

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

Инженерно-экономический факультет
Кафедра экономической информатики
Дисциплина: Программирование сетевых приложений

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
Ассистент кафедры ЭИ
_____._____.2024 Н.А. Шилов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:
**«РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ БАНКОВСКИМИ
ВКЛАДАМИ ФИЗИЧЕСКИХ ЛИЦ»**

БГУИР КР 1-40 05 01-02 013 ПЗ

Выполнил студент группы 273601
Малько Игнат Валерьевич

(подпись студента)
Курсовая работа представлена на
проверку _____._____.2024

(подпись студента)

Минск 2024

РЕФЕРАТ

БГУИР КР 11-40 05 01-02 013 ПЗ

Малько, И.В. Разработка системы управления банковскими вкладами физических лиц/И.В. Малько. – Минск: БГУИР, 2024. – 50 с.

Пояснительная записка 50 с., 44 рис., 11 источников, 3 приложения

УПРАВЛЕНИЕ БАНКОВСКИМИ ВКЛАДАМИ ФИЗИЧЕСКИХ ЛИЦ,
МОДЕЛИ UML, IDEF0, ПРОГРАММНОЕ СРЕДСТВО

Целью курсовой работы является совершенствование процесса управления банковскими вкладами физических лиц. Программа позволяет вносить корректировки в уже имеющиеся данные, создавать новые записи, удалять старые и сохранять отредактированные записи.

Объект исследования – процесс организации взаимодействия между пользователем и системой.

Предмет исследования – методы и способы организации взаимодействия между пользователем и системой.

Методология проведения работы: в процессе разработки системы использованы современные подходы к обработке данных, функциональный анализ процессов, моделирование системы с помощью UML-диаграмм.

Результаты работы: рассмотрены основные бизнес-процессы предметной области. Выполнена постановка задачи и определены основные методы ее решения; в ходе моделирования системы построен ряд UML-диаграмм; разработаны модели бизнес-процессов предметной области на основе нотаций IDEF-0; описаны основные алгоритмы работы программного средства; разработано руководство пользователя; выполнено тестирование программного средства, показавшее его соответствие функциональным требованиям, поставленным в задании на разработку.

СОДЕРЖАНИЕ

Введение.....	4
1 Анализ и моделирование предметной области программного средства.....	5
1.1 Описание предметной области.....	5
1.2 Разработка функциональной модели предметной области.....	8
1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований.....	13
1.4 Разработка информационной модели предметной области.....	15
1.5 UML-модели представления программного средства и их описание.	16
2 Проектирование и конструирование программного средства	17
2.1 Постановка задачи	17
2.2 Архитектурные решения	17
2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства	19
2.4 Проектирование пользовательского интерфейса	21
2.5 Обоснование выбора компонентов и технологий для реализации программного средства	22
3 Тестирование и проверка работоспособности программного средства	24
4 Руководство пользователя приложения.....	26
4.1 Инструкция по развертыванию приложения	26
4.2 Сквозной тестовый пример	26
Заключение	39
Список использованных источников	40
Приложение А	41
Приложение Б.....	42
Приложение В.....	49

ВВЕДЕНИЕ

В условиях быстроразвивающейся экономики и роста финансовой грамотности населения важным элементом работы банковских учреждений становится эффективное управление вкладами физических лиц. Банковские вклады — это не только способ сохранения и приумножения капитала, но и важный инструмент финансового планирования как для клиентов, так и для самих банков. С учетом высокой конкуренции на рынке финансовых услуг, банки стремятся предложить своим клиентам современные, удобные и безопасные инструменты управления их средствами.

Разработка системы управления банковскими вкладами физических лиц является актуальной задачей, поскольку от ее эффективности зависит не только уровень обслуживания клиентов, но и финансовая устойчивость самого банка. Современные технологии позволяют автоматизировать процессы, связанные с открытием, закрытием и управлением вкладами, что существенно сокращает временные затраты, минимизирует риски ошибок и повышает уровень безопасности операций.

Целью данной курсовой работы является создание приложения для системы управления банковскими вкладами физических лиц, которая будет учитывать потребности клиентов и требования законодательства, а также позволит банкам эффективно управлять своими ресурсами. В рамках работы будет проведен анализ существующих решений на рынке, выявлены их сильные и слабые стороны, а также разработаны рекомендации по внедрению инновационных функций, способствующих улучшению клиентского опыта.

Работа состоит из нескольких ключевых разделов. В первом разделе будет представлен обзор существующих систем управления вкладами, а также рассмотрены современные тенденции в этой области. Второй раздел будет посвящен анализу потребностей клиентов и специфики работы банков в условиях конкуренции. Третий раздел включает в себя проектирование архитектуры системы, выбор технологий и инструментов, а также описание функционала, который будет реализован в системе.

Таким образом, разработка системы управления банковскими вкладами физических лиц представляет собой важный шаг к улучшению финансовых услуг, что, в свою очередь, может способствовать повышению уровня доверия клиентов к банкам и их лояльности. В условиях постоянных изменений на финансовых рынках и растущей потребности населения в качественном обслуживании, создание такой системы становится не просто целесообразным, но и необходимым шагом для достижения конкурентных преимуществ.

1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА

1.1 Описание предметной области

В современном финансовом мире управление банковскими вкладами физических лиц представляет собой ключевую область, способствующую как стабильности банковской системы, так и финансовому благополучию клиентов. Банковские вклады – это средства, которые клиенты размещают в банках с целью сохранения и приумножения капитала. Вкладчики ожидают от банков надежности, привлекательных условий, а также качественного обслуживания. Существует несколько типов вкладов, включая срочные, бессрочные, накопительные и специальные предложения, что требует от банков гибкости и адаптивности в управлении этими продуктами.

Развитие технологий и цифровизация финансового сектора значительно изменили подходы к управлению вкладами. В современных условиях банки вынуждены внедрять автоматизированные системы, которые упрощают процесс открытия, изменения и закрытия вкладов. Клиенты ожидают, что смогут управлять своими вкладами через интернет-банкинг, мобильные приложения и другие цифровые каналы. Это требует от банков не только предоставления новых сервисов, но и обеспечения их безопасности, поскольку финансовые операции подвержены рискам мошенничества и кибератак.

Система управления банковскими вкладами должна учитывать разнообразие клиентских потребностей. Разные категории клиентов могут иметь различные ожидания и предпочтения: одни ищут высокие процентные ставки, другие – гибкость условий или возможность досрочного снятия средств без потери процентов. Важно, чтобы банковская система могла быстро адаптироваться к изменениям на рынке и реагировать на спрос клиентов, предлагая актуальные продукты. Это может быть достигнуто путем интеграции аналитических инструментов, позволяющих анализировать поведение клиентов и прогнозировать их потребности [11].

Одной из основных задач автоматизированной системы является минимизация временных затрат при обработке операций. Важно, чтобы клиенты могли быстро открывать вклады, проверять состояние своих счетов и получать информацию о возможных условиях. Это требует создания интуитивно понятного интерфейса и внедрения технологий, которые позволят обеспечить оперативность и высокую степень автоматизации. Кроме того, система должна обеспечивать надежное хранение данных о клиентах и их вкладах, что требует соблюдения строгих стандартов безопасности и конфиденциальности.

Объект исследования: управление банковскими вкладами физических лиц в современных условиях цифровизации финансовой сферы.

Предмет исследования: автоматизированные системы управления банковскими вкладами физических лиц, их архитектура, функциональные

возможности и методы обеспечения эффективности, безопасности и удобства использования.

Цель исследования: разработка требований и концепции автоматизированной системы управления банковскими вкладами физических лиц, обеспечивающей высокий уровень безопасности, удобства и функциональности для удовлетворения потребностей клиентов и повышения конкурентоспособности банка.

Задачи исследования:

1. Проанализировать существующие системы управления банковскими вкладами, их сильные и слабые стороны.

2. Определить основные потребности и ожидания клиентов в сфере управления вкладами.

3. Исследовать современные технологии, применяемые в автоматизации банковских операций, включая безопасность и интеграцию с другими продуктами.

4. Разработать концепцию и функциональные требования к системе управления банковскими вкладами, включая:

- обработку операций (открытие, изменение и закрытие вкладов);
- обеспечение конфиденциальности и защиты данных клиентов;
- удобство интерфейса и скорость выполнения операций.

5. Предложить механизмы интеграции системы управления вкладами с другими банковскими продуктами (кредитами, инвестициями и страхованием).

6. Разработать рекомендации по обучению и повышению квалификации персонала для работы с автоматизированной системой.

7. Оценить перспективы внедрения функций, учитывающих экологические и социальные аспекты в управлении вкладами, как конкурентного преимущества.

На сегодняшний день существует множество банковских систем, которые уже успешно функционируют на рынке, однако многие из них не способны полностью удовлетворить потребности клиентов или имеют устаревшие технологии. Таким образом, разработка новой системы управления банковскими вкладами физических лиц представляет собой актуальную задачу. При этом необходимо учитывать не только технологические, но и юридические аспекты, так как работа с личными данными клиентов регулируется законами о защите информации и финансовыми нормативами.

Важным аспектом является также взаимодействие с другими банковскими продуктами и услугами. Система управления вкладами должна интегрироваться с кредитными, инвестиционными и страховыми продуктами, чтобы предлагать клиентам комплексное решение для управления их финансами. Это создаст дополнительные возможности для кросс-продаж и увеличит лояльность клиентов. Например, вкладчики могут быть

заинтересованы в получении кредитов под залог своих вкладов, что требует четкой интеграции между системами.

Кроме того, необходимо учитывать растущий интерес к экологически устойчивым инвестициям. Современные клиенты всё чаще обращают внимание на то, как их деньги используются и какие проекты финансируются. Поэтому внедрение функций, позволяющих отслеживать экологические и социальные аспекты, может стать конкурентным преимуществом для банков, стремящихся привлечь новую аудиторию.

В заключение, управление банковскими вкладами физических лиц – это сложная и многоаспектная задача, требующая современного подхода и использования инновационных технологий. Эффективная система управления должна быть гибкой, безопасной и интуитивно понятной, обеспечивая высокий уровень обслуживания клиентов и удовлетворяя их разнообразные потребности. Разработка такой системы не только улучшит качество услуг банка, но и повысит его конкурентоспособность на рынке, что в условиях постоянных изменений является ключевым фактором для успешного функционирования.

А также, необходимо отметить важность постоянного обучения и повышения квалификации персонала банка, работающего с клиентами и управлением вкладами. В условиях быстрого технологического прогресса и изменений в законодательстве сотрудники должны быть осведомлены о новых инструментах, процедурах и трендах в банковской сфере. Регулярные тренинги и семинары помогут не только повысить уровень обслуживания, но и укрепить доверие клиентов к банку. Обученный персонал сможет более эффективно решать возникающие вопросы и предлагать клиентам оптимальные решения, что в конечном итоге повысит общую удовлетворенность клиентов и станет залогом долгосрочных отношений. Инвестиции в обучение сотрудников – это не только улучшение качества обслуживания, но и стратегический шаг к повышению конкурентоспособности банка на рынке финансовых услуг.

1.2 Разработка функциональной модели предметной области

IDEF0 – метод функционального моделирования, а также графическая нотация, которая используется для описания и формализации бизнес-процессов. Особенность IDEF0 заключается в том, что эта методология ориентирована на соподчиненность объектов.

Функциональная модель IDEF0 – это блоки, каждый из которых имеет несколько входов и выходов. В каждом блоке есть управление и механизмы, детализирующиеся до необходимого уровня. В левом верхнем углу расположена самая важная функция. Она соединяется с остальными стрелками и описаниями функциональных блоков. У каждой стрелки или активности есть свое значение. Благодаря этому такая модель используется для описания любых административных и организационных процессов [1].

Типы стрелок:

- входящими ставятся задачи (данные о пользователях, данные о текущих процессах);
- исходящими выводят результат деятельности (готовая система);
- управляющие (стрелки сверху вниз) – это механизмы управления (стратегия банка, требования безопасности);
- механизмы (стрелки снизу вверх) используются для проведения необходимых работ (команда разработки, программные технологии) [2].

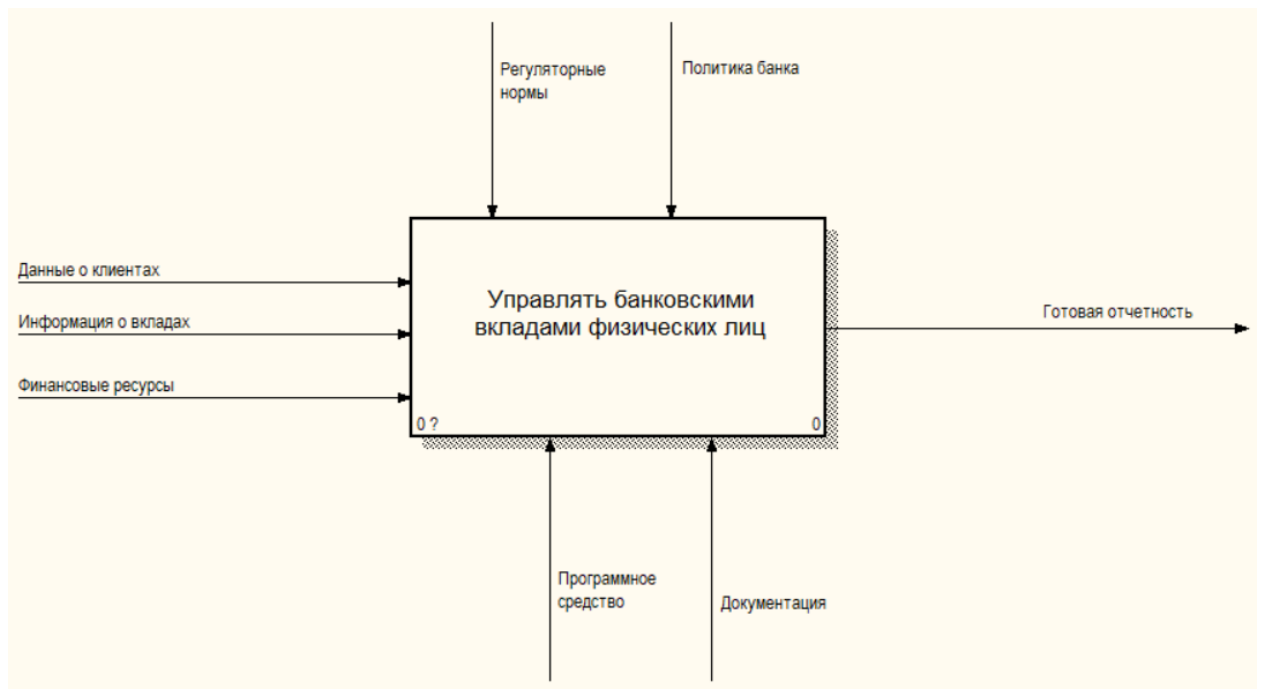


Рисунок 1.1 – Контекстная диаграмма модели уровня A0

Центральным элементом контекстной диаграммы IDEF0 (рис. 1.1) является функция, на схеме отображенная в виде функционального блока –

прямоугольника, внутри которого указано действие «Разработать систему управления банковскими вкладами физических лиц».

Диаграмма декомпозиции (рис. 1.2) детализирует отдельные элементы системы и связи между ними. Процедуру декомпозиции можно повторять до тех пор, пока не будет достигнут желаемый уровень детализации модели.

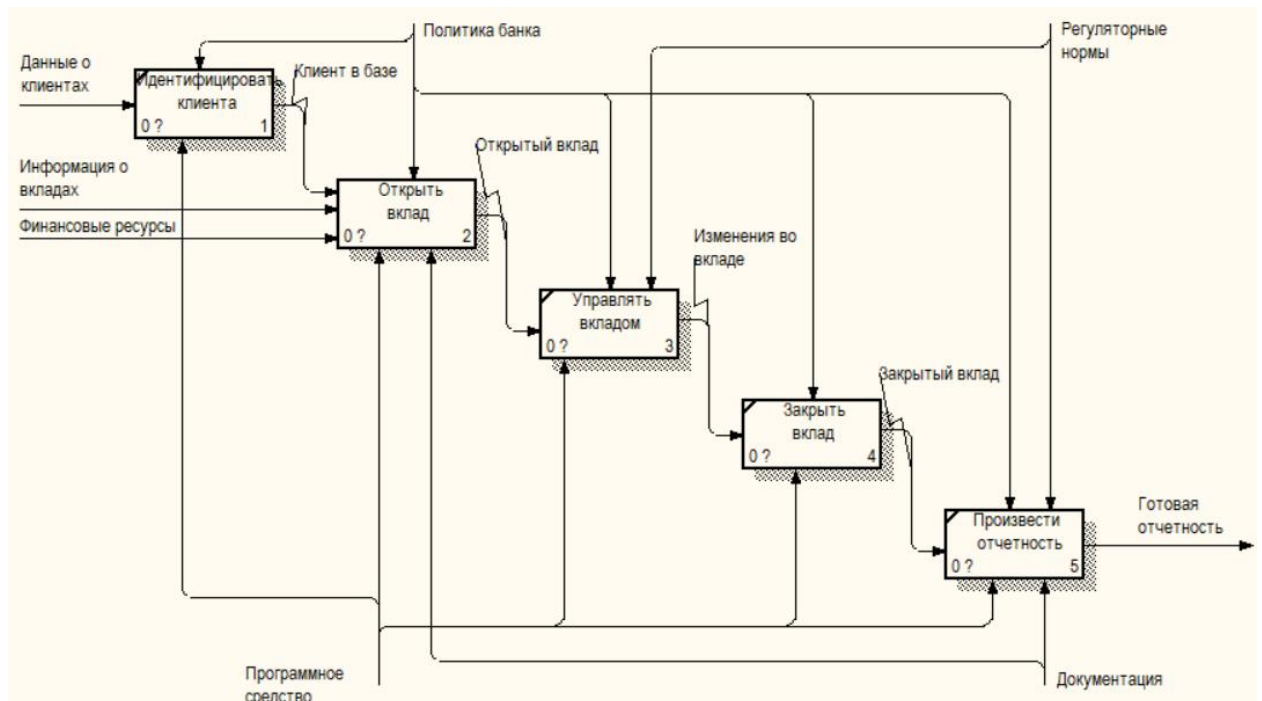


Рис. 1.2 – Диаграмма декомпозиции первого уровня А0

На данной диаграмме декомпозиции имеем уже пять процессов:

- идентифицировать клиента;
- открыть вклад;
- управлять вкладом;
- закрыть вклад;
- произвести отчетность.

