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

Факультет инженерно-экономический
Кафедра экономической информатики
Дисциплина «Программирование сетевых приложений»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
старший преподаватель
_____ Т.М. Унучек
___.____.2021

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему:

# «ПРОГРАММНАЯ ПОДДЕРЖКА УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ»

БГУИР КП 1-40 01 02-08 025 ПЗ

Выполнил студент группы 914301
ОЛЕШКЕВИЧ Дарья Андреевна

_____
(подпись студента)

Курсовой проект представлен на проверку 01.12.2021

_____
(подпись студента)

Минск 2021

РЕФЕРАТ

БГУИР КП 1-40 05 01 025 ПЗ

Олешкевич, Д.А. Программная поддержка управления взаимодействием с клиентами в банковской сфере: пояснительная записка к курсовому проекту / Д.А. Олешкевич. – Минск: БГУИР, 2021. – 62 с.

Пояснительная записка 62 с., 25 рис., 11 источников, 5 приложений

РЕДАКТИРОВАНИЕ, РАСЧЁТ, СИСТЕМА ВКЛАДОВ, ВКЛАД, РЕДАКТИРОВАНИЕ ДАННЫХ, ПОЛЬЗОВАТЕЛЬ, АДМИНИСТРАТОР, БАНК, ПРЕДПРИЯТИЕ, ПРОСМОТР ДАННЫХ, ДОБАВЛЕНИЕ, УДАЛЕНИЕ, БАЗА ДАННЫХ, ОТЧЁТЫ

Цель проектирования: проектирование графического пользовательского интерфейса для управления взаимодействием с клиентами в банковской сфере.

Методология проведения работы: в процессе решения поставленных задач использованы принципы системного подхода, теория разработки JavaFX, аналитические методы, методы компьютерной обработки экспериментальных данных, методы работы с базой данных.

Результаты работы: выполнен анализ литературно-патентных исследований, изучена область разработки, рассмотрено общетехническое обоснование разработки электронного модуля; проведён анализ спроектированной программы; осуществлено моделирование физических процессов, протекающих в процессе работы программы, разработана графическая часть проекта.

Модуль предназначен для расчёта прибыли с вкладов, хранения вклада.

Структура предполагает формы с определёнными возможностями для каждого пользователя, она включает в себя создание нового вклада, редактировании информации о пользователях, авторизацию, анализ и обработку данных.

Область применения результатов: могут быть использованы в финансовой сфере.

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

В современном мире IT позволяет человеку, не выходя из дома, совершить необходимые ему действия. Например, через приложение заказать продукты на дом, купить чайник в интернет магазине и многое другое. Практически во всех, если не во всех, отраслях есть возможность внедрить IT для облегчения взаимодействия человека с предприятием.

Вклады являются одной из основополагающих сфер жизнеобеспечения населения. Предприятия этой отрасли наиболее массовые в сфере предлагаемых услуг, к качеству работы которых предъявляются самые современные и жесткие требования, как со стороны вкладчиков, так и со стороны руководителей. Их эффективная работа напрямую зависит не только от материально-технической базы, но и от уровня оснащения организаций информационными средствами на основе современных компьютерных систем и автоматизированного складского и бухгалтерского учета.

Необходимо, чтобы программное обеспечение позволяло хранить и передавать все необходимые объемы информации. Идеально, если процесс обмена данных максимально ускорен по времени и упрощён в обслуживании. Это напрямую повлияет на один из важнейших показателей деятельности банка — прибыль. Человек, пользуясь приложением, ждет от него высокой скорости обработки запроса. Если же пользователь долго ожидает ответ на свои действия в приложение это негативно сказывается на его восприятии не только приложения, но и самой фирмы (в нашем случае, банка). Сам по себе банк может обладать выгодными для человека предложениями, хорошими процентными ставками для человека, хорошее обслуживание. Однако, если приложение у такого хорошего банка будет медленно работать, иметь интуитивно не понятный для клиента интерфейс, репутация банка может пострадать. Клиент будет рассказывать всем о плохом приложении, о том, как долго обновляется баланс его карт, например, и перспективные клиенты предпочтут другой банк исключительно из-за удобного приложения.

Целью данного курсового проекта является оптимизация работы банковской системы, с помощью которой можно управлять вкладами физических лиц.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ СИСТЕМЫ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ

В этой работе проектируется база данных «Bank for you (вкладчики и вклады)».

