При этом у людей, которые планируют брать кредит, различные цели и финансовые возможности. Кто-то может позволить себе кредит на автомобиль премиум класса, без рисков для финансового благополучия, для кого-то выплаты по кредиту на микроволновку могут снизить уровень жизни. Основной целью банка является получение прибыли. Исходя из данной цели, необходимо найти баланс между следующими принципами:

1. Кредит должен приносить доход, но при повышении процента кредитования и суммы кредита, выше риск его невыплаты или выплаты не в полном объеме, а также существенное потери в денежном выражении.

2. Риск его невыплаты должен быть минимален, чем меньше резервов, тем выше риск кассового разрыва и существенных потерь при кризисе.

3. Кредиторы должны быть надежными, для создания положительной репутации, но чем выше уровень финансовой надежности, тем меньше людей ему соответствует, соответственно меньше кредитов выдается, соответственно меньше прибыль.

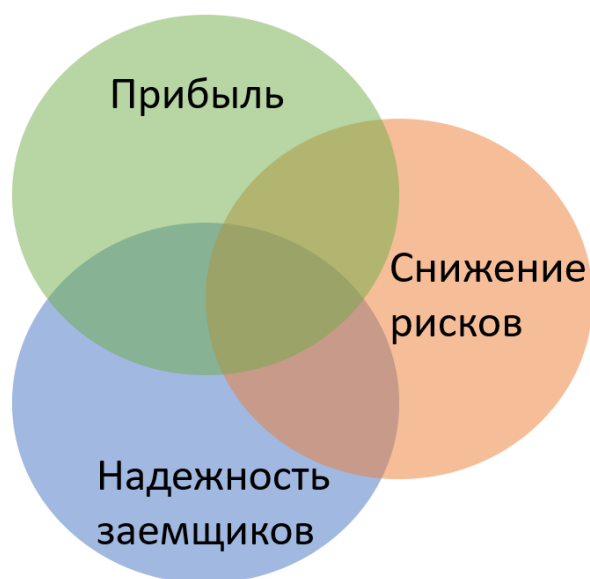


Рисунок 1.1 – Приоритетные факторы принятия решения при выдаче кредита

Также необходимо учитывать предложения других банков и быть конкурентноспособным.

1.2 Процесс выдачи кредита

Для выполнения процесса кредитование физических лиц, необходимы следующие сотрудники: кредитный менеджер, аналитик и продавец-консультант.

Продавец-консультант общается с клиентом, предлагает финансовые продукты, заполняет данные о клиенте и передает аналитику. Аналитик, используя математические модели, рассчитывает вероятные риски невыплат, ожидаемый доход для банка, выгодные условия кредитования, делает предварительное заключение и передает обобщенные расчеты менеджеру. Менеджер на основе данных от аналитика и общей финансовой ситуации в банке принимает финальное решение о выдаче кредита.

Таким образом, скорость и точность принятия решений зависит от количества и компетентности аналитиков, выполняющих стандартизированные процедуры, которые можно автоматизировать.

1.3 Автоматизация необходимых процессов

Автоматизация предприятия [3] — повышение эффективности его деятельности. В данном случае при помощи внедрения программного средства, в результате чего работа аналитика может проводиться полностью без участия человека, также часть функций продавца-консультанта и менеджера будет упрощена.

Автоматизировав функцию аналитика при помощи программного средства, продавец-консультант сможет моментально предлагать различные варианты условий кредита, что повысит привлекательность для клиентов. А с менеджера снимается часть ответственности за принятие решения так как снижается роль человеческого фактора в процессе и повышается роль системы, выполняемой строго запрограммированные функции.

2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

В данном курсовом проекте необходимо спроектировать клиент-серверное программное приложение, позволяющее автоматизировать процесс выдачи кредитов физическим лицам при некоторой кредитной политике.

Программный код должен быть написан на объектно-ориентированном языке Java, версии JDK7 и выше. Java отличается быстротой, высоким уровнем защиты и надежностью, а в версиях с 8-ой по 19-ую было реализовано множество функций и библиотек, способных помочь при решении поставленной задачи, например: введены лямбда-выражения, обеспечена безопасность транспортного уровня TLS, добавлены switch-выражения.

Для создания интерфейса была выбрана стандартная библиотека JavaFX. JavaFX позволяет создавать приложения с богатой насыщенной графикой при помощи использования аппаратного ускорения графики. По сравнению со Swing и рядом других подобных платформ, JavaFX предоставляет большие возможности. Это и широкий выбор элементов управления, и возможности по работе с мультимедиа, двух- и трехмерной графикой, декларативный способ описания интерфейса с помощью FXML, возможность изменения стиля интерфейса с помощью CSS, интеграция со Swing и другими графическими библиотеками.

Связь между сервером и клиентом в приложении осуществляется при помощи TCP. Передача данных по протоколу TCP (Transmission Control Protocol) предусматривает наличие подтверждений получения информации. Если же передающая сторона не получит в установленные сроки необходимого подтверждения, данные будут переданы повторно. Поэтому протокол TCP относят к протоколам, предусматривающим соединение, а UDP (User Datagram Protocol) - нет. UDP применяется в тех случаях, когда не требуется подтверждения приема (например, DNS-запросы или IP-телефония)[3]. То есть ключевая разница заключается в наличии подтверждения приема. В банковской сфере, при передаче информации между сервером и клиентом, потеря или повреждение данных недопустимы. Несмотря на то что реализация протокола TCP более сложная, и это отражается на его скорости, для небольшого количества пользователей внутренней сети банка разница в быстродействии будет незаметна. Поэтому был выбран более надежный протокол TCP.

Для соединения сервера с базой данных использовался JDBC-драйвер. JDBC – это стандарт взаимодействия с базами данных на Java. Его интерфейс поставляется в стандартной JDK в виде пакета java.sql. Низкоуровневый JDBC

лежит в основе большинства высокоуровневых библиотек работы с базой данных. JDBC-драйвер – реализация JDBC для определенной базы данных. В приложении может быть зарегистрировано несколько разных драйверов. При соединении к базе, нужный выбирается исходя из URL соединения.

Разработка приложения осуществлялась при помощи IntelliJ IDEA. IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java. Существенными преимуществами данной среды разработки являются удобный интерфейс, встроенный линтер, поддержка взаимодействия с различными СУБД, возможность установки плагинов, встроенный модуль JUnit для проведения тестов.

В качестве СУБД использовалась MySQL. Преимущества MySQL:

1. Открытый исходный код. Распространяется бесплатно для домашнего применения.
2. Простота. MySQL легко устанавливается, имеет понятный интерфейс, а разнообразие плагинов и дополнительных приложений упрощает работу с БД.
3. Функционал. Включает в себя практически весь необходимый набор инструментов, который может пригодиться при разработке любого проекта.
4. Безопасность. Многие системы безопасности уже встроены и работают по умолчанию.
5. Масштабируемость. Может использоваться в работе как с малым, так и с большим объемом данных.
6. Скорость. Является одной из самых быстрых среди имеющихся на современном рынке.

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

Основным процессом предметной области данного курсового проекта является процесс выдачи кредита физическому лицу. Это сложный процесс, включающий множество регламентированных процедур и операций, требующих особого внимания и выполнения всех требований и должностных инструкций. Следовательно, существует необходимость в изложении всех пунктов и действий каждой из сторон данной операции.

Главная бизнес-функция системы – это «миссия» системы, ее смысл в реальном мире. При её определении необходимо всегда иметь в виду цель моделирования и точку зрения на модель.

IDEF0 – нотация графического моделирования, используемая для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции. Особенность IDEF0 заключается в том, что эта методология ориентирована на соподчиненность объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность[4].

Впоследствии проведённого анализа предметной области курсового проекта можно представить функциональную модель процесса выдачи кредита физическому лицу. На рисунке 3.1 представлена контекстная диаграмма верхнего уровня.

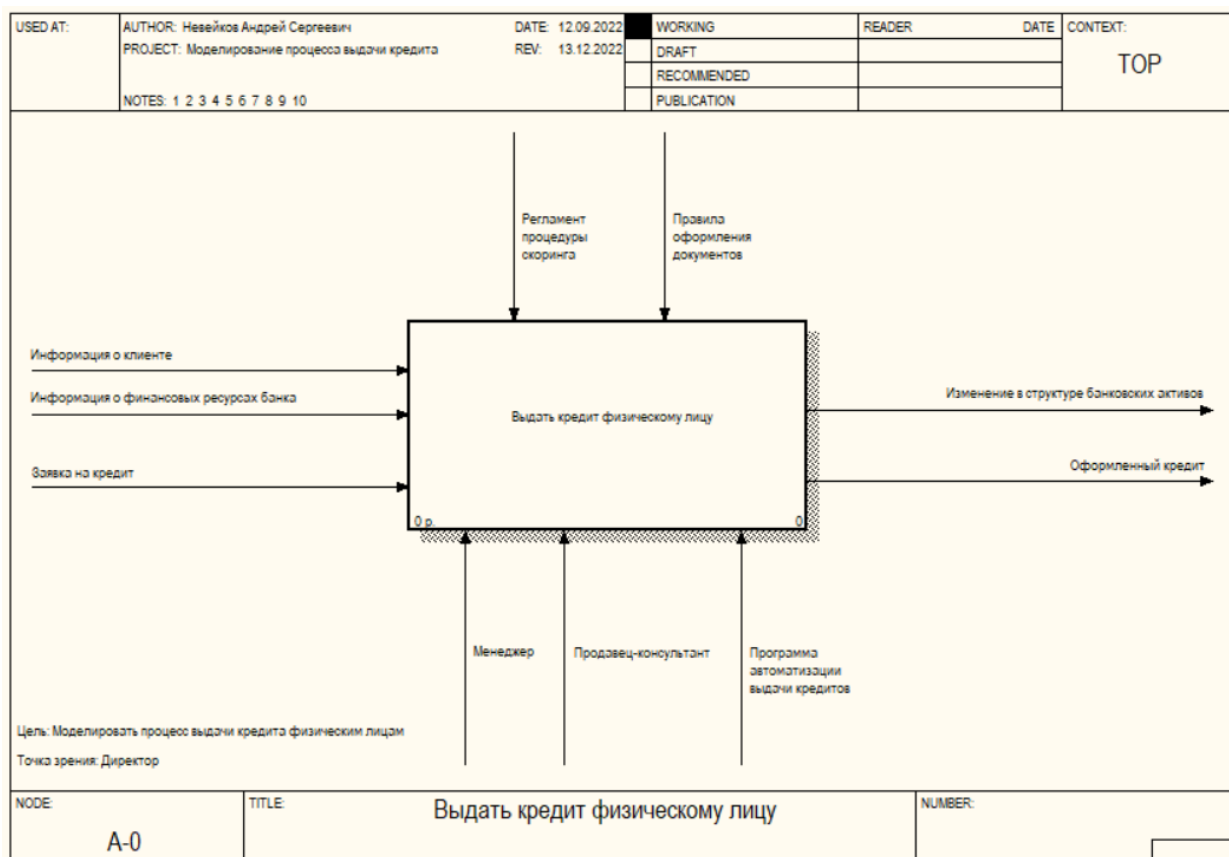


Рисунок 3.1 – Контекстная диаграмма

Далее представлена декомпозиция контекстной диаграммы, состоящая из трех блоков (рисунок 2.2):

1. Выбрать условия кредитования.
2. Анализировать платежеспособность клиента.
3. Оформить кредит.

Функциональная декомпозиция – разбиение системы на подсистемы, где каждая подсистема описывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности.

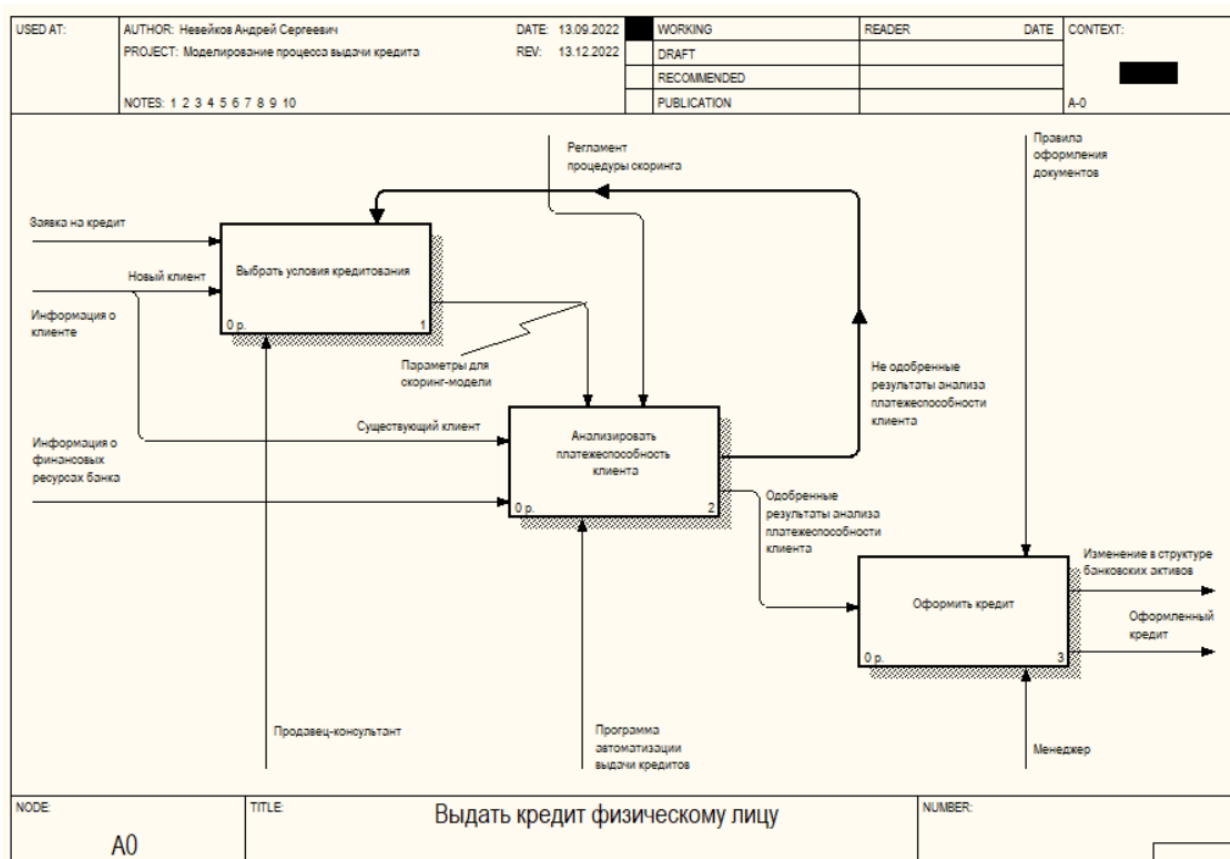


Рисунок 3.2 – Декомпозиция контекстной диаграммы

Этап «Выбрать условия кредитования» разбит на четыре функциональных блока (рисунок 3.3):

1. Направить к свободному консультанту.
2. Подобрать кредитный продукт.
3. Установить личность клиента.
4. Внести данные о клиенте в базу.