Для более детального изучения процессов, декомпозируем процесс «Открыть вклад» (рис. 1.3).

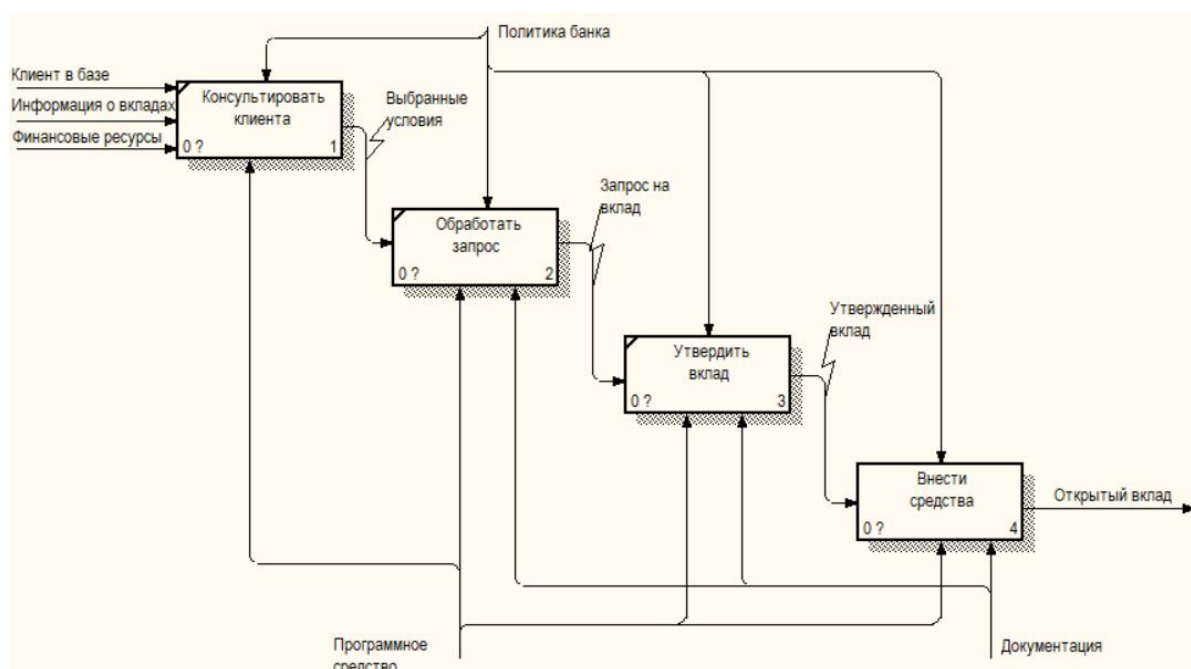


Рис. 1.3 – Диаграмма декомпозиции второго уровня A2 процесса «Открыть вклад»

Получилась диаграмма декомпозиции второго уровня с четырьмя функциональными блоками:

- консультировать клиента;
- обработать запрос;
- утвердить вклад;
- внести средства.

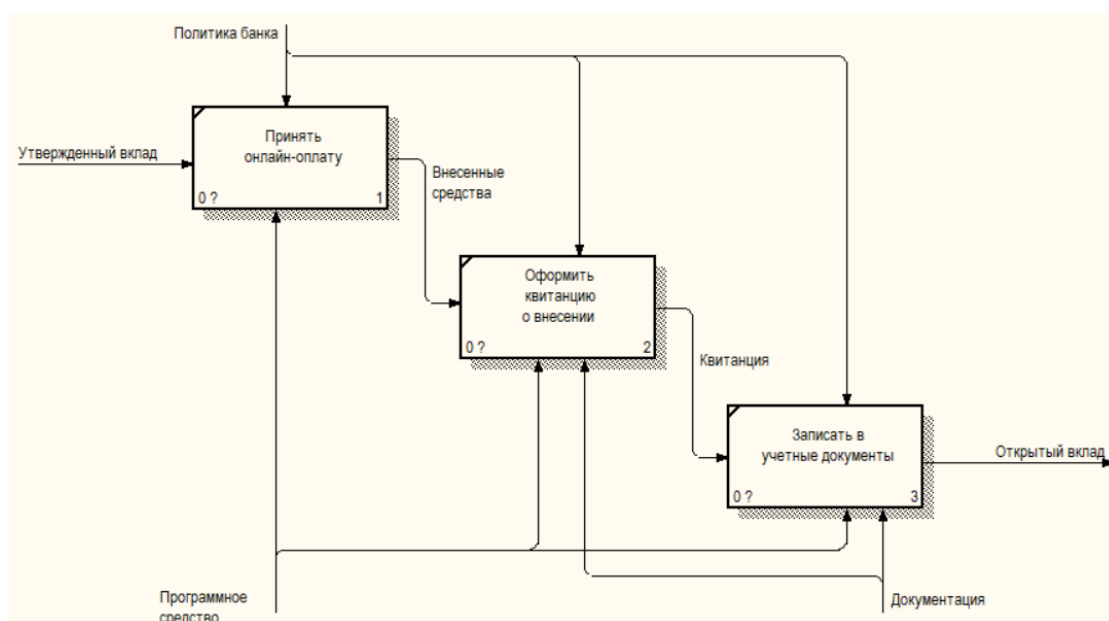


Рис. 1.4 – Диаграмма декомпозиции третьего уровня A24 процесса «Внести средства»

Далее, для более четкого понимания процессов, декомпозировали процесс «Внести средства» (рис. 1.4), получилась диаграмма декомпозиции третьего уровня с тремя функциональными блоками:

- принять онлайн-оплату;
- оформить квитанцию о внесении;
- записать в учетные документы.

Для детального понимания различий между процессами до и после внедрения ПО, рассмотрим также диаграмму собственно до внедрения (рисунки 1.5-1.8).

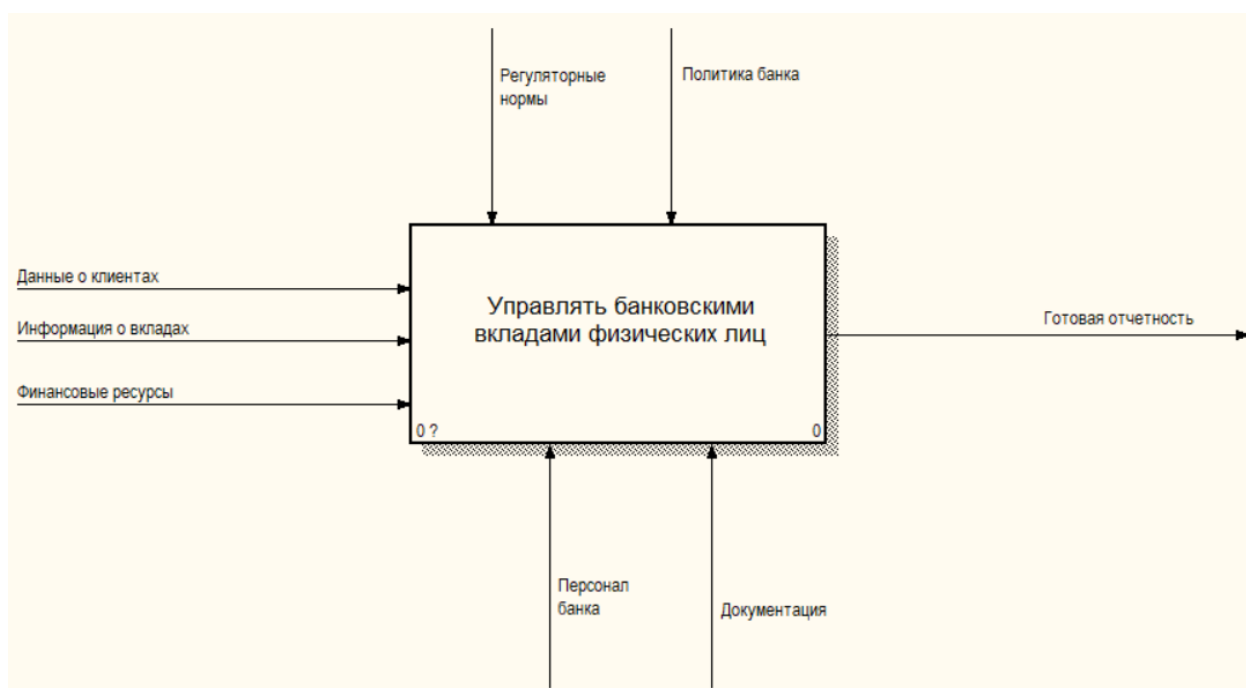


Рисунок 1.5 – Контекстная диаграмма модели уровня А0

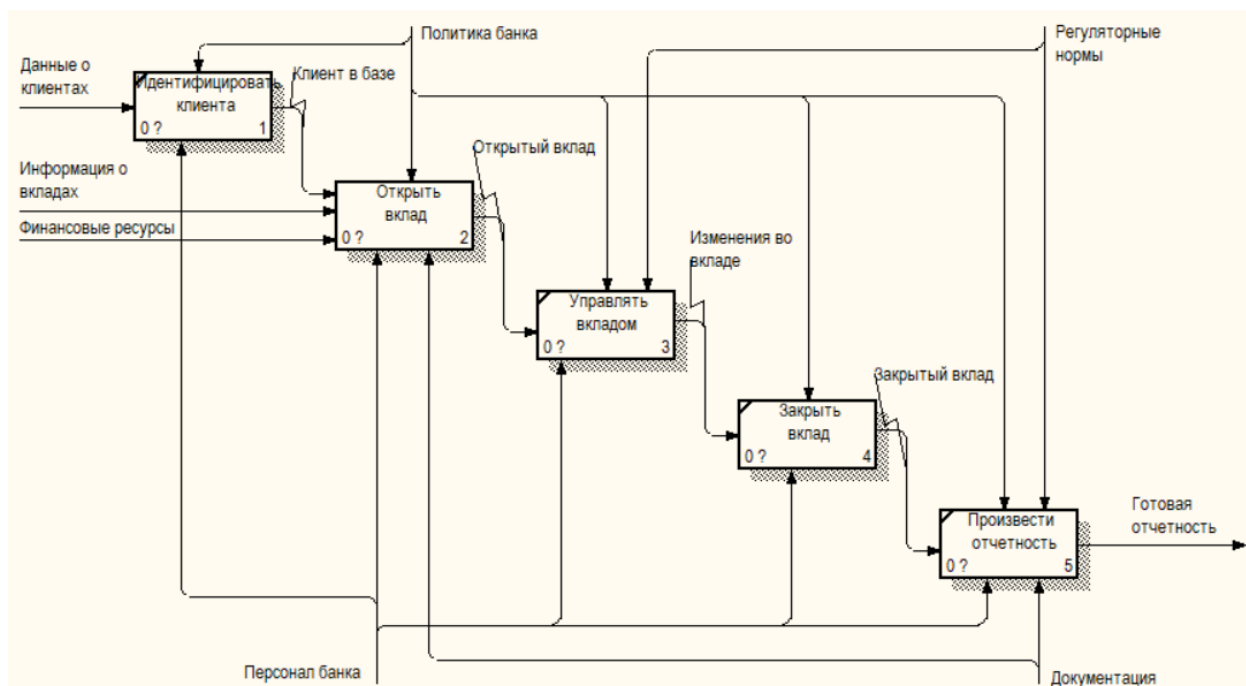


Рис. 1.6 – Диаграмма декомпозиции первого уровня A0

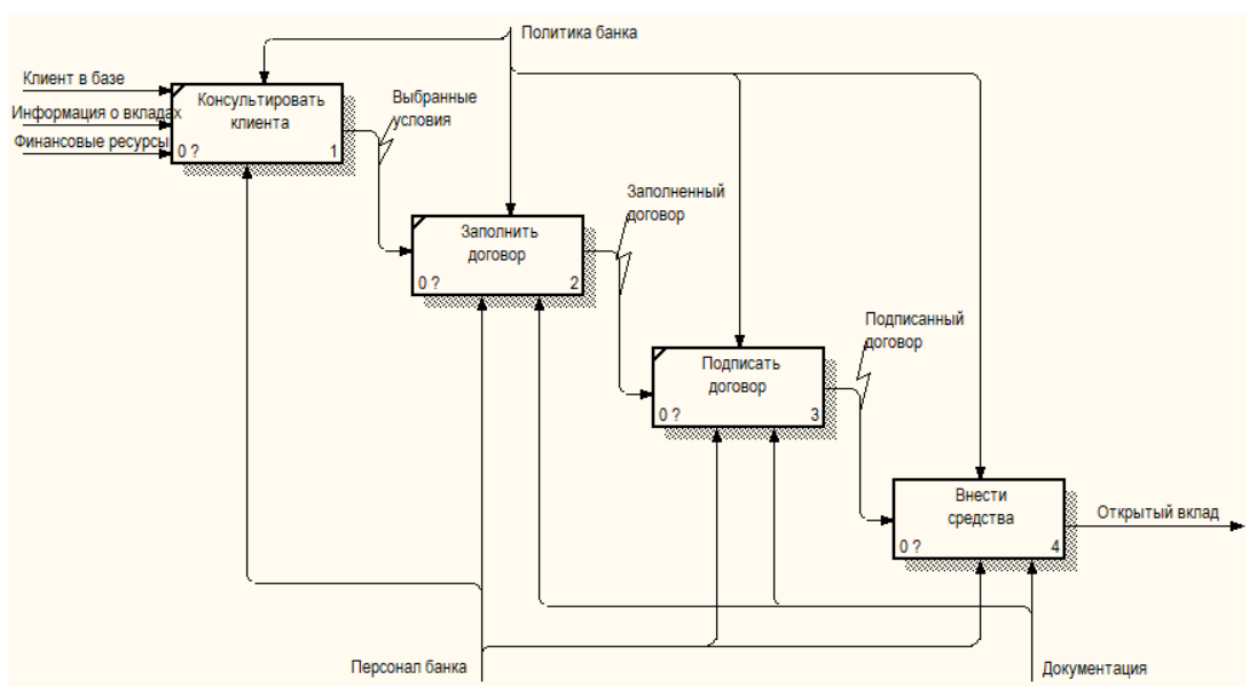


Рис. 1.7 – Диаграмма декомпозиции второго уровня A2 процесса «Открыть вклад»

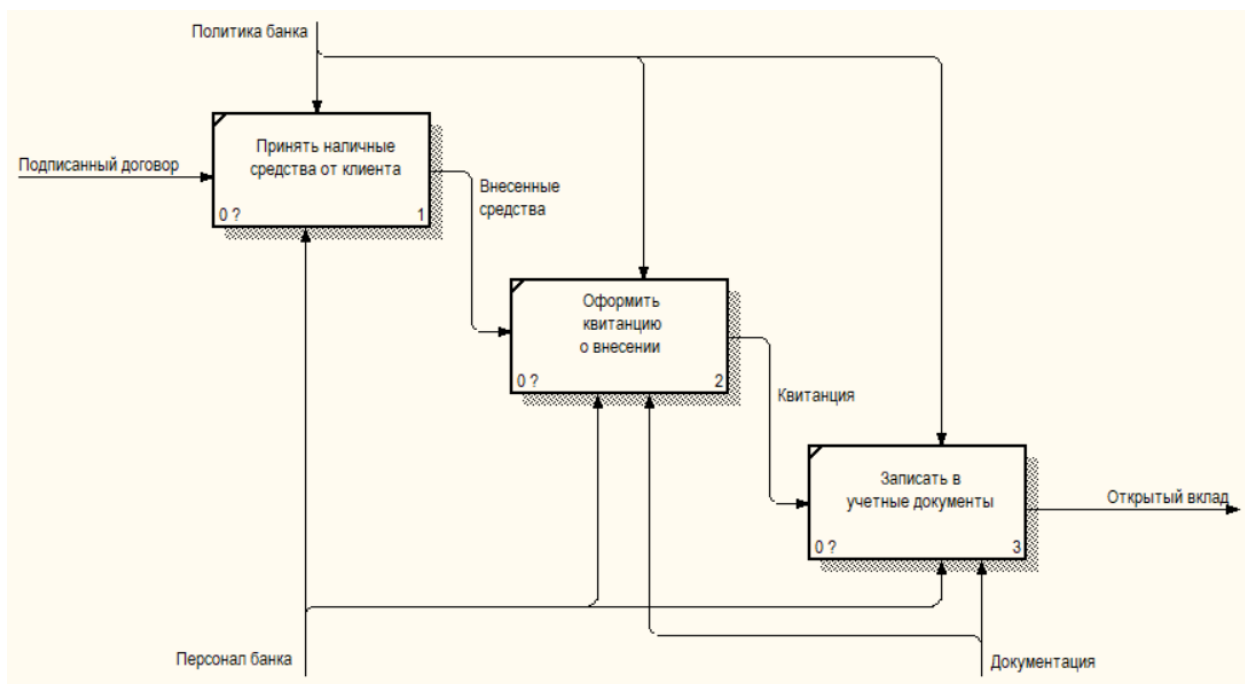


Рис. 1.8 – Диаграмма декомпозиции третьего уровня A24 процесса «Внести средства»

Благодаря рассмотрению процесса до и после внедрения ПО, мы более углубленно понимаем суть составления системы управления банковскими вкладами физических лиц.

1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Система управления банковскими вкладами физических лиц предназначена для автоматизации процессов открытия, ведения и закрытия вкладов, а также управления клиентскими данными. Основные цели разработки включают повышение эффективности работы банка, улучшение пользовательского опыта клиентов и обеспечение безопасности финансовых операций.