Вклад – денежные средства, внесенные физическим или юридическим лицом в финансовое учреждение. Таким финансовым учреждением в условиях рассматриваемой предметной области является Bank for you.

Виды вкладов:

– бессрочные, их еще называют вкладами до востребования;
– долгосрочные и краткосрочные;
– с фиксированной или плавающей процентной ставкой;
– пополняемые и фиксированной величины;
– отзывные и безотзывные;
– со снятием процентов в течение срока вклада, без такой возможности;
– с простым начислением процентов и с капитализацией;
– вклады в одной валюте и мультивалютные;
– условные.

Почему люди выбирают именное такой способ хранения денег?

Не нужно обладать специфическими знаниями, следить за ситуацией на рынке (в отличии, например, от вложения в ценные бумаги). Когда человек сделал вклад, он может не волноваться о том, что его деньги пропадут или уменьшатся в количестве. Он уверен, что они вернутся ему. Также легко подобрать финансовую организацию – отчетность банков и отзывы клиентов есть в свободном доступе в сети. Человек может выбрать необходимые ему условия для хранения и выдачи его вклада, что позволит всем остаться довольными. А также, депозиты до 1,4 млн рублей застрахованы государством. Значит, даже при неблагоприятном стечении обстоятельств, клиент сможет вернуть свои деньги.

Процентная ставка фиксируется в договоре и не меняется в течение всего срока.

Физическое или юридическое лицо может внести вклад определенного вида в «Bank for you» на определенных условиях. При этом заключается некий договор, включающий в себя права, обязанности и ответственности сторон, а также характеристики вклада. Банк открывает счет и записывает определенную сумму, оговоренную вкладчиком. Дополнительно указывается срок, на который вносятся денежные средства.

Банк принимает депозит и может распоряжаться вложенными средствами как собственными. Например, в банк приходят клиенты, которым необходимо взять кредит на определенную сумму денег. Правительство страны не может выделить необходимое количество денег банку на выдачу кредитов всем нуждающимся в нем клиентам. Тем более, в стране не один банк, который в этом нуждается. Для решения этой проблемы и были придуманы вкладом. Таким образом, банк берет деньги клиента, сделавшего вклад, и выдает кредиты другим клиентам банка. От данной системы все получают выгоду: кредит получен, те, кто сделал вклад, получают прибыль от этого. Банк же получает прибыль от действий обоих типов клиентов.

Вся сумма депозита остается собственностью вкладчика и должна быть возвращена либо немедленно, по желанию клиента, либо в другой срок, обозначенный в договоре.

В «Bank for you», как и в любом финансовом учреждении, ведется архив, хранящий информацию о вкладах, вкладчиках и сделанных операциях. Архивы оконченных операций позволяют сформировать отчет о работе за весь год. Это необходимо банкам для понимания того, насколько часто пользуются определенными услугами, какие следует улучшить. Так же, данная система предназначена и для формирования годовых отчётов. Эти отчёты необходимо составлять каждому предприятию, которое нацелено на увеличение прибыли и дальнейшего функционирования на рынке предоставления услуг. Отчёты помогают выявлять проблемы предприятий. Например, можно отследить, какими услугами пользуются реже всего, какие услуги пользуются большим спросом, чтобы в новом году делать упор на них. Так же отчёты позволят привлечь новых клиентов. Как это работает? Допустим, анализ годовой работы отражает то, что суммарно со вкладов клиенты получили большую сумму денег. Банк может заказать рекламные баннеры, запустить рекламу в социальных сетях. Таким образом, люди, не являющиеся клиентами банка, увидят, что можно получить большую сумму денег. Психология человека устроена так, что если он видит, что кто-то другой получил большую сумму денег практически ничего для этого не сделав, он считает, что «а почему бы и мне так не сделать? Это же просто». Он не думает о том, что заработанные людьми деньги на рекламных баннерах отображается за тысячи человек. Он просто видит простой способ заработка и радостный идет в банк вкладывать свои деньги. Хотя в итоге его ожидания и реальный исход событий не совпадают.