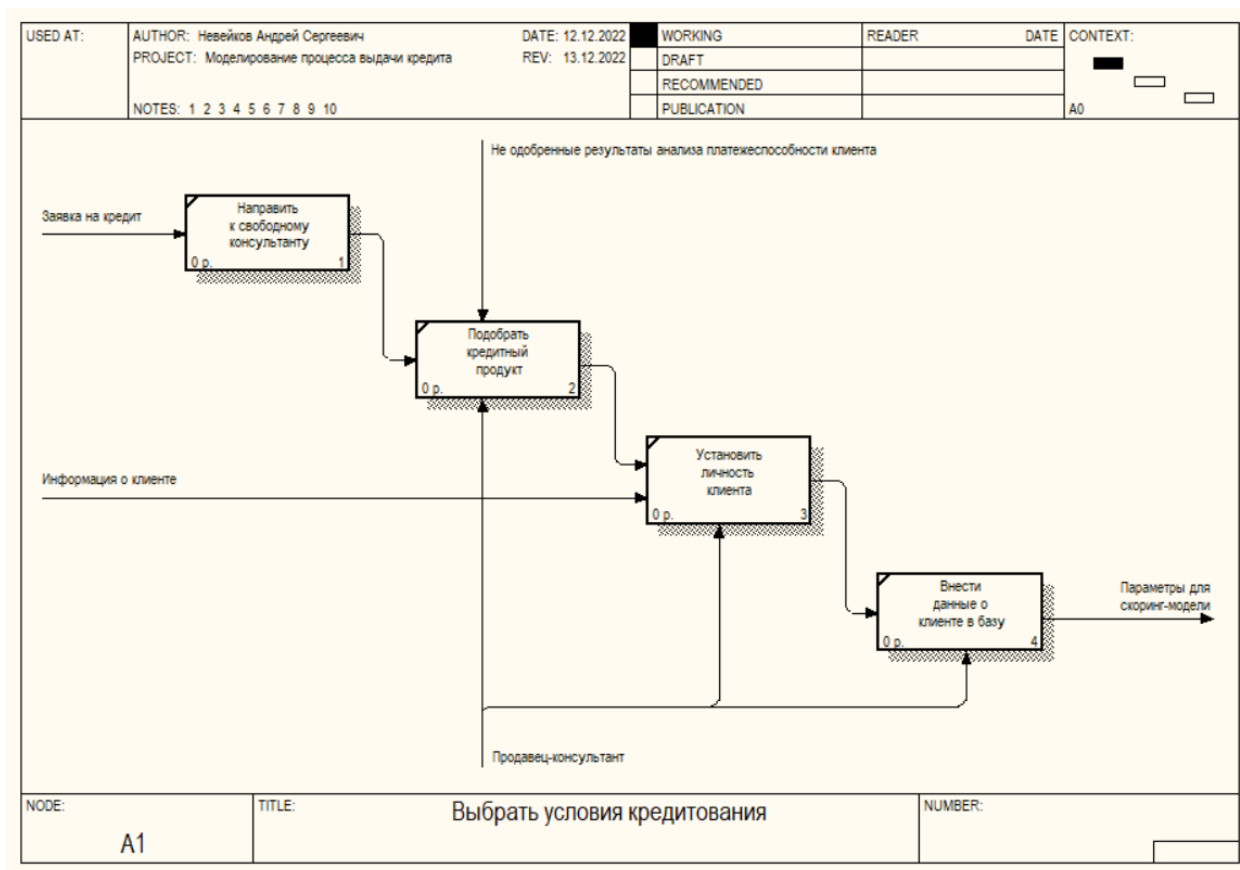


Рисунок 3.3 – Декомпозиция блока «Выбрать условия кредитования»

Декомпозиция блока «Анализировать платежеспособность клиента» представлена четырьмя блоками:

1. Проверить финансовые ресурсы банка.
2. Найти возможные варианты кредитования.
3. Найти оптимальный вариант кредитования.
4. Сформировать рекомендации согласно анализу.

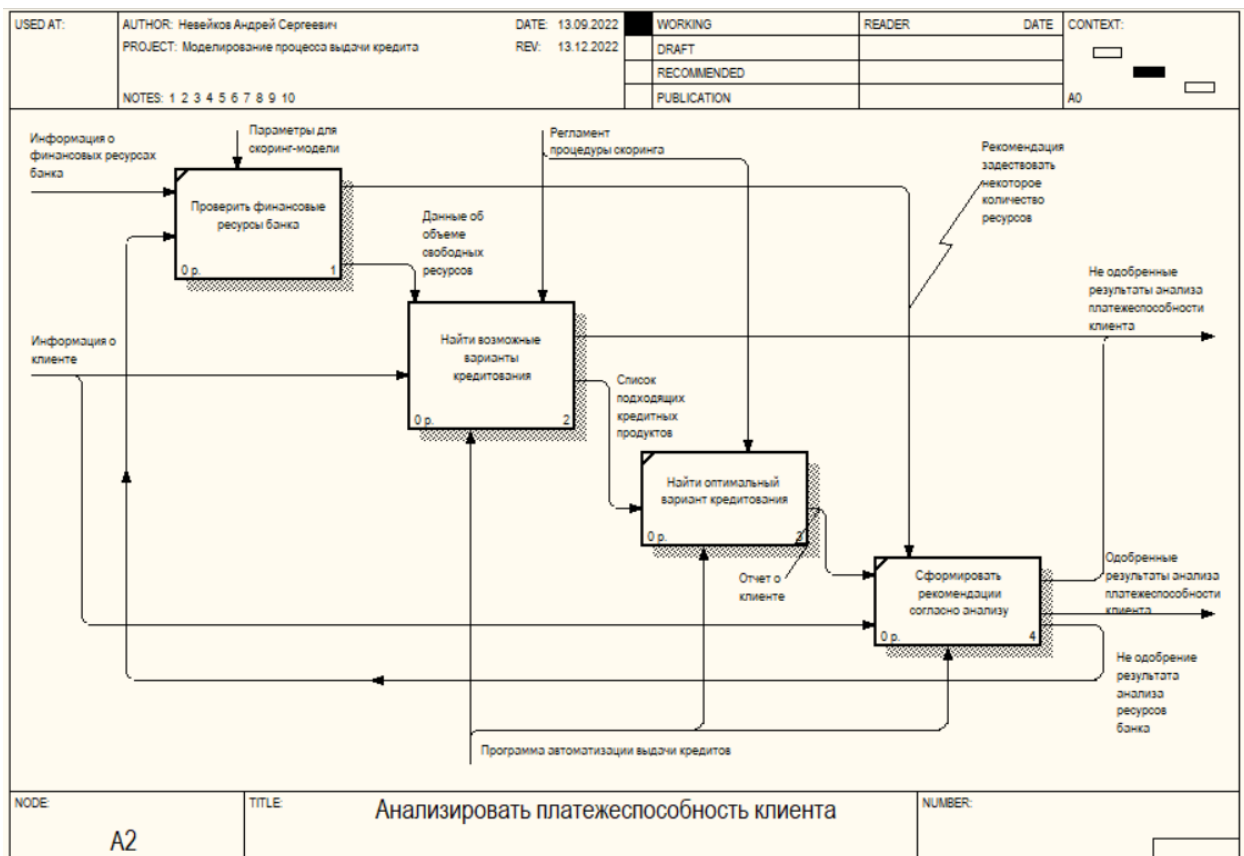


Рисунок 3.4 – Декомпозиция блока «Анализировать платежеспособность клиента»

Блок «Оформить кредит». Декомпозиция данного этапа представляет собой:

1. Одобрить выбранный системой вариант кредита.
2. Внести в договор согласованных параметров кредита.
3. Подписать кредитный договор.
4. Перевести деньги на счет клиента.

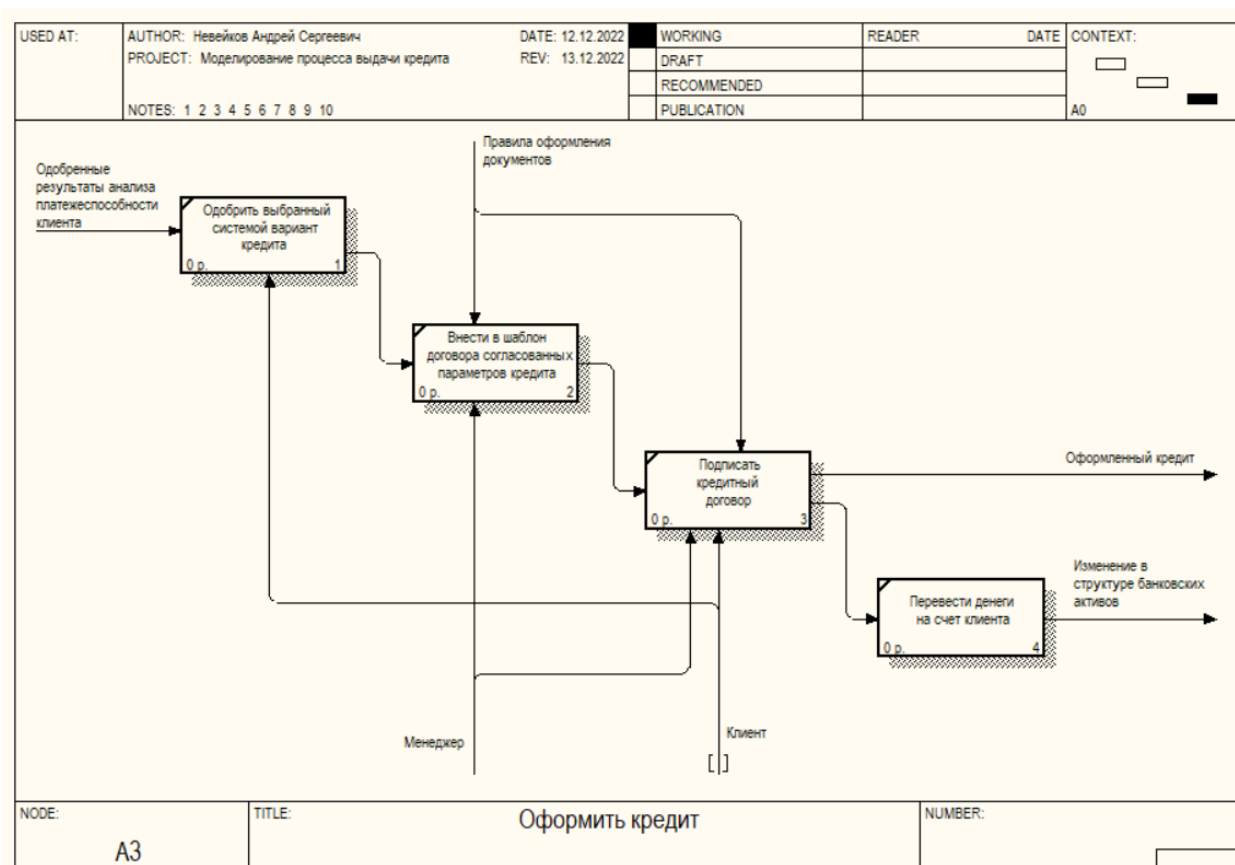


Рисунок 3.5 – Декомпозиция блока «Оформить кредит»

Таким образом, в результате описания функциональной модели данного курсового проекта, было отмечено, что для организации процесса выдачи кредита физическому лицу, необходимо учитывать потребности и платежеспособность клиента, чтобы не нести финансовых потерь от невозвращенных кредитов, создавать репутацию клиентоориентированного банка, а, следовательно, привлекать больше клиентов.

С помощью функционального моделирования были показаны все этапы процесса.

4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ

Структура базы данных создана по упрощенной методологии Инмона, с одной таблицей фактов `fct_clients_financial_data` и представленным в базе данных Star-уровнем. Функцию Data Sources выполняют данные, которые вносит пользователь непосредственно в программу, Stage Area и Cleansing Layer совмещены и находятся на сервере: сначала данные, пришедшие от пользователя сохраняются в объект класса, затем выполняются проверки на их корректность, после этого данные попадают в Star Area в базе данных. На рисунке 4.1 представлена схема базы данных, созданной по методологии Инмона[5].

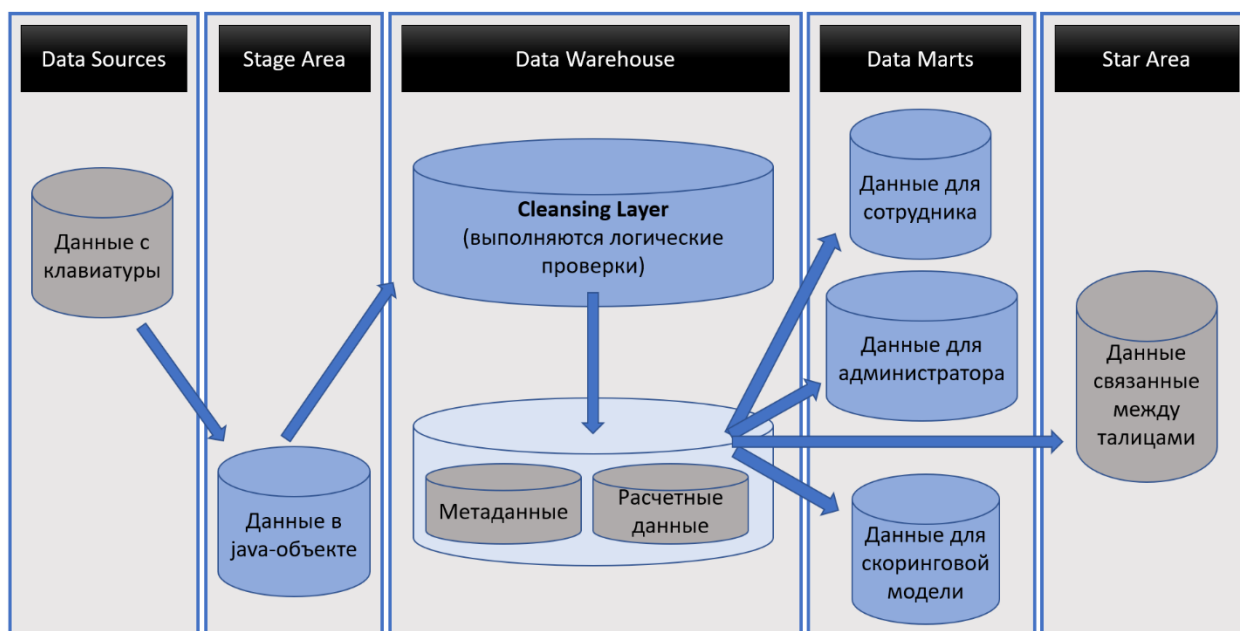


Рисунок 4.1 – Схема базы данных по методологии Инмона

Процесс взаимодействия с базой данных описан ниже. При обращении клиента в банк, сотрудник авторизуется в системе и вносит данные о клиенте, либо получает данные о существующем клиенте, если он ранее был зарегистрирован.

Клиент сообщает сотруднику параметры кредита (сумма, срок погашения, первоначальный взнос и т.д.).

На основе данных о пользователе и списке предоставляемых финансовых продуктов из базы данных скоринговая модель определяет, на каких условиях может быть выдан кредит или почему он не может быть выдан.

При входе в систему каждый сотрудник должен вводить аутентификационные данные, просматривать их может только администратор.

Разделение данных о клиенте на личную (`client_personal_data`) и финансовую (`fct_clients_financial_data`) информацию сделано с целью безопасности, чтобы предоставить сотрудникам доступ только к анонимной финансовой информации и избежать утечки паспортных данных клиентов.

В таблице `fct_clients_financial_data` вся информация о наличии непогашенных кредитов, ежемесячных платежах, количестве просроченных платежей, сотрудник, который выдал кредит.

Таблица `loan_product` хранит список финансовых продуктов, предоставляемых банком. Для каждого финансового продукта указываются необходимые параметры для предоставления кредита, например: высокий кредитный рейтинг, конкретная цель кредитования и так далее.

Таблица `bank_financial_flows` создана для демонстрации руководству финансового состояния банка. Также ее использует скоринговая модель, чтобы предотвратить выдачу высокорискованных кредитов, когда финансовое состояние плохое, избежать кассового разрыва, когда выданный сейчас кредит не позволит банку выполнять свои финансовые обязательства, ограничить выдачу необеспеченных кредитов.

Физическая модель базы данных банка представлена на рисунке 4.2.

