Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  старший преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Т.М. Унучек |
|  | \_\_\_.\_\_\_\_.2021 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«** **БАНКОВСКОЕ КРЕДИТОВАНИЕ И АВТОМАТИЗАЦИЯ ПОДСИСТЕМЫ ВЫДАЧИ КРЕДИТОВ ФИЗИЧЕСКИМ ЛИЦАМ»**

БГУИР КП 1-40 05 01-10 032 ПЗ

|  |  |
| --- | --- |
|  | Выполнила студентка группы 914301  ЖИЛИНСКАЯ Екатерина Витальевна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студентки) |
|  | Курсовой проект представлен на проверку 18.12.2021  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студентки) |

Минск 2021

**СОДЕРЖАНИЕ**

[Реферат](#введение) 3

[Введение](#введение) 4

[1 Описание предметной области](#требования) 5

[1.1 Кредитная функция банковской системы](#Исходные) 5

[1.2 Транзакционная функция банковской системы](#Исходные) 6

[2 Постановка задачи и обзор методов ее решения](#конструирование) 8

[3 Функциональное моделирование на основе стандарта IDEF0](#разработка_алгоритмов) 11

[4 Информационная модель системы](#описание) 16

[5 Обоснование оригинальных решений по использованию технических программных средств, не включенных в требования](#описание) 19

6 Описание алгоритмов, реализующих бизнес-логику серверной части проектируемой системы 21

[7 Руководство пользователя](#описание) 24

[8 Результаты тестирования разработанной системы](#разработка_алгоритмов) 35

[Заключение](#заключение) 38

[Список литературных источников](#Список) 39

[Приложение (рекомендуемое) А:Проверка на антиплагиат](#листинг) 40

[Приложение (обязательное) Б:Листинг алгоритмов, реализующих бизнес-логику](#листинг) 41

[Приложение (обязательное) В: Листинг части кода программы](#листинг) 44

[Приложение (обязательное) Г:Ведомость курсового проекта](#листинг) 55

**РЕФЕРАТ**

БГУИР КП 1-40 05 01-10 032 ПЗ

**Жилинская Е.В.** Банковское кредитование и автоматизация подсистемы выдачи кредитов физическим лицам : пояснительная записка к курсовому проекту / Е.В. Жилинская. – Минск : БГУИР, 2021. – 55 c.

Пояснительная записка 55 с., 30 рис., 10 источников, 4 приложений

ОФОРМЛЕНИЕ КРЕДИТОВ, ДЕНЕЖНЫЕ ПЕРЕВОДЫ, РЕДАКТИРОВАНИЕ ДАННЫХ, ПОЛЬЗОВАТЕЛЬ, АДМИНИСТРАТОР, ДОБАВЛЕНИЕ, УДАЛЕНИЕ, БАЗЫ ДАННЫХ

*Цель проектирования:* разработка программы для выдачи кредитов физическим лицам .

*Методология проведения работы:* В процессе решения поставленных задач были использованы методы компьютерной обработки данных.

*Результаты работы:* выполнен анализ литературных источников, спроектировано и написано приложение, разработана графическая часть проекта, проведен собственный анализ результатов.

*Область применения результатов:* могут быть использованы в банках для оформления кредитов и денежных переводов .

# **ВВЕДЕНИЕ**

Информационные технологии в банковском секторе позволяют накапливать и максимально использовать информацию о клиентах и их потребностях в услугах. В зависимости от особенностей бизнес-процессов банка, информационные технологии внедряются в различных подразделениях, одним из которых является фронт – офис банк, который автоматизирует такое направление, как ведение кредитов. Информационные технологии в данной области применяются для решения задач по ведению кредитных и расчетных счетов клиентов. В данном случае автоматизируется обслуживание клиента в учреждении банка, исполнение безналичных операций и получение отчетности по кредитным операциям.

Одна из функций – это кредитная функция банков. Так или иначе, но крупный бизнес и производства останутся при новой экономике. Без них человечество пока не может обойтись: что-то нужно кушать, на чем-то ездить. А раз так, то бизнесу нужны будут оборотные средства. Вряд ли частные лица будут обладать достаточным объемом денежных средств.

На сегодняшний день люди не могут без банковской системы и три основные функции банков – хранения, кредитная и транзакционная.

В связи с этим, есть необходимость, которая совпадает с целью курсовой работы, создания системы, с помощью которой будет происходить оформление кредитов.

Система позволит производить оформление кредитов, упростит внесение данных, дальнейшую обработку, анализ и составление отчётности для руководителей подразделений.

Для успешного достижения поставленной цели определены следующие этапы:

* изучение предметной области;
* проектирование структуры приложения, способов обработки и хранения данных;
* составление модели процесса для предметной области;
* построение информационной модели;
* выбор наиболее подходящих методов для реализации задачи.

Целью данного курсового проекта является разработка системы управления банковских кредитов. При разработке приложения будет использоваться язык программирования Java, и база данных MySQL.

Следуя из вышесказанного, тема, затронутая в курсовом проекте, является актуальной.

# **ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **Кредитная функция банковской системы**

Как выяснилось ранее одна из функций банка это кредитная, рассмотрим теперь это подробнее. Тема кредитов весьма противоречивая. С одной стороны, кредиты существенно упрощают людям жизнь, позволяя им иметь то, что они не могут себе позволить иметь без кредитов. Но, с другой стороны, подсаживаясь на кредитную иглу, человек может довольно сильно усложнить себе жизнь. Это, как минимум, лишняя головная боль, в которой не всегда есть смысл. Ведь быть кому-то должным, зная, что долг скорее всего придется отдавать, причем вместе с процентами, это значит жить в постоянном напряжении.

Итак, чтобы понять, что такое кредит и зачем он нужен, вначале необходимо разобраться с тем, что такое деньги. Если говорить коротко, то деньги – это то, за что люди должны работать. Они мотивируют людей к труду. Также можно сказать, что деньги – это идея, в которую все верят. Если в деньги никто верить не будет, никто не согласится за них работать. То, что деньги – это средство обмена, это уже третьестепенный факт. Работая за деньги, люди производят необходимые для жизни блага, которые мы все потребляем. Этими благами люди обмениваются с помощью денег. Таким образом, благодаря деньгам, люди приносят пользу друг другу. Люди хотят получать нужные им вещи сразу, прямо здесь и сейчас. Они не любят ждать, не любят копить, не любят планировать. Поэтому людям нужны быстрые деньги, чтобы удовлетворять свои желания сразу. Этому и способствует кредит – он позволяет людям обзавестись быстрыми деньгами ради осуществления своих желаний, либо для решения своих проблем. А потом они уже вынуждены будут работать, чтобы рассчитаться по кредиту.

Есть у кредитов и еще одна полезная функция – они дисциплинируют людей. Большинство людей недисциплинированны. Они часто не реализуют свои планы, не доводят начатые дела до конца, по нескольку раз на день меняют свои решения и так далее и тому подобное. Все это делает их ненадежными людьми, на которых нельзя положиться. А вот когда они берут кредит, то им приходится придерживаться определенных правил, чтобы более разумно управлять своими деньгами. В частности, им нужно рассчитывать их таким образом, чтобы вовремя выплачивать кредит. Простой пример: школа, в которой нас приучают учиться за оценки, а потом институт, в котором многие люди учатся только ради корочки, а не ради знаний. Так человек приучается служить кому-то за вознаграждение. Вот для таких людей кредит – это просто еще одна обязанность, которую они по привычке сами на себя возлагают. А вот человеку, который любит свободу, такой образ жизни не подходит. У него другие ценности, другие взгляды на жизнь. Ему лишние обязательства перед кем-либо не нужны.

При этом я не хочу сказать, что кредитом не могут и не должны пользоваться свободные люди, просто они это делают иначе – они стремятся приумножить кредитные деньги. Я взяла в кредит рубль, заработала с его помощью десять, а банку вернул два рубля. Подводя итог всему сказанному, могу заключить, что кредит, как стимул к труду, является очень эффективным инструментом.

## **Транзакционная функция банковской системы**

Что касается транзакционного бизнеса, то иностранные банки уже достаточно давно стали уделять внимание развитию направления транзакционного бизнеса. На Западе банки уже достаточно давно стали уделять внимание развитию этого направления. Банкам важно иметь стабильный источник доходов, поэтому транзакционный бизнес имеет высочайший приоритет в стратегиях развития крупнейших банков. Именно данное направление банковского бизнеса является стабильным источником формирования комиссионных доходов, позволяет банку диверсифицировать доходы и повысить устойчивость, что является очень важным при наступлении кризиса.

Рынок услуг транзакционного бизнеса является довольно конкурентным, при этом линейки транзакционных продуктов и услуг в банках не сильно отличаются между собой. Тем не менее есть составляющие, без которых банк не может быть успешен на этом рынке: в частности, сейчас наблюдается достаточно высокий интерес со стороны корпоративных клиентов к дистанционным продуктам и сервисам. А значит, банки должны развивать IT-инфраструктуру для того, чтобы суметь обеспечить этот спрос. Очень важно также суметь предложить клиенту такую услугу или сервис, которые смогут упростить ведение его повседневной операционной деятельности и позволят эффективно управлять ликвидностью компании.

Например, в банках есть такой продукт, как корпоративная карта для юридических лиц. Традиционно банки выпускают такие карты для того, чтобы клиенты могли начислять командировочные и рассчитываться по подотчетным целям. Однако имеется дополнительный сервис – инкассирование денежных средств через любой банкомат с функцией cash-in и возможность внесения наличных денежных средств с немедленным зачислением на текущий счет клиента.

Основная особенность продаж транзакционных продуктов состоит в том, чтобы увязать транзакционный продукт или банковскую услугу с непосредственной потребностью клиента. Корпоративные клиенты ориентировались на продукты финансирования и депозитные продукты, а весь комплекс расчетных сервисов называли РКО (расчетно-кассовое обслуживание). Но на самом деле транзакционные продукты – это достаточно широкий спектр продуктов и услуг, из которого каждый клиент может выбрать для себя комплекс, покрывающий потребности именно его бизнеса. Смс-информирование по счетам достаточно хорошо развит в рознице. Корпоративные клиенты его пока, на мой взгляд, воспринимают, как дополнительную опцию. На самом деле данный продукт позволяет реализовать такие потребности как защита средств компании от недобросовестных действий должностных лиц компании и защита от попытки незаконного списания денежных средств третьими лицами. Поэтому основной секрет успеха в продажах транзакционных продуктов – это понимание потребностей корпоративных клиентов и четкое формирование комплексного предложения в соответствии с запросами бизнеса.

# **ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ**

Цель программы состоит в разработке системы управления банковскими кредитами физических лиц. Объектом исследования является банковская система. Основными задачами является написание программы, которая будет выполнять основные денежные операции. Это будут кредиты, денежные переводы. Программа должна строиться на клиент – серверной архитектуре.

Клиентская часть позволяет отображать из базы данных все сделанные текущим пользователем кредиты, совершать оформление кредита, совершать денежные переводы, просматривать все сделанные денежные переводы. А модуль администратора позволяет просматривать все данные всего банка, то есть всех пользователей, все денежные переводы, совершенные всеми пользователями, все оформленные кредиты, произведенные пользователями информацию о типах кредитах. А также имеется функция поиска по данной таблице. Пользователь может так же перед оформлением кредита просматривать какую сумму он должен будет выплатить банку после окончания кредита. Задачей курсового проекта является создание клиент – серверного программного приложения, которое позволяет совершать основные операции, которые есть в любом банке. В базе данных храниться информация о пользователях: имя, пароль, сумма, которую он имеет на счету, дата регистрации, информацию о всех сделанных им кредитах, информацию о всех сделанных им денежных переводов.

Вход в приложение должен осуществляться для двух видов пользователей:

* администратор может просматривать все данные, всех оформленных кредитов, все денежные переводы, функции поиска;
* пользователь может так же оформить кредит, переводить деньги другим клиентам банка.

Бизнес – логика приложения организована на серверной части. В этой части программы обрабатываются полученные от клиента данные, а именно: внесение изменение в базу данных, внесение новых данных в базу данных, произведение расчета какую сумму пользователь будет должен банку по окончанию кредита. Для расчета суммы, которую пользователь должен будет по окончанию кредита предусмотрена отдельная функция вычисления, в зависимости от выбранного депозита.

Взаимодействие между клиентом и сервером должно быть организованно по технологии TCP/IP. Использование стека протоколов TCP/IP позволяет добиться надёжности связи между клиентом и сервером и осуществить параллельную обработку запросов.

Таким образом, для успешной реализации необходимо решить поставленную задачу изложенными выше методами.

Программный код должен быть написан на объектно – ориентированном языке Java. Java представляет собой язык программирования и платформу вычислений, которая была впервые выпущена Sun Microsystems в 1995 г. Существует множество приложений и веб-сайтов, которые не работают при отсутствии установленной Java, и с каждым днем число таких веб-сайтов и приложений увеличивается. Java отличается быстротой, высоким уровнем защиты и надежностью. От портативных компьютеров до центров данных, от игровых консолей до суперкомпьютеров, используемых для научных разработок, от сотовых телефонов до сети Интернет.