Анализ функциональных требований:

- управление клиентами: регистрация новых клиентов с проверкой личных данных; изменение личной информации; просмотр истории операций клиента;
- управление вкладами: открытие новых вкладов с выбором условий; закрытие вкладов с расчетом начисленных процентов; пополнение и снятие средств с вкладов; автоматическое продление вкладов по истечении срока;
- уведомления и отчеты: уведомление клиентов о важных событиях; генерация отчетов по движению средств по вкладам;

– финансовые операции: обработка операций пополнения и снятия средств с вкладов; ведение учета всех транзакций с возможностью фильтрации по дате и типу операции;

– управление пользователями: регистрация и аутентификация сотрудников банка; ведение истории действий пользователей в системе.

Для более детального понимания функциональных требований используется UML-диаграмма вариантов использования (Use Case) [3].

На рисунке 1.9 представлена диаграмма вариантов использования. Она демонстрирует, какой функционал разрабатываемого программного продукта будет доступен различным группам пользователей. Данная диаграмма является поведенческой диаграммой.

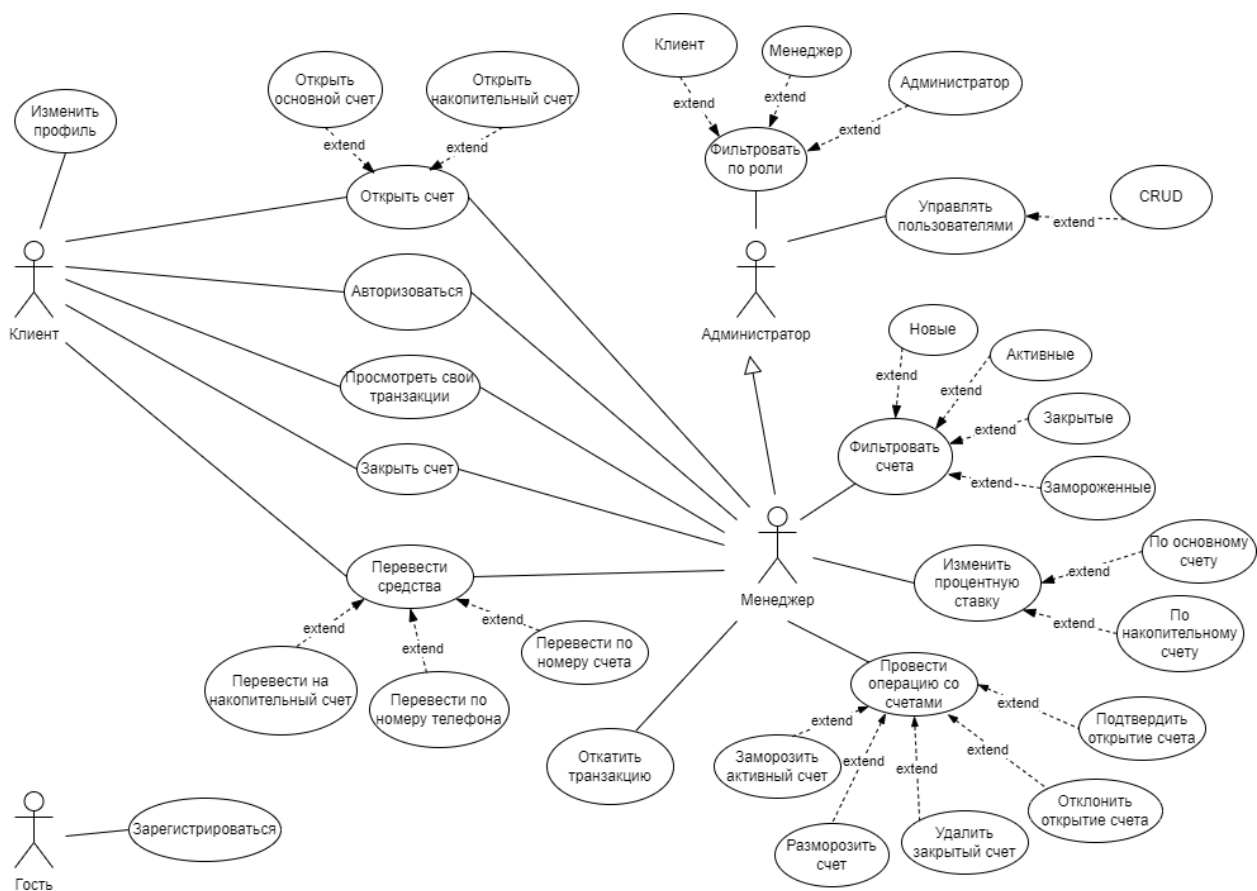


Рисунок 1.9 – Диаграмма вариантов использования

Из диаграммы видно, что пользователь обладает рядом возможностей: открытие вклада, закрытие вклада, просмотр истории операций по вкладу и др.

Если в систему входит гость, то ему необходимо зарегистрироваться, чтобы получить ряд возможностей пользователя.

В отличие от пользователя, администратор не имеет возможности снять средства с вклада, но имеет право добавлять и удалять аккаунты, изменять условия вклада и др. Все это функции, непосредственно, связаны с базой данных товаров.

1.4 Разработка информационной модели предметной области

Информационная модель представляет собой абстрактное представление данных, используемых в системе или приложении, и их отношений друг с другом. Она описывает структуру данных и способы их организации, представления и обработки (рисунок 1.10).

В контексте разработки программного обеспечения информационная модель определяет, как данные будут организованы и использованы в приложении. Она помогает проектировщикам и разработчикам понять, какие данные будут храниться, как они будут связаны друг с другом и каким образом они будут доступны для пользователей или других систем.



Рисунок 1.10 – Информационная модель предметной области

Информационная модель представляет собой абстрактное представление данных, используемых в системе или приложении, и их отношений друг с другом. Она описывает структуру данных и способы их организации, представления и обработки [5].

В контексте разработки программного обеспечения информационная модель определяет, как данные будут организованы и использованы в приложении. Она помогает проектировщикам и разработчикам понять, какие данные будут храниться, как они будут связаны друг с другом и каким образом они будут доступны для пользователей или других систем.

1.5 UML-модели представления программного средства и их описание

UML – это способ наглядно описать архитектуру, проектирование и реализацию комплексных программных систем. Код типичного приложения включает в себя тысячи строк, за связями и иерархиями которых очень не просто уследить. С помощью диаграмм UML структуру программы можно разделить на компоненты и подкомпоненты [4].

UML-диаграмма последовательности (Sequence Diagram) используется для моделирования взаимодействия между объектами системы во времени (рисунок 1.11). Она отображает порядок обмена сообщениями и взаимодействия участников (актеров или объектов) для выполнения определенного процесса или сценария [6].

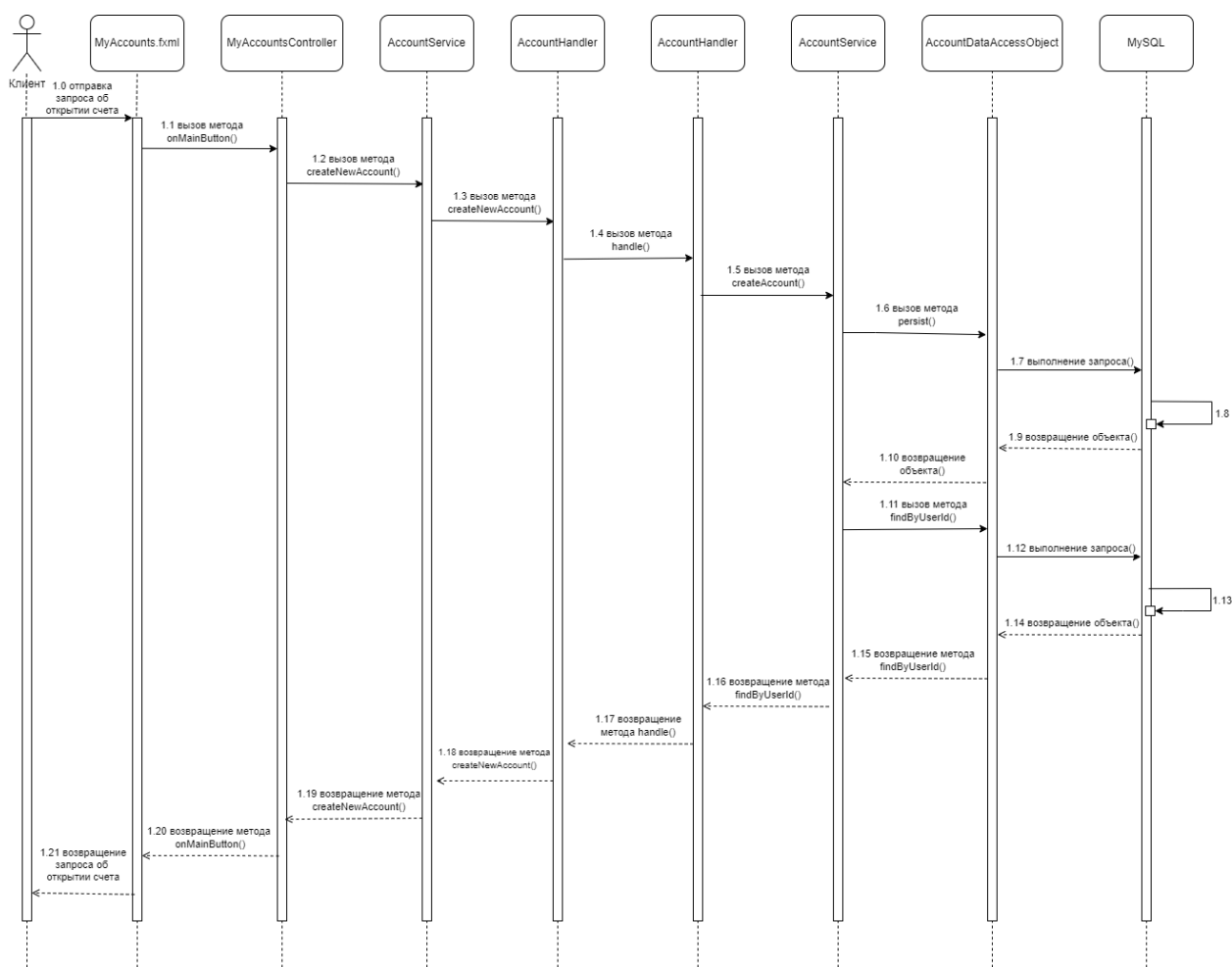


Рисунок 1.11 – Диаграмма последовательности процесса «Открытие счета»

Диаграмма последовательности представляет сценарий или поток событий в одном единственном случае использования. Поток сообщений диаграммы последовательностей основан на описании конкретного случая использования.

2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Постановка задачи

Общие требования. Приложение должно быть выполнено в архитектуре клиент-сервер с многопоточным сервером с организацией взаимодействия с базой данных на объектно-ориентированном языке Java.

Уровни архитектуры: Серверное приложение может быть реализовано в виде консольного приложения или GUI-приложения. Настройки сервера должны меняться без изменения исходного кода (аргументы командной строки, конфиг-файлы и т.д.). Клиентское приложение: оконное приложение с использованием стандартных библиотек пользовательского интерфейса (JavaFX, SWING, AWT).

В рамках работы над курсовым проектом должны быть использованы следующие техники:

- 1) разработка и использование собственной иерархии классов (не менее 5), расширение базовых классов, предоставляемых JDK 8 и выше;
- 2) реализация не менее 2-х паттернов проектирования на свой выбор (по желанию);
- 3) использовать сокрытие данных (инкапсуляция), перегрузку методов, переопределение методов, сериализацию, абстрактные типы данных (интерфейсы, абстрактные классы), статические методы, обработку исключительных ситуаций.

Конкретные версии фреймворков и технологий, применяемых для реализации программного средства, должны быть актуальными на начало 2024 года.

2.2 Архитектурные решения

В данном проекте, в соответствии с требованиями, необходимо сделать клиент-серверное приложение. Реализована слоистая архитектура приложения.

На стороне сервера есть слой сервисов, слой обработчиков, слой репозитория и слой моделей.

А на стороне клиента – слой контроллеров, слой сервисов, слой моделей и слой обработчиков.

На рисунках 2.1-2.2 изображены диаграммы классов нескольких слоев приложения.

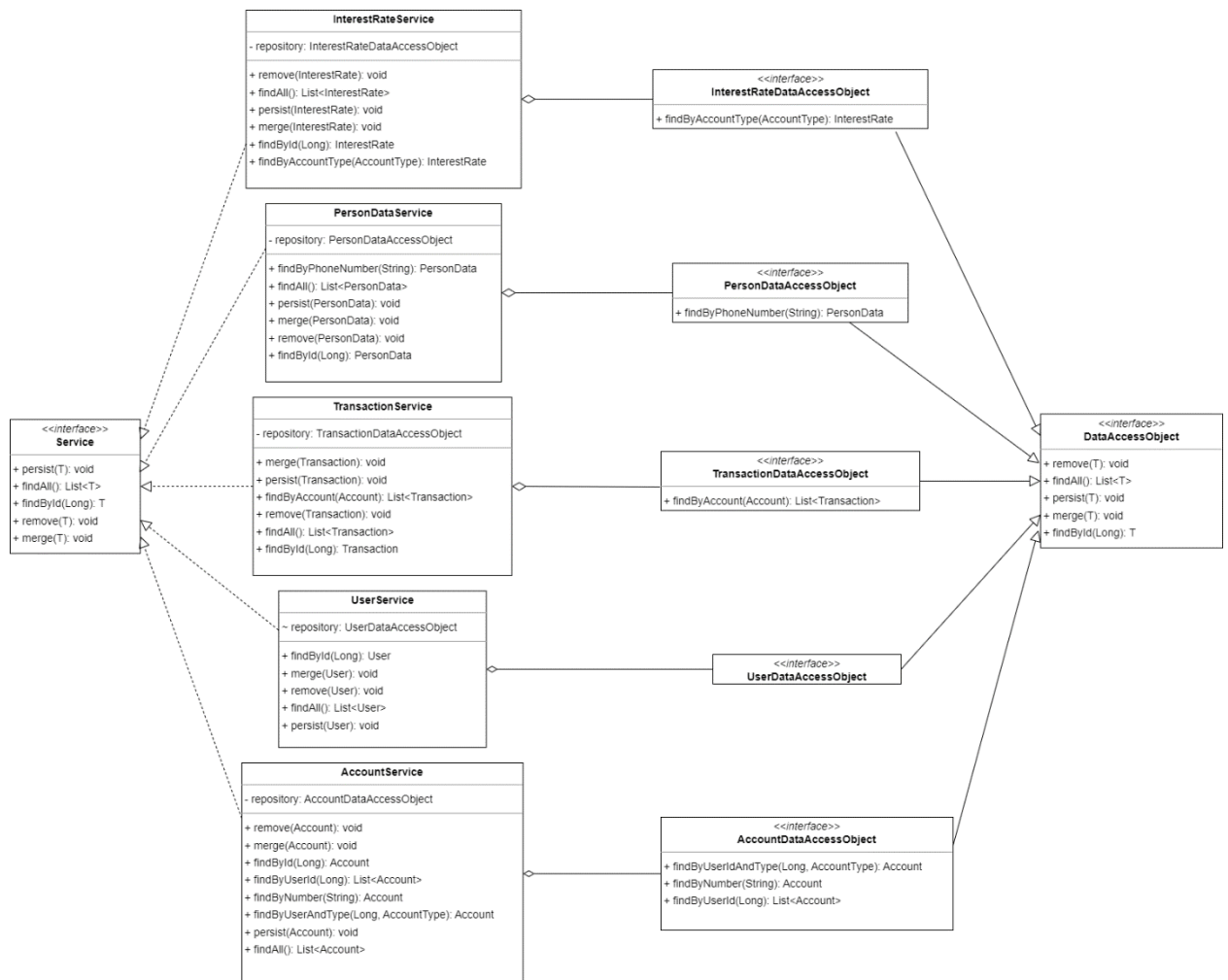


Рисунок 2.1 – Диаграмма классов пакетов service и repository

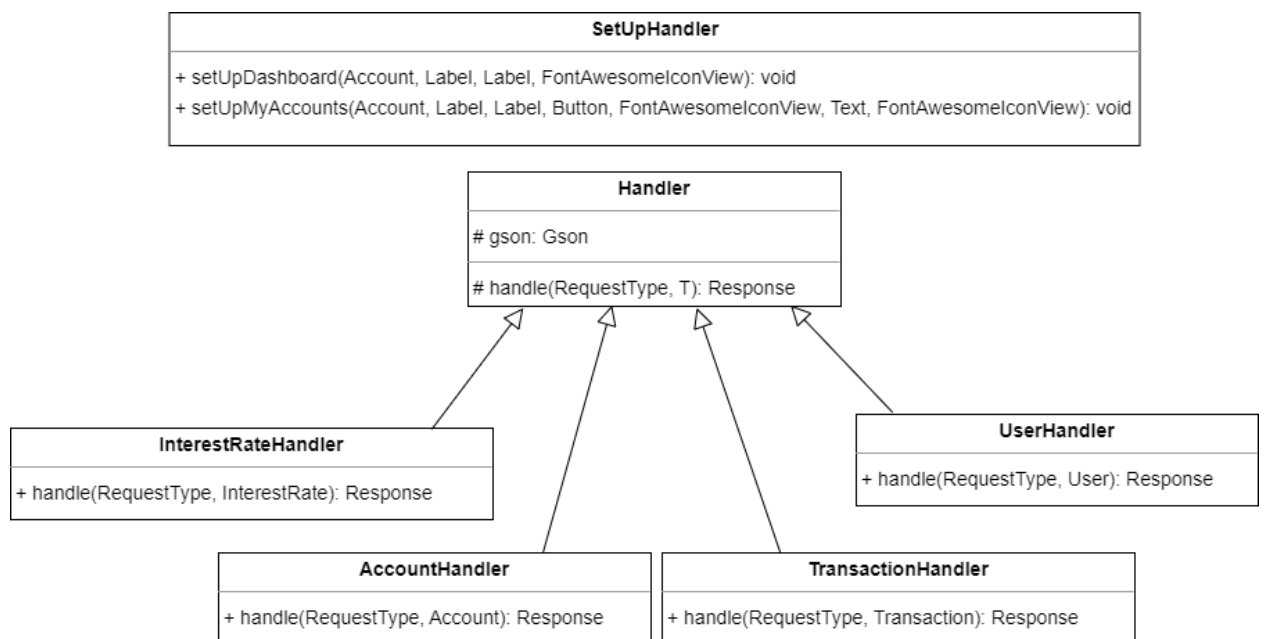


Рисунок 2.2 – Диаграмма классов пакета handler

Диаграмма классов – это UML-диаграмма, которая описывает систему, визуализируя различные типы объектов внутри системы и виды статических связей, которые существуют между ними. Он также иллюстрирует операции и атрибуты классов [7].