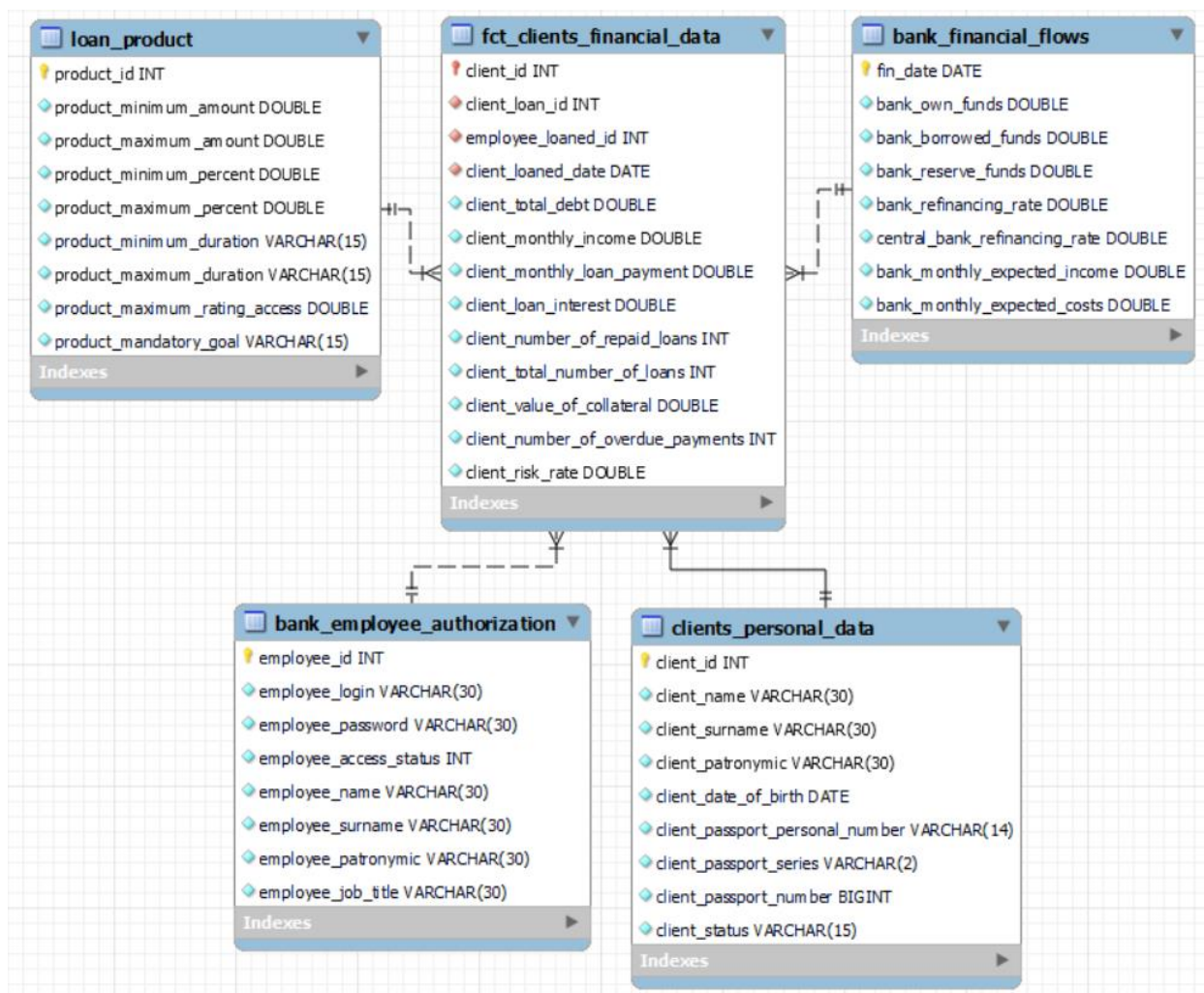


Рисунок 4.2 – Информационная модель банка

Таким образом, база данных использует эффективную с точки зрения безопасности, скорости доступа и логического разделения модель. Поскольку у каждой таблицы имеется всего один первичный ключ, а каждое не ключевое поле не транзитивно зависит от первичного ключа: при изменении любого столбца в таблице, не нужно менять связанный столбец в другой таблице.

5 ОБОСНОВАНИЕ ОРИГИНАЛЬНЫХ РЕШЕНИЙ ПО ИСПОЛЬЗОВАНИЮ ТЕХНИЧЕСКИХ И ПРОГРАММНЫХ СРЕДСТВ, НЕ ВКЛЮЧЕННЫХ В ТРЕБОВАНИЯ

UML – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур[6].

Язык UML предназначен для визуализации, спецификации, конструирования и документирования артефактов программных систем. UML подходит для содержательного описания классов, объектов и компонентов в сфере разработки программных продуктов.

Диаграмма в языке моделирования UML — наглядное представление некоей совокупности элементов модели системы в виде графа, на котором дуги связывают вершины.

5.1 Диаграмма вариантов использования

Диаграмма вариантов использования – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования. Каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

Администратору доступна работа со списком пользователей приложения (добавление, удаление, редактирование записей), так как с приложением могут работать только сотрудники банка, бизнес-логикой предусмотрена роль отвечающая за то, что все пользователи системы являются сотрудниками банка, однако администратору не доступна информация о клиентах банка и финансовых активах так как работа с финансовыми данными не входит в его служебные обязанности.

Подавцу-консультанту доступна вся информация о клиента банка: наличие кредитов, ежемесячных доход и даже паспортные данные. Также продавец-консультант может просматривать список кредитных продуктов

банка. Уникальная возможность продавца-консультанта – запустить процедуру скоринга для клиента.

Менеджер получает рекомендацию от скорринговой модели в отношении клиента банка, также он может просматривать состояние финансовых активов банка. Уникальной возможностью менеджера является возможность работы со списком кредитных продуктов банка (добавление, удаление редактирование)

Диаграмма вариантов использования представлена на рисунке 5.1.1.

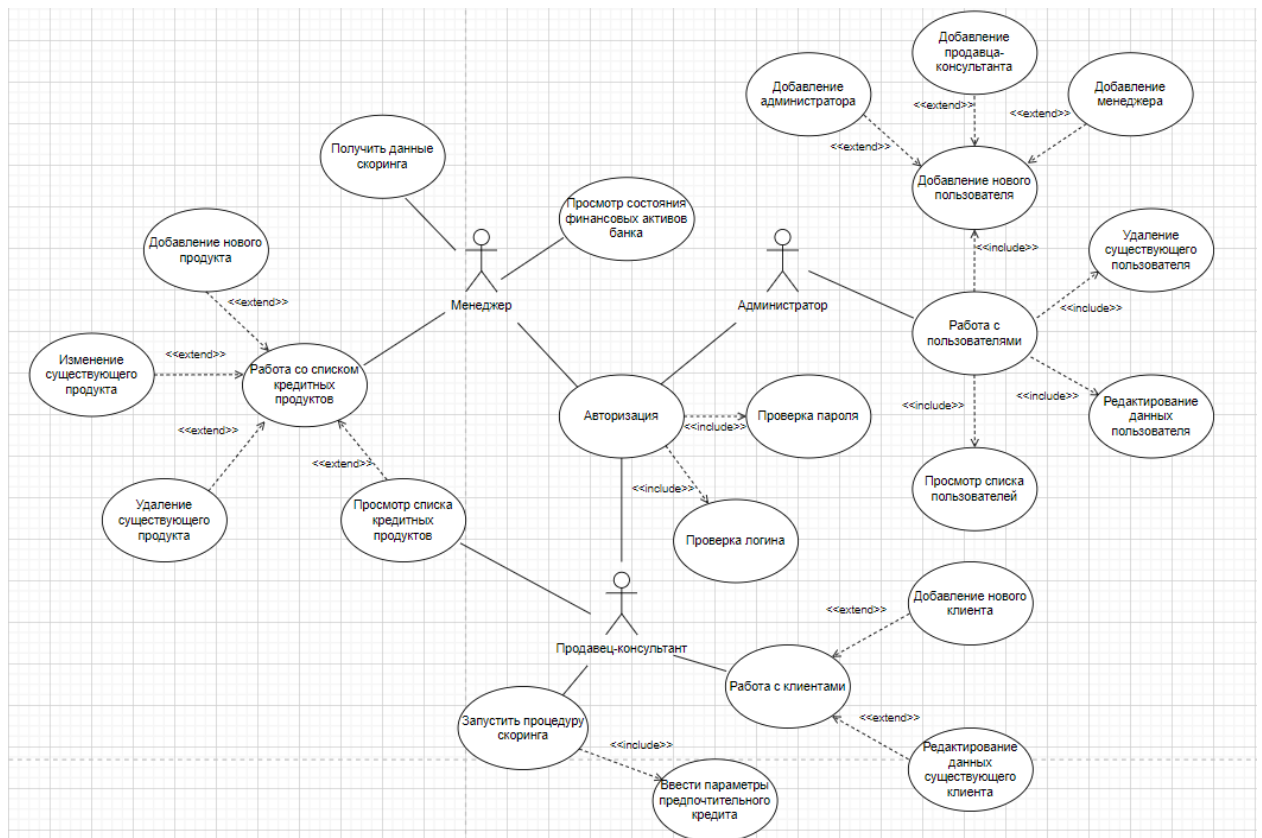


Рисунок 5.1.1 – Диаграмма вариантов использования

Общим для всех трех актеров является возможность авторизации. В зависимости от прав доступа им будет представлен разный функционал приложения.

5.2 Диаграмма развертывания и компонентов системы

Диаграммы компонентов используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы[7].

Компонентами могут быть аппаратные компоненты, такие как микросхема, модуль или устройство; или бизнес-подразделение, такое как продавцы-консультанты, кредитный скоринг или квартальный отчет; или программные компоненты, такие как база данных, сервер или пользовательский интерфейс.

Диаграмма компонентов представлена на рисунке 5.2.1.

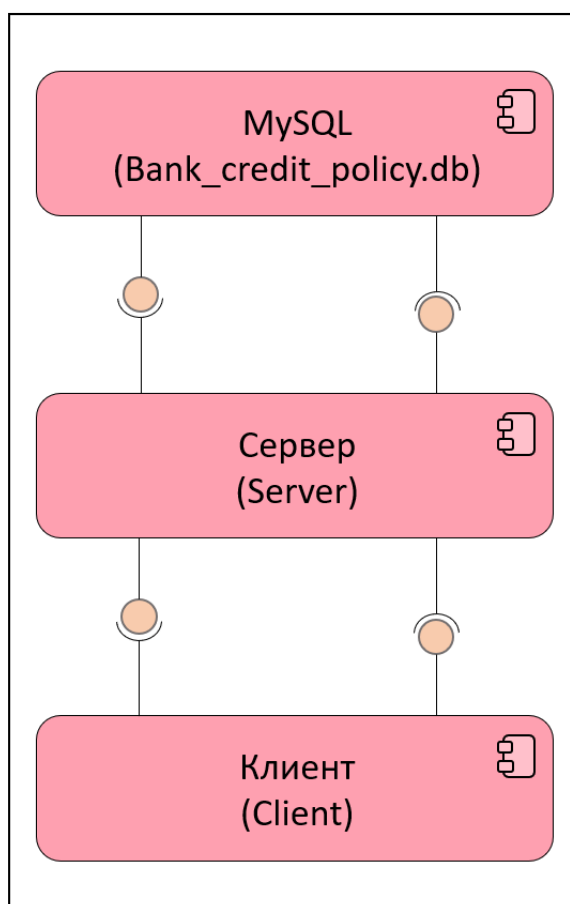


Рисунок 5.2.1 – Диаграмма компонентов системы

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их. Например: ПК пользователя и сервер приложения связаны через интернет протокол TCP/IP, а связь между базой данных и сервером происходит через JDBC Driver. Это единственная диаграмма, на которой применяются “трехмерные” обозначения: узлы системы обозначаются кубиками. Диаграмма развертывания представлена на рисунке 5.3.

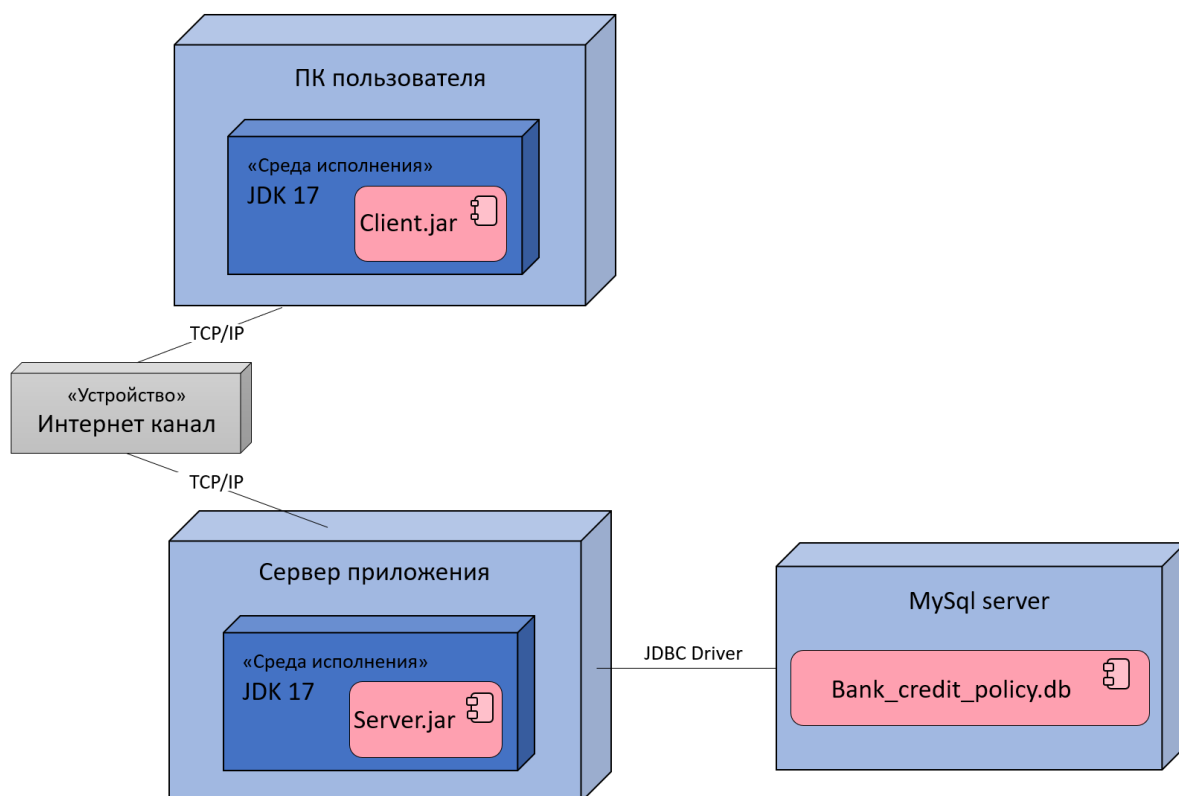


Рисунок 5.2.1 – Диаграмма развертывания приложения

В качестве узлов выступают база данных, сервер приложения и ПК пользователя.

Для запуска разрабатываемого приложения необходимо наличие исполняемой среды JDK17 на компьютерах сотрудников банка.

5.3 Диаграмма классов

Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов, методов, интерфейсов и взаимосвязей между ними.

Как результат смешивания приёмов композиции и декомпозиции, в программе получилось четыре основных класса, три из них отвечают за реализацию функционала продавца-консультанта, менеджера и администратора, последний отвечает за взаимодействие с базой данных. Для хранения данных SQL-запросов созданы классы с переопределённым методом toString() так, чтобы использовать один метод передачи сообщения с сервера на клиент. На рисунке 5.3 представлена диаграмма классов программы.