Как правило компьютеры и программы, входящие в состав информационной системы, не являются равноправными. Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам. Компьютер (или программу), управляющий ресурсом, называют сервером этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер). Клиент и сервер какого-либо ресурса могут находится как в рамках одной вычислительной системы, так и на различных компьютерах, связанных сетью.

Основной принцип технологии "клиент – сервер" заключается в разделении функций приложения на три группы:

* ввод и отображение данных (взаимодействие с пользователем);
* прикладные функции, характерные для данной предметной области;
* функции управления ресурсами (файловой системой, базой данных и т.д.).

Поэтому, в любом приложении выделяются следующие компоненты:

* компонент представления данных;
* прикладной компонент;
* компонент управления ресурсом.

Таким образом, клиент–сервер – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами.

Фактически клиент и сервер – это программное обеспечение. Программы – сервера, ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями, просмотр web-страниц во всемирной паутине).

Преимущества клиент-серверных систем:

* отсутствие дублирования кода программы-сервера программами-клиентами;
* так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются;
* все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов и на сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

Связь между сервером и клиентом в приложении осуществляется при помощи TCP/IP. Также в реализации системы были использованы следующие техники:

* разработаны и использованы собственные иерархии, расширение базовых классов, предоставляемых JDK;
* реализованы паттерны проектирования;
* использовано сокрытие данных (инкапсуляция), перегрузка методов, переопределение методов, абстрактные типы данных (интерфейсы, абстрактные классы), статические методы, обработка исключительных ситуаций.

Требуется предусмотреть возможность просмотра, редактирования, удаления данных, создания новых записей. Сервер должен обладать возможностью параллельной обработки запросов, так как управлять кредитами может не один, а несколько человек. Данную вещь следует добиться через использование в программном обеспечении многопоточности. Связь с базой данных должна осуществляться с помощью SQL-запросов через сервер.

В качестве основной среды разработки в проекте используется Intell IJ Idea. В качестве СУБД в курсовом проекте используется MYSQL. Для создания UML-диаграмм в работе было использовано средство draw.io, которое позволяет осуществлять создание диаграмм вариантов использования, состояний, последовательностей и развертывания с помощью моделирования. Для выполнения UML-моделей в стандарте IDEF0 и информационного моделирования применялось средство CA AllFusion ERwin Data Modeler r7 (ERwin).

# **ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0**

Для грамотного описания основного процесса предметной области используются различные графические методики и инструменты. Одной из таких методик является методология IDEF0.

Методология создания функциональной модели производственной среды или системы. Она была разработана в середине 70-х годов в США в ходе реализации проекта ICAM (Integrated Computer-Aided Manufacturing). Его целью являлась разработка подходов, обеспечивающих повышение эффективности производства благодаря систематическому внедрению компьютерных технологий.

IDEF0-методология предназначена для создания описания систем и основана на концепциях системного моделирования. Моделированием называется процесс создания точного описания системы.

В IDEF0-моделях используются как естественный, так и графический языки. Данная модель дает полное и точное описание, адекватное системе и имеющее конкретное назначение. Назначение описания называют целью модели.

Основным процессом данной предметной области является процесс управления банковскими кредитами. Для того, чтобы осуществить моделирование, необходимо составить список вопросов к IDEF0-модели. Вопросы будут следующие:

1. Что необходимо для управления кредитами?
2. Каким образом происходит данный процесс?
3. Источники данных, использующиеся при работе банка?
4. В какой валюте будут происходить денежные переводы?

Таким образом, можно сформулировать цель модели: определить основные этапы процесса управления банковскими кредитами физических лиц. Цель становится критерием окончания моделирования.

Также для составления IDEF0-модели необходимо определить претендентов на точку зрения и выбрать точку зрения.

Точка зрения – позиция, с которой описывается система. Точка зрения диктует автору модели выбор нужной информации о субъекте и форму ее подачи.

Описание модели IDEF0 организовано в виде иерархии взаимосвязанных диаграмм. Вершина этой древовидной структуры представляет самое общее описание системы, а ее основание состоит из наиболее детализированных описаний.

На вершине иерархии находится контекстная диаграмма модели (Рис. 3.1). В качестве входных данных выступают «информация о кредите» и «данные пользователя», в качестве выходных – «выдача кредита». Механизмами, осуществляющими преобразования данных, являются «Системный администратор» «Бизнес-логика» и «Программное обеспечение». Управлением являются «ГОСТ», «Внутренний порядок обслуживания», «Закон «О защите информации»».

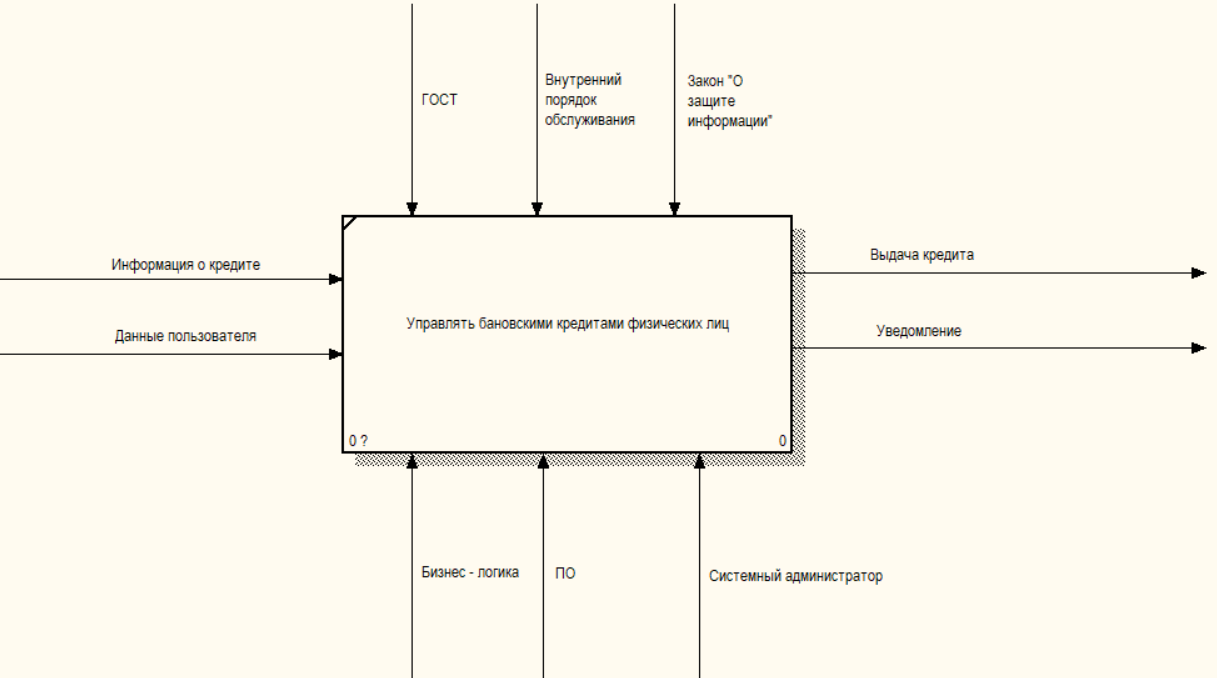


Рисунок 3.1 – Контекстная диаграмма верхнего уровня

Декомпозиция контекстной диаграммы (Рис. 3.2) описывает разделение процесса управления банковскими кредитами физических лиц на следующие этапы:

* авторизовать пользователя;
* оформить кредит;
* изменить баланс клиента;
* уведомить об успешном оформлении кредита.

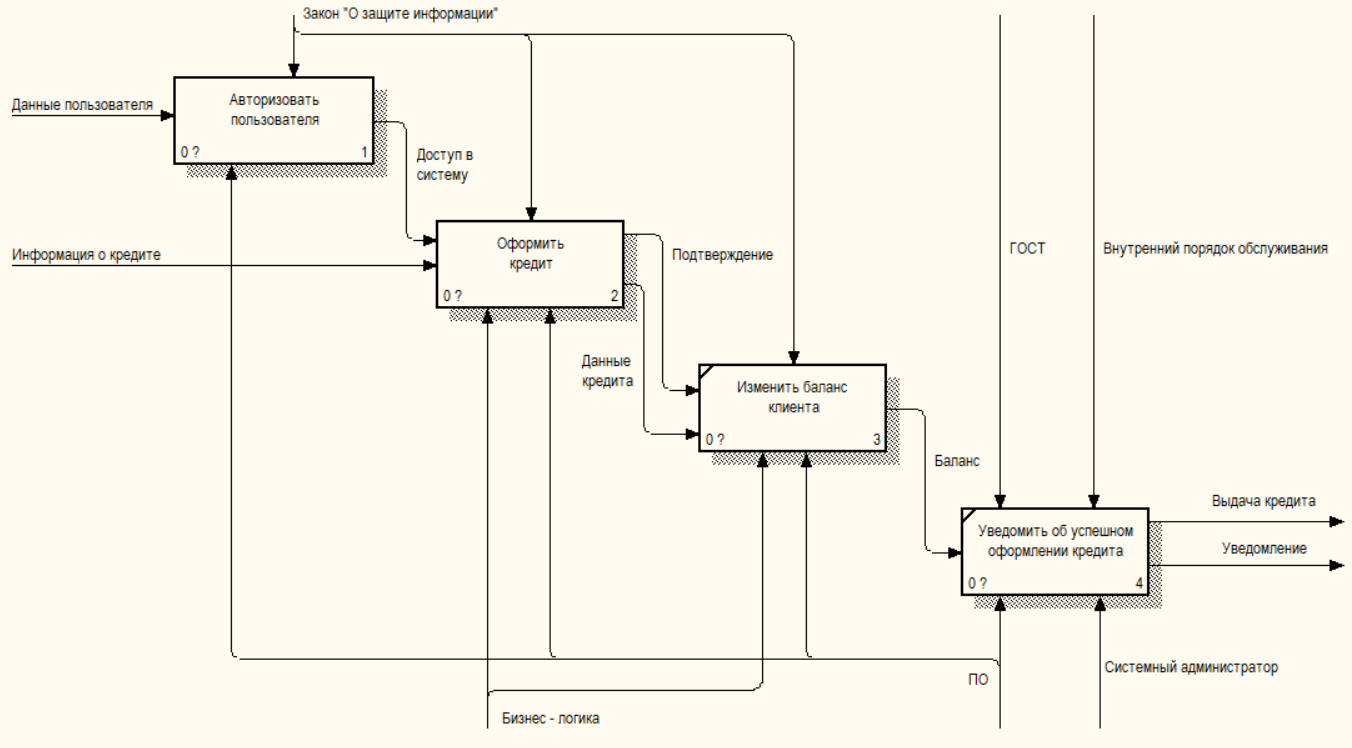


Рисунок 3.2 – Декомпозиция диаграммы верхнего уровня

Можно провести декомпозицию блока «Авторизовать пользователя» (Рис. 3.3). Данный блок можно разбить следующим образом:

* считать логин и пароль;
* проверить наличие пользователя в системе;
* предоставить доступ к системе.

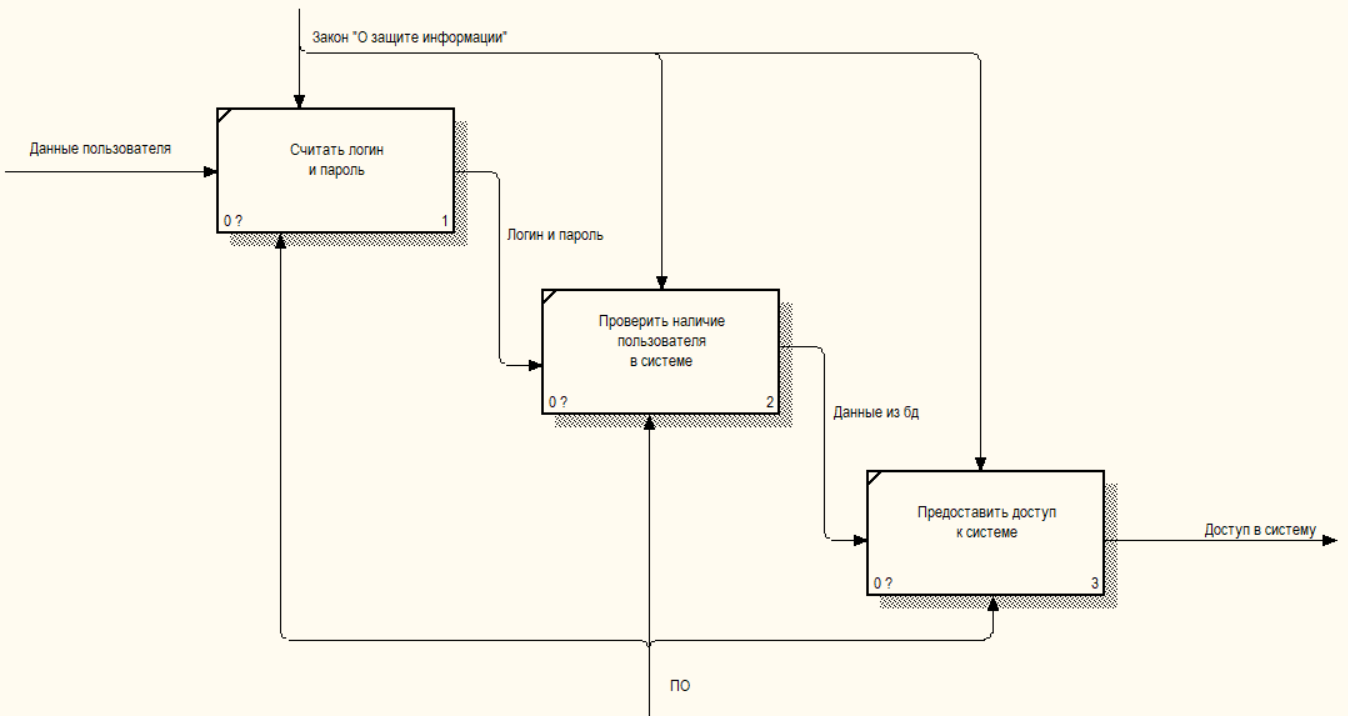


Рисунок 3.3 – Декомпозиция блока «Авторизовать пользователя»

Следующим блоком для декомпозиции является «Оформить кредит» (Рис. 3.4). Данный блок состоит из:

* выбрать кредит;
* считать данные о выбранной кредите;
* подтвердить оформления кредита.