2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства

Блок-схема алгоритма отображает в графическом виде последовательность операций и переходные фазы. Каждому действию соответствует определенная фигура (ромб, квадрат, овал и т. д.), поэтому располагать их нужно в правильном порядке [8].

Одним из основных алгоритмов приложения является перевести деньги по номеру телефона. Блок-схема данного алгоритма представлена на рисунке 2.3.

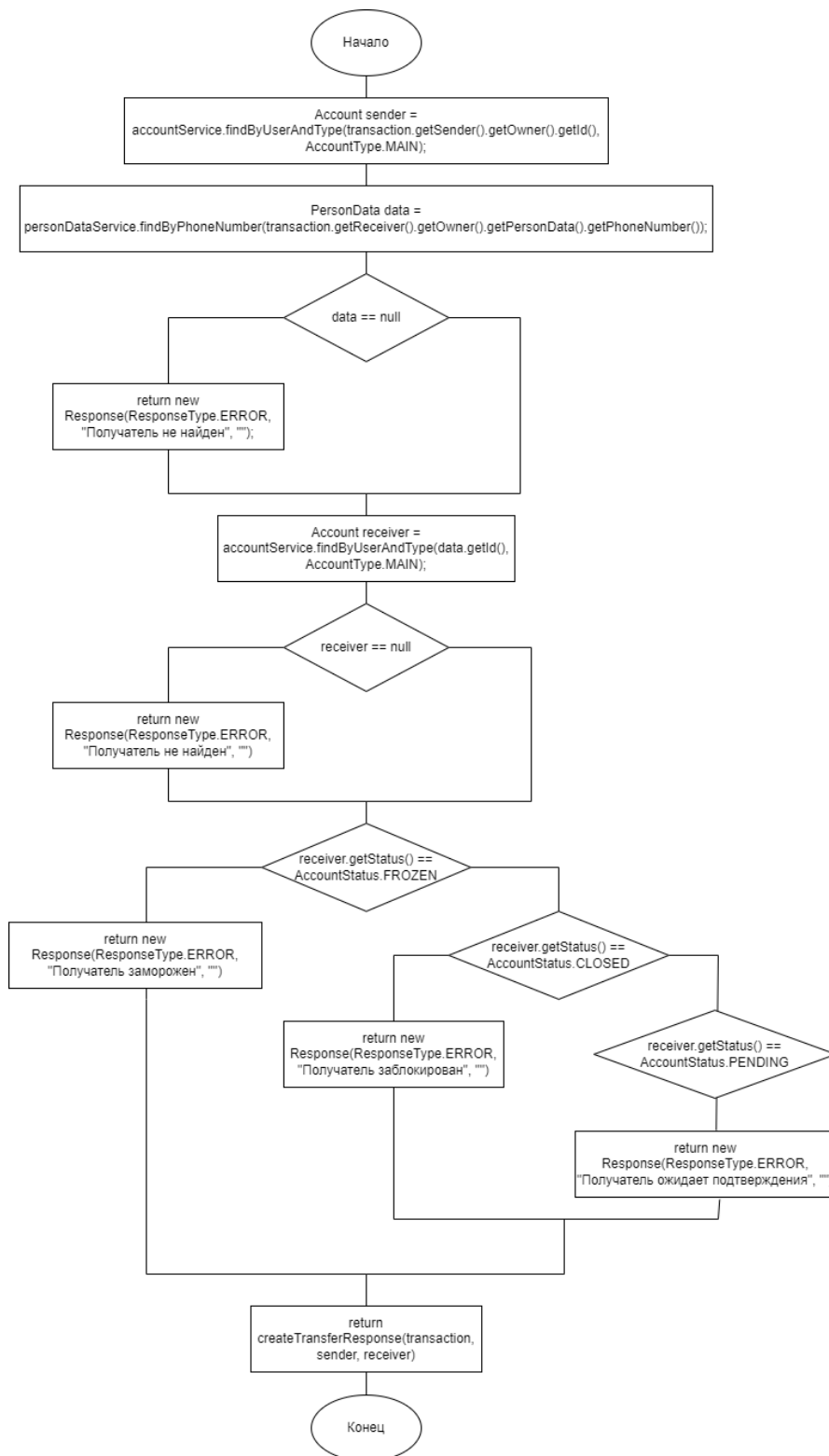


Рисунок 2.3 – Блок-схема алгоритма transferMoneyByNumber (Transaction transaction)

Листинг кода алгоритма:

```
public Response transferMoneyByNumber(Transaction transaction) {
```

```

        Account sender =
accountService.findByUserAndType(transaction.getSender().getOwner().getId(),
AccountType.MAIN);
        PersonData data =
personDataService.findByPhoneNumber(transaction.getReceiver().getOwner().getPersonDat
a().getPhoneNumber());
        if (data == null) {
            return new Response(ResponseType.ERROR, "Получатель не найден",
""");
        }
        Account receiver = accountService.findByUserAndType(data.getId(),
AccountType.MAIN);
        if (receiver == null) {
            return new Response(ResponseType.ERROR, "Получатель не найден",
""");
        }
        if (receiver.getStatus() == AccountStatus.FROZEN) {
            return new Response(ResponseType.ERROR, "Получатель заморожен",
""");
        } else if (receiver.getStatus() == AccountStatus.CLOSED) {
            return new Response(ResponseType.ERROR, "Получатель заблокирован",
""");
        } else if (receiver.getStatus() == AccountStatus.PENDING) {
            return new Response(ResponseType.ERROR, "Получатель ожидает
подтверждения", "");
        }
        return createTransferResponse(transaction, sender, receiver);
    }

```

2.4 Проектирование пользовательского интерфейса

В качестве среды разработки была выбрана IntelliJIDEA за счет удобства работы, поддержки JavaFX и Maven, интеграции с системами контроля версий (например Git) и плагиновой архитектуры (широкий выбор плагинов, интеграция с базами данных).

IntelliJIDEA значительно обгоняет свои аналоги (Eclipse, NetBeans), так как имеет большее быстродействие и оптимизацию, современный интерфейс и поддержку современных технологий.

Как современный UI-фреймворк JavaFX предоставляет мощные инструменты для создания графических интерфейсов (например, CSS для стилизации, FXML для декларативной разметки).

JavaFX модернизированный фреймворк, чем устаревший Swing. Стилизация интерфейса в JavaFX проще благодаря поддержке CSS. А также JavaFX быстрее и лучше оптимизирован для создания динамических и графически насыщенных приложений.

Инструментом управления проектами в приложении был выбран Maven. Он автоматизирует процесс сборки, управления зависимостями и генерации документации, его использование приводит к единообразной структуре проектов, что упрощает поддержку и развитие. Можно отметить и легкий доступ к миллионам библиотек через центральный репозиторий Maven.

Maven использует XML-конфигурацию, которая проще для пользователей по сравнению с Gradle (использует Groovy/Kotlin DSL), и

предоставляет множество стандартных плагинов, минимизируя необходимость создания кастомных решений.

MySQL использована как база данных из-за высокой производительности и удобной интеграции с Java (интеграция через JDBC, с поддержкой ORM-фреймворков, таких как Hibernate). Благодаря своей популярности, MySQL имеет огромное количество руководств и готовых решений.

MySQL легче освоить, чем PostgreSQL, который имеет более сложный синтаксис.

2.5 Обоснование выбора компонентов и технологий для реализации программного средства

Диаграмма развёртывания показывает топологию системы и распределение компонентов системы по ее узлам, а также соединения - маршруты передачи информации между аппаратными узлами. Это единственная диаграмма, на которой применяются “трехмерные” обозначения: узлы системы обозначаются кубиками. Все остальные обозначения в UML - плоские фигуры [9].

На рисунке 2.4 представлена диаграмма развёртывания, совмещенная с диаграммой компонентов приложения.

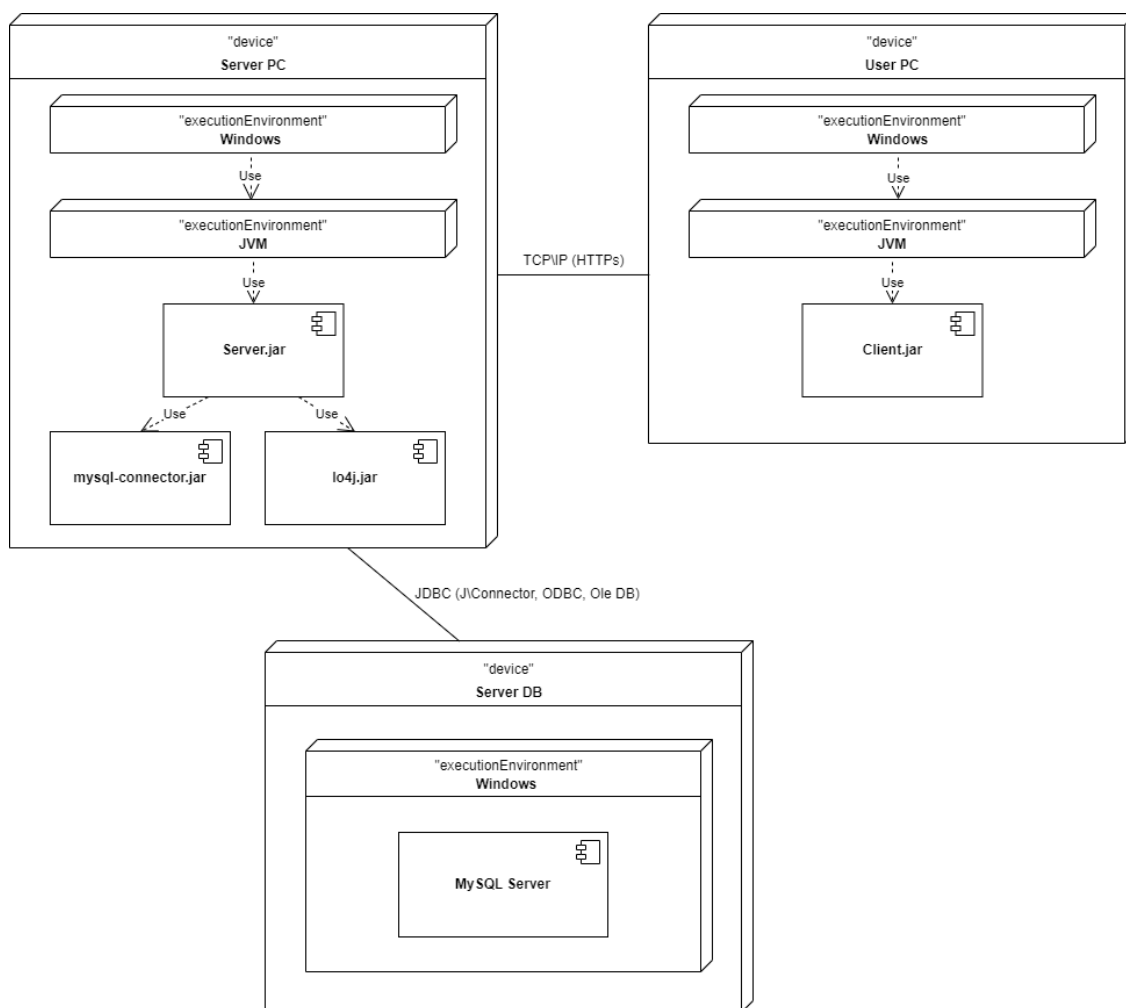


Рисунок 2.4 – UML-диаграмма компонентов

Диаграммы компонентов используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы [10].

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного обеспечения (Software Testing) - проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

GUI — это графический интерфейс, то есть то, что пользователь видит на экране. Это, пожалуй, самая сложная для тестирования вещь, если речь идет, например, о проверке работы сайта, то мы должны как-то эмулировать работу браузера, который довольно сложно устроен, анализировать информацию, которая выводится на странице. Но этот вид тестирования очень важен, так как он взаимодействует с приложением так же, как и пользователь.

JUnit применяется для модульного тестирования, которое позволяет проверять на правильность отдельные модули исходного кода программы. Преимущество данного подхода заключается в изолировании отдельно взятого модуля от других. При этом, цель такого метода позволяет программисту удостовериться, что модуль, сам по себе, способен работать корректно. JUnit представляет из себя библиотеку классов.

Тестовые классы хранятся в отдельном пакете test (рисунок 3.1).

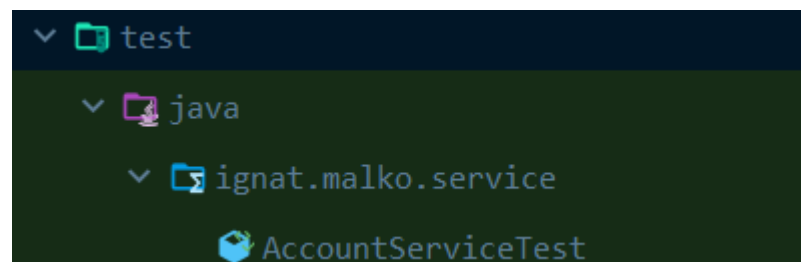


Рисунок 3.1 – Хранение тестового класса

В тестовом классе использованы две аннотации @BeforeEach (аннотированный метод будет запускаться перед каждым тестовым методом в тестовом классе) и @Test (используется, чтобы пометить метод как тест JUnit).

Также в классе используются моки — классы-заглушки, которые используются, чтобы проверить, что определенная функция была вызвана с определенными аргументами.

На рисунке 3.2 приведены результаты JUnit тестирования нахождения счетов пользователей, добавление, редактирование и удаление аккаунтов.

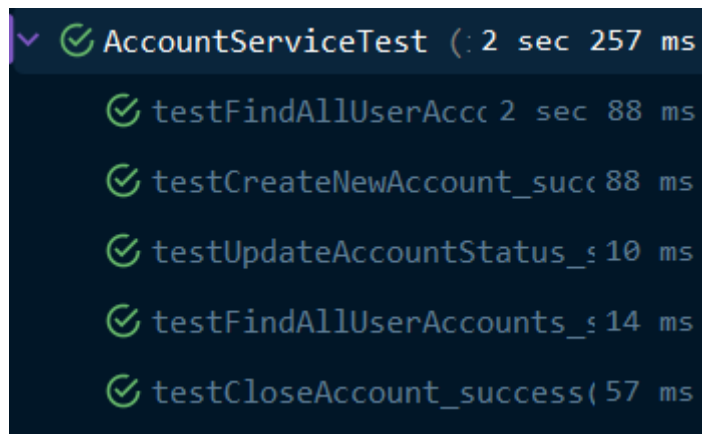


Рисунок 3.2 – Результаты JUnit тестирования

Сегодня все большую популярность приобретает test-driven development (TDD), техника разработки ПО, при которой сначала пишется тест на определённый функционал, а затем пишется реализация этого функционала. На практике все, конечно же, не настолько идеально, но в результате код не только написан и протестирован, но тесты как бы неявно задают требования к функционалу, а также показывают пример использования этого функционала.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРИЛОЖЕНИЯ

4.1 Инструкция по развертыванию приложения

Для запуска приложения без IDE понадобится несколько шагов:

- убедиться, что Maven установлен;
- собрать пакет;
- запустить серверное приложение;
- запустить клиентское приложение;
- использовать конфигурацию Maven для упрощения;
- создать исполняемые JAR.

4.2 Сквозной тестовый пример

При запуске программы нас встречает авторизация (рисунок 4.1).