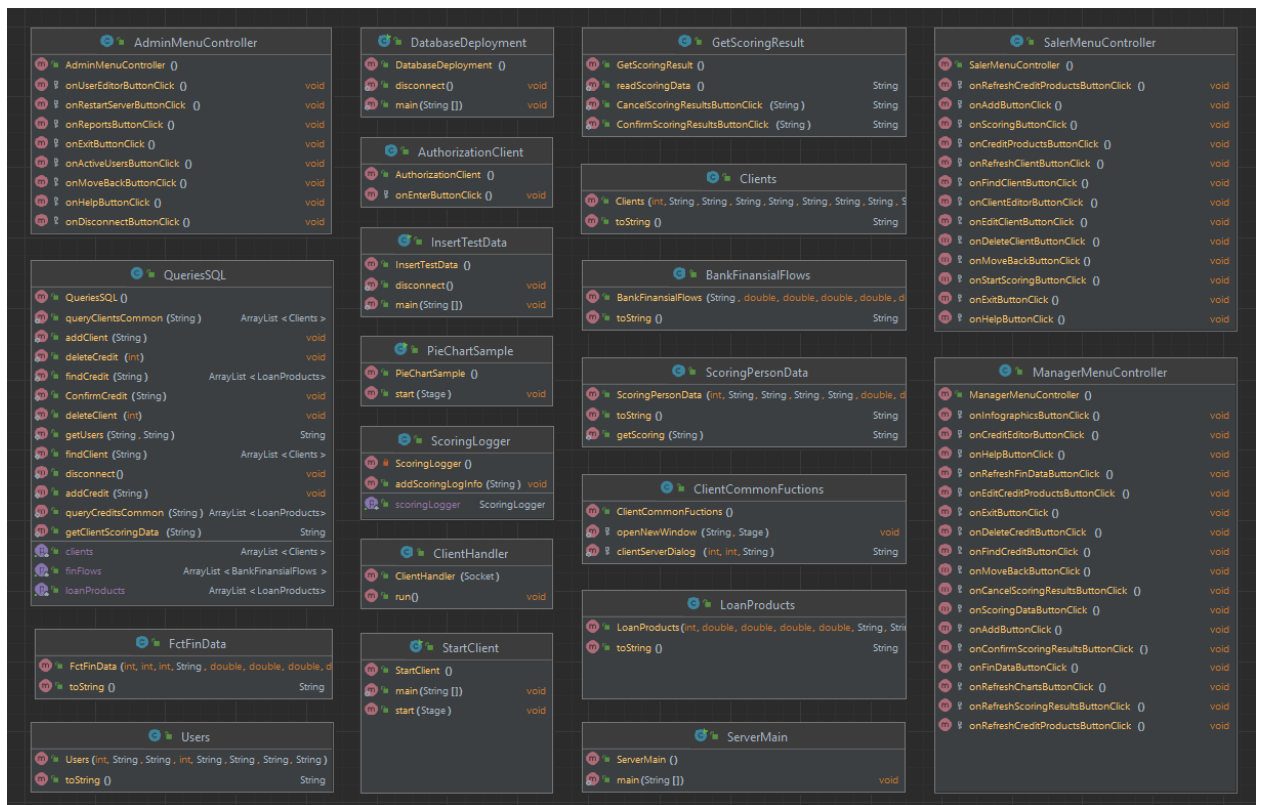


Рисунок 5.3.1 – Диаграмма классов

В проекте реализован паттерн проектирования singleton. При помощи данного паттерна функционирует сборщик логов, чтобы реализовать singleton необходимо определить статический создающий метод, который возвращает один и тот же объект. Реализация этого паттерна представлена на рисунке 5.4.

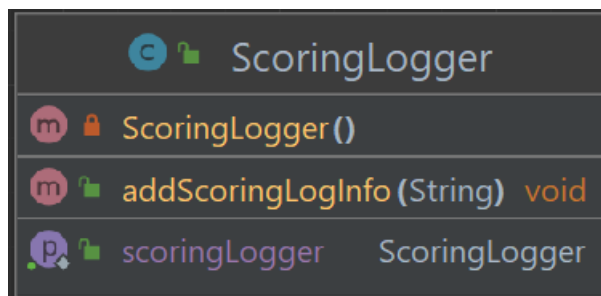


Рисунок 5.3.2 – Реализация паттерна Singleton

Singleton относится к порождающим паттернам. Этот паттерн гарантирует, что у класса есть только один экземпляр класса и к этому объекту предоставляется глобальная точка доступа. Отлично подходит для ведения log-файлов.

Также в курсовом проекте реализован архитектурный подход MVC (Model, View, Controller). Пользователь получает информацию через графический интерфейс, вносит изменения в контроллере, который передает необходимые параметры модели, а модель обновляет графический интерфейс (рисунок. 5.5).

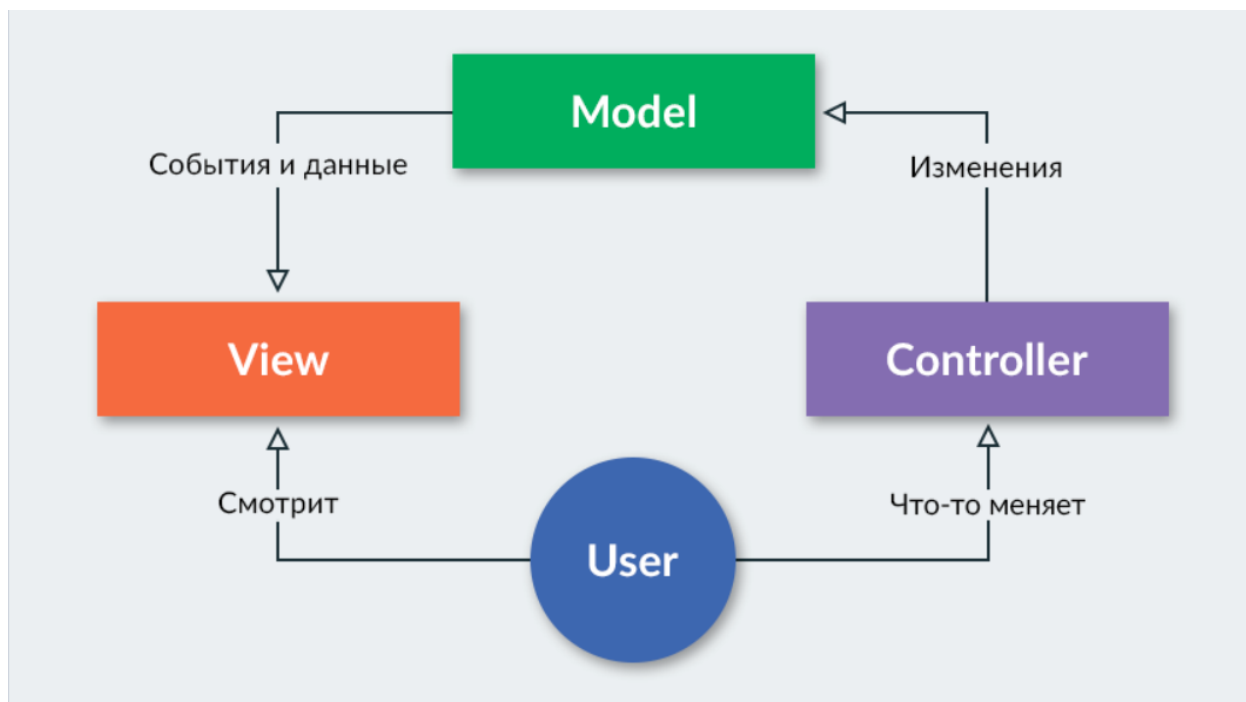


Рисунок 5.3.3 – Реализация принципа MVC

На данный момент MVC крайне популярен при разработке клиент-серверных приложений.

5.4 Диаграмма последовательности

Диаграмма последовательности — UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта и взаимодействие актеров информационной системы в рамках прецедента.

На рисунке 5.4.1 представлена диаграмма последовательности процесса авторизации.

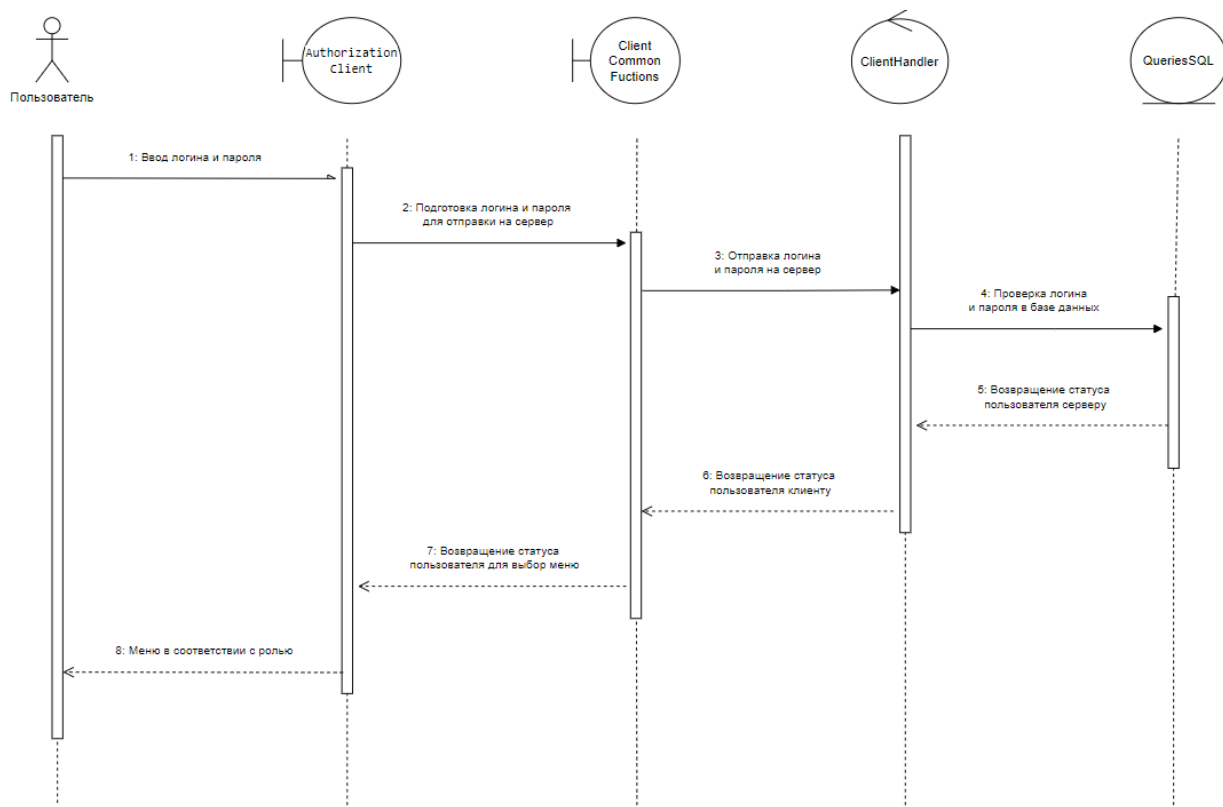


Рисунок 5.4.1 – Последовательности процесса авторизации

На представленной диаграмме пользователь вводит данные для авторизации в окно графического приложения, которые считывает контроллер окна и передает в адаптер, который отправляет передат информацию о запросе по протоколу TCP на сервер. Сервер подключается к базе данных и отправляет нужный запрос.

В БД происходит поиск соответствия полученных данных с имеющимися.

В случае успеха или ошибки сервер возвращает клиенту полученный от базы данных статус, от которого зависит открывшееся меню. 0 – не авторизованный пользователь, 1 – администратор, 2 – продавец-консультант, 3 – менеджер.

5.5 Диаграмма состояний

Диаграмма последовательности — UML-диаграмма, в виде ориентированного графа для конечного автомата, в котором вершины обозначают состояния дуги показывают переходы между двумя состояниями (рисунок 5.5.1).

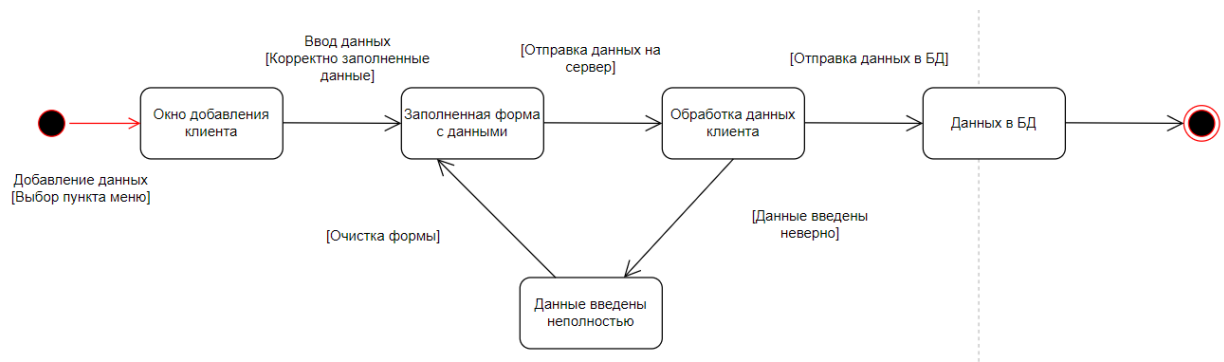


Рисунок 5.5.1 – Диаграмма состояний при добавлении нового клиента

Начальным состоянием данного процесса является выбор пользователем в меню необходимого пункта для начала создания объекта класса «client_personal_data». Далее необходимо заполнить форму добавления, то есть добавить в поля информацию из паспорта клиента: имя, фамилия, дата рождения и т.д.

После чего поля начинают обрабатываться, выполняя проверку наличия введенных значений.

В случае корректного заполнения формы поля объекта записываются в базу данных и объект прекращает существование в системе, в противном случае форма с некорректными данными очищается.

6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ

Схема алгоритма — графическое представление определения, анализа или метода решения задач, в котором используются символы для отображения данных и операций. Схемы алгоритмов и программ состоят из имеющих заданное значение символов, краткого пояснительного текста и соединяющих линий[8].

6.1 Алгоритм авторизации в программе

Так как программа разработана для работников банка, первое с чем они будут сталкиваться – это авторизация. Чтобы избежать лишней нагрузки на сервер во время работы за счет неактивных пользователей, подключение устанавливается только после ввода логина и пароля. Когда пользователь нажимает кнопку «Войти», устанавливается соединение с сервером, сервер направляет запрос в базу данных и, получив ответ, либо запускает меню согласно уровню доступа, либо стирает неверно вводимые данные пользователя. Ограничений на количество неверных вводов пароля не предусмотрено.

Когда пользователь запускает окно авторизации, изначально его уровень доступа 0, новый уровень присвоится вместе с подтверждением логина и пароля от базы данных.

Алгоритм авторизации в программе представлен на схеме 6.1.1.

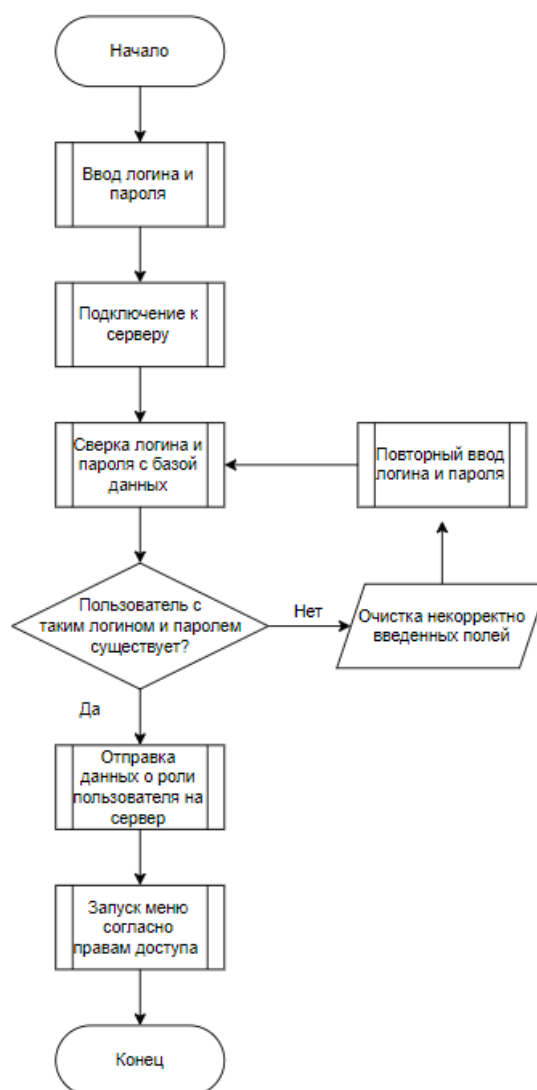


Рисунок 6.1 – Алгоритм авторизации в программе

Программой предусмотрено три уровня доступа: администратор, менеджер и продавец-консультант.