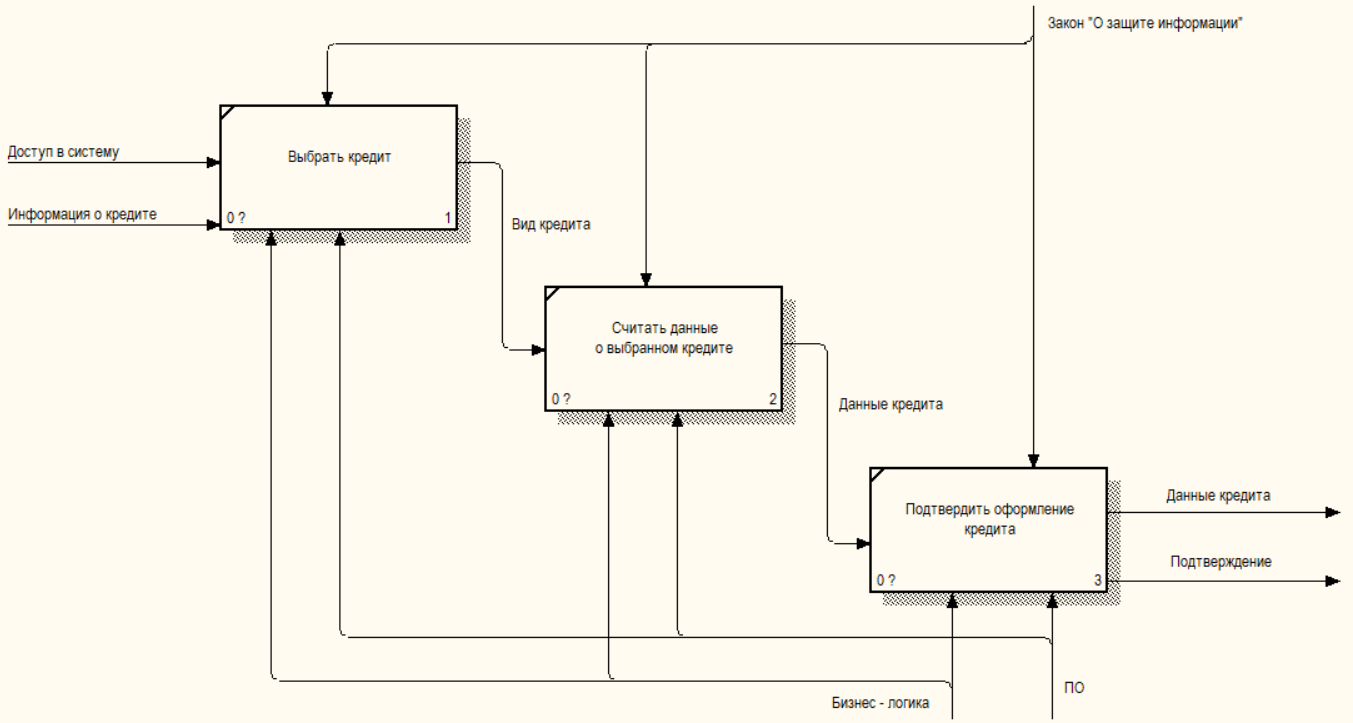


Рисунок 3.4 – Декомпозиция блока «Оформить кредит»

Последним блоком для декомпозиции является «Изменить баланс клиента» (Рис. 3.5). Данный блок состоит из:

* отправить код подтверждения;
* произвести расчет;
* отобразить баланс.

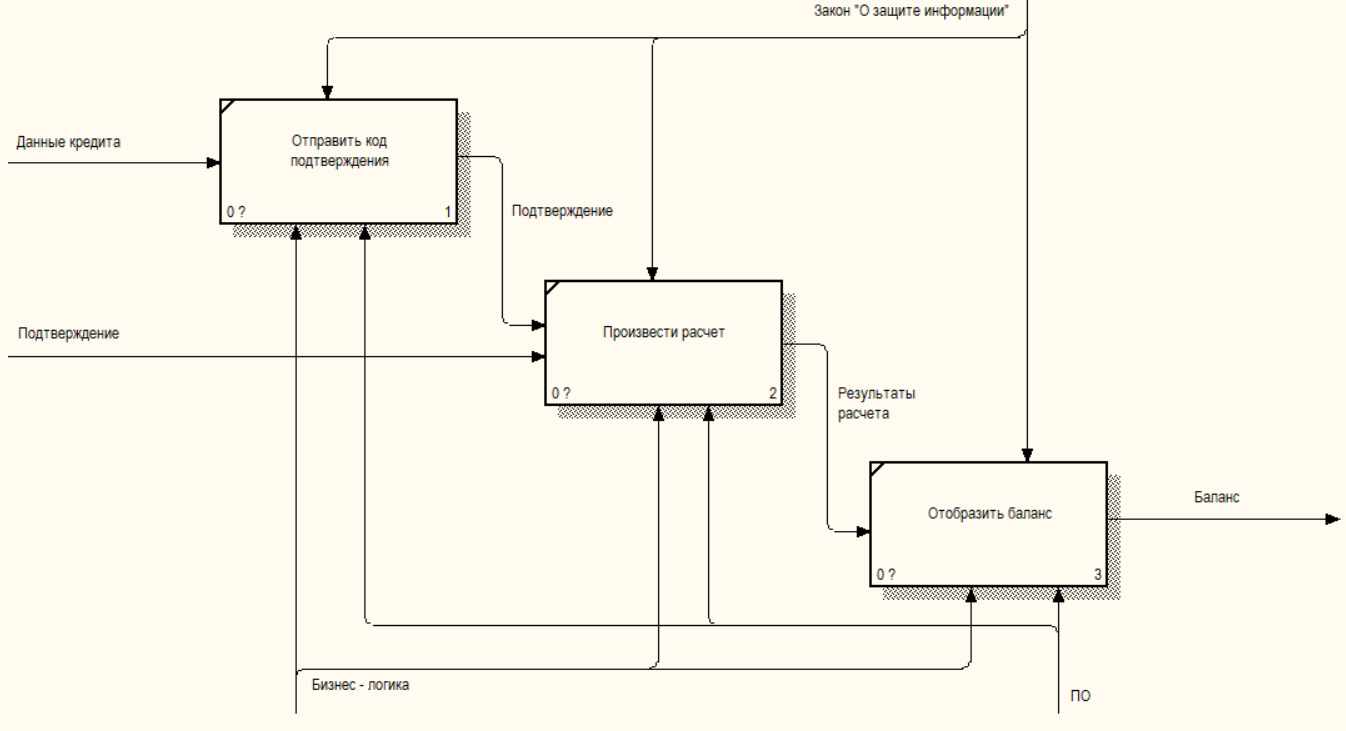


Рисунок 3.5 – Декомпозиция блока «Изменить баланс клиента»

С помощью представленной IDEF-0 диаграммы можно более подробно и тщательно рассмотреть процесс управления банковскими кредитами физических лиц. Можно рассмотреть процесс уведомления об успешном оформлении кредита (Рис 3.6).

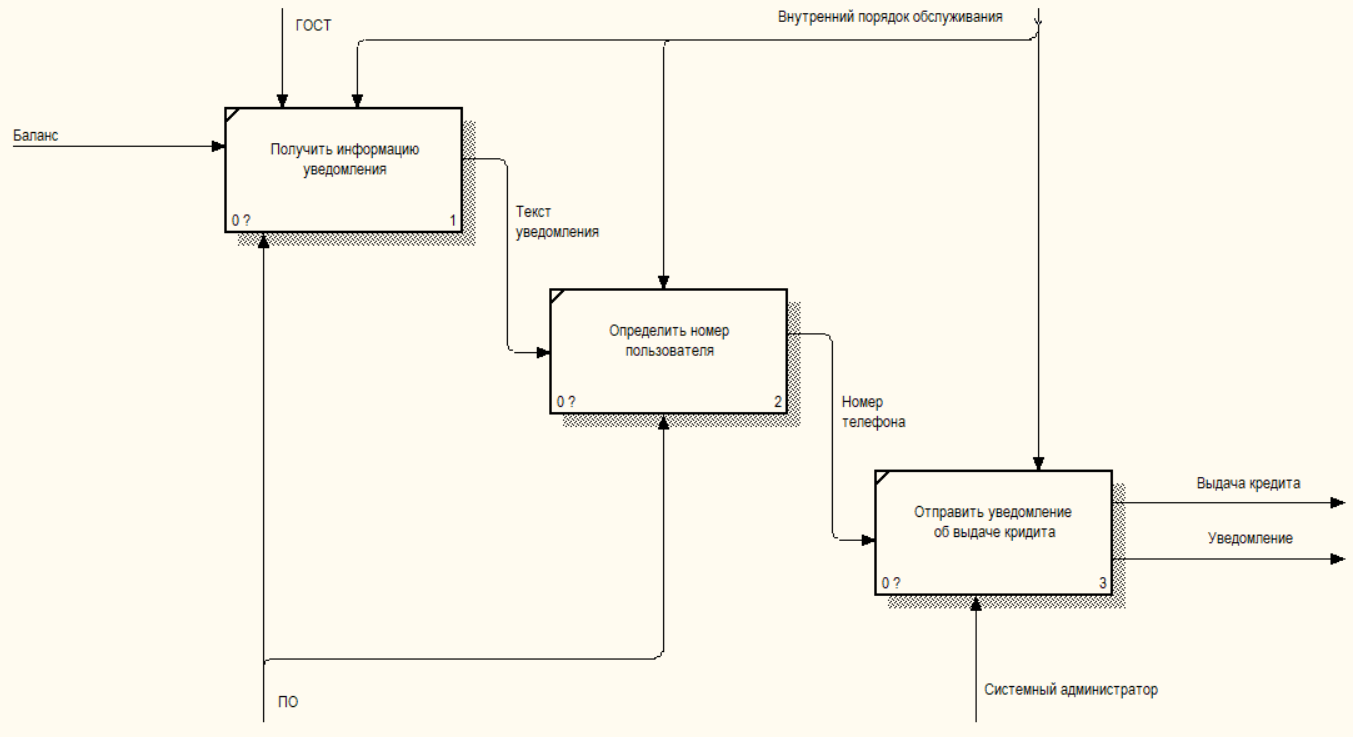


Рисунок 3.6 – Декомпозиция блока «Уведомить об успешном оформлении кредита»

# **ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ**

Моделирование данных – это процесс описания информационных структур данных и установления связей и отношений между ними с целью определения информационных потребностей системы.[10]

Модель данных в наиболее общем понятии, представляет собой баланс между определёнными потребностями специфического проекта приложения и между определёнными потребностями специфического проекта приложения и общими информационными требованиями предметной области. Модель данных включает все объекты, их атрибуты и отношения, требуемые данным отдельным проектом.

При моделировании базы данных использовалась методология IDEFX, служащая для разработки реляционных баз данных, использующая условный синтаксис при разработке концептуальной схемы.

Для разработки схемы проектирования IDEFX1 используется CASE-средство AllFusion Erwin Data Modeler.

При построении информационной модели используется алгоритм:

* определение сущностей и связей между сущностями;
* определение первичных и вторичных ключей;
* определение атрибутов сущностей;
* переход к физическому описанию.

В результате информационного моделирования мною были выделены следующие сущности:

* пользователи;
* кредиты;
* виды кредитов;
* сохранения в файл;
* денежные переводы.

Для хранения информации о пользователях необходимо использовать сущность «users». В ней администратор может добавлять новых пользователей либо новый пользователь может сам зарегистрироваться. Таблица пользователей содержит в себе информацию об имени, пароле, дате регистрации, сумме, которую он имеет на счету и номер своего аккаунта. Номер аккаунта пригодится для того, чтобы другие пользователи по номеру аккаунта могли переводить пользователю денежные средства.