У банков есть своя особенность – очереди. Человек, чаще всего, идет в банк после своей работы, так как с утра ему явно не до походов в банк. Из-за

одинаковой логики у людей в банках формируются «километровые» очереди, которые портят настроение клиентам. Даже если взять не окошко в банке, в котором ты можешь поменять валюту, к примеру, а взять информационный киоск, через который можно оплатить счета за воду, свет, отопление и прочее. Даже к нему всегда большая очередь. Это происходит чаще всего из-за того, что в очереди стоит старый человек, который не разбирается как работать с информационным киосков и из-за одного человека приходится ждать по полчаса или больше. А за этим клиентом в очереди стоит несколько человек, которым необходимо оплатить телефон, например, и эта операция занимает достаточно мало времени, половина из которого ожидание ответа от киоска.

Для решения этого вопроса нужно разработать информационную систему, которая включает в себя максимальное количество операций, которые можно выполнить сидя дома.

Это решение позволит улучшить отзывы банка, привлечь новых клиентов, так как людям свойственно делиться друг с другом новыми приложениями, а тем более если приложение позволяет быстро и качественно выполнить большое количество операций.

Хорошее приложение - одна из лучших реклам для банка.

Как итог: необходимо внедрение информационных технологий в сферу банковского обслуживания для улучшения взаимодействия с клиентами.

В настоящее время автоматизация банка и расчет финансового результата в нём, являются очень актуальным и необходимым. Внедрение программного продукта поможет организовать эффективную работу банка, взять под контроль финансовые потоки, контролировать работу менеджеров и управлять бизнесом без каких-либо проблем.

# 2 ПОСТАНОВКА ЗАДАЧИ ПРОЕКТИРОВАНИЯ И РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ

## 2.1 Детализация задач в области разработки системы

Для решения поставленной задачи необходимо разработать комфортную для использования IT-систему. При помощи которой пользователи банка смогут работать со вкладами.

В приложении необходимо реализовать:

– интуитивно понятный интерфейс для комфортного пользования приложением;

– реализовать клиент-серверное приложение;

– реализовать возможность пользователю сохранять информацию о всех своих депозитах в файл;

– авторизация пользователей (для разделения прав доступа администратору и пользователю);

– администратора, который имеет доступ к таблицам всех вкладов, пользователей, работников;

– обработка исключений, которые могут возникнуть при использовании приложения.

У администратора должны быть следующие возможности:

– видеть всех пользователей, редактировать их профили;

– видеть депозиты всех пользователей;

– добавлять в архив депозиты пользователей;

– просматривать архив;

– назначать должность пользователя;

– назначать нового администратора;

– просматривать всех пользователей, у которых есть должность.

У пользователя должны быть следующие возможности:

– регистрация;

– авторизация;

– создание вклада;

– просмотр всех своих активных вкладов;

– забрать вклад;

– получить итоговую прибыль от вклада;

– поиск вклада по величине вклада;

– редактировать свой профиль;
– обновлять страницу;
– получить информацию о всех своих вкладах в документе.

### 2.2 Обзор методов решения поставленных задач

Решение задача курсового проекта реализовано с помощью приложения, в основе работы которого лежит модель взаимодействия клиент-сервера на основе TCP соединения. Так же было реализовано GUI-приложение, которое работает с сервером, который посылает SQL-запросы базе данных. В качестве СУБД используется PosgreSQL.

Вся реализация производилась на языке Java. Java является объектно-ориентированным языком программирования и имеет в комплектации достаточно объемную библиотеку классов, которая значительно облегчает разработку программного обеспечения, предлагая программисту мощные средства решения распространенных задач. Для написания кода программы была использована среда разработки IntelliJ IDEA.

С помощью Draw.io были созданы UML-диаграммы. Выбор был остановлен на нём в связи с интуитивно-понятным интерфейсом: слева набор блоков, справа настройки внешнего вида и связей, в центре сам редактор.

В данной работе использовались два паттерна проектирования:
– singletone;
– delegate.

В курсовом проекте присутствует авторизация пользователей, возможность просмотра, редактирования, удаления данных и создания новых записей.

Для выполнения UML-моделей в стандарте IDEF0 использовалось CASE-средство CA AllFusion Process Modeler r7 (BPwin). Для информационного моделирования применялось средство CA AllFusion ERwin Data Modeler r7 (ERwin).

#### 2.2.1 Паттерн проектирования «Singleton»

Одиночка (Singleton, Синглтон) - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа. Одиночка позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан. В этом отличие одиночки от глобальных переменных.