6.2 Алгоритм работы программы

После входа в программу, открытия меню с уникальным функционалом для каждого уровня доступа пользователя.

Нельзя сказать, что у какого-то из пользователей функционал более широкий, так как разделение происходит по должностным обязанностям и с соблюдением принципа банковской тайны.

Продавец-консультант может редактировать список клиентов, просматривать предлагаемые кредитные продукты и запустить процедуру скоринга для клиента.

Менеджер отвечает за редактирование списка предлагаемых кредитных продуктов, имеет возможность просматривать кредитные ресурсы банка, для принятия решений об подтверждении или отмене рекомендации скоринговой модели.

Администратор в основном отвечает за добавление сотрудников, по мере трудоустройства и может взаимодействовать с сервером.

Алгоритм работы программы представлен на схеме 6.2.1 и 6.2.2.

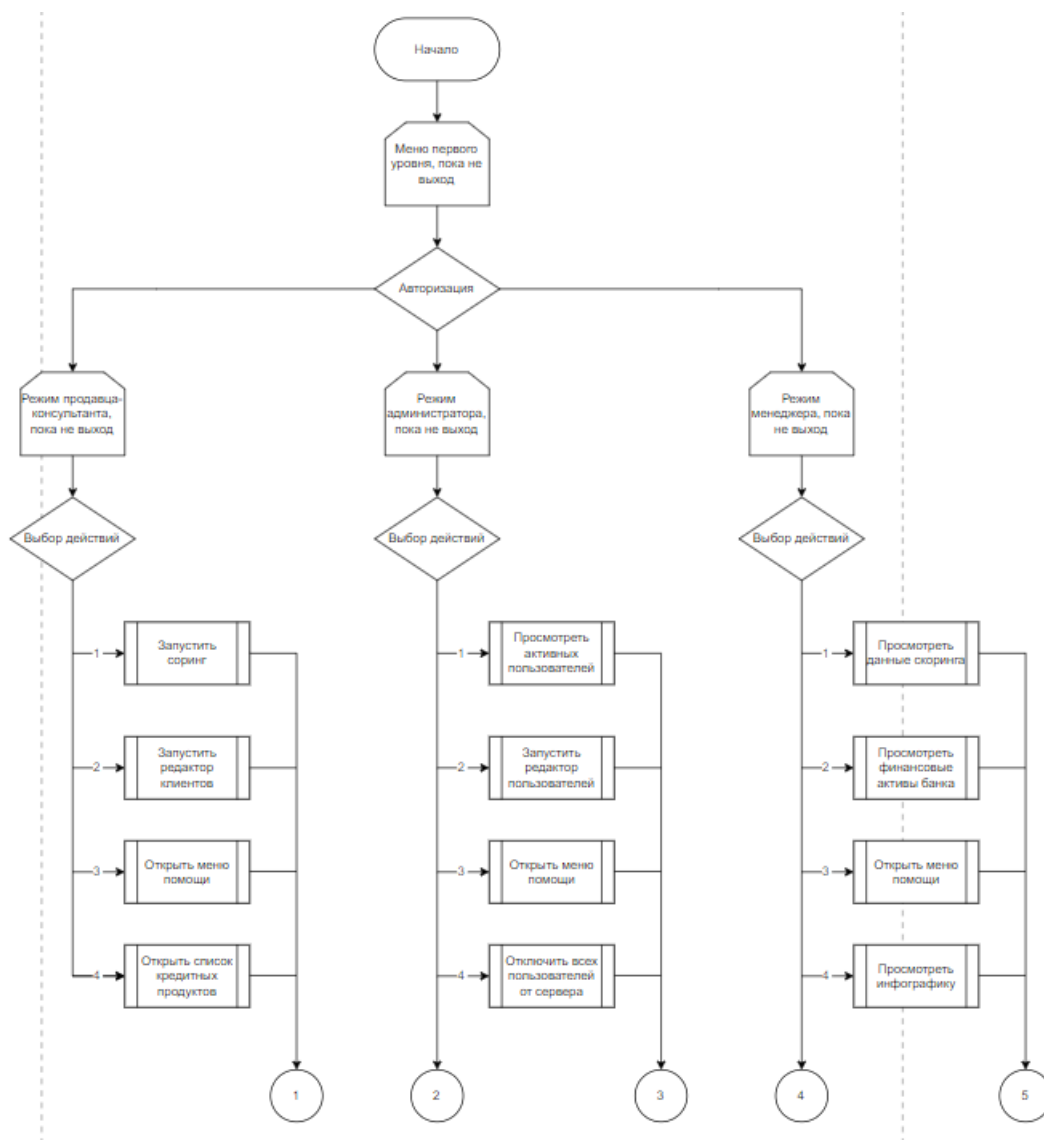


Рисунок 6.2.1 – Алгоритм работы программы

Своими решениями менеджер косвенно влияет на изменение структуры финансовых ресурсов банка. Он может их редактировать вручную, но подтверждаемые кредиты влияют на структуру активов банка.

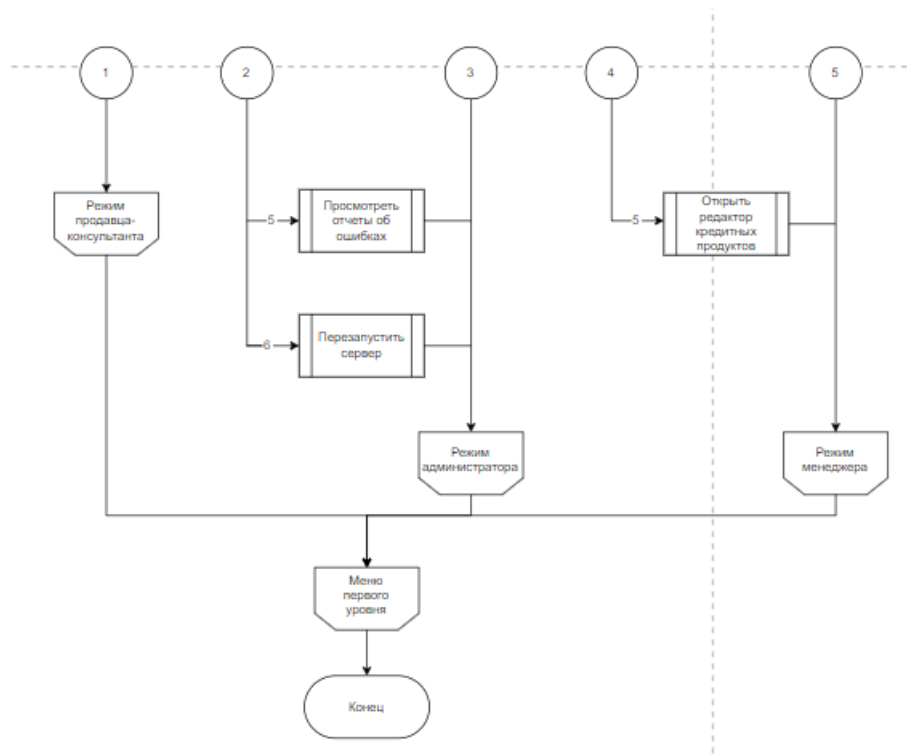


Рисунок 6.2.2 – Алгоритм работы программы (продолжение)

К некоторым таблицам в базе данных доступ есть только у скоринговой модели. Это сделано в целях автоматизации и безопасности.

6.3 Алгоритм создания нового кредитного продукта

В целях маркетинга или при избытке свободных денежных средств есть необходимость в скорейшем привлечении клиентов. Для этого добавляется новые кредитные продукты, которые отличаются процентной ставкой, максимально и минимальной суммой, сроками погашения.

После получения распоряжения руководства, менеджер должен авторизироваться. Открыв «Редактор кредитных продуктов», менеджер смотрит есть ли такие продукты, которые закрывают востребованную нишу. Если нет, после нажатия кнопки «Изменить», вносит актуальные параметры кредитного продукта и нажимает кнопку «Добавить». Данные через сервер будут переданы в базу данных. В случае успешного добавления, новый кредитный продукт отобразится в обновленном списке.

Алгоритм создания нового кредитного продукта представлен на схеме 6.3.1.

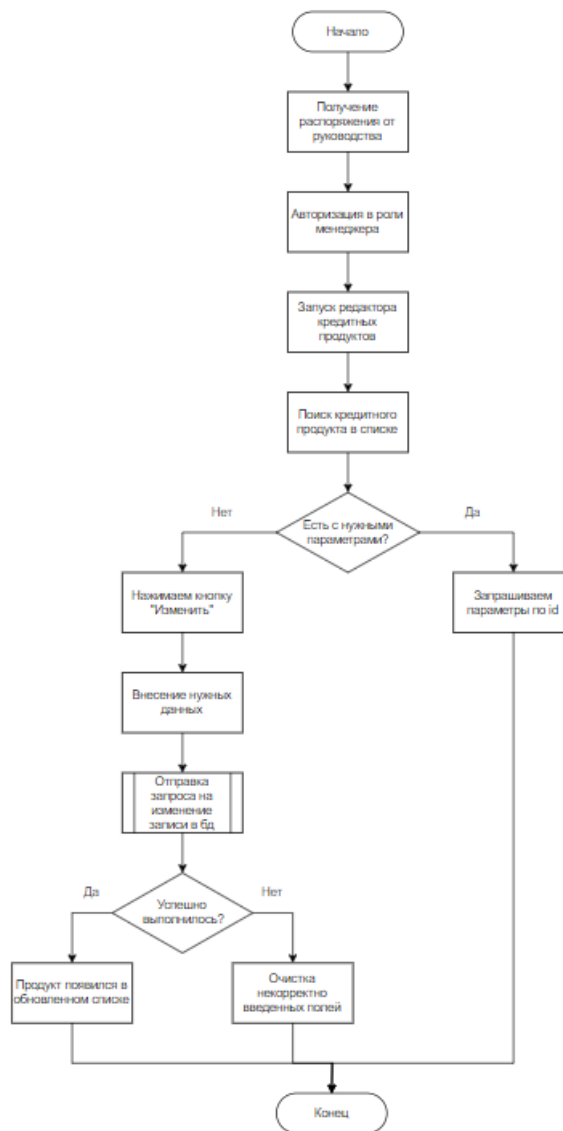


Рисунок 6.3.1 – Алгоритм создания нового кредитного продукта

Подобным образом работают все алгоритмы, добавляющие ручную данные в базу. Отличия заключаются в параметрах самого объекта и пользователе, который его создает.

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для исправной работы системы необходимы три компонента: клиент, сервер и база данных. Перед началом работы убедитесь, что у вас установлена MySQL Workbench. Если вы впервые запускаете приложение и на вашем компьютере нет базы данных, вы можете сгенерировать ее, выполнив метод `main`, класса `DatabaseDeployment`. После выполнения будет сгенерировано 5 таблиц, связанные внешними ключами. Либо выполнить SQL-скрипт из папки `SQL`.

Если вы хотите проверить работоспособность приложения на тестовых данных, необходимо выполнить метод `main`, класса `InsertTestData`.

Для запуска серверной части необходимо установить JDK 17. Для успешной работы рекомендуется сначала запускать серверную часть, а затем клиентскую.

7.1 Запуск приложения

После запуска серверной части, можно запустить приложение для работы клиента. В самом начале вас встретит окно авторизации (рисунок 7.1.1).

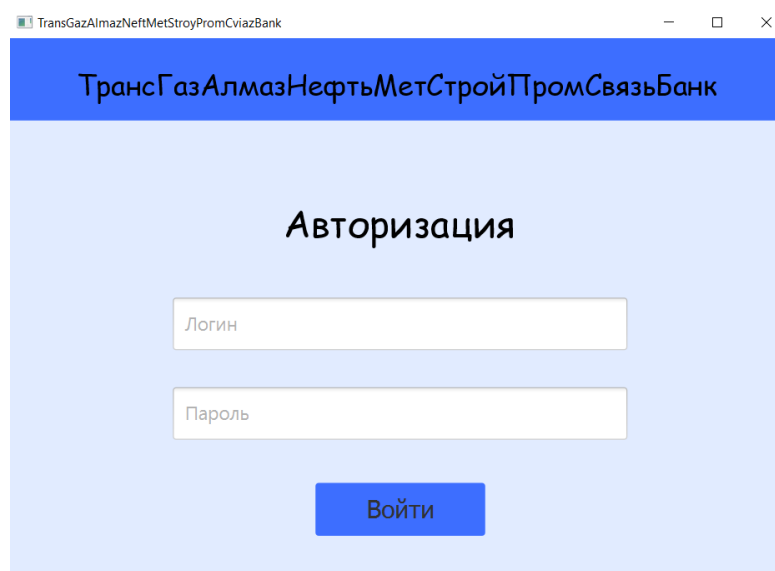


Рисунок 7.1.1 – Окно авторизации

Предусмотрены 3 роли: менеджер, продавец-консультант и администратор, которые различаются предоставляемым функционалом. Чтобы авторизоваться необходимо ввести логин и пароль и нажать кнопку войти,

далее приложение само определит, кто входит в систему и откроет соответствующее меню. На рисунке 7.1.2 показано меню администратора.

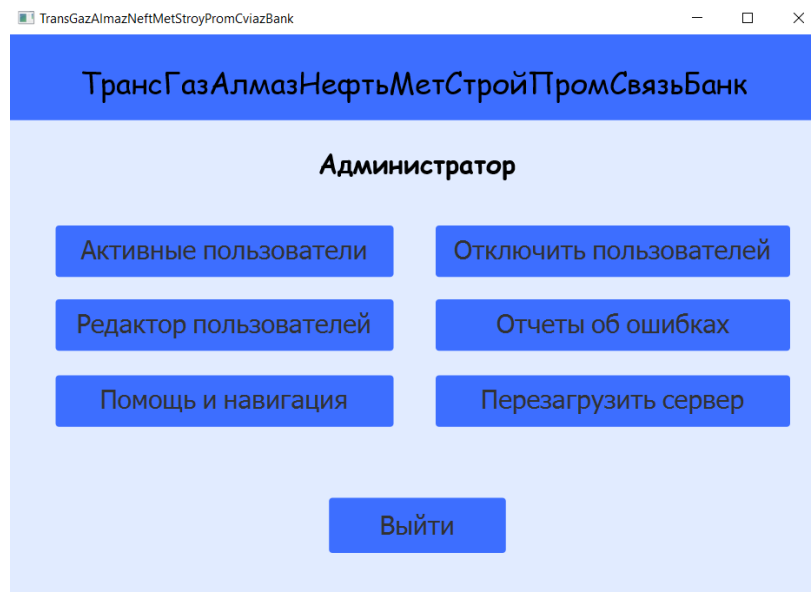


Рисунок 7.1.2 – Меню администратора

Для удобства пользователя в каждом меню предусмотрена кнопка «Помощь и навигация», в этом разделе описаны функциональные возможности меню (рисунок 7.1.3).