Для хранения всех кредитов создана сущность «deposit». Каждый раз, когда пользователь оформляет кредит, в таблицу кредитов заносится информация об идентификаторе пользователя, который оформляет кредит, идентификаторет вида кредита который совершается, сумма, которую пользователь оформляет, дата совершения кредита. Минимальной суммой кредита является 50$.

Для хранения вида кредитов создана таблица «deposit\_type». Данная таблица содержит в себе имя кредита, срок данного кредита, и процентную ставку текущего кредита. Чем больше срок, тем больше устанавливается процентная ставка. Пользователь может выбирать для себя кредит с удобным для себя сроком.

Для хранения денежных переводов создана таблица «transfer\_money». Таблица хранит в себе идентификатор пользователя, который совершает денежный перевод, сумму, которую переводит пользователь, номер счёта куда пользователь переводит деньги, и дату совершенного перевода.

Для хранения всех сохранений в файл создана таблица «save\_to\_file». Таблица позволяет владеть информацией о том, когда пользователь совершает сохранения в файл текущего действия. Таблица содержит в себе идентификатор пользователя, который сохраняет информацию и дату сохранения.

Для более подробного описания были построена физическая модель базы данных (Рис. 4.1).

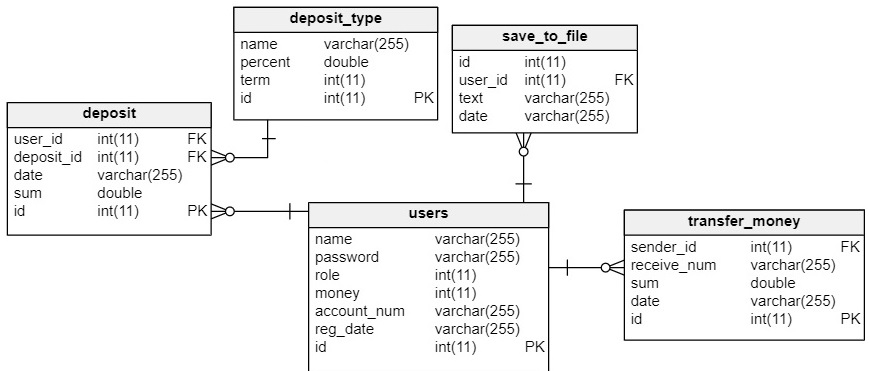


Рисунок 4.1 – Физическая модель базы данных

Любой пользователь может сохранять информацию в файл множество раз, поэтому между таблицей «users» и таблицей «save\_to\_file» связь один ко многим. Любой пользователь может переводить деньги другому пользователю множество количество раз, поэтому между таблицей «users» и таблицей «transfer\_money» связь один ко многим. Так же таблица «deposit\_type» связывается связью один ко многим с таблицей «deposit» так как один и тот же вид кредита может совершаться множество раз. Пользователь может не один раз оформлять кредит следовательно таблица «users» связана с таблицей «deposit» связью один ко многим.

# **ОБОСНОВАНИЕ ОРИГИНАЛЬНЫХ РЕШЕНИЙ ПО ИСПОЛЬЗОВАНИЮ ТЕХНИЧЕСКИХ ПРОГРАММНЫХ СРЕДСТВ, НЕ ВКЛЮЧЕННЫХ В ТРЕБОВАНИЯ**

В данной программе было использовано оформление кредита и просмотр того, какую сумму пользователь будет иметь. Многие банки могут предложить пользователям свои виды кредитов. Кредит пользователя должен иметь максимальную прибыль для него, и максимально благоприятные условия, чтобы пользователь воспользовался именно услугами моего банка, а не другого. При оформлении кредита пользователь может выбрать кредит, наиболее подходящий для него. Кредиты отличаются между собой процентной ставкой, сроком использования и, очевидно, названием. Таким образом следует выбрать кредит, который будет устраивать пользователя и по сроку длительности, и по процентной ставке. Чем длительнее срок кредита, тем больше процентная ставка. Пользователь может ввести сумму в поле ввода и просмотреть какую сумму он будет должен после завершения срока кредита. После оформления кредита пользователь на главном окне может просмотреть свой текущий счет и сумму.

Одной из отличительной особенности программы является использование паттернов проектирования, которые позволяют упростить работу программы. Первый патерн который я использовала это Singleton. Паттерн Singleton гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа. В системе будет существовать не более одного экземпляра заданного класса. Экземпляр должен быть легко доступен для всех клиентов данного класса. Часто в системе могут существовать сущности только в единственном экземпляре, например, текущая дата. Текущая дата, которая может храниться в Java в формате long равна количеству миллисекунд, пройденному с первого января 1970 года. Это число не может равняться каким-то двум значениям, в таких случаях нам и понадобиться паттерн Singleton. В таких случаях необходимо уметь создавать единственный экземпляр некоторого типа.

Первым делом нужно создать объект текущего класса, который будет глобальный и иметь модификатор видимости public. Данная переменная будет доступна всегда и во всех классах. Однако, использовать глобальную переменную некоторого типа непосредственно невозможно. Чтобы обратиться к объекту текущего класса или создать его нужно вызвать статический метод getInstance() , который создается разработчиком вручную. Данный метод будет возвращать текущий объект класса. В самом методе происходит проверка. Если объект текущего класса не равен null, то нам возвращается текущий объект, который уже создан. Если же наш объект равен null, то тогда наш объект создается привычным для нас путём создания объекта. Таким образом пользователь не сможет создать два объекта, объект будет только один.

Второй паттерн, который я использовала это Builder. Если ваш класс содержит большое количество конструкторов и статических методов, то возникает проблема – трудно запомнить порядок и назначение необязательных параметров при вызове класса. В таких случаях рекомендуется использовать шаблон Builder. Для того чтобы создать объект нашего класса без использования шаблона строителя нам придется создавать объект следующим образом:

MyClass myClass = new MyClass(3,6,7,2,”name”,8,2,9,10).

Глядя на набор чисел сложно понять, какая цифра за что отвечает. Можно вызвать конструктор без параметров, чтобы создать объект и потом вызывать сеттеры. Таким образом количество кода в классе увеличивается. Но тогда класс находится в неустойчивом состоянии, так как зависим от множества отдельных вызовов метода и не может обеспечить проверку своего состояния. Поэтому и был придумать Builder. Вместо непосредственного создания объекта класса вызывается конструктор со всеми необходимыми параметрами. Затем вызывается сеттеры для установления необходимых параметров. В завершение вызывается метод build() для генерации объекта. Реализуется такой подход через статический внутренний класс. И таким образом создание объекта класса будет выглядеть иным образом:

MyClass myClass = new MyClass(10,20).setAge(10).setWeight(85).setYear(2018).

Данный код выглядит намного проще, и при задании какого-либо свойства объекту мы вызовем нужный нам метод с понятным для разработчика названием.

# **ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС –** **ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ**

Для реализации серверной части проектируемой системы в программе используется средство для обмена информацией между компьютерами TCP/IP. TCP/IP – это не зависящий от платформы стандарт, который перекидывает мосты через пропасть, лежащую между разнородными компьютерами, операционными системами и сетями. Это протокол, который глобально управляет Internet, и в значительной мере благодаря сети TCP/IP завоевал свою популярность.

Понимание TCP/IP главным образом подразумевает способность разбираться в наборах таинственных протоколов, которые используются главными компьютерами TCP/IP для обмена информацией. Давайте рассмотрим некоторые из этих протоколов и выясним, что составляет оболочку TCP/IP.

Программное обеспечение для TCP/IP, на вашем компьютере, представляет собой специфичную для данной платформы реализацию TCP, IP и других членов семейства TCP/IP. Обычно в нем также имеются такие высокоуровневые прикладные программы, как FTP (File Transfer Protocol, Протокол передачи файлов), которые дают возможность через командную строку управлять обменом файлами по Сети.

Причина, по которой TCP/IP столь важен сегодня, заключается в том, что он позволяет самостоятельным сетям подключаться к Internet или объединяться для создания частных интрасетей. Вычислительные сети, составляющие интрасеть, физически подключаются через устройства, называемые маршрутизаторами или IP-маршрутизаторами. Маршрутизатор – это компьютер, который передает пакеты данных из одной сети в другую. В интрасети, работающей на основе TCP/IP, информация передается в виде дискретных блоков, называемых IP-пакетами (IP packets) или IP-дейтаграммами (IP datagrams). Благодаря программному обеспечению TCP/IP все компьютеры, подключенные к вычислительной сети, становятся "близкими родственниками". По существу, оно скрывает маршрутизаторы и базовую архитектуру сетей и делает так, что все это выглядит как одна большая сеть. Точно так же, как подключения к сети Ethernet распознаются по 48-разрядным идентификаторам Ethernet, подключения к интрасети идентифицируются 32-разрядными IP-адресами, которые мы выражаем в форме десятичных чисел, разделенных точками (например, 128.10.2.3). Взяв IP-адрес удаленного компьютера, компьютер в интрасети или в Internet может отправить данные на него, как будто они составляют часть одной и той же физической сети.

TCP/IP дает решение проблемы данными между двумя компьютерами, подключенными к одной и той же интрасети, но принадлежащими различным физическим сетям. Решение состоит из нескольких частей, причем каждый член семейства протоколов TCP/IP вносит свою лепту в общее дело. IP – самый фундаментальный протокол из комплекта TCP/IP – передает IP-дейтаграммы по интрасети и выполняет важную функцию, называемую маршрутизацией, по сути дела это выбор маршрута, по которому дейтаграмма будет следовать из пункта А в пункт B, и использование маршрутизаторов для "прыжков" между сетями.

TCP – это протокол более высокого уровня, который позволяет прикладным программам, запущенным на различных главных компьютерах сети, обмениваться потоками данных. TCP делит потоки данных на цепочки, которые называются TCP-сегментами, и передает их с помощью IP. В большинстве случаев каждый TCP-сегмент пересылается в одной IP-дейтаграмме. Однако при необходимости TCP будет расщеплять сегменты на несколько IP-дейтаграмм, вмещающихся в физические кадры данных, которые используют для передачи информации между компьютерами в сети. Поскольку IP не гарантирует, что дейтаграммы будут получены в той же самой последовательности, в которой они были посланы, TCP осуществляет повторную "сборку" TCP – сегментов на другом конце маршрута, чтобы образовать непрерывный поток данны.

В данной программе мы должны будем сначала запустить сервер, затем клиент будет подключаться к серверу и будет происходить логическое соединение. Приложение является многопоточным поэтому сколько угодно клиентов могут одновременно подключаться к одному серверу.

В самом начале нужно будет создать сокет и присвоить ему номер порта. Далее создать объект server object output stream для того, чтобы мы могли нашему клиенту отправлять какой-либо ответ, и создать server object input stream. Этот объект нам понадобиться для того, чтобы мы могли прочитать что нам прислал наш клиент.

Для того чтобы реализовать многопоточный клиент сервер нужно наследовать наш класс сервера от класса Thread. И далее когда каждый раз клиент подключается для него создавать отдельный поток с помощью метода start() , который в свою очередь будет вызывать метод run() который будет выполнять всю нашу программу.

Каждый раз, когда клиент будет обращаться к серверу он будет отсылать ему строку, в которой будет наше действие, которое должен будет выполнить сервер. Например, нам нужно будет авторизоваться. В данном случае клиент отправит строку вида: «signUp login password». На стороне сервера нам нужно будет разбить эту строчку по пробелам и записать в массив строк. Первый элемент нашего массива это и будет действие, которое должен выполнить сервер. В зависимости от того какое это действие мы будем вызывать соответствующий метод на стороне сервера. Сервер производит наши действия. Идет запрос в базу данных, и в зависимости от того, наши данные принадлежат обычному пользователю или администратору сервер будет отсылать клиенту какую-ту строку. Для того чтобы отправить строку нам нужно будет обратить к нашему объекту server object output stream и записать в него нашу строку, к примеру «ADMIN» или «USER».

На стороне клиента тоже создается два объекта. Client object output stream который нужен будет для того, чтобы отправить нашему серверу какую-либо строку. И client object input stream. Для того чтобы считать то, что нам отправил сервер. Помимо этого, нам понадобиться сокет для установления логического соединения. В сокете нужно будет указать IP-адрес по которому происходит соединение и номер порта. Клиент, как было сказано раньше отправляет серверу строку, сервер выполняет свои действия, отправляет клиенту ответ, и с помощью объекта client object input stream клиент получает ответ и на клиентской стороне происходят какие-либо действия. К примеру, сервер отослал клиенту строку «ADMIN» – клиент открывает окно администратора. Сервер отослал клиенту строку «USER» – клиент открывает окно пользователя.

Взаимодействие достаточно простое. А что делать если требуется считать целый массив каких-то объектов? Большое количество записей в базу данных? В таком случае нам нужно на стороне сервера все наши данные записать в коллекцию ArrayList типа String и отправлять клиенту массив строк. В свою очередь клиент будет преобразовывать наш массив строк в нужный нам формат.