Singleton является одним из самых популярных паттернов. Скорее всего, причина его популярности как раз и кроется в этой простоте — всего лишь один класс, ничего сложного. Это, наверное, самый простой для изучения и реализации паттерн. Если вы встретите человека, который только что узнал о существовании паттернов проектирования, можете быть уверены, что он уже знает про Singleton. Проблема заключается в том, что, когда из инструментов у вас есть только молоток, всё вокруг выглядит как гвозди. Из-за этого «Одиночкой» часто злоупотребляют.

В объектно-ориентированном программировании существует правило хорошего тона — (Single Responsibility Principle, первая буква в аббревиатуре SOLID). Согласно этому правилу, каждый класс должен отвечать лишь за один какой-то аспект. Совершенно очевидно, что любой Singleton-класс отвечает сразу за две вещи: за то, что класс имеет лишь один объект, и за реализацию того, для чего этот класс вообще был создан.

Принцип единственной обязанности был создан не просто так — если класс отвечает за несколько действий, то, внося изменения в один аспект поведения класса, можно затронуть и другой, что может сильно усложнить разработку. Так же разработку усложняет тот факт, что переиспользование (reusability) класса практически невозможно. Поэтому хорошим шагом было бы, во-первых, вынести отслеживание того, является ли экземпляр класса единственным, из класса куда-либо во вне, а во-вторых, сделать так, чтобы у класса, в зависимости от контекста, появилась возможность перестать быть Singleton’ом, что позволило бы использовать его в разных ситуациях, в зависимости от необходимости (т.е. с одним экземпляром, с неограниченным количество экземпляров, с ограниченным набором экземпляров и так далее).

### 2.2.2 Паттерн проектирования «Delegate»

Паттерн делегирования является поведенческим (behavioral) паттерном проектрования.

Это техника, в которой объект выражает определенное поведение снаружи, но в реальности делегирует ответственность за реализацию этого поведения связанному объекту.

Паттерн делегирования обеспечивает механизм отвлечения от реализации и контроля желаемого действия. Класс, вызываемый для выполнения действия, не выполняет его, а фактически делегирует вспомогательному классу. Потребитель не имеет или не требует знания

фактического класса, выполняющего действие, только контейнера, к которому производится вызов.

# 3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ

Описание данной предметной области произведено посредством построения IDEF0-модели в системе Erwin Process Modeler.

Главный концептуальный принцип методологии IDEF - представление любой изучаемой системы в виде набора взаимодействующих и взаимосвязанных блоков, отображающих процессы, операции, действия, происходящие в изучаемой системе. В IDEF0 все, что происходит в системе и ее элементах — это функции, каждая из которых ставится в соответствие блок, который представляет собой прямоугольник.

Для IDEF0 имеет значение сторона процесса и связанная с ней стрелка:

– слева входящая стрелка – вход бизнес-процесса – информация (документ) или ТМЦ, который будет преобразован в ходе выполнения процесса;

– справа исходящая стрелка – выход бизнес-процесса – преобразованная информация (документ) или ТМЦ;

– сверху входящая стрелка – управление бизнес-процесса – информация или документ, который определяет, как должен выполняться бизнес-процесс, как должно происходить преобразование входа в выход;

– снизу входящая стрелка – механизм бизнес-процесса – то, что преобразовывает вход в выход: сотрудники или техника. Считается, что за один цикл процесса не происходит изменения механизма.

Основными понятиями методологии и языка IDEF0 являются блок, ветвление, внутренняя, входная, выходная, граничная стрелки, декомпозиция, дерево узлов.

Блок – это прямоугольник, который содержит имя, номер. Он используется для описания функции.

Ветвление – это разделение стрелки на два и большее число сегментов. Внутренняя стрелка – это входная, выходная или управляющая стрелка, концы которой связывают источник и потребителя, являющиеся блоками одной диаграммы.

Входная стрелка – стрелка, которая отображает вход IDEF0-блока, то есть данные, которые будут преобразованы функцией в выход. Они располагаются с левой стороны блока.

Выходная стрелка – стрелка, которая отображает выход IDEF0-блока, то есть данные, преобразованные функцией. Они располагаются с правой стороны блока.

Граничная стрелка отображает связь диаграммы с другими блоками системы.

Декомпозиция – разделение моделируемой функции на функции-компоненты.