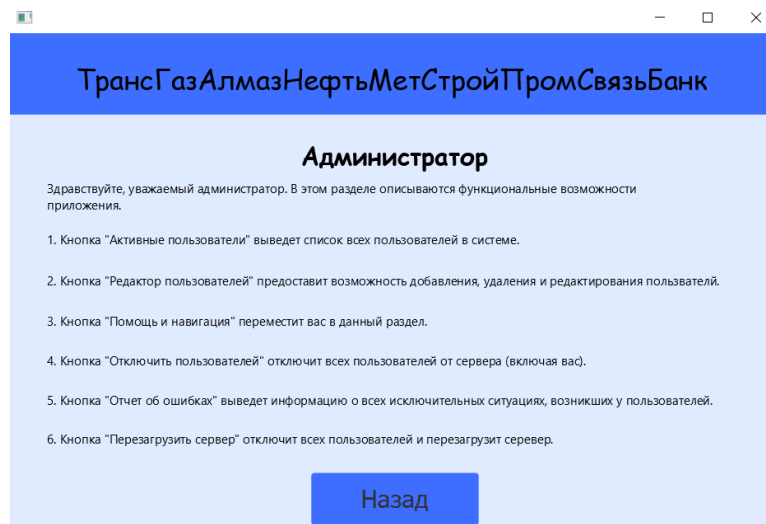


Рисунок 7.1.2 – Раздел «Помощь и навигация» в меню администратора

Из раздела «Помощь и навигация» можно вернуться в основное меню при помощи кнопки «Назад».

7.2 Функции продавца-консультанта

При авторизации в качестве продавца-консультанта, у вас появится меню, в котором доступны пять функций: запуск процедуры скоринга, запуски редактора клиентов, страница с помощью и страница просмотра кредитных продуктов (рисунок 7.2.1).

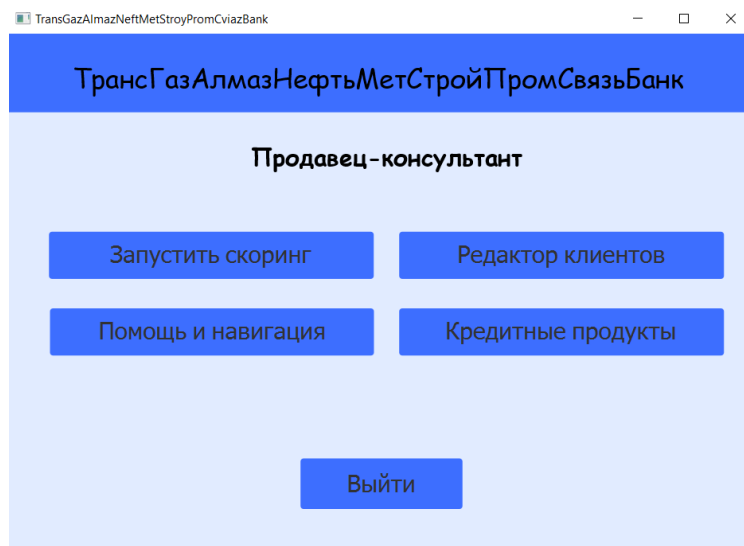


Рисунок 7.2.1 – Меню продавца-консультанта

Уникальная способность продавца-консультанта – это работа с клиентами банка. В первую очередь к ней относится просмотр списка клиентов банка (рисунок 7.2.2).

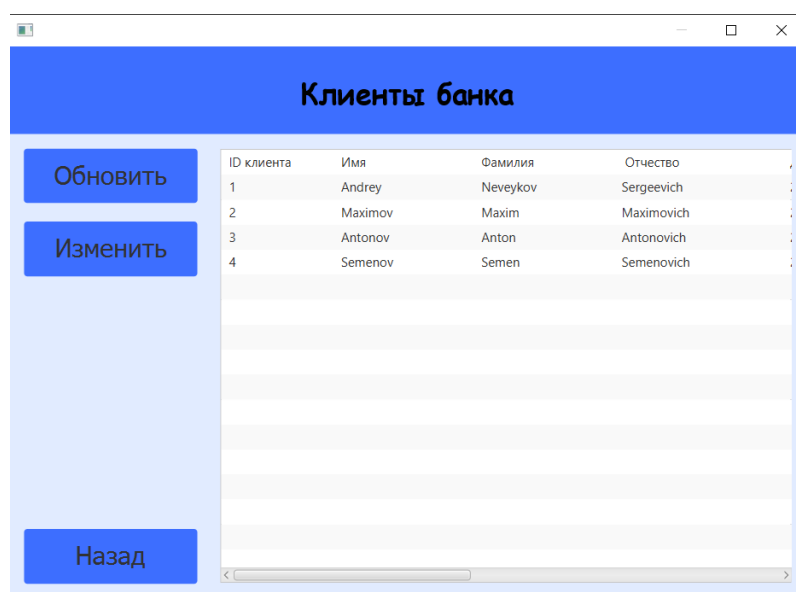


Рисунок 7.2.2 – Окно списка клиентов банка

Продавец-консультант может искать нужного клиента с помощью его персональных данных. Когда клиент обращается за кредитом, он передает свой паспорт, для подтверждения личности. По уникальному ID паспорта можно определить, находится ли он уже в базе (рисунок 7.2.3).

Клиенты банка

ID 2

Имя Maximov ID паспорта PB59V78CG3CGP9

Фамилия Maxim Серия паспорта MP

Отчество Maximovich № паспорта 1616889

Дата рождения 2000-01-13 Статус new;

Назад Добавить Найти Удалить

Рисунок 7.2.3 – Поиск клиента банка по id паспорта

Продавец-консультант может добавить нового клиента в этом же окне, внося необходимую информацию и нажав кнопку «Добавить». Удаление происходит также по id паспорта, после нажатия кнопки «Удалить».

Чтобы знать, что можно предложить клиентам, продавец-консультант имеет доступ к списку кредитных продуктов через соответствующее окно (рисунок 7.2.4).

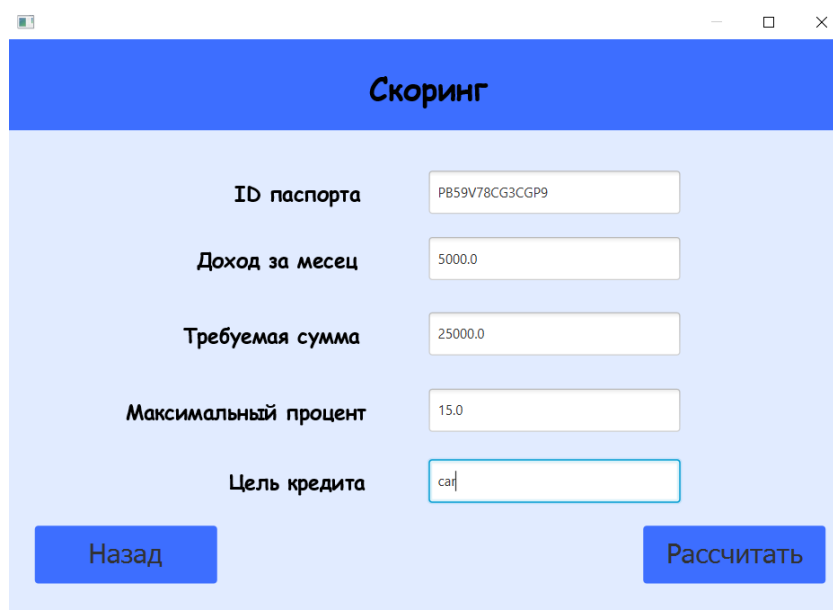
Кредитные продукты

Назад Обновить

ID продукта	мин. сумма	макс. сумма	мин. %	макс. %
1	500.0	10000.0	3.0	25.0
2	15000.0	500000.0	7.0	25.0
3	1500.0	20000.0	5.0	17.0
4	100.0	5000.0	3.5	12.7
5	50000.0	900000.0	6.0	16.0
6	3000.0	60000.0	7.0	12.0

Рисунок 7.2.3 – Поиск клиента банка по id паспорта

Главная функция продавца-консультанта – запускать процедуру скоринга для клиентов. Потребуется заполнить только пять полей, всю остальную информацию скоринг-модель возьмет сама из базы данных (рисунок 7.2.5).



Скоринг	
ID паспорта	PB59V78CG3CGP9
Доход за месяц	5000.0
Требуемая сумма	25000.0
Максимальный процент	15.0
Цель кредита	car
<div>Назад Рассчитать</div>	

Рисунок 7.2.5 – Окно ввода данных для скоринг-модели

После нажатия кнопки «Рассчитать», модель запустится и передаст результаты вычисления менеджерам, которые смогут одобрить кредит или отказать в его выдаче.

7.3 Функции менеджера

Основные цели менеджера – работа с кредитными продуктами и одобрение результатов скоринг-модели. Также присутствуют дополнительные функции в виде просмотра данных о финансовом состоянии банка. Меню менеджера представлено на рисунке 7.3.1.



Рисунок 7.3.1 – Меню менеджера

После нажатия кнопки «Редактор кредитных продуктов», откроется окно, в котором можно найти, добавить или удалить кредитный продукт.

Рисунок 7.3.2 – Окно редактор кредитных продуктов

Кнопки «Финансовые активы», позволит увидеть информацию о текущем состоянии основных экономических показателей (рисунок 7.3.3).

Финансовое состояние банка

Собственные средства

Заемные средства Ожидаемые поступления

Резервные средства Ожидаемые траты

Ставка ЦБ Расчетная ставка банка

Рисунок 7.3.3 – Окно просмотра финансовых показателей

Как и у всех остальных ролей, менеджеру предусмотрена вкладка помощи с описанием всех функций (рисунок 7.3.4).

ТрансГазАлмазНефтьМетСтройТромСвязьБанк

Менеджер

Здравствуйте, уважаемый менеджер. В этом разделе описываются функциональные возможности приложения.

1. Кнопка "Данные скоринга" откроет логи всех производимых скоринг операций.
2. Кнопка "Финансовые активы" позволяет посмотреть финансовое состояние банка банка.
3. Кнопка "Помощь и навигация" переместит вас в данный раздел.
4. Кнопка "Инфоргафика" откроет графики с основными бизнес-метриками банка.
5. Кнопка "Редактор кредитных продуктов" откроет меню, позволяющее добавлять, удалять и искать кредитные продукты.

Рисунок 7.3.4 – Окно помощи и навигации

Основная функция менеджера просмотр результатов скоринга и одобрение кредитов (рисунке 7.3.5).

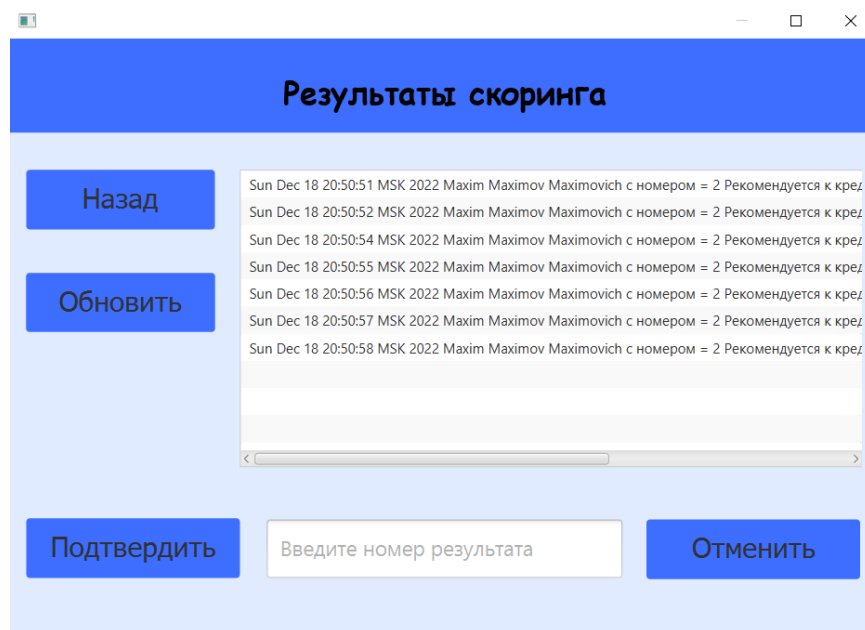


Рисунок 7.3.5 – Окно просмотра логов скоринга

Для подтверждения результата необходимо ввести номер строки и нажать «Подтвердить» либо «Отменить»

На следующем рисунке показано окно с графиком состояния активов, его можно увидеть в разделе «Инфографика» (рисунок 7.3.6).

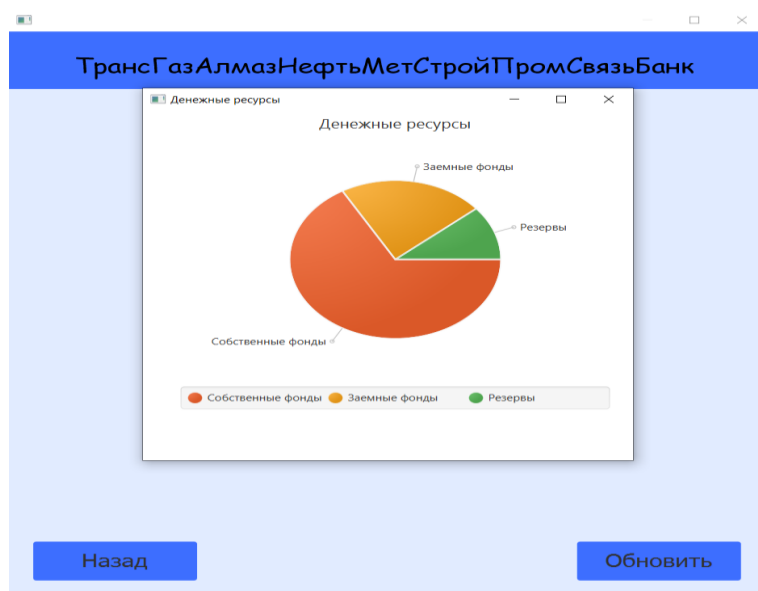


Рисунок 7.3.5 – Окно просмотра логов скоринга

Параметры для графика клиент запрашивает автоматически в базе данных через сервер.

7.4 Функции администратора

Основная функция администратора – работать с пользователями, но также есть несколько дополнительных функций (рисунок 7.4.1).

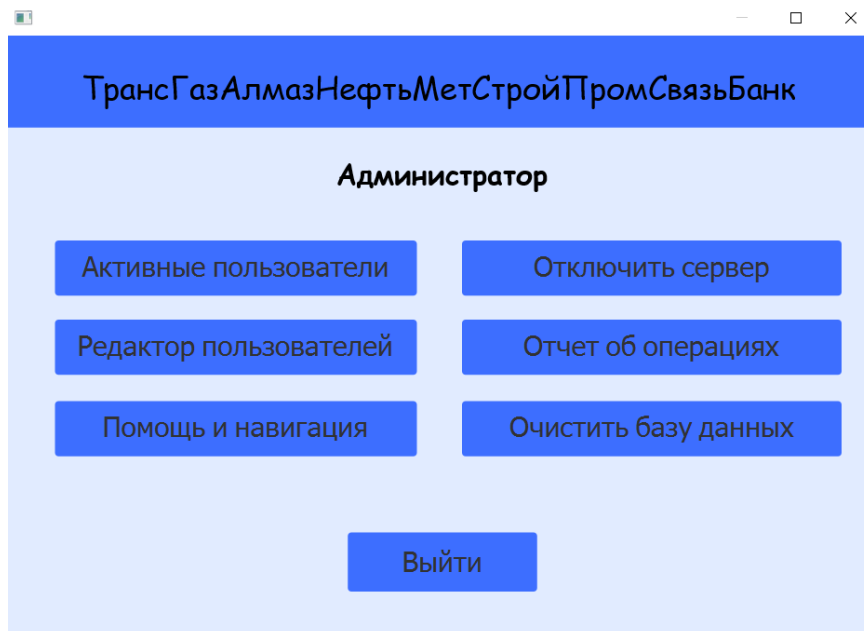


Рисунок 7.4.1 – Меню администратора

Про каждую из функций подробно написано в разделе «Помощь и навигация» (рисунок 7.4.2).