Одной из задач курсовой работы было расчет суммы, которую получит пользователь после окончания срока депозита. Клиент посылает на сервер команду и сумму, которую нужно рассчитать. Клиент получается данную строку, разбивает на пробелы, определяет команду и переходит к расчету. Расчет суммы, полученной по окончанию депозита равен сумме текущей суммы плюс сумме, умноженной на процентную ставку текущего депозита. Сервер отправляет клиенту преобразованную в строку сумму и клиент уже обрабатывает эту строку и выводит пользователю на экран.

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Серверная и клиентская часть программы реализована в разных проектах. Для начала работы запускаем сервер, затем запускаем клиент. После этого в окне авторизации вводим данные пользователя. (Рис.7.1).

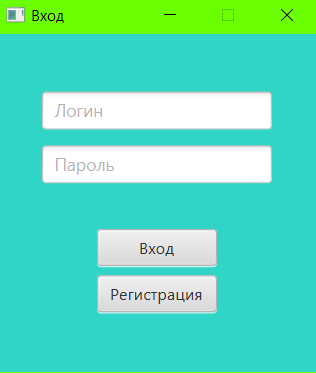


Рисунок 7.1– Окно авторизации

Окно авторизации позволяет нам авторизоваться, либо если вы не имеете учетной записи, то зарегистрироваться (Рис 7.2).

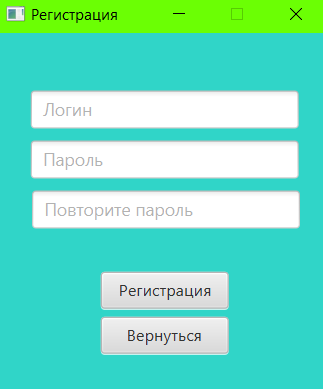


Рисунок 7.2 – Окно регистрации

В окне авторизации происходит проверка на то, введенные данные являются данными пользователя или администратора. Если же это пользователь, то открывается окно пользователя (Рис 7.3).

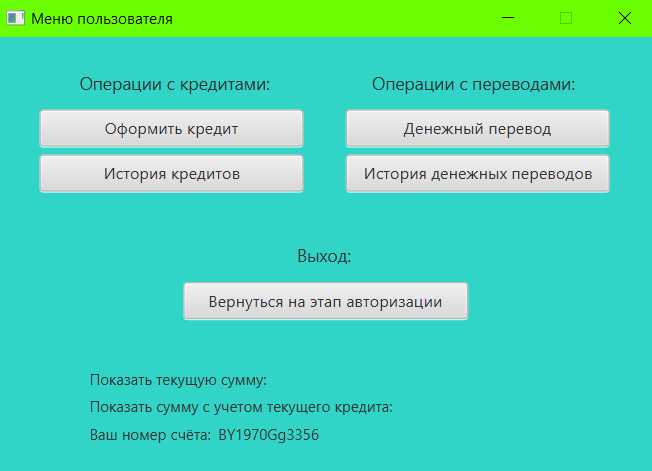


Рисунок 7.3 – Окно пользователя

Окно пользователя содержит в себе четыре основных действия: оформление кредита, просмотр истории кредитов, осуществление денежного перевода, просмотр истории денежных переводом. Первое из которых это оформить кредит. Есть несколько видов кредита каждый из которых имеет свое название, процентную ставку, и срок. В зависимости от выбранного вида кредита меняется процентная ставка и срок, что позволяет при оформлении депозита, выбрать наиболее приемлемый в настоящее время. Также стоит быть внимательным при вводе суммы, но в случае некорректного ввода появится сообщение с предупреждением (Рис 7.4).

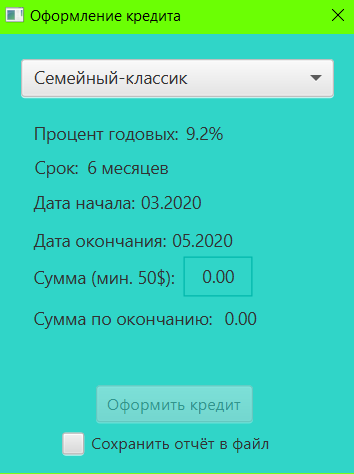


Рисунок 7.4 – Оформить кредит

Второе действие — это просмотреть все оформленные пользователем кредиты. В таблице отображается имя оформленного кредита, сумма, процентная ставка, срок кредита, и дата его оформления (Рис 7.5).



Рисунок 7.5 – Просмотр всех сделанных кредитов

Следующим действием является перевод денежных средств. Пользователь может перевести деньги на другой счет, минимальная сумма перевода составляет 5 долларов. Следует ввести номер счета, на который нужно перевести сумму (Рис 7.6).

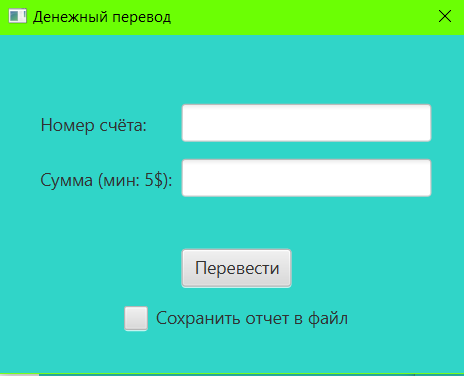


Рисунок 7.6 – Перевод денежных средств

Четвертым действием является просмотр истории денежных переводов. Пользователь может просматривать все денежные переводы, сделанные им за все время (Рис 7.7).



Рисунок 7.7 – История денежных переводов.

Так же пользователь может воспользоваться поиском по сделанным денежным переводов. Для этого необходимо ввести в окно поиска счёт пользователя (Рис 7.8).

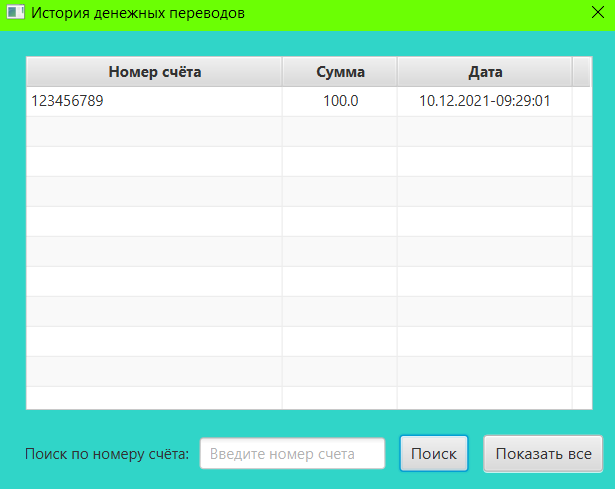


Рисунок 7.8 – Поиск денежных переводов

Так же в окне совершении кредита пользователь может выбрать любой вид кредита. И в зависимости от понравившегося ему, пользователь оставляет его и приступает к оформлению кредита. А также при желании пользователь может сохранить отчёт об оформленном кредите. Для этого необходимо лишь поставить галочку напротив фразы «Сохранить отчёт в файл», и информация об кредите будет сохранена в файле. При оформлении кредита можно ввести сумму. (Рис 7.9).

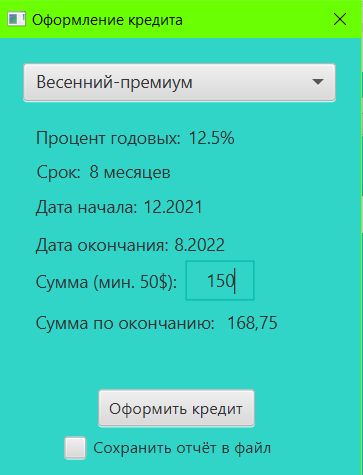


Рисунок 7.9- Ввод суммы кредита

Окно администратора имеет три действия — это просмотр всех пользователей, просмотр всех денежных переводов и просмотр всех кредитов. Просмотр всех денежных переводов содержит в себе таблицу, которая демонстрирует все переводы всех пользователей (Рис 7.10).

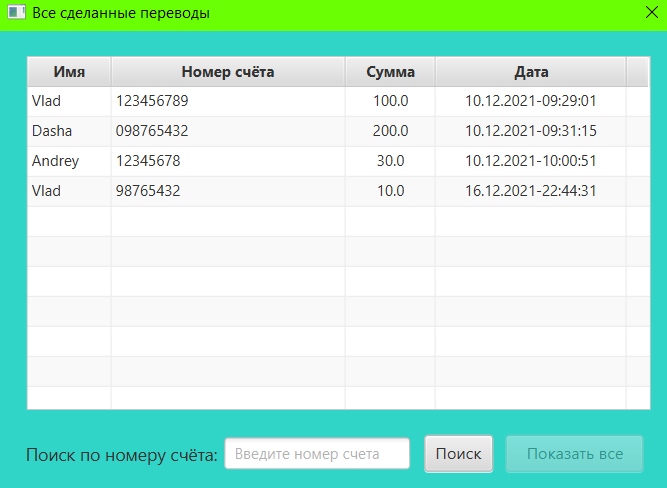


Рисунок 7.10 – Просмотр всех сделанных переводов

Также есть возможность поиска информации по номеру счёта, на который был сделан перевод (Рис.7.11).

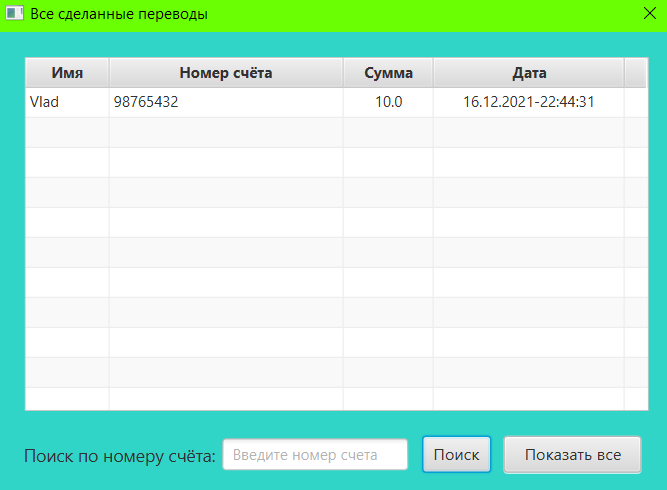


Рисунок 7.11 – Поиск денежных переводов по номеру счёта

Просмотр всех пользователей содержит в себе таблицу, которая демонстрирует всех пользователей (Рис 7.12).

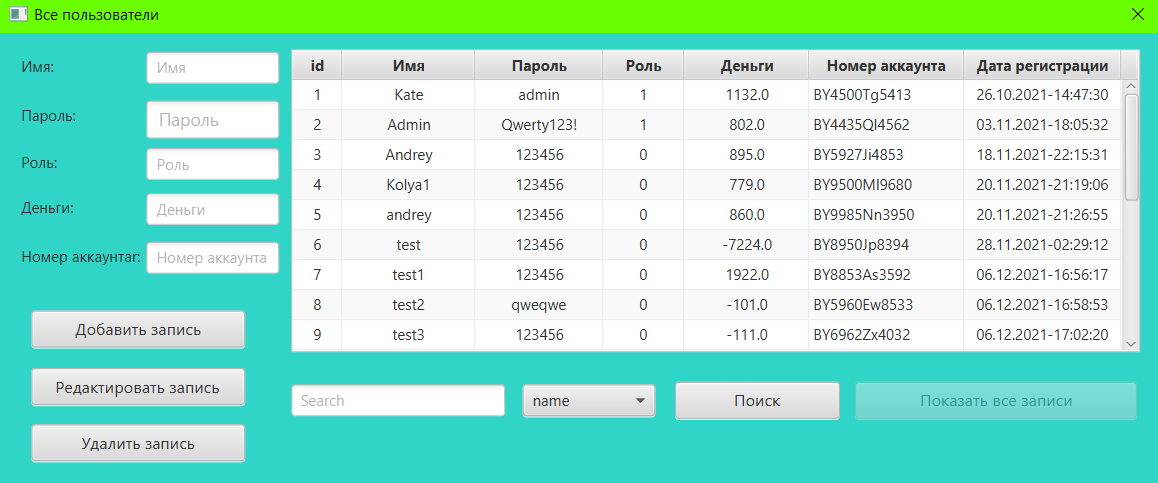


Рисунок 7.12 – Просмотр всех пользователей

В окне просмотра всех пользователей есть возможно добавить, удалить, изменить или найти пользователя. Для поиска необходимо написать в строке поиска слово, по которому хотим искать и выбрать из выпадающего списка то поле, по которому ищем (Рис 7.13).