Дерево узлов – разделение моделируемой функции на функции-компоненты.

Методология описания бизнес-процессов IDEF0 наиболее широко используется, так как в ней рассматриваются логические отношения между работами.

Входными данными системы будет являться только вклад.

Результатом системы – выходом – будет служить рассчитанная стоимости вклада.

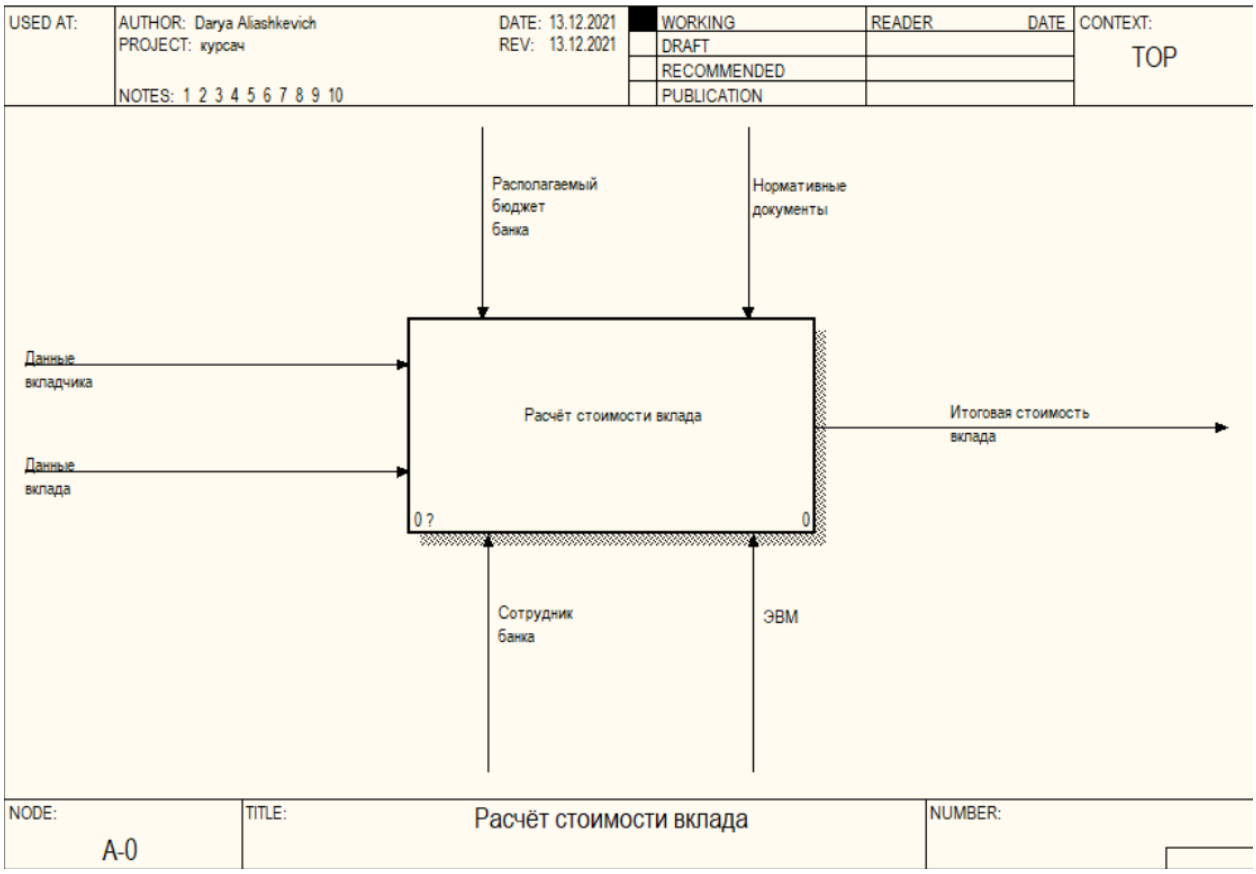



Рисунок 3.1 – Контекстная диаграмма расчёта вклада

Для того чтобы описать работу функционального блока более подробно, выполняется его декомпозиция, и моделируется диаграмма второго уровня,

которая представлена на рисунке 3.2. На ней представлено три функциональных блока декомпозиции.