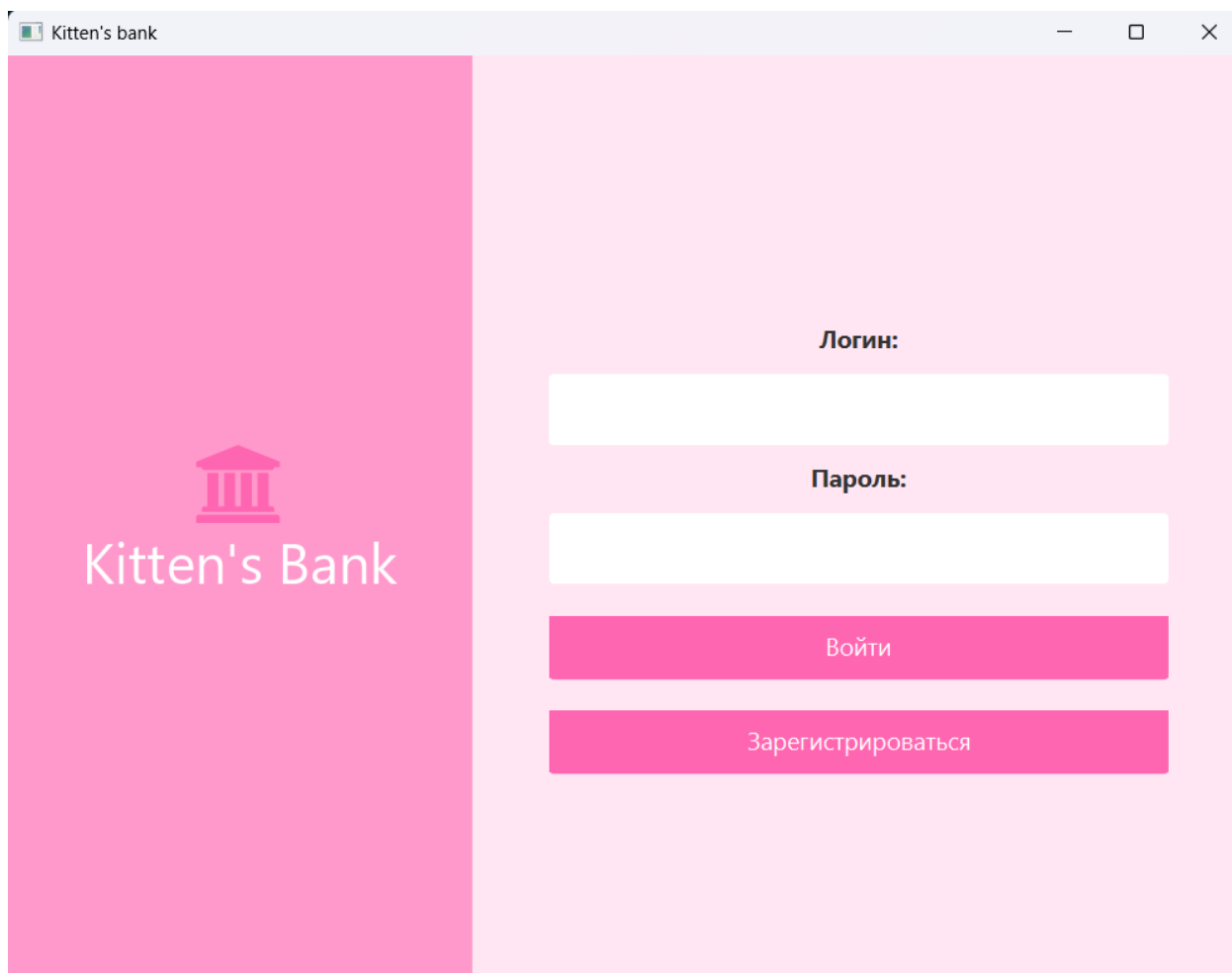


Рисунок 4.1 – Авторизация

При неверном заполнении полей с логином или паролем появляются ошибки (рисунки 4.2-4.3)

Такого пользователя не существует!

Логин:

Рисунок 4.2 – Ошибка ввода логина

Неправильный пароль!

Логин:

Пароль:

Войти

Зарегистрироваться

Рисунок 4.3 – Ошибка ввода пароля

Если пользователь еще не зарегистрирован, то при нажатии на кнопку регистрации открывается форма регистрации. При вводе некорректных данных программа отправляет ошибки (рисунки 4.4-4.8).

Kitten's bank

Все поля должны быть заполнены

Логин:

admin

Пароль:

Повторите пароль:

Фамилия:

Имя:

Email:

Номер телефона:

Возраст:

14

Пол:

☐ Мужской ☐ Женский ☐ Другое

Назад Зарегистрироваться

Рисунок 4.4 – Регистрация

Kitten's bank

Пароли не совпадают

Рисунок 4.5 – Ошибка ввода пароля

Kitten's bank

Такой пользователь уже существует!

Рисунок 4.6 – Ошибка ввода существующего логина

Kitten's bank

Некорректный email

Рисунок 4.7 – Ошибка ввода email-адреса

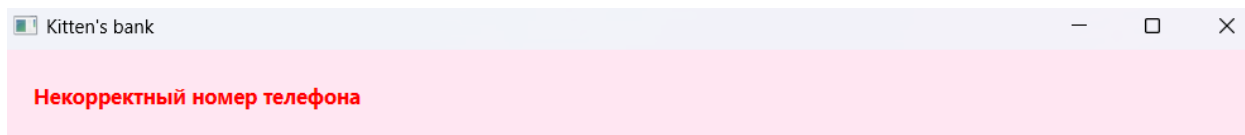


Рисунок 4.8 – Ошибка ввода номера телефона

После успешной авторизации попадаем на главную страницу, где видны счета пользователя (рисунок 4.9).

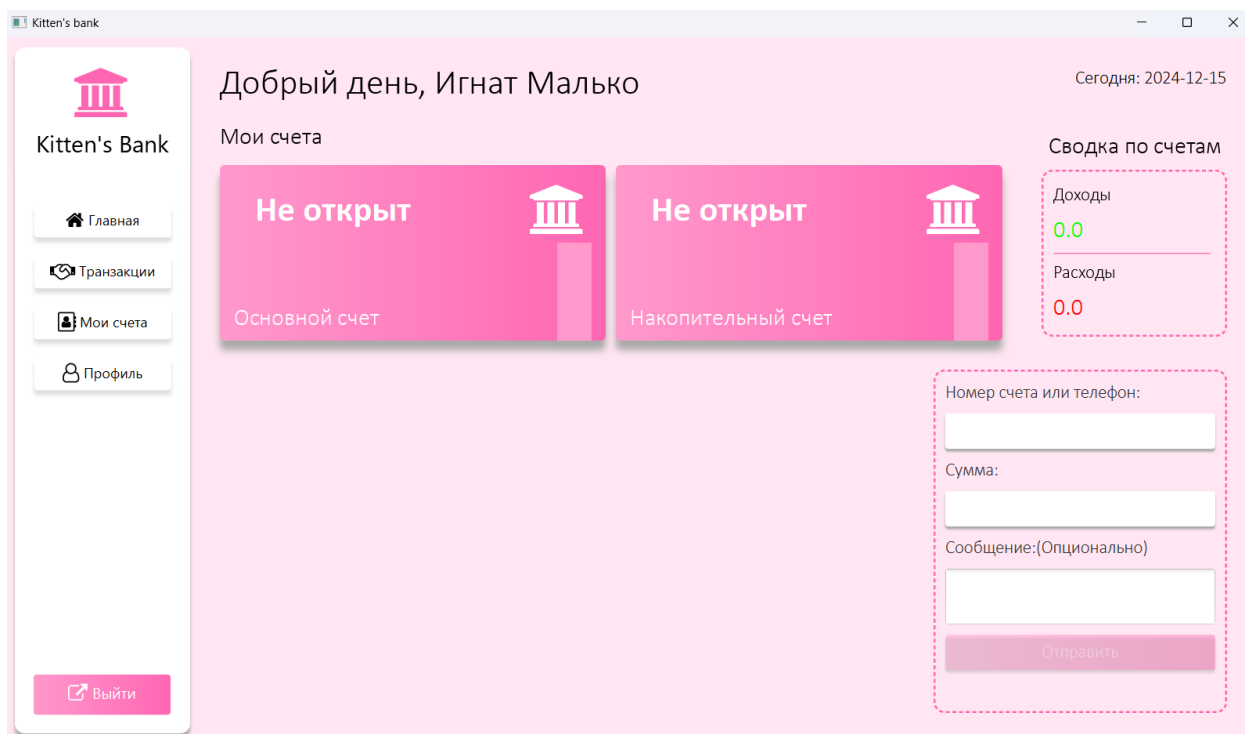


Рисунок 4.9 – Главная страница

Видно, что счета не открыты, для открытия следует перейти на вкладку счетов пользователя и запросить открытие счета, который уже может одобрить менеджер или администратор.

На рисунке 4.10 представлены уже открытые основной и накопительный счета с добавленными на них средствами.

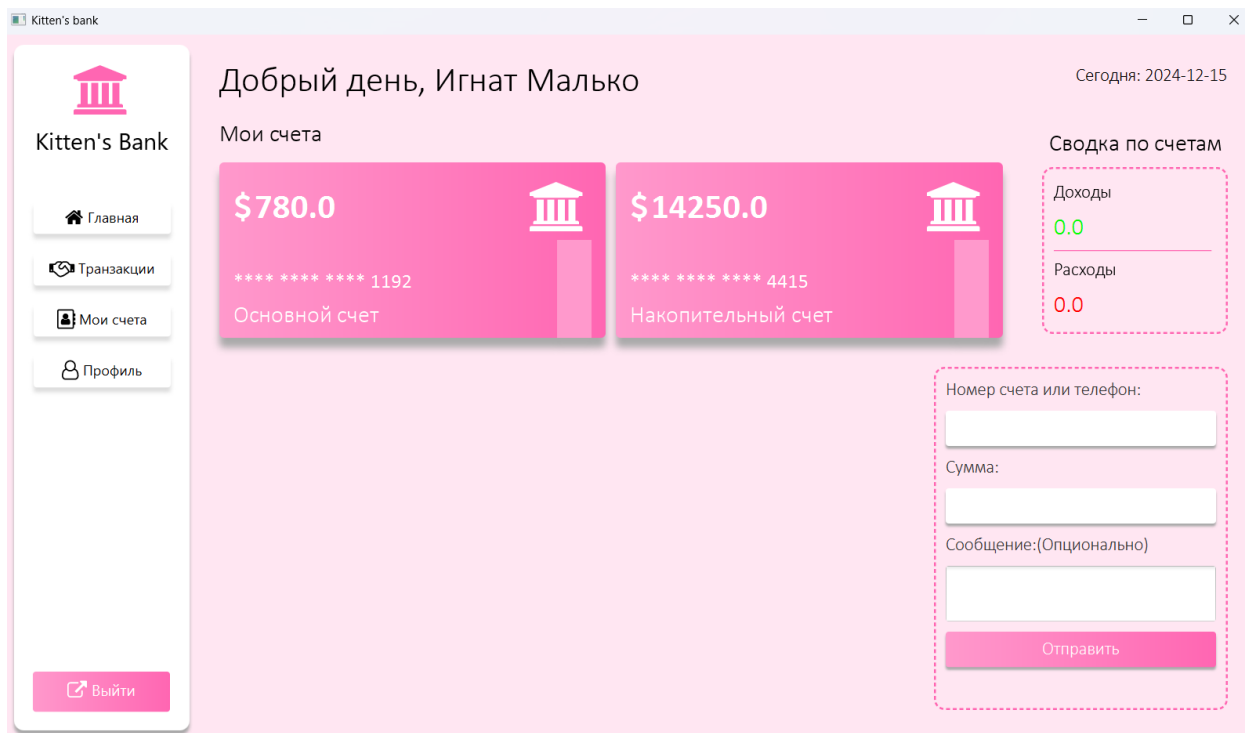


Рисунок 4.10 – Главная страница

В разделе счетов пользователя можно перевести средства с основного на накопительный счет (рисунок 4.11).

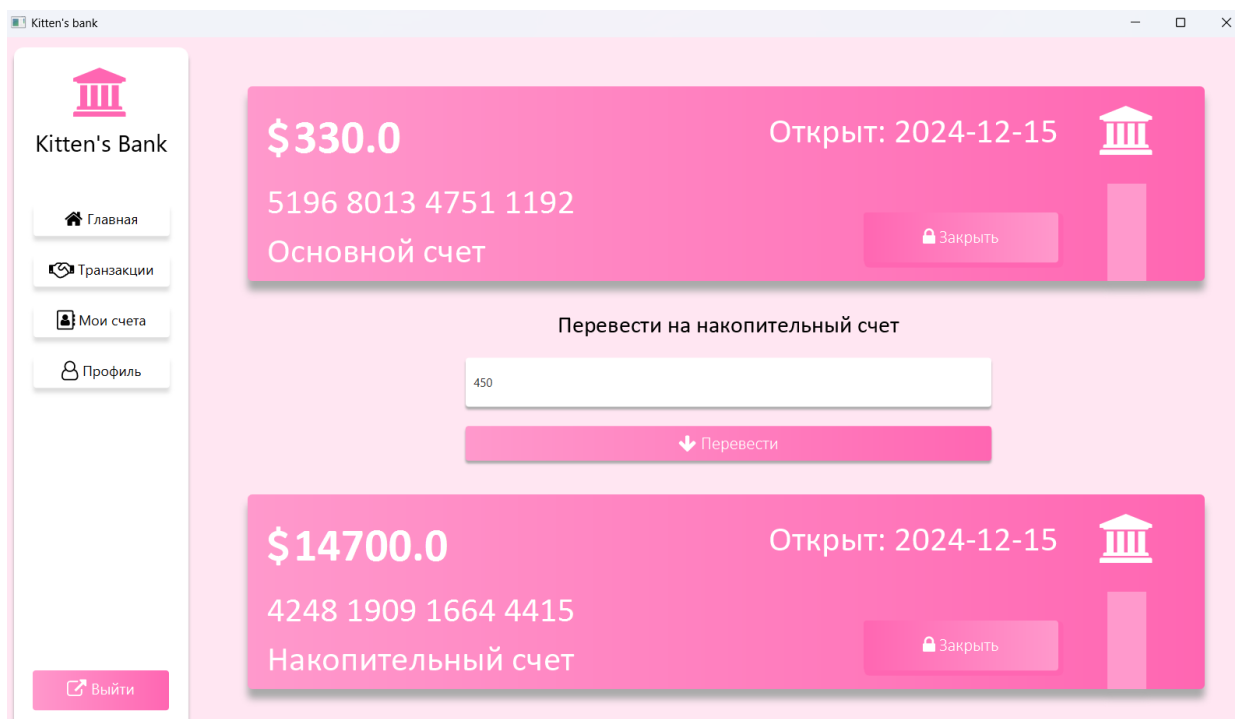


Рисунок 4.11 – Мои счета

На рисунке 4.12 продемонстрированы все транзакции, которые совершал пользователь.

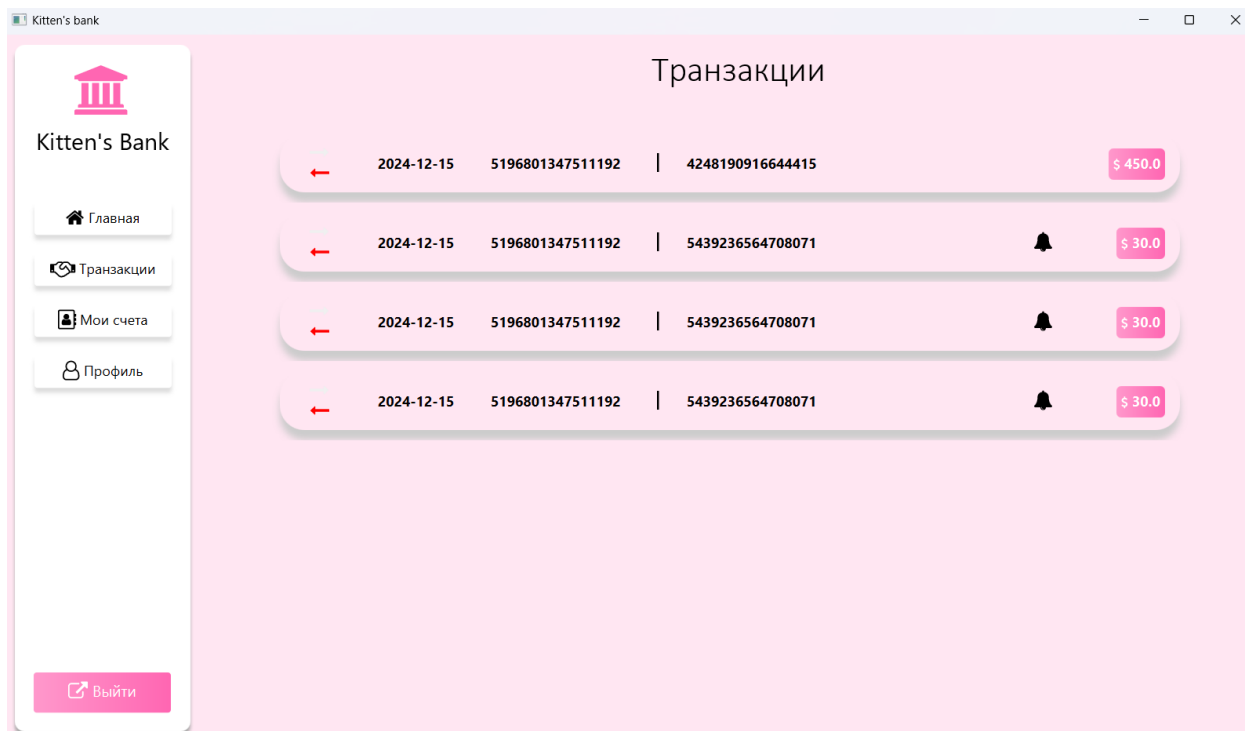


Рисунок 4.12 – Транзакции

Изменить свои данные можно нажатием на кнопку профиля (рисунок 4.13).