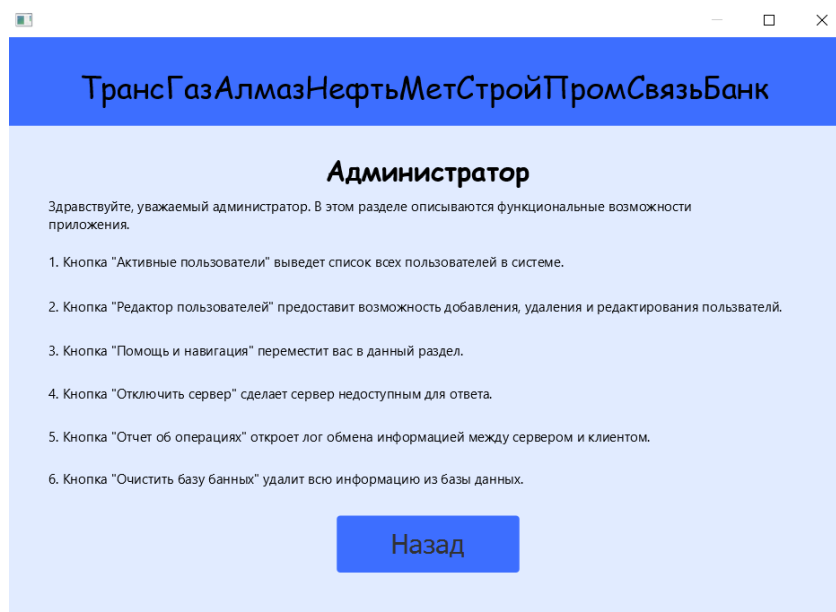


Рисунок 7.4.1 – Меню администратора

8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

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

В качестве способа тестирования было выбрано ручное тестирование. Ручное тестирование [9] — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно производится тестировщиком без использования программных средств, для проверки программы или сайта путём моделирования действий пользователя.

Мной были проверены функции авторизации, добавления, поиска и удаления.

В качестве параметров для авторизации я передавал несуществующие значения и нажимал «Войти». Программа, получив ответ от сервера, что такого пользователя нет, очистила некорректные данные (рисунок 8.1).

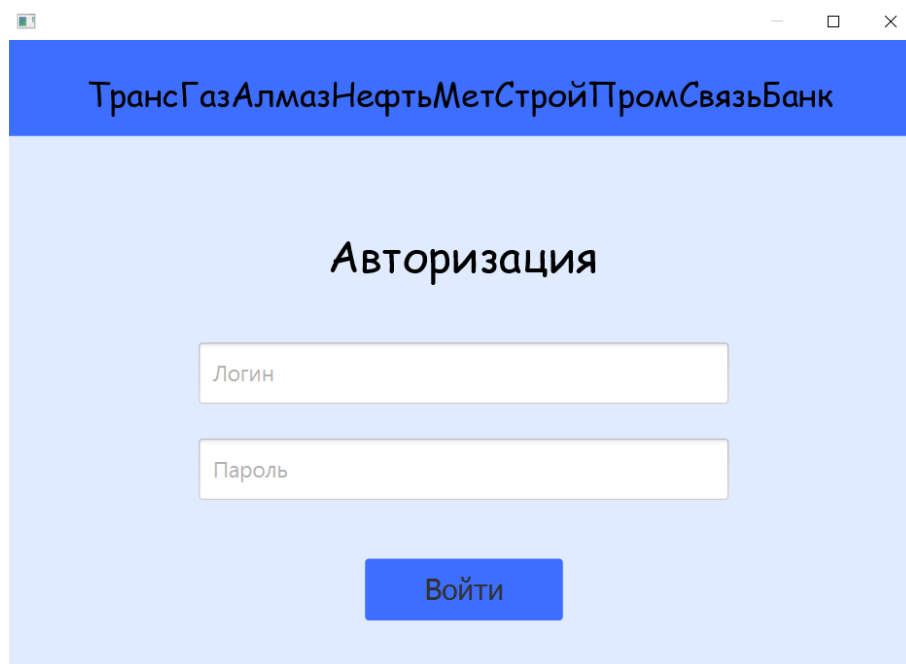


Рисунок 8.1 – Результат ввода некорректных данных авторизации

В меню для взаимодействия с информацией из БД я вводил неуникальные значения, по которым невозможно вернуть однозначный результат поиска, программа оставалась в режиме ожидания, до момента поиска по уникальному полю (рисунок 8.2).

The screenshot shows a window titled "Кредитные продукты" (Credit Products). It contains a search form with the following fields: "ID" (empty), "мин. размер" (empty), "мин. период" (empty), "макс. размер" (empty), "макс. период" (empty), "мин. %" (empty), "мин. рейтинг" (empty), "макс. %" (empty), and "цель" (containing "car"). At the bottom, there are four buttons: "Назад" (Back), "Добавить" (Add), "Найти" (Find), and "Удалить" (Delete).

Рисунок 8.2 – Результат поиска по неуникальному полю

Попытка ввода не всех аргументов для добавления записи в базу данных тоже не приводит к ошибке (рисунок 8.3).

The screenshot shows the same "Кредитные продукты" window. The "ID" field now contains the value "8". The "мин. размер" field contains "4000.0", "макс. размер" contains "500000.0", "мин. %" contains "3.0", and "макс. %" contains "12.0". The "мин. период", "макс. период", "мин. рейтинг", and "цель" fields remain empty. The buttons at the bottom are the same as in the previous screenshot.

Рисунок 8.3 – Результат добавления объекта с частью аргументов

Далее была произведена попытка удаления объекта по несуществующему индексу (рисунок 8.4).

Рисунок 8.4 – Результат поиска и удаления по несуществующему индексу

Подобное поведение пользователя также не приводит к ошибке, что говорит о высоком уровне надежности разработанного программного продукта.

ЗАКЛЮЧЕНИЕ

При разработке данного приложения были детально исследованы способы автоматизации процесса выдачи кредита физическому лицу.

Данное программное средство было создано с использованием современных технологий и стандартов программирования для упрощения бизнес-процессов банка. Оно сокращает время из-за автоматизации процессов передачи, хранения и изменения данных и высвобождает дополнительные ресурсы.

Информация хранится в базе данных и log-файле.

В программе предусмотрено использование следующих аспектов:

- программа работает в среде 32x и 64x разрядной ОС Windows 7 и выше, при наличии JDK 17 и выше;
- предусмотрена возможность развертывания базы данных внутри проекта;
- предусмотрен скрипт генерации тестовых данных.
- интерфейс программы русскоязычный.

Выполнено моделирование с использованием стандартов IDEF0 с использованием CASE средства ERwin process modeler. Также разработаны диаграммы по стандарту UML 2.0 с использованием программного средства Draw.io

Для описания работы приложения представлен следующий графический материал:

- диаграмма вариантов использования (Use Case);
- диаграмма состояний (Statechart);
- диаграмма последовательностей (Sequence diagram);
- диаграмма компонентов (component diagram);
- диаграмма развертывания (deployment diagram);
- информационная модель;
- блок-схемы алгоритмов, реализующих бизнес-логику
- IDEF0 – диаграмма выдачи кредита физическому лицу.

Итогом курсового проекта является разработанное графическое, клиент-серверное приложение, которое оптимизирует процесс выдачи кредита физическому лицу, обеспечивает сотрудникам быстрый доступ к данным и автоматические вычисления.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Финансовые информационные системы [Электронный ресурс]. – Режим доступа: <https://fisgroup.ru/blog/banking-automation/>
- [2] Экспоцентр Москва [Электронный ресурс]. – Режим доступа: <https://www.expocentr.ru/ru/articles-of-exhibitions/2016/avtomatizaciya-proizvodstva/>.
- [3] ИТ База знаний [Электронный ресурс]. – Режим доступа: <https://wiki.merionet.ru/seti/23/tcp-i-udp-v-chem-raznica/>
- [4] Язык моделирования IDEF0. Знакомство с нотацией IDEF0 и пример использования [Электронный ресурс]. – Электронный данные. – Режим доступа: <https://www.badarminsk.ru/yazyk-modelirovaniya-idef0-znakomstvo-s-notaciei-idef0-i-primer-ispolzovaniya/>
- [5] Архитектура хранилищ данных: традиционная и облачная [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/441538/>
- [6] Моделирование бизнеса IDEF, UML, ARIS [Электронный ресурс]. – Режим доступа: <https://analytics.infozone.pro/business-modeling-idef-uml-aris/>
- [7] Простое руководство по диаграмме компонентов <https://createlly.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/>
- [8] Электронный фонд правовых и нормативно-технических документов [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.cntd.ru/document/9041994>
- [9] Ручное тестирование [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Ручное_тестирование

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

```
//Реализация класса со скоринг-моделью

package com.example.course_project.database;

import java.sql.SQLException;

import static com.example.course_project.database.QueriesSQL.getFinFlows;
import static com.example.course_project.database.QueriesSQL.getLoanProducts;

public class ScoringPersonData {
    public int client_id;
    public String client_name;
    public String client_surname;
    public String client_patronymic;
    public String client_date_of_birth;

    public double client_total_debt;
    public double client_monthly_income;
    public double client_monthly_loan_payment;
    public int client_number_of_repaid_loans;
    public int client_total_number_of_loans;
    public double client_value_of_collateral;
    public int client_number_of_overdue_payments;

    public ScoringPersonData(int client_id, String client_name, String
client_surname, String client_patronymic, String client_date_of_birth, double
client_total_debt, double client_monthly_income, double
client_monthly_loan_payment, int client_number_of_repaid_loans, int
client_total_number_of_loans, double client_value_of_collateral, int
client_number_of_overdue_payments) {
        this.client_id = client_id;
        this.client_name = client_name;
        this.client_surname = client_surname;
        this.client_patronymic = client_patronymic;
        this.client_date_of_birth = client_date_of_birth;
        this.client_total_debt = client_total_debt;
        this.client_monthly_income = client_monthly_income;
        this.client_monthly_loan_payment = client_monthly_loan_payment;
        this.client_number_of_repaid_loans = client_number_of_repaid_loans;
        this.client_total_number_of_loans = client_total_number_of_loans;
        this.client_value_of_collateral = client_value_of_collateral;
        this.client_number_of_overdue_payments =
client_number_of_overdue_payments;
    }

    @Override
    public String toString() {
        return client_id +
            "," + client_name +
            "," + client_surname +
            "," + client_patronymic +
            "," + client_date_of_birth +
            "," + client_total_debt +
```

```

        "," + client_monthly_income +
        "," + client_monthly_loan_payment +
        "," + client_number_of_repaid_loans +
        "," + client_total_number_of_loans +
        "," + client_value_of_collateral +
        "," + client_number_of_overdue_payments +
        ' ';
    }

    public static String getScoring(String credit_order) throws SQLException
    {
        double total_score = 0.0;

        String client = QueriesSQL.getClientScoringData(credit_order);
        String loan_products = String.valueOf(getLoanProducts());
        String fin_flows = String.valueOf(getFinFlows());

        client = client.replaceAll(";", ",");

        String[] client_splited = client.split(",");

        if (Double.parseDouble(client_splited[6]) <
Double.parseDouble(client_splited[12])) {
            total_score += 1.5;}
        if (Integer.parseInt(client_splited[8]) > 0) {
            total_score += Integer.parseInt(client_splited[8]) * 1.5;}
        if (Double.parseDouble(client_splited[12]) > 1500.0) {
            total_score += 0.001 * Double.parseDouble(client_splited[12]);}
        if (Double.parseDouble(client_splited[10]) > 0.7 *
Double.parseDouble(client_splited[12])) {
            total_score += 0.003 * Double.parseDouble(client_splited[10]);}
        if (Integer.parseInt(client_splited[11].substring(0,
client_splited[11].length()-1)) > 2) {
            total_score -= Integer.parseInt(client_splited[11]);}
        if (Double.parseDouble(client_splited[13]) <
Double.parseDouble(client_splited[12]) * 12) {
            total_score += Double.parseDouble(client_splited[12]) *
12/Double.parseDouble(client_splited[13]);}
        if (client_splited[15].equals("car") ||
client_splited[15].equals("flat")) {
            total_score += 20.0;}
        if (Double.parseDouble(client_splited[14]) > 10.0 &&
Double.parseDouble(client_splited[14]) < 20.0) {
            total_score += 1.5 * (Double.parseDouble(client_splited[14]) -
10);}
        if (Integer.parseInt(client_splited[4].substring(0, 4)) > 2003) {
            total_score += 10.0;}

        loan_products = loan_products.substring(1, loan_products.length() -
1);

        loan_products = loan_products.replaceAll(";", ",");
        String[] loan_products_splited = loan_products.split(";");

        String best_product_id = "";
        Double product_persent = 10000.0;

        for (int i = 0; i < loan_products_splited.length; i++){
            String[] loan_products_args =
loan_products_splited[i].split(",");
            if (Double.parseDouble(loan_products_args[1]) <
Double.parseDouble(client_splited[13]) &&

```

```

        Double.parseDouble(loan_products_args[2]) >
Double.parseDouble(client_splited[13]) &&
        Double.parseDouble(loan_products_args[3]) <
Double.parseDouble(client_splited[14]) &&
        Double.parseDouble(loan_products_args[7]) < total_score
&&

        (loan_products_args[8].equals(client_splited[15]) ||
loan_products_args[8].equals("unknown") &&
            product_persent >
Double.parseDouble(loan_products_args[3]))
    ) {
        best_product_id = loan_products_args[0];
        product_persent = ((Double.parseDouble(loan_products_args[4])
- Double.parseDouble(loan_products_args[3]))
            / (0.05 * total_score)) +
Double.parseDouble(loan_products_args[3]);
    }

    fin_flows = fin_flows.substring(1, fin_flows.length() - 1);
    String[] fin_flows_args = fin_flows.split(",");

    String result = "";

    if (!product_persent.equals(10000.0) && total_score > 30.0 &&
(Double.parseDouble(fin_flows_args[1]) -
Double.parseDouble(fin_flows_args[2])
    - Double.parseDouble(fin_flows_args[3])) >
Double.parseDouble(client_splited[13])) {
        result = client_splited[2] + " " + client_splited[1] + " " +
client_splited[3]
            + " с номером = " + client_splited[0] + " Рекомендуется к
кредитованию кредитом с id = " + best_product_id
            + " по ставке: " + String.format("%.3f",product_persent)
+ " На сумму " + client_splited[13];
    }
    else {
        result = client_splited[2] + " " + client_splited[1] + " " +
client_splited[3]
            + " с номером = " + client_splited[0] + " Не
рекомендуется к кредитованию";
    }

    ScoringLogger.getScoringLogger().addScoringLogInfo(result);

    return result;
}

// Реализация сбора данных для скоринга

public static String getClientScoringData(String credit_order) throws
SQLException {

    String[] credit_order_data = credit_order.split("/");

    ResultSet resSetClientPers = statement.executeQuery("SELECT client_id
FROM clients_personal_data " +
        "WHERE client_passport_personal_number='" +
credit_order_data[0] + "';");

    int client_found_id = 0;