Данный уровень включает в себя:

- анализ входных данных;
- выбор параметров расчета вклада;
- расчёт итоговой стоимости вклада.

Конечная цель системы: итоговая стоимость вклада.

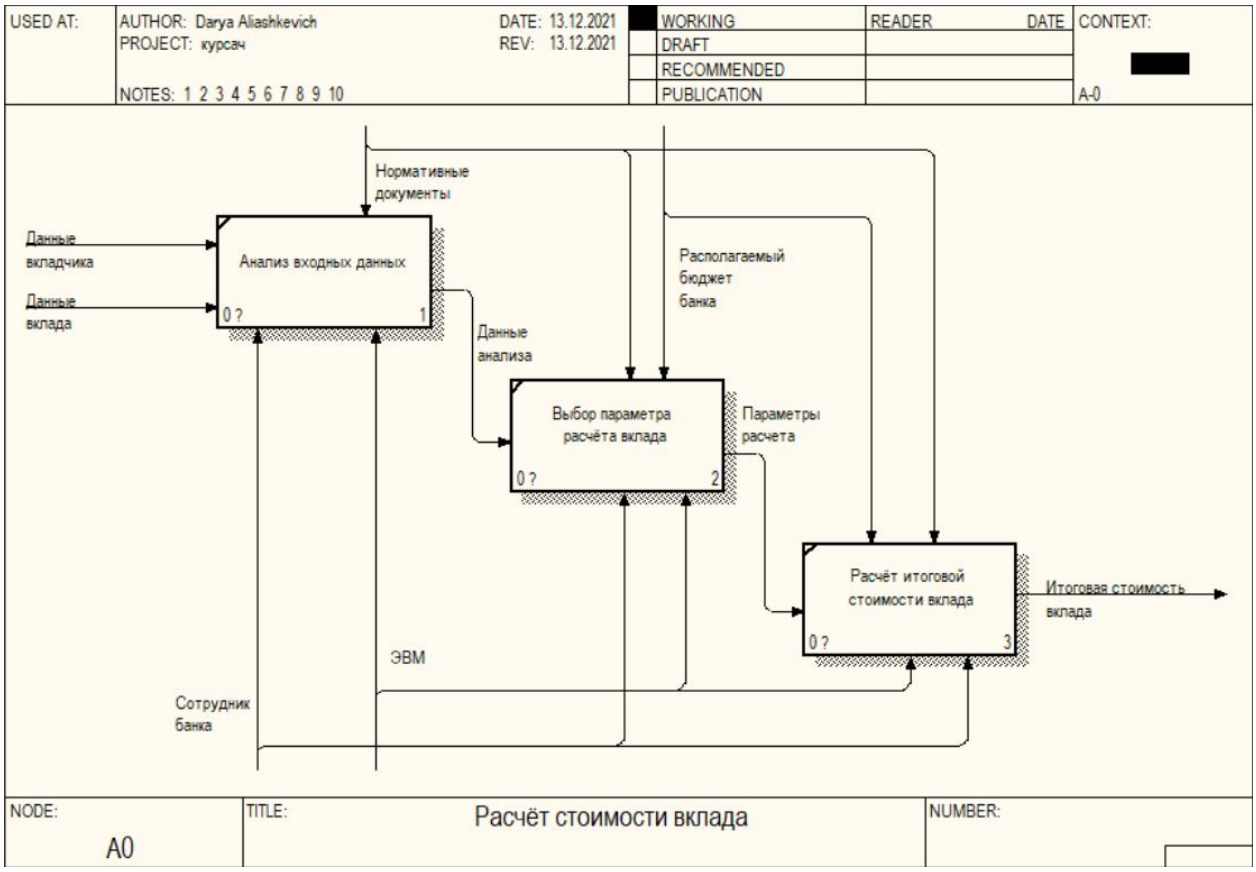



Рисунок 3.2 – Декомпозиция функционального блока

Далее следует декомпозиция процесса «Анализ входных данных», результат которой показан на рисунке 3.3.

На данном уровне выделены следующие подпроцессы:

- определение затрат на выдачу вклада;
- определение затрат на реализацию вклада;
- оценка прибыли от вклада.

Итоговая цель: данные анализа.