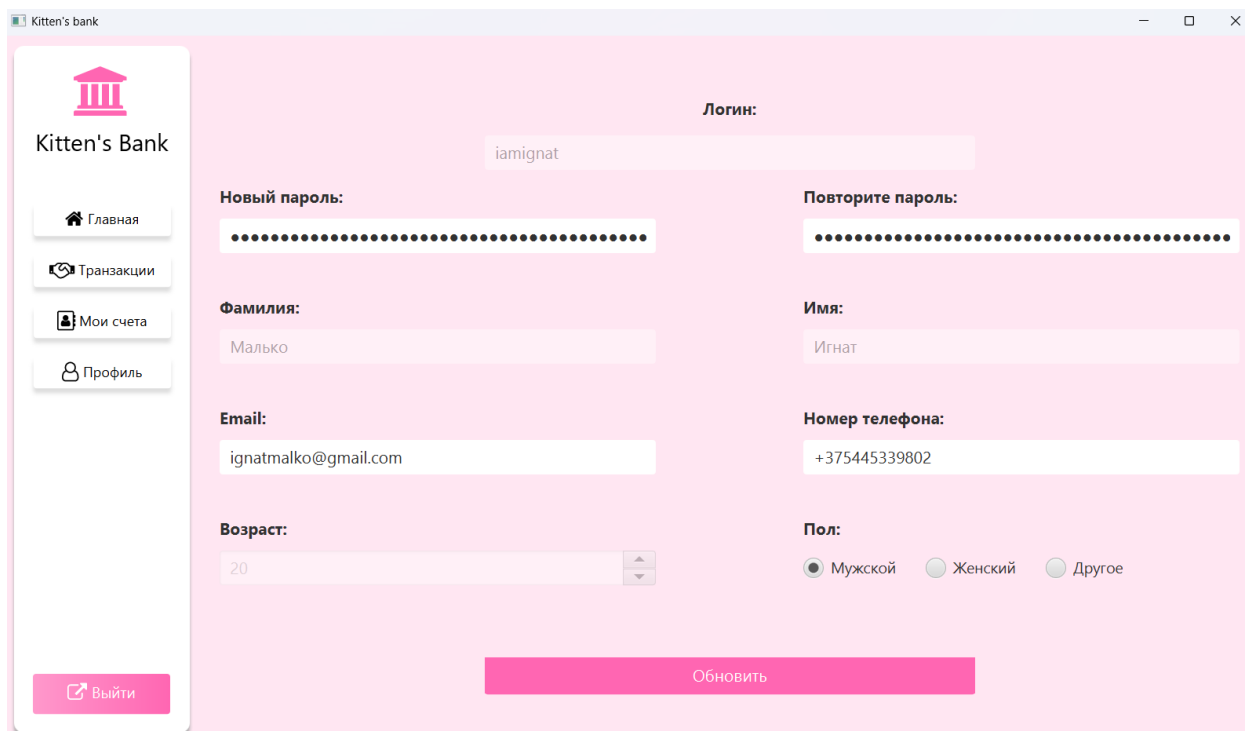


Рисунок 4.13 – Профиль

Далее разберем роль менеджера. В данной программе с увеличением роли расширяются возможности, то есть все разделы пользователя остаются и у менеджера, и у администратора. Поэтому рассмотрим разделы, которые идут после профиля.

В разделе счета есть четыре вкладки: новые, активные, замороженные и закрытые (рисунки 4.14-4.20).

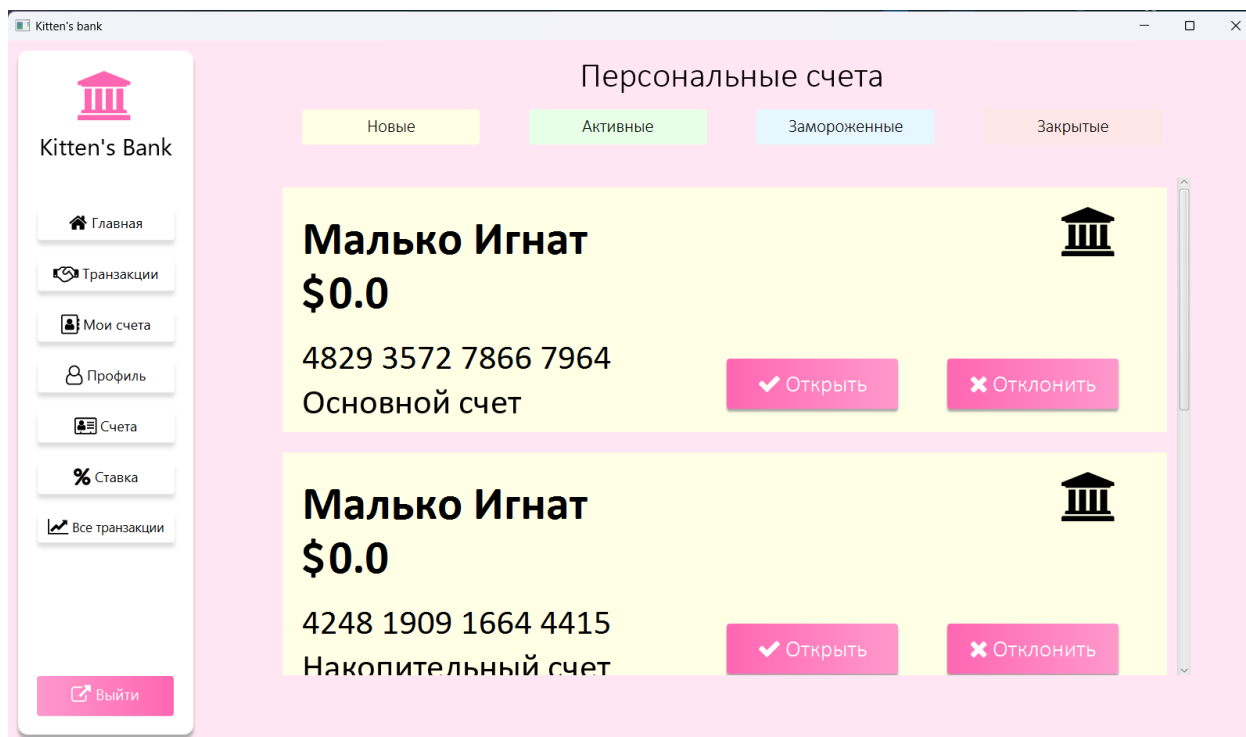


Рисунок 4.14 – Новые счета

В новых счетах менеджер или администратор могут подтвердить либо отклонить запрос на открытие счета от пользователя.

После каждого шага требуется подтверждение действия.

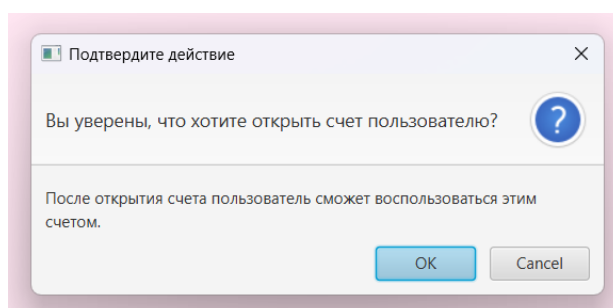


Рисунок 4.15 – Подтверждение открытия счета

В активных счетах видны все открытые счета пользователей.

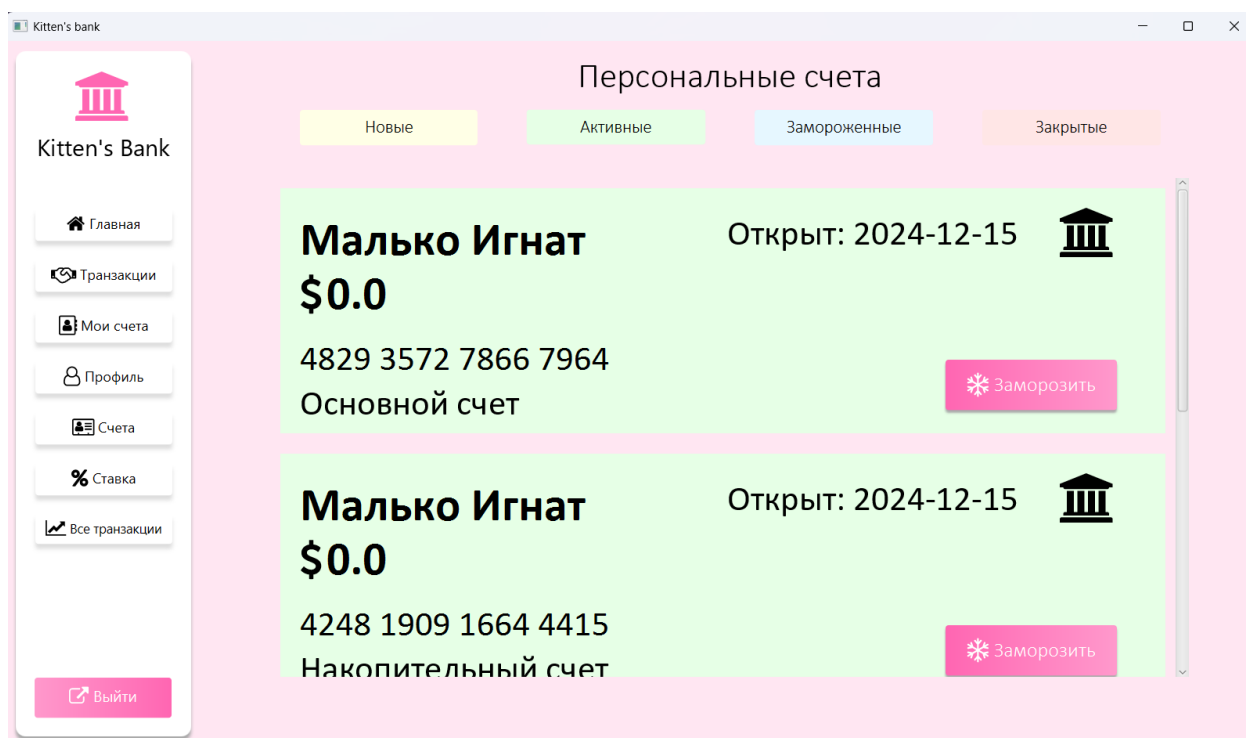


Рисунок 4.16 – Активные счета

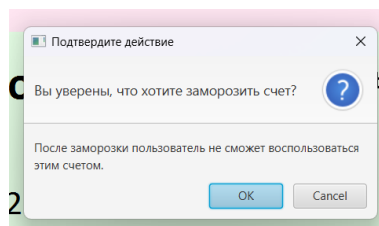


Рисунок 4.17 – Подтверждение заморозки счета

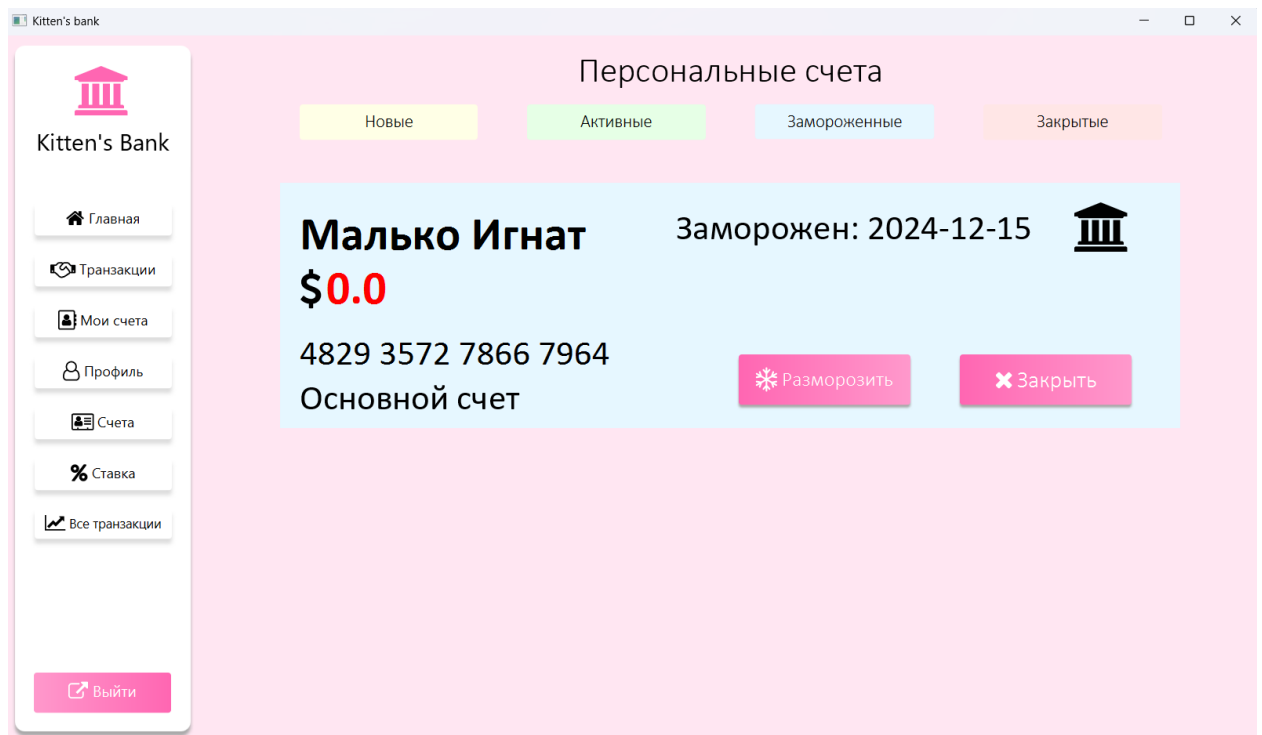


Рисунок 4.18 – Замороженные счета

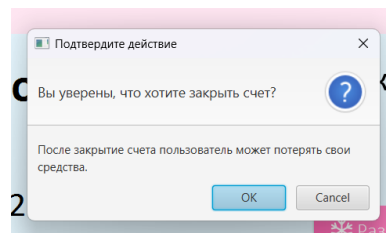


Рисунок 4.19 – Подтверждение удаления счета

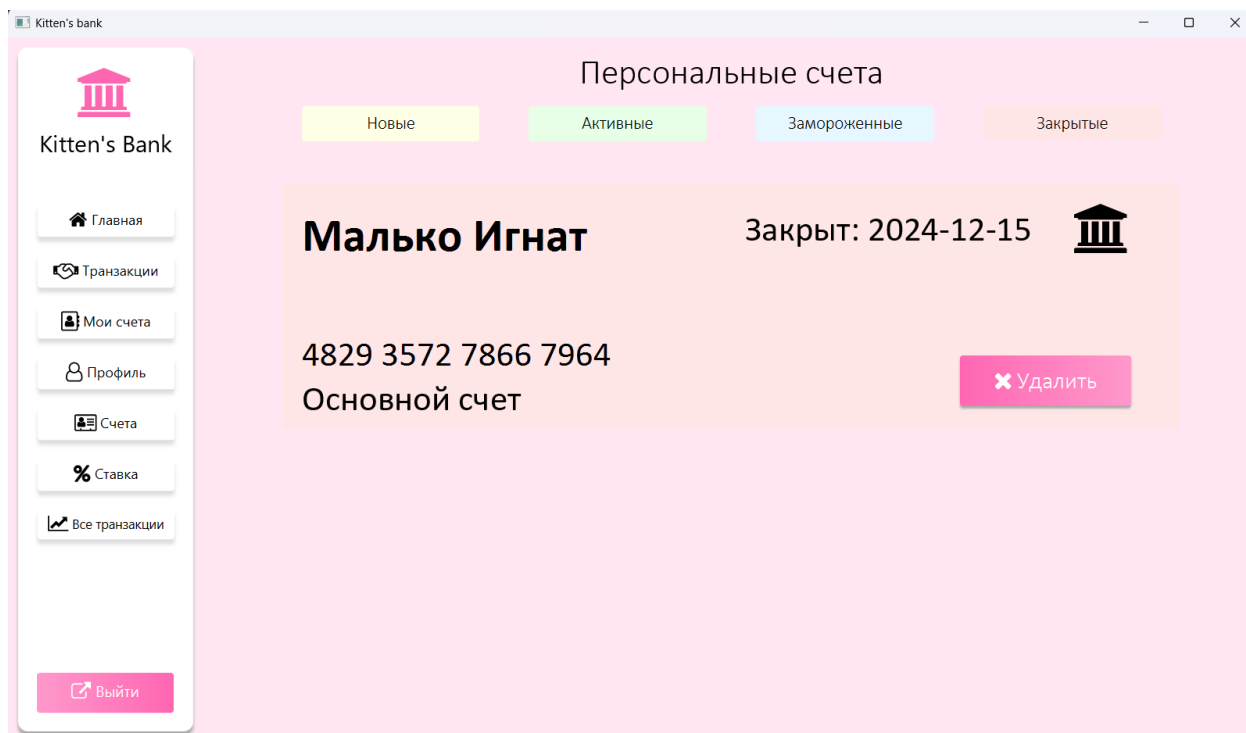


Рисунок 4.20 – Закрытые счета

Перейдя на вкладку ставки, можно изменить процентные ставки счетов (рисунок 4.21).

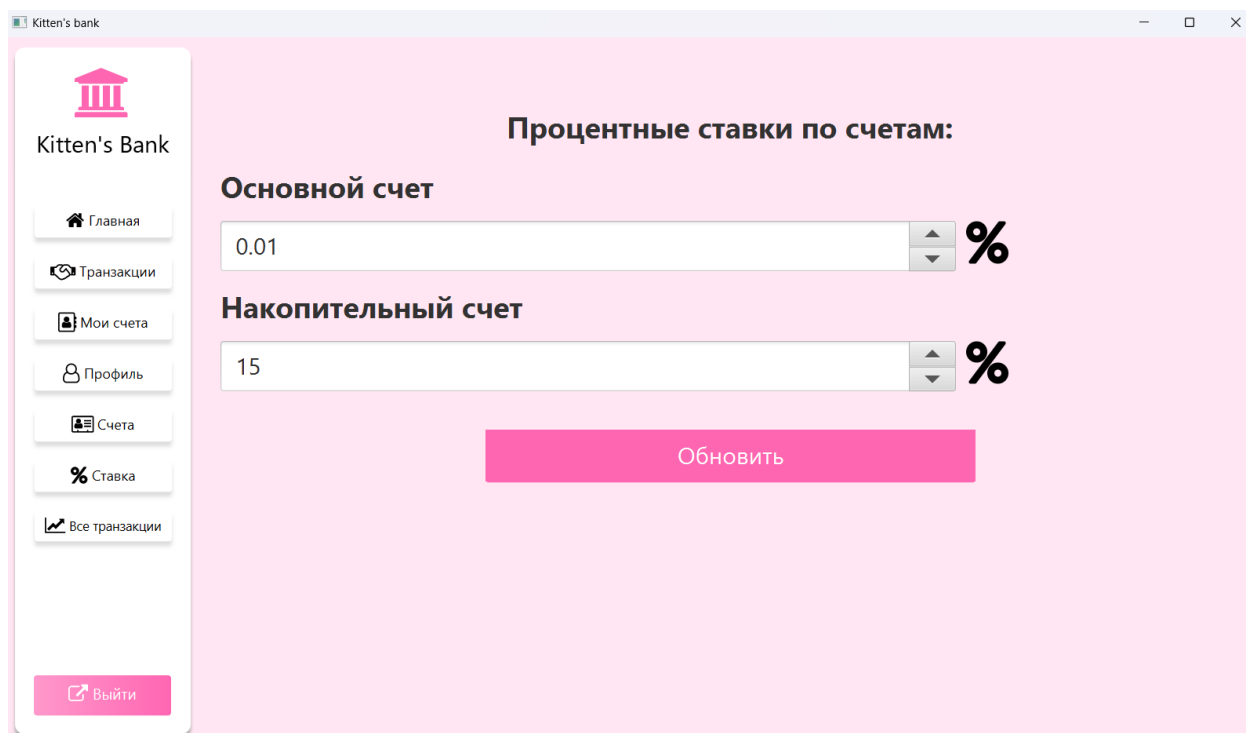


Рисунок 4.21 – Процентные ставки

Далее доступен просмотр всех транзакций пользователей (рисунок 4.22).