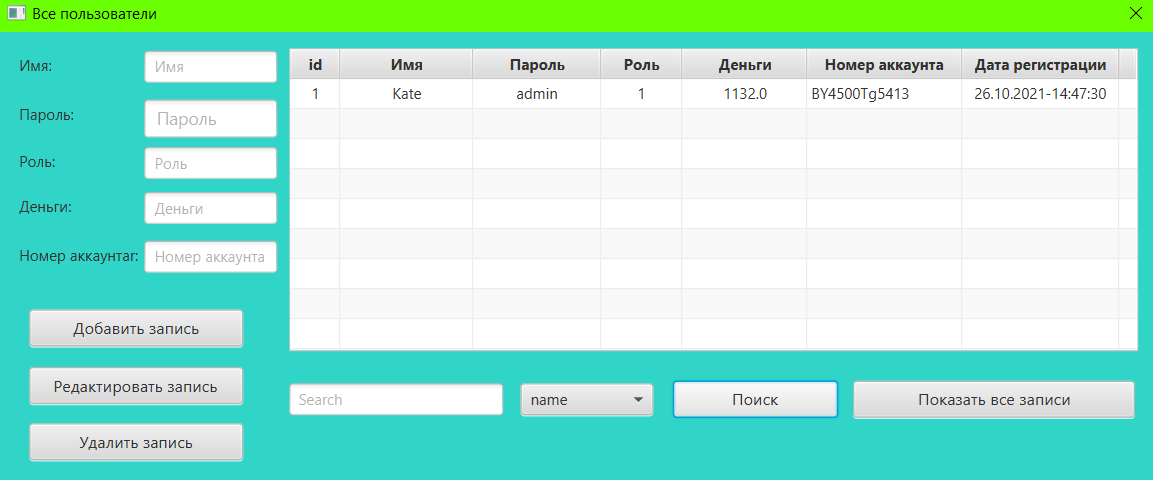


Рисунок 7.13 – Поиск пользователя по имени

Для изменения данных пользователя необходимо выбрать пользователя, чтобы его данные отобразились слева в полях. Затем выбирается желаемое, для изменения поле, и вводятся новые данные (Рис. 7.14).

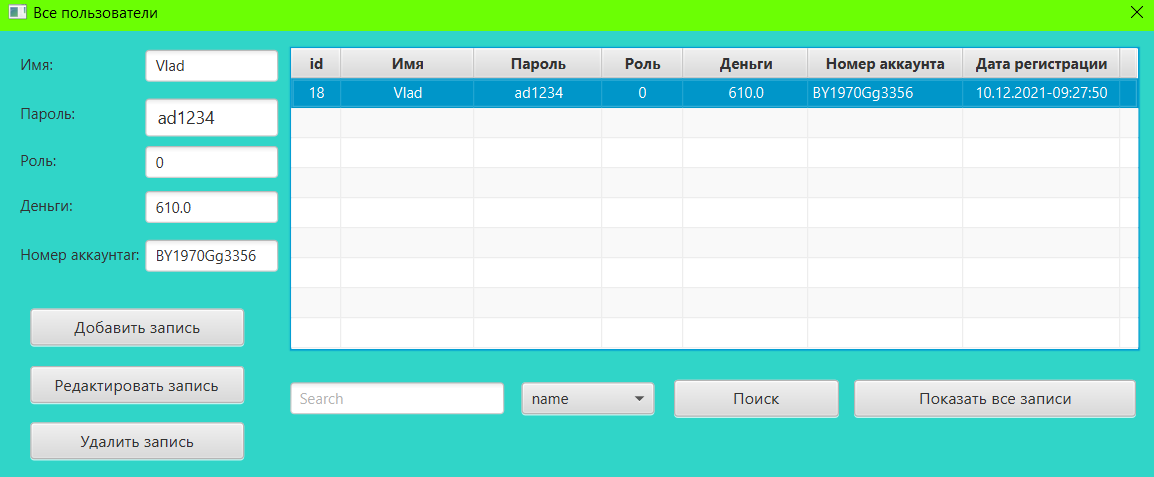


Рисунок 7.14 – Изменение пароля пользователя

Удаление пользователя происходит подобным образом, как и изменение, за исключением того, что не нужно менять данные. Просто выделяется необходимая строка в таблице и нажимается кнопка удаления (Рис. 7.15).

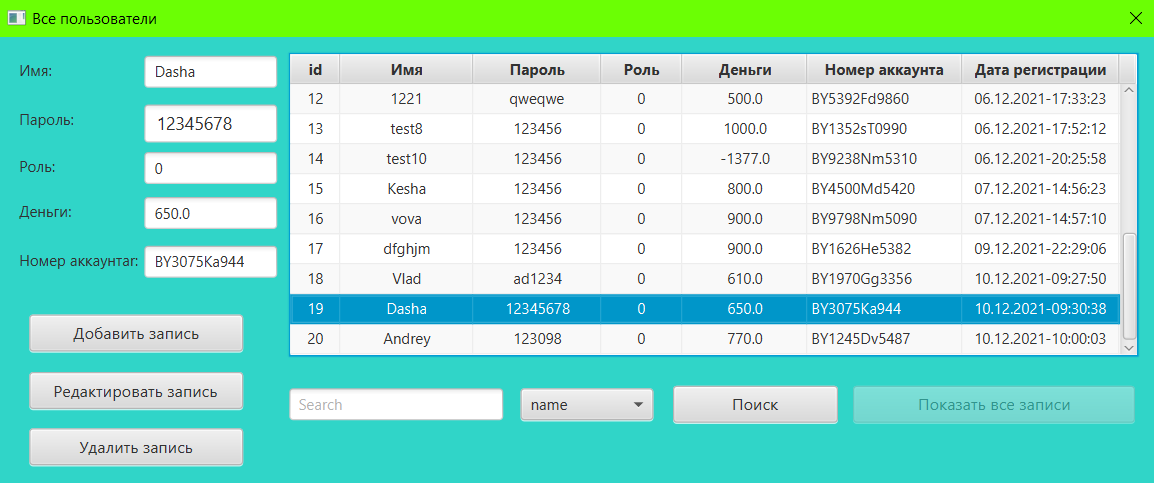


Рисунок 7.15 – Удаление записи

Для добавления пользователя необходимо сначала написать в пустые поля данные, а уже после этого нажать на кнопку добавления (Рис 7.16).

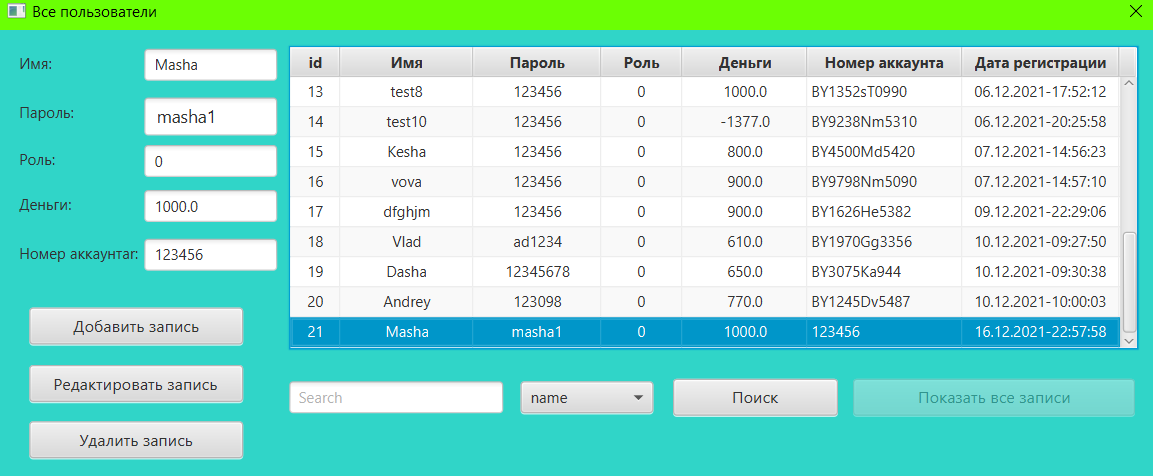


Рисунок 7.16 – Добавление записи

Просмотр всех кредитов содержит в себе таблицу, которая демонстрирует все оформленные кредиты всеми пользователями. Помимо таблицы можно увидеть круговую диаграмму и столбчатую диаграмму для более наглядного вида количества оформленных кредитов (Рис 7.17).



Рисунок 7.16 – Просмотр всех кредитов

Также есть возможность поиска данных по сумме. Для этого необходимо ввести в поле поиска сумму, по которой нужно ориентироваться при выборке данных. После указания необходимой суммы необходимо нажать кнопку поиска и на экран выведутся строки, сумма которых меньше или равна сумме, указанной вами в строке поиска (Рис. 7.17).



Рисунок 7.16 – Поиск кредитов до 100$

# **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ**

Для регистрации следует придумать имя и дважды повторить пароль. Пароль должен иметь больше пяти символов. Если же пароль слишком короткий регистрация не произойдет (Рис 8.1).

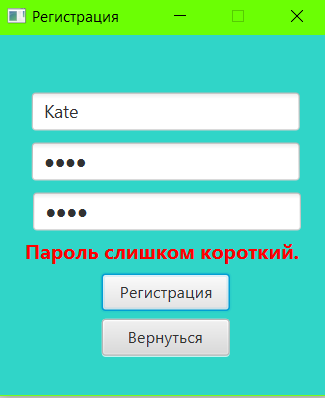


Рисунок 8.1 – Попытка регистрации с коротким паролем

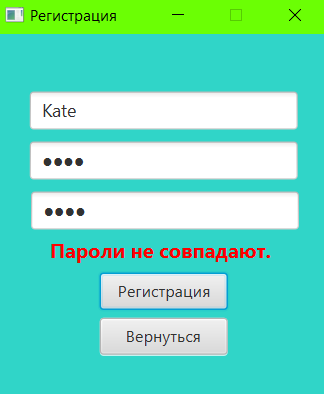


Рисунок 8.2 – Окно регистрации с разными паролями

Пользователю нужно придумать пароль и дважды повторить его. Если же пароли не эквивалентны пользователь не будет зарегистрирован (Рис 8.2).

Чтобы оформить кредит следует ввести сумму. Минимальная сумма оформления кредита является 50$. Если же сумма отрицательная или поле пустое или же сумма маленькая, то кнопка оформления будет недоступной и кредит нельзя будет оформить (Рис 8.3).

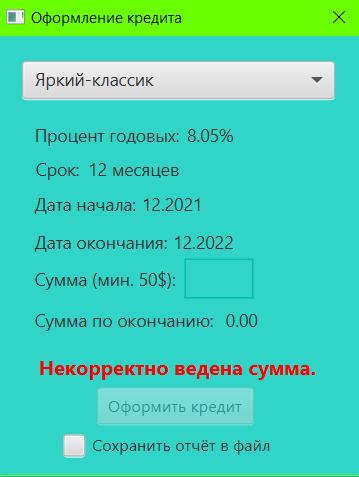


Рисунок 8.3 – Проверка корректности ведёной суммы

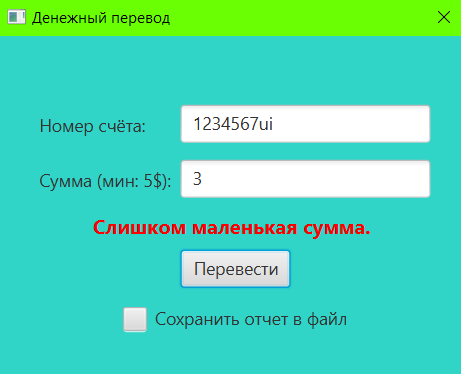


Рисунок 8.4 – Проверка суммы перевода

Так же в окне денежных переводов предусмотрена проверка на пустое поле и маленькую сумму. Минимальная сумма перевода 5$ (Рис 8.4).

При просмотре всех пользователей, может возникнуть необходимость поиска пользователя. Для корректной работы программы стоит проверка на заполненность поля поиска (Рис. 8.5).

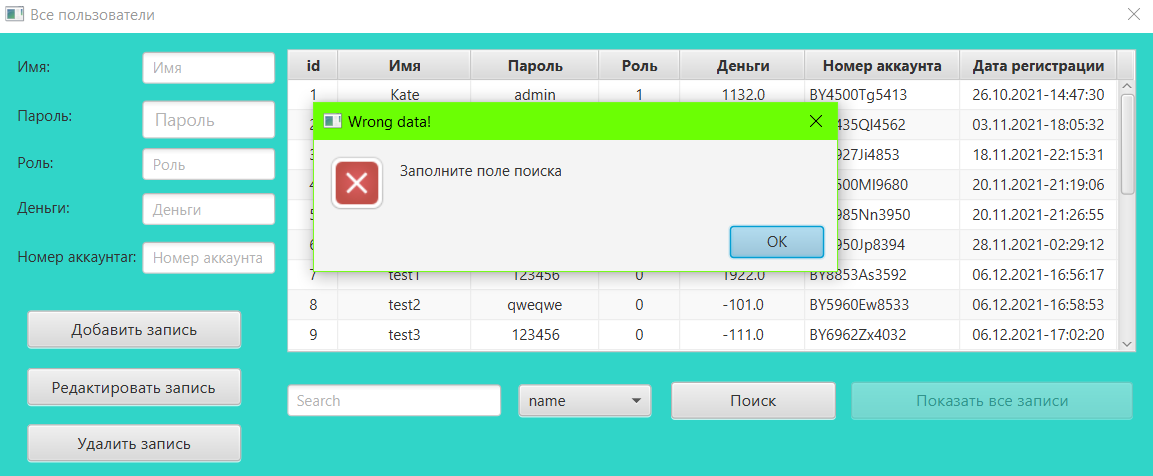


Рисунок 8.5 – Проверка поля поиска

# **ЗАКЛЮЧЕНИЕ**

В процессе разработки и написания работы были использованы знания объектно – ориентированного программирования полученные ранее, понимание работы клиент – серверных приложений и их взаимодействие, также знания были употреблены на практике, в ходе чего было допущено немало ошибок, вследствие чего были приобретены навыки по выявлению и устранению этих ошибок. Был получен опыт отображения данных в элементе TableView. Так же получен опыт разработки визуальных приложений с использованием технологии JavaFX. JavaFX позволяет разрабатывать визуальные desktop – приложения любой сложности. Получен опыт разработки визуальной части приложения и графического отображения каких-либо данных посредством таблиц, программы, которая имеет больше, чем одно окно и реализации их взаимодействия.