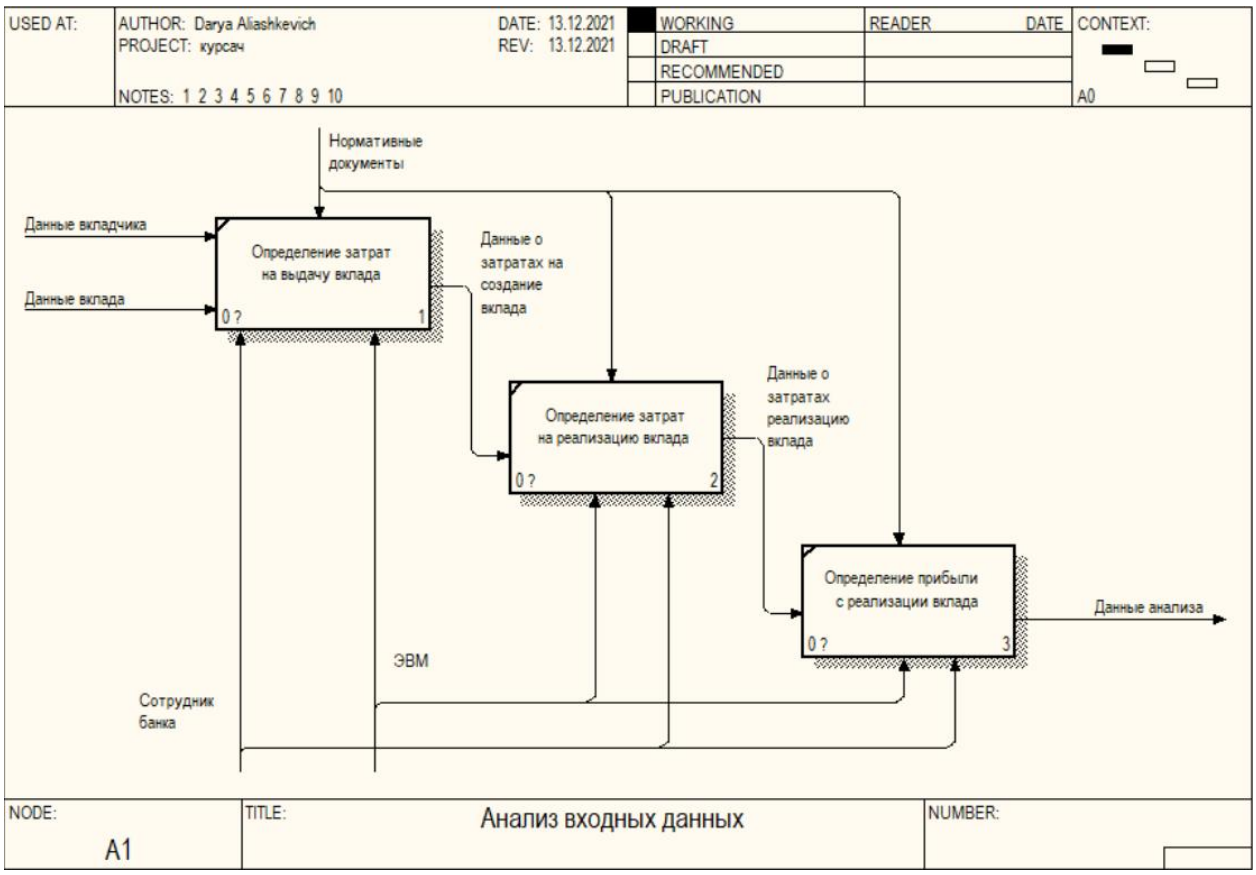



Рисунок 3.3 – Анализ входных данных

На следующем этапе разработки модели выполняется декомпозиция подпроцесса «Определение затрат на производство», показанная на рисунке 3.4.

В данном уровне существуют следующие подпроцессы:

- расчет стоимости вклада;
- расчет затрат при использовании определенной технологии;
- суммарный расчет вклада.

Итоговая цель подпроцесса: затраты на производство изделия.