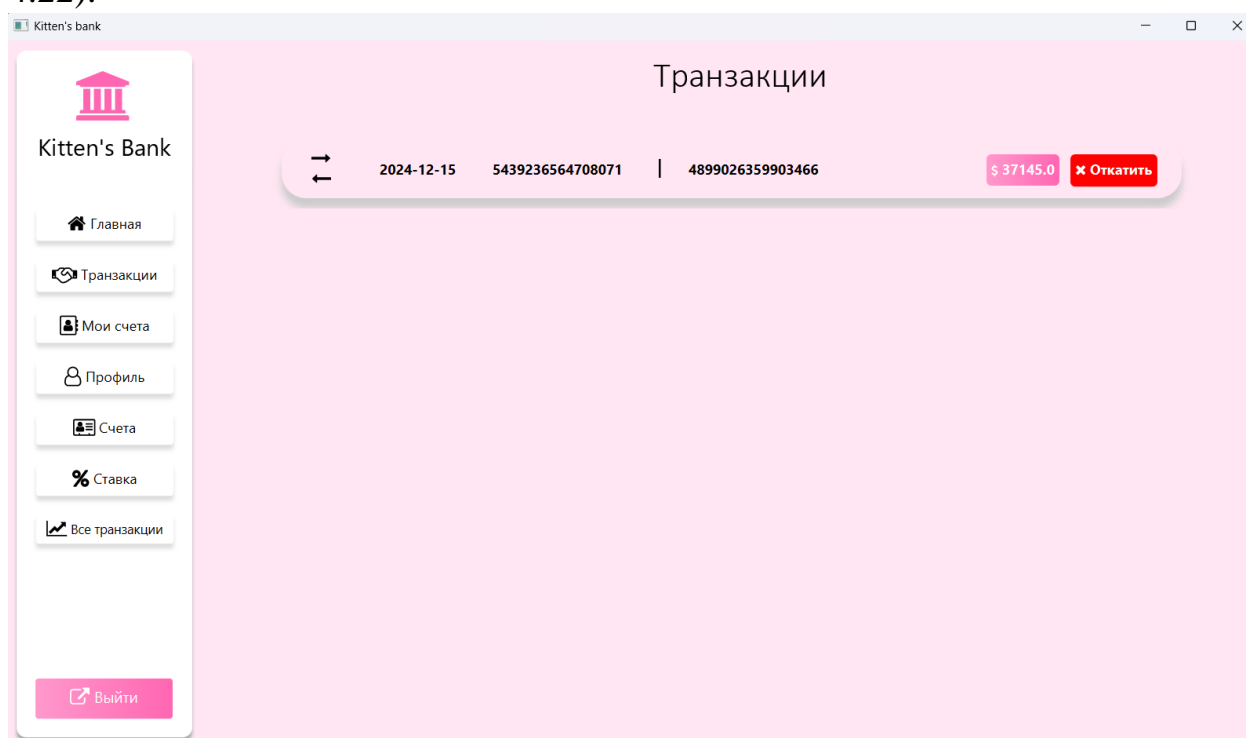


Рисунок 4.22 – Все транзакции

Прав у менеджера больше нет. У администратора доступен еще один раздел просмотра клиентов.

Можно отфильтровать отдельно менеджеров и администраторов (рисунки 4.23-4.26).

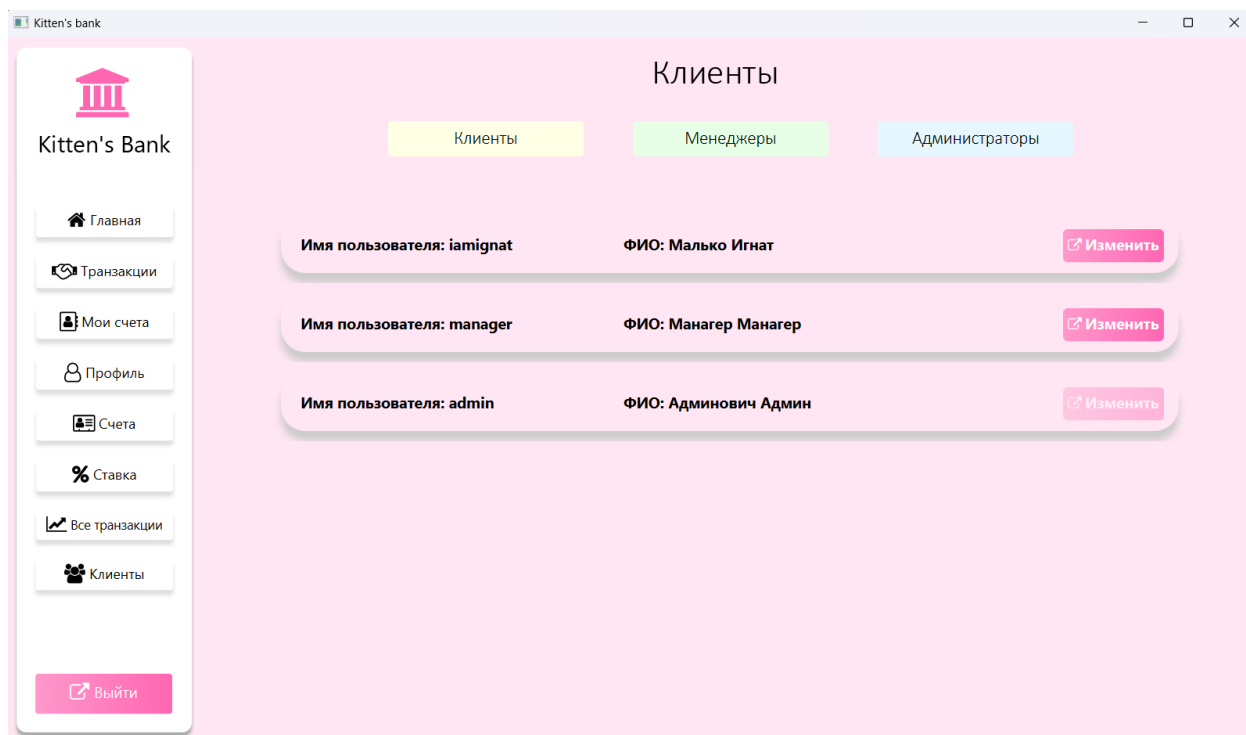


Рисунок 4.23 – Клиенты

Каждого пользователя, кроме себя самого, администратор может изменить (рисунок 4.27).

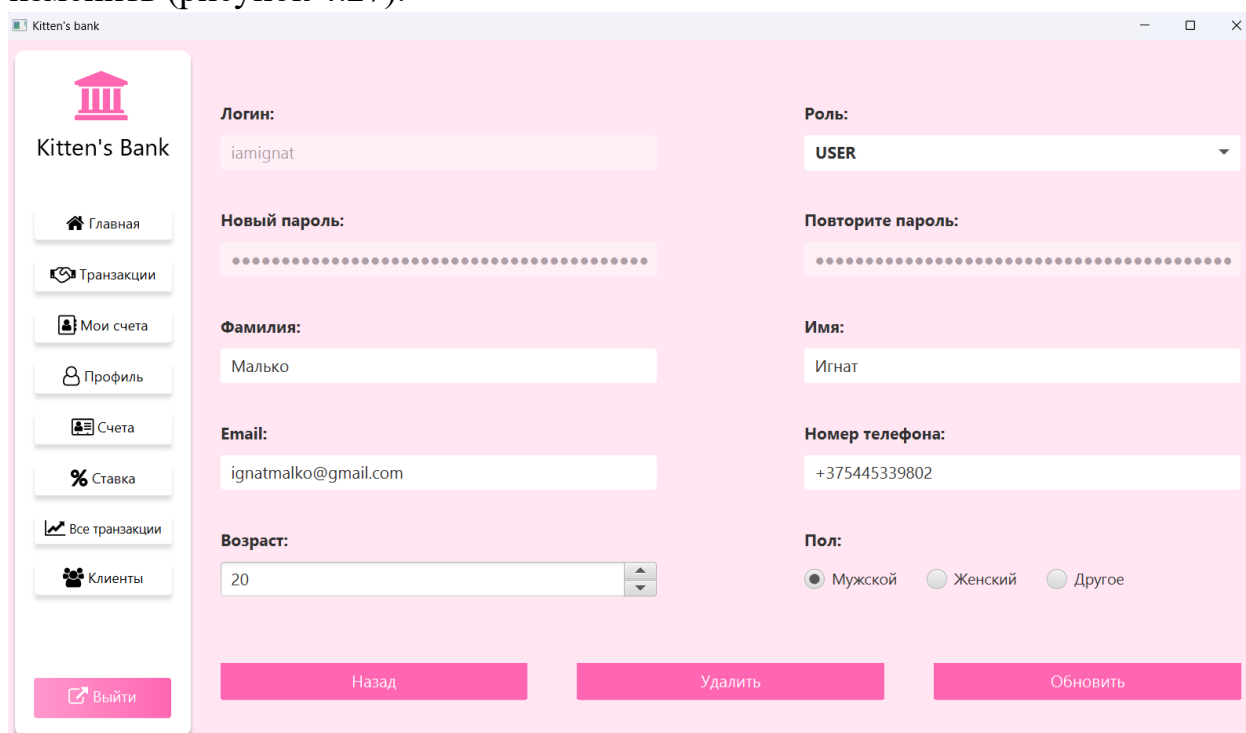


Рисунок 4.24 – Изменить пользователя

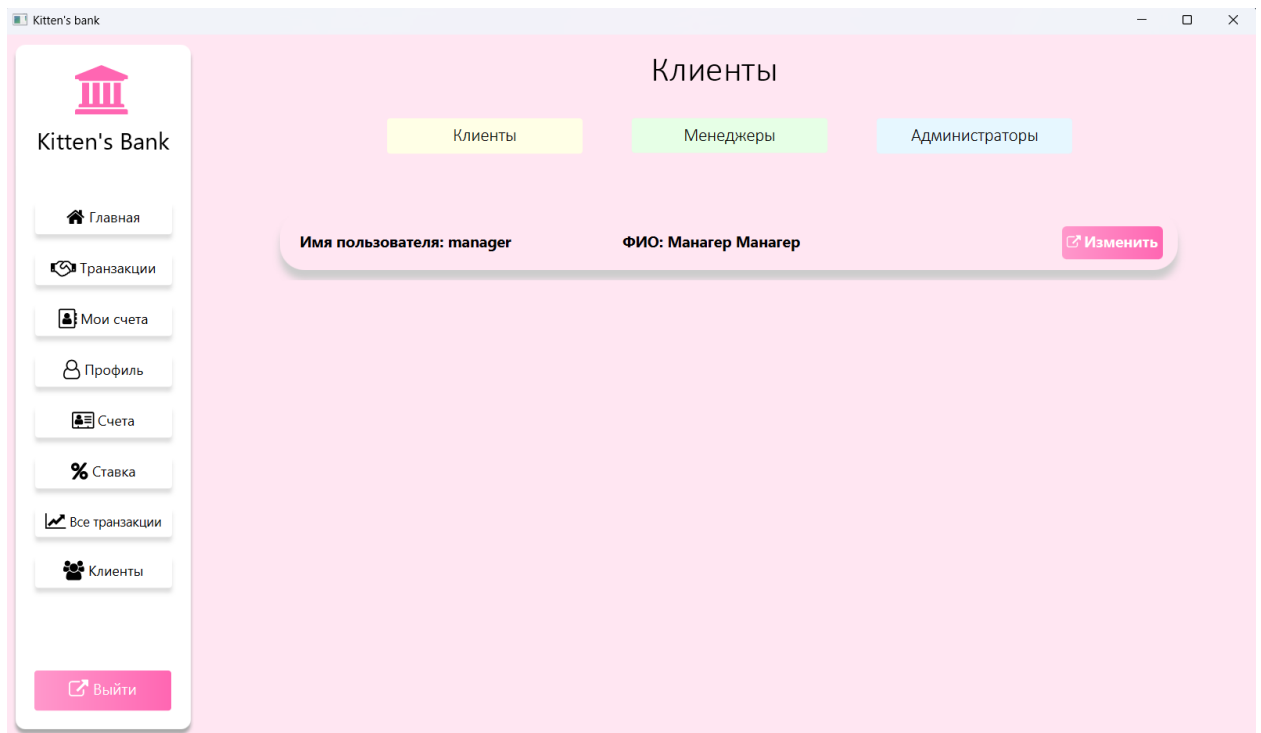


Рисунок 4.25 – Менеджеры

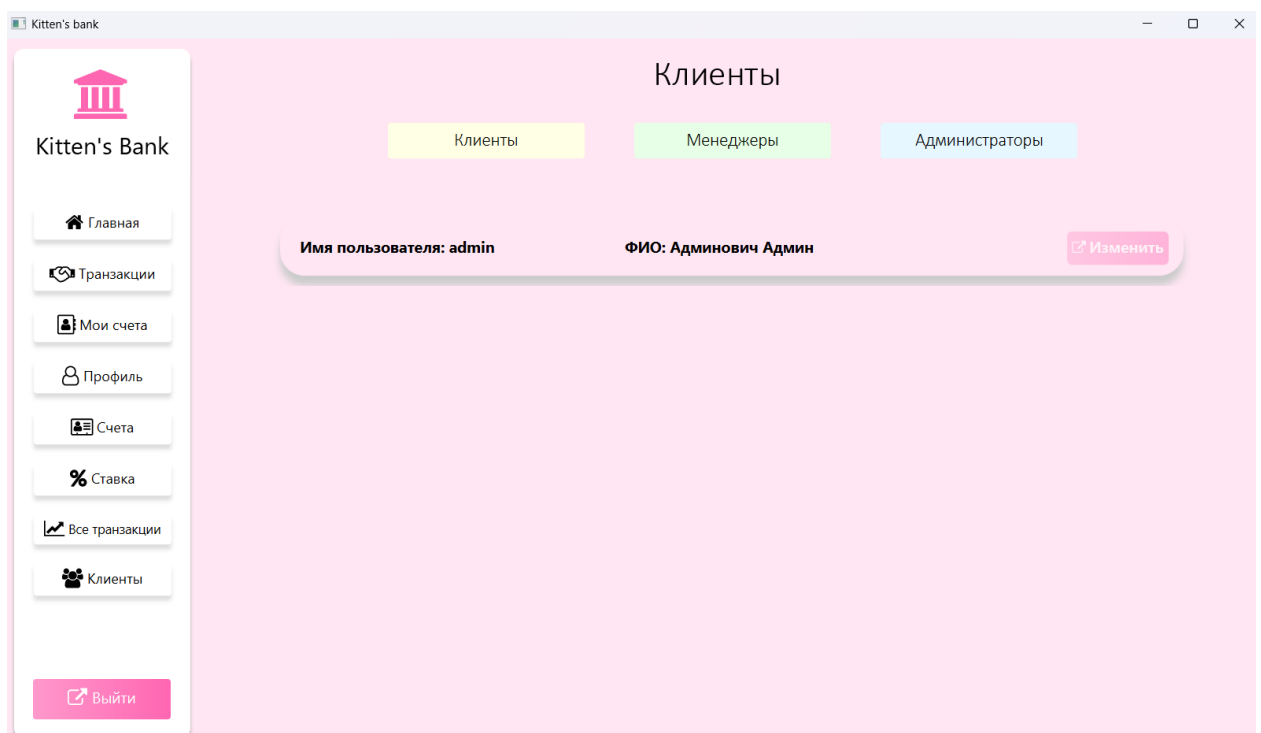


Рисунок 4.26 – Администраторы

В главе была описана инструкция по развертыванию приложения на любом персональном компьютере, а также полностью рассмотрен и описан функционал приложения для клиента, менеджера и администратора системы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта на тему "Разработка системы управления банковскими вкладами физических лиц" была создана программа, которая упрощает и оптимизирует процессы работы с банковскими вкладами. Используя современные технологии разработки на Java, такие как JavaFX и Maven, удалось реализовать удобную и функциональную систему для учета и управления банковскими счетами физических лиц.

Применение JavaFX позволило создать интуитивно понятный и визуально привлекательный пользовательский интерфейс, обеспечивающий легкий доступ ко всем функциям системы. Инструменты Maven упростили процесс управления зависимостями и сборки проекта, обеспечив его структурированность и масштабируемость.

Разработанная система предоставляет возможности для регистрации клиентов, открытия и управления банковскими вкладами, отображения истории транзакций, а также расчетов процентных начислений. Благодаря использованию JavaFX удалось добавить интерактивность интерфейса, что делает работу с системой удобной и понятной даже для пользователей без технических знаний.

Разработка данной программы показала, что автоматизация процессов управления банковскими вкладами не только повышает их эффективность, но и улучшает качество обслуживания клиентов. Внедрение таких решений позволяет сотрудникам банка сосредоточиться на решении более сложных задач, оставив рутинные операции автоматизированной системе.

Дальнейшее развитие проекта может включать интеграцию с платежными системами, реализацию дополнительных аналитических инструментов, а также создание мобильного приложения для еще большего удобства клиентов. Таким образом, система обладает высоким потенциалом для дальнейшего развития и может быть успешно внедрена в банках и финансовых организациях.


СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


- [1] IDEF0. Знакомство с нотацией и пример использования [Электронный ресурс]. – Режим доступа: <https://www.trinion.org/blog/idef0-znakomstvo-s-notaciey-i-primer-ispolzovaniya>
- [2] IDEF0: что такое и как используется [Электронный ресурс]. – <https://software.by/info/blogs/1c/idef0-what-is-and-how-to-use/>
- [3] UML – что это за язык моделирования и зачем нужен [Электронный ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/uml/>
- [4] Использование диаграммы вариантов использования UML при проектировании программного обеспечения [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/566218/>
- [5] Информационные модели в базах данных [Электронный ресурс]. – Режим доступа: <https://studfile.net/preview/9743806/page:2/>
- [6] Диаграмма последовательности (sequence-диаграмма) [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/814769/>
- [7] UML-диаграммы классов [Электронный ресурс]. – Режим доступа: <https://prog-cpp.ru/uml-classes/>
- [8] Блок-схема алгоритма: разбираемся в особенностях [Электронный ресурс]. – Режим доступа: <https://gb.ru/blog/blok-shema-algoritma/>
- [9] Диаграмма развёртывания (Deployment diagram) [Электронный ресурс]. – https://flexberry.github.io/ru/fd_deployment-diagram.html
- [10] Простое руководство по диаграммы компонентов [Электронный ресурс]. – Режим доступа: <https://creatly.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/>
- [11] Виды банковских вкладов [Электронный ресурс]. – Режим доступа: <https://www.raiffeisen.ru/wiki/vidy-bankovskih-vkladov/>


ПРИЛОЖЕНИЕ А


(обязательное)


Отчет о проверке на заимствования в системе «Антиплагиат»


**АНТИПЛАГИАТ**
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

ИИТЦ МГУ
www.iitc.mgu.ru

ТАРИФЫ
Demo 
[ИЗМЕНИТЬ](#)

ПРОВЕРКИ
1 в 6 минут 
[ПРОВЕРИТЬ ДОКУМЕНТ](#)

ПОЛЬЗОВАТЕЛЬ 
mcmikekenter@gmail.com
[ВОЙТИ В КАБИНЕТ](#)

МЕНЮ ru ▼

ГЛАВНАЯ / КАБИНЕТ / РЕЗУЛЬТАТЫ ПРОВЕРКИ

Оригинальность98,52%

Совпадения1,48%

Цитирования0%

Самоцитирования0%

[ПОЛНЫЙ ОТЧЕТ](#)[КРАТКИЙ ОТЧЕТ](#)[ИСТОРИЯ ОТЧЕТОВ](#)

 РАСПЕЧАТАТЬ ▼

 ВЫГРУЗИТЬ ▼

 СОЗДАТЬ ССЫЛКУ ▼

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг кода алгоритмов, реализующих основную бизнес-логику

Класс AccountHandler:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class AccountHandler {
    private AccountService accountService;
    private TransactionService transactionService;
    private PersonDataService personDataService;
    private InterestRateService interestRateService;
    private Gson gson;
    private final double TRANSFER_LIMIT = 10431.00;

    public Response createAccount(Account account) {
        account.setBalance(0);
        account.setStatus(AccountStatus.PENDING);
        account.setNumber(AccountNumberGenerator.generateAccountNumber());

        account.setInterestRate(interestRateService.findByAccountType(account.getType()));
        accountService.persist(account);
        return new Response(ResponseType.OK, "Вы успешно оформили заявку на открытие счета!", gson.toJson(account));
    }

    public Response findAllUserAccounts(User user) {
        List<Account> accounts = accountService.findById(user.getId());
        if (accounts != null) {
            return new Response(ResponseType.OK, "Аккаунты клиента успешно найдены",
gson.toJson(accounts));
        }
        return new Response(ResponseType.ERROR, "Аккаунты клиента не найдены", "");
    }

    public Response closeAccount(Account account) {
        List<Account> accounts =
accountService.findById(account.getOwner().getId());
        if (accounts.size() == 2) {
            if (accounts.getFirst().getType() == account.getType()) {
                double balance = accounts.getFirst().getBalance();
                accounts.getFirst().setBalance(0);
                accounts.getFirst().setCreated(LocalDate.now());
                if (accounts.get(1).getStatus() != AccountStatus.CLOSED) {
                    accounts.get(1).setBalance(accounts.get(1).getBalance() +
balance);
                }
                accounts.getFirst().setStatus(AccountStatus.CLOSED);
            } else {
                double balance = accounts.get(1).getBalance();
                accounts.get(1).setBalance(0);
                accounts.get(1).setCreated(LocalDate.now());
                if (accounts.getFirst().getStatus() != AccountStatus.CLOSED) {
                    accounts.getFirst().setBalance(accounts.getFirst().getBalance() +
balance);
                }
                accounts.get(1).setStatus(AccountStatus.CLOSED);
            }
        }
    }
}
```

Продолжение приложения Б

```

    }
    } else {
        accounts.getFirst().setStatus(AccountStatus.CLOSED);
        accounts.getFirst().setBalance(0);
        accounts.getFirst().setCreated(LocalDate.now());
    }
    for (Account a : accounts) {
        accountService.merge(a);
    }
    return new Response(ResponseType.OK, "Вы успешно закрыли счет!",
gson.toJson(accounts));
}

    public Response transferMoneyToSavings(Transaction transaction) {
        Account mainAccount =
accountService.findByUserAndType(transaction.getSender().getOwner().getId(),
AccountType.MAIN);
        Account savingsAccount =
accountService.findByUserAndType(transaction.getSender().getOwner().getId(),
AccountType.SAVINGS);
        Transaction newTransaction = new Transaction();
        newTransaction.setSender(mainAccount);
        newTransaction.setReceiver(savingsAccount);
        newTransaction.setAmount(transaction.getAmount());
        newTransaction.setMessage(null);
        newTransaction.setDate(LocalDate.now());
        transactionService.persist(newTransaction);
        mainAccount.setBalance(mainAccount.getBalance()
newTransaction.getAmount());
        savingsAccount.setBalance(savingsAccount.getBalance()
newTransaction.getAmount());
        accountService.merge(mainAccount);
        accountService.merge(savingsAccount);
        return new Response(ResponseType.OK, "Вы положили деньги на накопительный
счет", gson.toJson(List.of(mainAccount, savingsAccount)));
    }

    public Response transferMoneyByNumber(Transaction transaction) {
        Account sender =
accountService.findByUserAndType(transaction.getSender().getOwner().getId(),
AccountType.MAIN);
        PersonData data =
personDataService.findByPhoneNumber(transaction.getReceiver().getOwner().getPersonDat
a().getPhoneNumber());
        if (data == null) {
            return new Response(ResponseType.ERROR, "Получатель не найден", "");
        }
        Account receiver = accountService.findByUserAndType(data.getId(),
AccountType.MAIN);
        if (receiver == null) {
            return new Response(ResponseType.ERROR, "Получатель не найден", "");
        }
        if (receiver.getStatus() == AccountStatus.FROZEN) {
            return new Response(ResponseType.ERROR, "Получатель заморожен", "");
        } else if (receiver.getStatus() == AccountStatus.CLOSED) {
            return new Response(ResponseType.ERROR, "Получатель заблокирован", "");
        } else if (receiver.getStatus() == AccountStatus.PENDING) {
            return new Response(ResponseType.ERROR, "Получатель ожидает
подтверждения", "");
        }
    }

```

Продолжение приложения Б

```
        return createTransferResponse(transaction, sender, receiver);
    }

    private Response createTransferResponse(Transaction transaction, Account sender,
Account receiver) {
        Transaction newTransaction = new Transaction();
        newTransaction.setSender(sender);
        newTransaction.setReceiver(receiver);
        newTransaction.setAmount(transaction.getAmount());
        newTransaction.setMessage(transaction.getMessage());
        newTransaction.setDate(LocalDate.now());
        transactionService.persist(newTransaction);
        if (newTransaction.getAmount() > TRANSFER_LIMIT) {
            receiver.setStatus(AccountStatus.FROZEN);
        }
        sender.setBalance(sender.getBalance() - newTransaction.getAmount());
        receiver.setBalance(receiver.getBalance() + newTransaction.getAmount());
        accountService.merge(sender);
        accountService.merge(receiver);
        return new Response(ResponseType.OK, "Вы успешно перевели деньги",
gson.toJson(sender));
    }

    public Response transferMoneyByAccount(Transaction transaction) {
        Account sender =
accountService.findByUserAndType(transaction.getSender().getOwner().getId(),
AccountType.MAIN);
        Account receiver =
accountService.findByNumber(transaction.getReceiver().getNumber());
        System.out.println(receiver);
        if (receiver == null) {
            System.err.println("Получатель не найден");
            return new Response(ResponseType.ERROR, "Получатель не найден", "");
        }
        if (receiver.getStatus() == AccountStatus.FROZEN) {
            return new Response(ResponseType.ERROR, "Получатель заморожен", "");
        } else if (receiver.getStatus() == AccountStatus.CLOSED) {
            return new Response(ResponseType.ERROR, "Получатель заблокирован", "");
        } else if (receiver.getStatus() == AccountStatus.PENDING) {
            return new Response(ResponseType.ERROR, "Получатель ожидает
подтверждения", "");
        }
        return createTransferResponse(transaction, sender, receiver);
    }

    public Response findAllAccounts() {
        List<Account> accounts = accountService.findAll();
        if (accounts != null) {
            return new Response(ResponseType.OK, "Список аккаунтов успешно получен",
gson.toJson(accounts));
        }
        return new Response(ResponseType.ERROR, "Список аккаунтов не получен", "");
    }

    public Response updateStatus(Account account) {
        account.setCreated(LocalDate.now());
        accountService.merge(account);
        return new Response(ResponseType.OK, "Статус успешно обновлен",
gson.toJson(accountService.findById(account.getId())));
    }
}
```

Продолжение приложения Б

```
public Response removeAccount(Account account) {
    List<Transaction> transactions = transactionService.findByAccount(account);
    for (Transaction t : transactions) {
        if (Objects.equals(t.getReceiver().getId(), account.getId())) {
            t.setReceiver(null);
        } else if (Objects.equals(t.getSender().getId(), account.getId())) {
            t.setSender(null);
        }
        transactionService.merge(t);
    }
    accountService.remove(account);
    return new Response(ResponseType.OK, "Счет успешно удален", "");
}
}
```

Класс TransactionHandler:

```
@Data
@AllArgsConstructor
@NoArgsConstructor

public class TransactionHandler {
    private TransactionService transactionService;
    private AccountService accountService;
    private Gson gson;

    public Response findAllUserTransactions(User user) {
        List<Transaction> transactions =
transactionService.findByAccount(accountService.findByUserAndType(user.getId(),
AccountType.MAIN));
        if (transactions != null) {
            return new Response(ResponseType.OK, "Транзакции клиента успешно найдены",
gson.toJson(transactions));
        }
        return new Response(ResponseType.ERROR, "Транзакции клиента не найдены", "");
    }

    public Response findAllTransactions() {
        List<Transaction> transactions = transactionService.findAll();
        if (transactions != null) {
            return new Response(ResponseType.OK, "Список транзакций успешно получен",
gson.toJson(transactions));
        }
        return new Response(ResponseType.ERROR, "Список транзакций не получен", "");
    }

    public Response rollbackTransaction(Transaction transaction) {
        if (transaction.getReceiver() != null) {
            Account receiver =
accountService.findById(transaction.getReceiver().getId());
            receiver.setBalance(receiver.getBalance() - transaction.getAmount());
            accountService.merge(receiver);
        }
        if (transaction.getSender() != null) {
            Account sender =
accountService.findById(transaction.getSender().getId());
            sender.setBalance(sender.getBalance() + transaction.getAmount());
            accountService.merge(sender);
        }
    }
}
```

Продолжение приложения Б

```
    }
    transaction.setReceiver(null);
    transaction.setSender(null);
    transactionService.remove(transaction);
    return new Response(ResponseType.OK, "Транзакция успешно отменена", "");
}
}
```

Класс UserHandler:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class UserHandler {

    private UserService userService;
    private PersonDataService personDataService;
    private AccountService accountService;
    private TransactionService transactionService;
    private Gson gson;

    public Response updateUser(User user) {
        Response response;
        user.getPersonData().setId(user.getId());
        user.setPassword(PasswordEncryptor.hashPassword(user.getPassword()));
        userService.merge(user);
        User responseUser = userService.findById(user.getId());
        response = new Response(ResponseType.OK, "Обновление прошло успешно!",
gson.toJson(responseUser));
        return response;
    }

    public Response register(User user) {
        Response response;
        if (userService.findAll().stream().noneMatch(x ->
x.getLogin().equalsIgnoreCase(user.getLogin()))) {
            user.setPassword(PasswordEncryptor.hashPassword(user.getPassword()));
            userService.persist(user);
            response = new Response(ResponseType.OK, "Регистрация прошла успешно!",
""");
        } else {
            response = new Response(ResponseType.ERROR, "Такой пользователь уже
существует!", "");
        }
        return response;
    }

    public Response login(User user) {
        Response response;
        if (userService.findAll().stream().anyMatch(x ->
x.getLogin().equalsIgnoreCase(user.getLogin()))) {
            if (userService.findAll().stream().anyMatch(x ->
PasswordEncryptor.verify(user.getPassword(), x.getPassword()))) {
                Optional<User> optionalUser = userService.findAll().stream().filter(x
-> x.getLogin().equalsIgnoreCase(user.getLogin())).findFirst();
                if (optionalUser.isPresent()) {
                    User authorizedUser = optionalUser.get();
                    authorizedUser = userService.findById(authorizedUser.getId());
                }
            }
        }
        return response;
    }
}
```

Продолжение приложения Б

```
        response = new Response(ResponseType.OK, "Авторизация прошла успешно!", gson.toJson(authorizedUser));
    } else {
        response = new Response(ResponseType.ERROR, "Такого пользователя не существует!", "");
    }
    } else {
        response = new Response(ResponseType.ERROR, "Неправильный пароль!", "");
    }
    } else {
        response = new Response(ResponseType.ERROR, "Такого пользователя не существует!", "");
    }
    return response;
}

public Response findAllUsers() {
    Response response;
    if (! userService.findAll().isEmpty()) {
        response = new Response(ResponseType.OK, "Список пользователей успешно получен", gson.toJson(userService.findAll()));
    } else {
        response = new Response(ResponseType.ERROR, "Список пользователей не получен", "");
    }
    return response;
}

public Response deleteUser(User user) {
    Response response;
    if (userService.findAll().stream().anyMatch(x -> Objects.equals(x.getId(), user.getId()))) {
        List<Account> accounts = accountService.findById(user.getId());
        if (accounts.isEmpty()) {
            userService.remove(user);
            response = new Response(ResponseType.OK, "Удаление прошло успешно!", "");
        } else {
            for (Account account : accounts) {
                List<Transaction> transactions = transactionService.findByAccount(account);
                for (Transaction transaction : transactions) {
                    if (Objects.equals(transaction.getSender().getId(), account.getId())) {
                        transaction.setSender(null);
                    } else if (Objects.equals(transaction.getReceiver().getId(), account.getId())) {
                        transaction.setReceiver(null);
                    }
                }
                transactionService.merge(transaction);
            }
            accountService.remove(account);
        }
    }
    userService.remove(user);
    response = new Response(ResponseType.OK, "Удаление прошло успешно!", "");
} else {
    response = new Response(ResponseType.ERROR, "Такого пользователя не существует!", "");
}
```

Продолжение приложения Б

```
    }  
    return response;  
}  
}
```

Класс InterestRateHandler:

```
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class InterestRateHandler {  
    private InterestRateService interestRateService;  
    private Gson gson;  
  
    public Response updateInterestRate(InterestRate interestRate) {  
        interestRateService.merge(interestRate);  
        return new Response(ResponseType.OK, "Процентная ставка успешно обновлена",  
gson.toJson(interestRateService.findById(interestRate.getId())));  
    }  
  
    public Response findAllInterestRates() {  
        List<InterestRate> interestRates = interestRateService.findAll();  
        if (interestRates != null) {  
            return new Response(ResponseType.OK, "Список процентных ставок успешно        }  
        return new Response(ResponseType.ERROR, "Список процентных ставок не получен",  
"");  
    }  
}
```


ПРИЛОЖЕНИЕ В
(обязательное)
Листинг скрипта генерации базы данных

```
create table interest_rates
(
    type tinyint null,
    value double not null,
    id bigint auto_increment
        primary key,
    check (`type` between 0 and 1)
);

create table person_data
(
    age int null,
    Id bigint auto_increment
        primary key,
    email varchar(255) null,
    firstName varchar(255) null,
    lastName varchar(255) null,
    phoneNumber varchar(255) null,
    sex varchar(255) null
);

create table users
(
    role tinyint null,
    id bigint auto_increment
        primary key,
    person_data_id bigint null,
    login varchar(255) null,
    password varchar(255) null,
    constraint UKdcghoi5v9l2edcvtosk96k0c
        unique (person_data_id),
    constraint FK89cgxw185b3qd4oibj3vh67cp
        foreign key (person_data_id) references person_data (Id),
    check (`role` between 0 and 2)
);

create table accounts
(
    balance double not null,
    created date null,
    status tinyint null,
    type tinyint null,
    id bigint auto_increment
        primary key,
    owner_id bigint null,
    rate_id bigint null,
    number varchar(255) null,
    constraint FKjln86358moqf5k5pw89oiq8ur
        foreign key (owner_id) references users (id),
    constraint FKnrwymjhlvpfnv3ec6plbabf2
        foreign key (rate_id) references interest_rates (id),
    check (`status` between 0 and 3),
    check (`type` between 0 and 1)
);

create table transactions
```

Продолжение приложения В

```
(
    amount      double      not null,
    date         date        null,
    id           bigint auto_increment
           primary key,
    receiver_id  bigint      null,
    sender_id    bigint      null,
    message      varchar(255) null,
    constraint FKep3ko5p4fdvnw79p1uhw2q6nf
           foreign key (sender_id) references accounts (id),
    constraint FKjord7to517f34oa9ni6puyt85
           foreign key (receiver_id) references accounts (id)
);
```