Также при написании проекта была изучена различная документация по использованию диаграмм, по процессу работы с данными из БД, по оформлению кредитов и денежных переводов. При изучении технологии посредством документации изучение тех или иных сфер очень продуктивно, так как помимо изучения тех или иных бизнес – процессов, при внедрении их в программный код, работа этих процессов становится понятна и наглядна, т.е. происходит большее понимание процесса, работающего во внешней среде.

Архитектура клиент – серверного приложения выстроена таким образом, чтобы её можно было расширять впоследствии без нанесения ошибок и корректировок в уже существующий проект, так называемая масштабируемость. Логическое визуальное представление интерфейса повышает скорость изучения системы для её применения на практике.

Для предотвращения возможных ошибок были реализованы методы, позволяющие их предотвратить и свести исправление ошибок администраторами системы к минимуму.

В ходе программы достигнута основная цель программы – разработать подсистему выдачи кредитов физическим лицам.

Итоговым результатом выполненной работы является созданное приложения, которое соответствует всем требованиям безопасности и которое можно применять в банковской сфере. Проект может быть полезен всем банкам, которые ещё не используют подобную систему. Также для клиентов была сделана удобная запись в файл, что позволит хранить полезную информацию об оформленных кредитах и контролировать их.

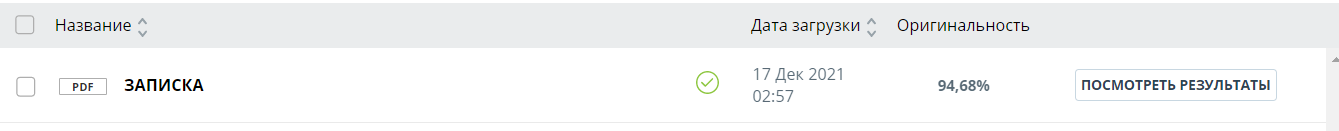
# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Зачем человечеству нужны банки [Электронный ресурс]. – Режим доступа: https://www.money-talks.ru/finance/2013/11/27/zachem-chelovechestvu-nuzhny-banki.html
2. Роль кредитов в жизни человека [Электронный ресурс]. – Режим доступа: https://mirden.ru/zachem-nuzhen-kredit/
3. Особенности транзакционного бизнеса [Электронный ресурс]. –  Режим доступа: http://www.banki.ru/news/bankpress/?id=5999131
4. Испытано на себе. Как сохранить и вырастить деньги? [Электронный ресурс]. –  Режим доступа: https://infobank.by/infolineview/ispytano-na-sebe-kak-soxranitj-i-vyrastitj-denjgi/
5. Что такое технология Java и каково ее применение? [Электронный ресурс]. – Режим доступа: https://java.com/ru/download/help/whatis\_java.html
6. Клиент – серверная Архитектура [Электронный ресурс]. – Режим  доступа: https://qastart.by/class-2/60-klient-servernaya-arkhitektura
7. Методология моделирования [Электронный ресурс]. –  Режим доступа: <https://www.uamconsult.com/book_443_chapter_18_3.3._IDEF_-_metodologija_modelirovanija.html>
8. Методология IDEF0 [Электронный ресурс]. – Режим доступа: https://itteach.ru/bpwin/metodologiya-idef0
9. Проектирование, разработка и сопровождение баз данных с использованием CASE – средств [Электронный ресурс]. – Режим доступа: https://www.bsuir.by/m/12\_100229\_1\_90142.pdf
10. Протокол TCP/IP моделирования [Электронный ресурс]. – Режим доступа: <http://chernykh.net/content/view/553/751/>

# **ПРИЛОЖЕНИЕ А**

# **(Обязательное)**

**Проверка на антиплагиат**

****

# **Приложение Б**

# **(обязательное)**

# **Листинг алгоритмов реализующих бизнес-логику**

package sample.classes;

public class ShowUsers {

private int id, role;

private String name, accountNum, password, regDate;

private double money;

public ShowUsers(int id, String name, String password, int role, double money, String accountNum, String regDate) {

this.id = id;

this.role = role;

this.name = name;

this.money = money;

this.accountNum = accountNum;

this.password = password;

this.regDate = regDate;

}

public int getId() {

return id;

}

public int getRole() {

return role;

}

public String getName() {

return name;

}

public String getAccountNum() {

return accountNum;

}

public String getPassword() {

return password;

}

public String getRegDate() {

return regDate;

}

public double getMoney() {

return money;

}

}

package sample;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Modality;

import javafx.stage.Stage;

import java.io.\*;

import java.net.Socket;

public class Main extends Application {

static private ObjectOutputStream coos;

static private ObjectInputStream cois;

static private Socket clientSocket;

@Override

public void start(Stage primaryStage) throws Exception{

Parent root = FXMLLoader.load(getClass().getResource("/sample/views/start.fxml"));

primaryStage.setTitle("Вход");

primaryStage.setResizable(false);

Scene scene = new Scene(root);

primaryStage.setScene(scene);

primaryStage.show();

root.requestFocus();

}

public static void main(String[] args) {

try {

System.out.println("Соединение с сервером..");

clientSocket = new Socket("127.0.0.1", 2528);

System.out.println("Соединение установлено....");

coos = new ObjectOutputStream(clientSocket.getOutputStream());

cois = new ObjectInputStream(clientSocket.getInputStream());

}catch (Exception e){

e.printStackTrace();

}

launch(args);

}

public void openNewWindow(Stage stage,boolean isCurrentClose,String path,String title,boolean isModal){

try {

if(isCurrentClose) stage.close();

FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(path));

Parent root1 = (Parent) fxmlLoader.load();

stage = new Stage();

if(isModal) stage.initModality(Modality.APPLICATION\_MODAL);

stage.setResizable(false);

stage.setTitle(title);

stage.setScene(new Scene(root1));

stage.show();

root1.requestFocus();

} catch (IOException e) {

e.printStackTrace();

}

}

public static String sendToServerString(String sendMessages) {

String receive = "";

try{

coos.writeObject(sendMessages);

receive = (String)cois.readObject();

}catch(IOException | ClassNotFoundException e){

e.printStackTrace();

}

return receive;

}

public static Object getObjectFromServer(String sendMessage){

Object obj = null;

try{

coos.writeObject(sendMessage);

obj = cois.readObject();

}catch (IOException | ClassNotFoundException e){

e.printStackTrace();

}

return obj;

}

}

# **Приложение В**

# **(обязательное)**

# **Листинг кода**

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.sql.\*;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Random;

public class DatabaseWork {

Connection dbConnection;

private double userCurrentSum = 0;

private double userAfterDepositSum = 0;

private int userId = 0;

private Statement statement;

String insertStr = "";

double percent = 0;

String userAccNumber = "";

public DatabaseWork() {

dbConnection();

}

public void dbConnection() {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

dbConnection = DriverManager.getConnection("jdbc:mysql://localhost/course?serverTimezone=GMT%2B8", "root", "root");

statement = dbConnection.createStatement();

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

}

}

public void getUserAccNumber(ObjectOutputStream soos){

try {

soos.writeObject(userAccNumber);

} catch (IOException e) {

e.printStackTrace();

}

}

public void getUserMoney(ObjectOutputStream soos){

try {

soos.writeObject(Double.toString(userCurrentSum));

} catch (IOException e) {

e.printStackTrace();

}

}

public void getUserMoneyWithDeposit(ObjectOutputStream soos){

try {

soos.writeObject(Double.toString(userCurrentSum + userAfterDepositSum));

} catch (IOException e) {

e.printStackTrace();

}

}

public void signUp(String[] action, ObjectOutputStream soos) {

ResultSet users = null;

try {

users = statement.executeQuery("SELECT \* FROM users");

String answer = "Error";

while (users.next()) {

if (users.getString("name").equals(action[1]) && users.getString("password").equals(action[2])) {

userId = Integer.parseInt(users.getString("id"));

userCurrentSum = Double.parseDouble(users.getString("money"));

userAccNumber = users.getString("account\_num");

if (users.getString("role").equals("0")) answer = "User";

else answer = "Admin";

}

}

users.close();

soos.writeObject(answer);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void showAllStatisticsDeposit(String[] arr, ObjectOutputStream soos) {

ArrayList<String> typeDepositList = new ArrayList<>();

ResultSet typeDeposit = null;

ArrayList<String> listForClient = new ArrayList<>();

int count = 0, tmp = 1;

try {

typeDeposit = getTypeDepositTable();

while (typeDeposit.next()) typeDepositList.add(typeDeposit.getString("name"));

typeDeposit.close();

ResultSet allDeposit = null;

for (String str : typeDepositList) {

allDeposit = statement.executeQuery("SELECT \* FROM deposit WHERE deposit\_id = " + quotate(Integer.toString(tmp)));

while (allDeposit.next()) count++;

listForClient.add(str + " " + Integer.toString(count));

count = 0;

tmp++;

}

allDeposit.close();

soos.writeObject(listForClient);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void showAllTransferMoney(String[] tmp, ObjectOutputStream soos) {

ArrayList<String> list = new ArrayList<>();

ResultSet allTransferMoney = null;

try {

ArrayList<String> listUsers = new ArrayList<>();

ResultSet users = getUserFromTable();

while (users.next()) listUsers.add(users.getString("name"));

users.close();

if (tmp.length == 2)

allTransferMoney = statement.executeQuery(Query.TRANSFER\_MONEY\_WHERE\_ACOOUNT\_NUM + quotate(tmp[1]));

else allTransferMoney = getTransferMoneyFromTable();

while (allTransferMoney.next()) {

list.add(listUsers.get(Integer.parseInt(allTransferMoney.getString("sender\_id")) - 1) + " " +

allTransferMoney.getString("receive\_num") + " " +

allTransferMoney.getString("date") + " " +

allTransferMoney.getString("sum"));

}

allTransferMoney.close();

soos.writeObject(list);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void showAllUsers(String[] tmp, ObjectOutputStream soos) {

try {

ArrayList<String> listUsers = new ArrayList<>();

ResultSet allUsers = getUserFromTable();

while (allUsers.next()) listUsers.add(allUsers.getString("id") + " " + allUsers.getString("name") + " " + allUsers.getString("password") + " " + allUsers.getString("role") + " " +

allUsers.getString("money") + " " + allUsers.getString("account\_num") + " " + allUsers.getString("reg\_date"));

allUsers.close();

soos.writeObject(listUsers);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void deleteUser(String[] tmp, ObjectOutputStream soos) {

try {

insertStr = "DELETE FROM `deposit` WHERE `user\_id` = (" + quotate(tmp[1]) + ")";

statement.executeUpdate(insertStr);

insertStr = "DELETE FROM `transfer\_money` WHERE `sender\_id` = (" + quotate(tmp[1]) + ")";

statement.executeUpdate(insertStr);

insertStr = "DELETE FROM `users` WHERE `id` = (" + quotate(tmp[1]) + ")";

statement.executeUpdate(insertStr);

soos.writeObject("Success");

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void editUser(String[] tmp, ObjectOutputStream soos) {

try {

insertStr = "UPDATE `users` SET "

+ " name = " + quotate(tmp[2]) + ","

+ " password = " + quotate(tmp[3]) + ","

+ " role = " + quotate(tmp[4]) + ","

+ " money = " + quotate(tmp[5]) + ","

+ " account\_num = " + quotate(tmp[6])

+ " WHERE id = " + quotate((tmp[1]));

statement.executeUpdate(insertStr);

soos.writeObject("Success");

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void searchUser(String[] tmp, ObjectOutputStream soos) {

try {

ArrayList<String> listUsers = new ArrayList<>();

ResultSet allUsers = getUserFromTable();

while (allUsers.next()) {

if (allUsers.getString(tmp[1]).equals(tmp[2]))

listUsers.add(allUsers.getString("id") + " " + allUsers.getString("name") + " " + allUsers.getString("password") + " " + allUsers.getString("role") + " " +

allUsers.getString("money") + " " + allUsers.getString("account\_num") + " " + allUsers.getString("reg\_date"));

System.out.println(allUsers.getString(tmp[1]) + " " + tmp[2]);