```

```

        while(resSetClientPers.next()){
            client_found_id = resSetClientPers.getInt("client_id");
        }

        ResultSet resSetClientFin = statement.executeQuery("SELECT
client_total_debt, client_monthly_income, " +
            "client_monthly_loan_payment, client_number_of_repaid_loans,
" +
            " client_total_number_of_loans, client_value_of_collateral,
client_number_of_overdue_payments, " +
            " clients_personal_data.client_id, client_name,
client_surname, client_patronymic, client_date_of_birth " +
            "FROM fct_clients_financial_data " +
            "INNER JOIN clients_personal_data " +
            "ON clients_personal_data.client_id =
fct_clients_financial_data.client_id " +
            "WHERE clients_personal_data.client_id='" + client_found_id +
            "'" +
            "ORDER BY client_loaned_date ASC " +
            "LIMIT 1;");

        double client_total_debt;
        double client_monthly_income;
        double client_monthly_loan_payment;
        int client_number_of_repaid_loans;
        int client_total_number_of_loans;
        double client_value_of_collateral;
        int client_number_of_overdue_payments;
        int client_id;
        String client_name;
        String client_surname;
        String client_patronymic;
        String client_date_of_birth;

        ArrayList<ScoringPersonData> scoringDataList = new ArrayList<>();

        while (resSetClientFin.next()) {

            client_total_debt =
resSetClientFin.getDouble("client_total_debt");
            client_monthly_income =
resSetClientFin.getDouble("client_monthly_income");
            client_monthly_loan_payment =
resSetClientFin.getDouble("client_monthly_loan_payment");
            client_number_of_repaid_loans =
resSetClientFin.getInt("client_number_of_repaid_loans");
            client_total_number_of_loans =
resSetClientFin.getInt("client_total_number_of_loans");
            client_value_of_collateral =
resSetClientFin.getDouble("client_value_of_collateral");
            client_number_of_overdue_payments =
resSetClientFin.getInt("client_number_of_overdue_payments");
            client_id =
resSetClientFin.getInt("clients_personal_data.client_id");
            client_name = resSetClientFin.getString("client_name");
            client_surname = resSetClientFin.getString("client_surname");
            client_patronymic =
resSetClientFin.getString("client_patronymic");
            client_date_of_birth =
String.valueOf(resSetClientFin.getDate("client_date_of_birth"));

```



```

        ScoringPersonData scoringPersonData = new
        ScoringPersonData(client_id, client_name, client_surname,
                           client_patronymic, client_date_of_birth,
                           client_total_debt, client_monthly_income,
                           client_monthly_loan_payment,
                           client_number_of_repaid_loans, client_total_number_of_loans,
                           client_value_of_collateral,
                           client_number_of_overdue_payments);

        scoringDataList.add(scoringPersonData);
    }
    return String.valueOf(scoringDataList).substring(1,
String.valueOf(scoringDataList).length() - 1)
        + "," + credit_order_data[1] + "," + credit_order_data[2] +
    "," +
        credit_order_data[3] + "," + credit_order_data[4];

// Реализация приема команд от пользователей с различным статусом
package com.example.course_project.server;

import com.example.course_project.database.GetScoringResult;
import com.example.course_project.database.QueriesSQL;
import com.example.course_project.database.ScoringPersonData;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.sql.SQLException;

import static com.example.course_project.database.QueriesSQL.*;

public class ClientHandler implements Runnable {

    private static Socket clientDialog;

    public ClientHandler(Socket client) {
        ClientHandler.clientDialog = client;
    }

    @Override
    public void run() {

        String[] switch_params;
        try {

            while (!clientDialog.isClosed()) {
                DataOutputStream out = new
                DataOutputStream(clientDialog.getOutputStream());
                DataInputStream in = new
                DataInputStream(clientDialog.getInputStream());

                String entry = in.readUTF();
                System.out.println("READ from clientDialog message - " +
entry);

                switch_params = entry.split(";");

                System.out.println(switch_params[2]);

```

```

        switch (switch_params[0]) {
            case "0":
                String[] login_password;
                login_password = switch_params[2].split("/");
                out.writeUTF(QueriesSQL.getUsers(login_password[0],
login_password[1]));
                break;
            case "1":
                break;
            case "2":
                switch (switch_params[1]) {
                    case "1":
                        String loan_products_list =
String.valueOf(getLoanProducts());
                        out.writeUTF(loan_products_list);
                        break;
                    case "2":
                        String bank_clients_list =
String.valueOf(getClients());
                        out.writeUTF(bank_clients_list);
                        break;
                    case "3":
                        QueriesSQL.deleteClient(Integer.parseInt(switch_params[2]));
                        out.writeUTF("Deleted");
                        break;
                    case "4":
                        String client =
String.valueOf(findClient((switch_params[2])));
                        out.writeUTF(client);
                        break;
                    case "5":
                        QueriesSQL.addClient(switch_params[2]);
                        out.writeUTF("Added");
                    case "6":
                        out.writeUTF(ScoringPersonData.getScoring(switch_params[2]));
                }
                break;
            case "3":
                switch (switch_params[1]) {
                    case "1":
                        String fin_flows =
String.valueOf(getFinFlows());
                        out.writeUTF(fin_flows);
                        break;
                    case "2":
                        String loan_products_list =
String.valueOf(getLoanProducts());
                        out.writeUTF(loan_products_list);
                        break;
                    case "3":
                        QueriesSQL.addCredit(switch_params[2]);
                        out.writeUTF("Added");
                        break;
                    case "4":
                        QueriesSQL.deleteCredit(Integer.parseInt(switch_params[2]));
                        out.writeUTF("Deleted");

```

```

        break;
        case "5":
            String credit =
String.valueOf(findCredit((switch_params[2])));
            out.writeUTF(credit);
            break;
        case "6":
            String scoring_results =
GetScoringResult.readScoringData();
            out.writeUTF(scoring_results);
            break;
        case "7":
            String cancel_confirmation =
GetScoringResult.CancelScoringResultsButtonClick(switch_params[2]);
            out.writeUTF(cancel_confirmation);
            break;
        case "8":
            String confirm_scoring =
GetScoringResult.ConfirmScoringResultsButtonClick(switch_params[2]);
            out.writeUTF(confirm_scoring);
            break;
    }
    break;
}

out.flush();

in.close();
out.close();

}
clientDialog.close();

} catch (IOException e) {
    e.printStackTrace();
} catch (SQLException e) {
    throw new RuntimeException(e);
}
}

}

// Реализация добавления клиента в базу данных

public static void addClient(String args) throws SQLException {

    String[] client_data = args.split("/");

    statement.executeUpdate("USE bank_credit_policy");
    statement.executeUpdate("INSERT INTO clients_personal_data (" +
        "client_name, client_surname, client_patronymic,
client_date_of_birth, client_passport_personal_number," +
        "client_passport_series, client_passport_number,
client_status)" +
        "VALUES ('" + client_data[0] + "', '" + client_data[1] + "',
'" + client_data[2] + "', '" + client_data[3] + "
        "'", '" + client_data[4] + "', '" + client_data[5] + "', '" +
client_data[6] + "', '" + client_data[7] + "')");
    }

// Реализация удаления кредитного продукта из базы данных

```

```

        public static void deleteCredit(int id) throws SQLException {

            statement.executeUpdate("USE bank_credit_policy");
            statement.executeUpdate("DELETE FROM loan_product WHERE product_id='"
+ id + "'");
        }

// Реализация метода авторизации

@FXML
protected void onEnterButtonClick() throws IOException {
    int status = 0;
    String response = ClientCommonFuctions.clientServerDialog(status, 0,
        login.getText().trim() + "/" + password.getText().trim());

    System.out.println("Client connected to socket.");

    enter_the_application.getScene().getWindow().hide();
    switch (response) {

        case "1":
            ClientCommonFuctions.openNewWindow("admin_menu.fxml",
                (Stage)
enter_the_application.getScene().getWindow());
            break;
        case "2":
            ClientCommonFuctions.openNewWindow("saler_menu.fxml",
                (Stage)
enter_the_application.getScene().getWindow());
            break;
        case "3":
            ClientCommonFuctions.openNewWindow("manager_menu.fxml",
                (Stage)
enter_the_application.getScene().getWindow());
            break;
        case "0":
            ClientCommonFuctions.openNewWindow("authorization.fxml",
                (Stage)
enter_the_application.getScene().getWindow());
            break;
    }
}

// Реализация функции обмена информации сервера и клиента

protected static String clientServerDialog(int status, int
function_identifider, String args_list) {

    String response = "";

    try(Socket socket = new Socket("127.0.0.1", 2222);
        DataOutputStream oos = new
DataOutputStream(socket.getOutputStream());
        DataInputStream ois = new
DataInputStream(socket.getInputStream()); )
    {

        System.out.println("Client connected to socket.");
    }
}

```

```

        while(!socket.isOutputShutdown() && response.equals("")){

            if(access) {

                System.out.println(status + ";" + function_identifier +
";" + args_list);

                oos.writeUTF(status + ";" + function_identifier + ";" +
args_list);

                oos.flush();

                response = ois.readUTF();
                System.out.println("Server response " + response);
            }
        }

    } catch (UnknownHostException e) {

        e.printStackTrace();
    } catch (IOException e) {

        e.printStackTrace();
    }

    return response;
}

// Реализация общей функции открытия окон для клиента

protected static void openNewWindow(String pathToNewWindow, Stage stage)
throws IOException {

    FXMLLoader fxmlloader = new
FXMLLoader(ClientCommonFuctions.class.getResource(pathToNewWindow));

    Parent root = (Parent) fxmlloader.load();
    stage = new Stage();
    stage.initModality(Modality.APPLICATION_MODAL);
    stage.setScene(new Scene(root));
    stage.show();
}

// Функция вывода информации о клиентах банка в listView

@FXML
protected void onRefreshClientButtonClick() {
    bank_clients.getItems().clear();
    String bank_clients_list =
ClientCommonFuctions.clientServerDialog(status, 2,
"0");

    bank_clients_list = bank_clients_list.substring(1,
bank_clients_list.length() - 1);
    bank_clients_list = bank_clients_list.replaceAll(";", " ");
    bank_clients_list = bank_clients_list.replaceAll(",", "\t\t\t\t");
    String[] loan_products_splited = bank_clients_list.split(";");

    String table_header = "ID клиента \t\tИмя \t\t\t\t\tФамилия \t\t\t\t\t
Отчество" +

```



```

    public String product_minimum_duration;
    public String product_maximum_duration;
    public double product_minimum_rating_access;
    public String product_mandatory_goal;

    public LoanProducts(int product_id, double product_minimum_amount, double
product_maximum_amount,
                        double product_minimum_percent, double
product_maximum_percent, String product_minimum_duration,
                        String product_maximum_duration, double
product_minimum_rating_access, String product_mandatory_goal){
        this.product_id = product_id;
        this.product_minimum_amount = product_minimum_amount;
        this.product_maximum_amount = product_maximum_amount;
        this.product_minimum_percent = product_minimum_percent;
        this.product_maximum_percent = product_maximum_percent;
        this.product_minimum_duration = product_minimum_duration;
        this.product_maximum_duration = product_maximum_duration;
        this.product_minimum_rating_access = product_minimum_rating_access;
        this.product_mandatory_goal = product_mandatory_goal;
    }

    @Override
    public String toString() {
        return product_id +
            "," + product_minimum_amount +
            "," + product_maximum_amount +
            "," + product_minimum_percent +
            "," + product_maximum_percent +
            "," + product_minimum_duration +
            "," + product_maximum_duration +
            "," + product_minimum_rating_access +
            "," + product_mandatory_goal +
            ';';
    }
}

// Реализация построения графика

public class PieChartSample extends Application {

    @Override public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Imported Fruits");
        stage.setWidth(500);
        stage.setHeight(500);

        String fin_flow_data = ClientCommonFuctions.clientServerDialog(3, 1,
"0");
        fin_flow_data = fin_flow_data.substring(1, fin_flow_data.length() -
1);
        String[] splited_info = fin_flow_data.split(",");

        ObservableList<PieChart.Data> pieChartData =
            FXCollections.observableArrayList(
                new PieChart.Data("Собственные фонды",
Double.parseDouble(splited_info[1])),
                new PieChart.Data("Заемные фонды",
Double.parseDouble(splited_info[2])),
                new PieChart.Data("Резервы",
Double.parseDouble(splited_info[3])));
        final PieChart chart = new PieChart(pieChartData);
    }
}

```

```
chart.setTitle("Денежные ресурсы");  
  
((Group) scene.getRoot()).getChildren().add(chart);  
stage.setScene(scene);  
stage.show();  
}  
}
```


ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг скрипта генерации базы данных

```
CREATE DATABASE bank_credit_policy;
USE bank_credit_policy;

CREATE TABLE clients_personal_data (
    client_id                int                auto_increment
primary key,
    client_name              varchar(30) not null,
    client_surname           varchar(30) not null,
    client_patronymic        varchar(30) not null,
    client_date_of_birth     date             not null,
    client_passport_personal_number varchar(14) not null,
    client_passport_series   varchar(2)  not null,
    client_passport_number   bigint(7)   not null,
    client_status            varchar(15) not null);

CREATE TABLE fct_clients_financial_data (
    client_id                int                primary key,
    client_loan_id           int                not null,
    employee_loaned_id       int                not null,
    client_loaned_date       date              not null,
    client_total_debt        double            not null,
    client_monthly_income    double            not null,
    client_monthly_loan_payment double        not null,
    client_loan_interest     double            not null,
    client_number_of_repaid_loans int          not null,
    client_total_number_of_loans int          not null,
    client_value_of_collateral double          not null,
    client_number_of_overdue_payments int       not null,
    client_risk_rate         double            not null);

CREATE TABLE loan_product (
    product_id              int                primary key,
    product_minimum_amount  double            not null,
    product_maximum_amount  double            not null,
    product_minimum_percent double            not null,
    product_maximum_percent double            not null,
    product_minimum_duration varchar(15) not null,
    product_maximum_duration varchar(15) not null,
    product_maximum_rating_access double       not null,
    product_mandatory_goal  varchar(15) not null);

CREATE TABLE bank_employee_authorization ("
    employee_id             int                primary key,
    employee_login          varchar(30) not null,
    employee_password        varchar(30) not null,
    employee_access_status  int                not null,
    employee_name            varchar(30) not null,
    employee_surname        varchar(30) not null,
    employee_patronymic     varchar(30) not null,
    employee_job_title      varchar(30) not null);

CREATE TABLE bank_financial_flows (
    fin_date                date              primary key,
```

bank_own_funds	double not null,
bank_borrowed_funds	double not null,
bank_reserve_funds	double not null,
bank_refinancing_rate	double not null,
central_bank_refinancing_rate	double not null,
bank_monthly_expected_income	double not null,
bank_monthly_expected_costs	double not null);

```

ALTER TABLE fct_clients_financial_data
ADD CONSTRAINT fk_t_fct_clients_fin_data_t_clients_pers_data
FOREIGN KEY (client_id)
REFERENCES clients_personal_data (client_id);

```

```

ALTER TABLE fct_clients_financial_data
ADD CONSTRAINT fk_t_fct_clients_fin_data_t_loan_product
FOREIGN KEY (client_loan_id)
REFERENCES loan_product (product_id);

```

```

ALTER TABLE fct_clients_financial_data
ADD CONSTRAINT fk_t_fct_clients_fin_data_t_bank_empl_auth
FOREIGN KEY (employee_loaned_id)
REFERENCES bank_employee_authorization (employee_id);

```

```

ALTER TABLE fct_clients_financial_data
ADD CONSTRAINT fk_t_fct_clients_fin_data_t_bank_fin_flows
FOREIGN KEY (client_loaned_date)
REFERENCES bank_financial_flows (fin_date);

```

ПРИЛОЖЕНИЕ В

(обязательное)

Отчет о проверке на заимствования в системе «Антиплагиат»

<input type="checkbox"/> Название ▾	Дата загрузки ▾	Оригинальность	
<input type="checkbox"/> PDF Пояснительная записка	<input checked="" type="checkbox"/> 19 Дек 2022 05:59	95,41%	ПОСМОТРЕТЬ РЕЗУЛЬТАТЫ

Рисунок В.1 – Отчет о проверке на заимствования в системе «Антиплагиат»