}

allUsers.close();

soos.writeObject(listUsers);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void showAllDeposit(String[] tmp, ObjectOutputStream soos) {

ArrayList<String> list = new ArrayList<>();

ArrayList<String> listDepositType = new ArrayList<>();

try {

ResultSet allDepositType = getTypeDepositTable();

while (allDepositType.next())

listDepositType.add(allDepositType.getString("name") + " " + allDepositType.getString("percent") + " " + allDepositType.getString("term"));

allDepositType.close();

ArrayList<String> listDepositUsers = new ArrayList<>();

ResultSet allUsers = getUserFromTable();

while (allUsers.next()) listDepositUsers.add(allUsers.getString("name"));

allUsers.close();

ResultSet allDeposit = null;

if (tmp.length == 2) allDeposit = statement.executeQuery(Query.DEPOSIT\_WHERE\_SUM\_LESS + tmp[1]);

else allDeposit = getDepositFromTable();

String[] strTmp = {""};

while (allDeposit.next()) {

strTmp = listDepositType.get(Integer.parseInt(allDeposit.getString("deposit\_id")) - 1).split(" ");

list.add(listDepositUsers.get(Integer.parseInt(allDeposit.getString("user\_id")) - 1) + " " +

allDeposit.getString("sum") + " " +

strTmp[0] + " " +

strTmp[1] + " " +

strTmp[2] + " " +

allDeposit.getString("date"));

}

allDeposit.close();

soos.writeObject(list);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void showMyDeposit(String[] tmp, ObjectOutputStream soos) {

ArrayList<String> typeDepositList = new ArrayList<>();

try {

ResultSet depositType = getTypeDepositTable();

while (depositType.next())

typeDepositList.add(depositType.getString("name") +

" " + depositType.getString("percent") +

" " + depositType.getString("term"));

depositType.close();

ResultSet myDeposit = statement.executeQuery(Query.DEPOSIT\_WHERE\_USER\_ID + quotate(Integer.toString(userId)));

ArrayList<String> myDepositList = new ArrayList<>();

while (myDeposit.next())

myDepositList.add(typeDepositList.get(Integer.parseInt(myDeposit.getString("deposit\_id")) - 1) +

" " + myDeposit.getString("sum") + " " + myDeposit.getString("date"));

myDeposit.close();

soos.writeObject(myDepositList);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void registration(String[] tmp, ObjectOutputStream soos) {

int countId = 1;

try {

ResultSet users = statement.executeQuery("SELECT \* FROM users ");

while (users.next()) countId++;

users.close();

userId = countId;

Random rnd = new Random();

char smallLet = (char) (rnd.nextInt(26) + 'a');

char bigLet = (char) (rnd.nextInt(26) + 'A');

userAccNumber = "BY" + (rnd.nextInt((10000 - 0) + 1) + 0) + bigLet + smallLet + (rnd.nextInt((10000 - 0) + 1) + 0);

if (tmp.length == 3) {

insertStr = "INSERT INTO users VALUES ("

+ quotate(Integer.toString(countId)) + ","

+ quotate(tmp[1]) + ","

+ quotate(tmp[2]) + ","

+ quotate("0") + ","

+ quotate(Double.toString(1000)) + ","

+ quotate(userAccNumber) + ","

+ quotate(getDate()) + ")";

statement.executeUpdate(insertStr);

userCurrentSum = 1000;

}else{

insertStr = "INSERT INTO users VALUES ("

+ quotate(Integer.toString(countId)) + ","

+ quotate(tmp[1]) + ","

+ quotate(tmp[2]) + ","

+ quotate(tmp[3]) + ","

+ quotate(tmp[4]) + ","

+ quotate(tmp[5]) + ","

+ quotate(getDate()) + ")";

statement.executeUpdate(insertStr);

userCurrentSum = Double.parseDouble(tmp[4]);

}

soos.writeObject("SUCCESS");

} catch (IOException | SQLException e) {

e.printStackTrace();

}

}

public void showTransferMoney(String[] stringFromClient, ObjectOutputStream soos) {

ArrayList<String> list = new ArrayList<>();

try {

ResultSet users = getUserFromTable();

ArrayList<String> usersList = new ArrayList();

while (users.next())

usersList.add(users.getString("account\_num"));

users.close();

ResultSet transferMoney = statement.executeQuery(Query.TRANSFER\_MONEY\_WHERE\_SENDER\_ID + quotate(Integer.toString(userId)));

if (stringFromClient.length == 1)

while (transferMoney.next())

list.add(transferMoney.getString("receive\_num") + " " + transferMoney.getString("sum") + " " +

transferMoney.getString("date"));

else

while (transferMoney.next())

if (stringFromClient[1].equals(transferMoney.getString("receive\_num")))

list.add(transferMoney.getString("receive\_num") + " " + transferMoney.getString("sum") + " " +

transferMoney.getString("date"));

transferMoney.close();

soos.writeObject(list);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void makeDeposit(String[] stringFromClient, ObjectOutputStream soos) {

try {

ResultSet tmp = getDepositFromTable();

String deposit\_id = "";

int countId = 1;

while (tmp.next()) countId++;

tmp.close();

ResultSet deposit\_type = getTypeDepositTable();

while (deposit\_type.next()) {

if (deposit\_type.getString("name").equals(stringFromClient[1])) {

deposit\_id = deposit\_type.getString("id");

percent = Double.parseDouble(deposit\_type.getString("percent"));

}

}

deposit\_type.close();

insertStr = "INSERT INTO deposit VALUES ("

+ quotate(Integer.toString(countId)) + ","

+ quotate(Integer.toString(userId)) + ","

+ quotate(deposit\_id) + ","

+ quotate(getDate()) + ","

+ quotate(stringFromClient[2]) + ")";

statement.executeUpdate(insertStr);

insertStr = "UPDATE `users` SET "

+ " money = " + quotate(Double.toString(userCurrentSum - Double.parseDouble(stringFromClient[2])))

+ " WHERE (" + "id = " + quotate(Integer.toString(userId)) + ")";

userCurrentSum -= Double.parseDouble(stringFromClient[2]);

statement.executeUpdate(insertStr);

userAfterDepositSum = (Double.parseDouble(stringFromClient[2]) + (Double.parseDouble(stringFromClient[2]) \* percent / 100));

if (stringFromClient.length == 5)

saveToFile("\nВы оформили депозит " + stringFromClient[1] + " на " + stringFromClient[3] + " месяцев " +

"сумой " + stringFromClient[2] + "$",soos);

soos.writeObject("");

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

public void getTypeDeposit(ObjectOutputStream soos) {

ArrayList<String> list = new ArrayList<>();

try {

ResultSet tmp = getTypeDepositTable();

while (tmp.next()) {

list.add(tmp.getString("name") + " " + tmp.getString("percent") + " " +

tmp.getString("term"));

}

tmp.close();

soos.writeObject(list);

} catch (SQLException | IOException e) {

e.printStackTrace();

}

}

private void saveToFile(String string,ObjectOutputStream soos){

try (FileOutputStream fos = new FileOutputStream("otchyot.txt", true)) {

byte[] buffer = string.getBytes();

fos.write(buffer, 0, buffer.length);

} catch (IOException ex) {

ex.printStackTrace();

}

int countId = 1;

try {

ResultSet saveFile = statement.executeQuery("SELECT \* FROM save\_to\_file ");

while (saveFile.next()) countId++;

saveFile.close();

userId = countId;

insertStr = "INSERT INTO save\_to\_file VALUES ("

+ quotate(Integer.toString(countId)) + ","

+ quotate(Integer.toString(userId)) + ","

+ quotate(string) + ","

+ quotate(getDate()) + ")";

statement.executeUpdate(insertStr);

soos.writeObject("SUCCESS");

} catch (IOException | SQLException e) {

e.printStackTrace();

}

}

public void makeTransferMoney(String[] action, ObjectOutputStream soos){

double sum = Double.parseDouble(action[2]);

String numberAcc = action[1];

try {

String receiver\_num = getReceiverUserId(numberAcc);

int idNewCountry = getIdNewCountryTransferMoney();

double otherCurrentSum = 0;

ResultSet tmp = getUserFromTable();

if (!receiver\_num.equals("NOT FOUND"))

while (tmp.next()) {

if (tmp.getString("id").equals(receiver\_num))

otherCurrentSum = Integer.parseInt(tmp.getString("money"));

}

insertStr = "INSERT INTO transfer\_money VALUES ("

+ quotate(Integer.toString(idNewCountry)) + ","

+ quotate(Integer.toString(userId)) + ","

+ quotate(numberAcc) + ","

+ quotate(Double.toString(sum)) + ","

+ quotate(getDate()) + ")";

statement.executeUpdate(insertStr);

insertStr = "UPDATE `users` SET "

+ " money = " + quotate(Double.toString(userCurrentSum - sum))

+ " WHERE (" + "id = " + quotate(Integer.toString(userId)) + ")";

statement.executeUpdate(insertStr);

userCurrentSum -= sum;

if (!receiver\_num.equals("NOT FOUND")) {

insertStr = "UPDATE `users` SET "

+ " money = " + quotate(Double.toString(sum + otherCurrentSum))

+ " WHERE (" + "id = " + quotate(receiver\_num) + ")";

statement.executeUpdate(insertStr);

}

if (action.length == 4) saveToFile("\nВы перевели на счёт " + numberAcc + " " + sum + "$ " + getDate(),soos);

soos.writeObject(receiver\_num);

}

catch (SQLException | IOException e){

e.printStackTrace();

}

}

private int getIdNewCountryTransferMoney(){

int count = 1;

try {

ResultSet transferMoney = getTransferMoneyFromTable();

while (transferMoney.next()) count++;

}

catch (SQLException e){

e.printStackTrace();

}

return count;

}

private String getReceiverUserId(String acc){

String answer = "";

try {

ResultSet users = getUserFromTable();

while (users.next()) {

if (users.getString("account\_num").equals(acc)) {

answer = users.getString("id");

}

}

users.close();

if (answer.equals("")) answer = "NOT FOUND";

}catch (SQLException e){

e.printStackTrace();

}

return answer;

}

private ResultSet getTypeDepositTable() throws SQLException {

return statement.executeQuery(Query.SHOW\_TYPE\_DEPOSIT);

}

private ResultSet getDepositFromTable() throws SQLException {

return statement.executeQuery(Query.SHOW\_ALL\_DEPOSIT);

}

private ResultSet getTransferMoneyFromTable() throws SQLException {

return statement.executeQuery(Query.SHOW\_TRANSFER\_MONEY);

}

private ResultSet getUserFromTable() throws SQLException {

return statement.executeQuery(Query.SHOW\_ALL\_USERS);

}

private static String quotate(String content) {

return "'" + content + "'";

}

private static String getDate() {

return new SimpleDateFormat("dd.MM.yyyy-HH:mm:ss").format(Calendar.getInstance().getTime());

}

}

**ПРИЛОЖЕНИЕ Г**

**Ведомость курсового проекта**

**(обязательное)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Перв. примен.* | ГУИР. ГУИР.506413.032 Д1 | *Зона* | *Обозначение* | | | | | *Наименование* | | | *Дополнительные, сведения* | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | | *Текстовые документы* | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  | *ГУИР КП 1–40 05 01 032 ПЗ* | | | | | *Пояснительная записка* | | | *55 с.* | | | |
|  |  | | | | |  | | |  | | | |
| *Справочный №* |  |  |  | | | | | *Графические документы* | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  | *ГУИР.506413.032 Д1* | | | | | *Схема алгоритма* | | | *Формат А3* | | | |
|  |  | | | | |  | | |  | | | |
|  | *ГУИР.506413.032 Д2* | | | | | *IDEF0-модель процессов предметной области* | | | *Формат А4* | | | |
|  |  | | | | |  | | |  | | | |
|  | *ГУИР.506413.032 Д3* | | | | | *Диаграммы* | | | *Формат А4, 2 л.* | | | |
|  | |  |  | | | | |  | | |  | | | |
|  | *ГУИР.506413.021 Д4* | | | | | *Пользовательский графический интерфейс* | | | *Формат А4* | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
| *ПоКПись и* |  |  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
| *Инв. №* |  |  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
| *Взам. Инв. №* |  |  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
| *ПоКПись и* |  |  |  | | | | |  | | |  | | | |
|  |  | | | | |  | | |  | | | |
|  |  |  |  |  | *ГУИР КП 1-40 05 01 032 Д1* | | | | | | | |
|  |  |  |  |  |
| *Изм.* | *Лист* | *№ докум.* | *Подп.* | *Дата* |
| *Инв. № подл.* |  | *Разраб.* | | *Жилинская* |  |  | *Банковское кредитование и автоматизация подсистемы выдачи кредитов физическим лицам*  *Ведомость курсового проекта* | | *Лит.* | | | | *Лист* | *Листов* |
| *Пров.* | | *Унучек* |  |  |  | *Т* | |  | *55* | *55* |
|  | |  |  |  | *Кафедра ПИКС, группа 914301* | | | | | |
| *Н.контр.* | | *Унучек* |  |  |
| *Утв.* | | *Хорошко* |  |  |