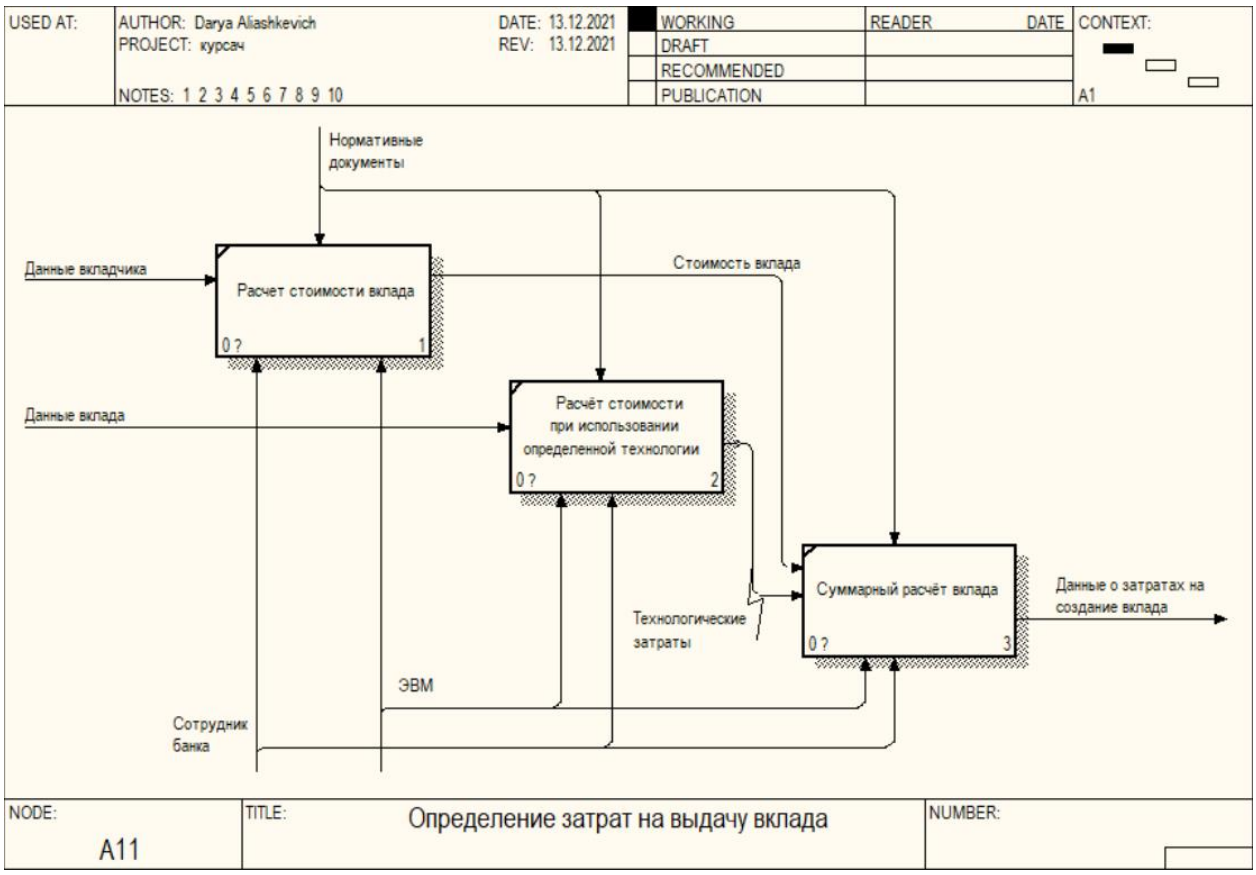



Рисунок 3.4 – Определение затрат на производство

В результате последовательного выполнения всех процессов происходит вычисление итоговой суммы прибыли.

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ

При проектировании системы автоматизации работы банка было сформировано 5 сущностей, а именно:

– пользователь;
– вклад;
– история вкладов;
– работник;
– сумма выдачи.

Сущность Пользователь нужна для авторизации пользователей и содержит в себе следующие атрибуты:

– user_id – хранит уникальный номер пользователя;
– user_email – логин пользователя, необходимый для авторизации;
– user_name – хранит имя пользователя;
– user_lastName – хранит отчество пользователя;
– user_password – пароль пользователя, нужный для авторизации;
– user_role – роль пользователя.

Сущность Вклад необходима для учета вкладов пользователей:

– contribution_id – уникальный номер вклада;
– contribution _userId – атрибут унаследованный от сущности Пользователь;
– contribution _deposit_amount – сумма вклада;
– contribution _percent – процентная ставка;
– contribution _currency – валюта вклада;
– contribution _collect_money – возможность забрать вклад;
– contribution _type_deposit – тип вклада;
– contribution _count_year – количество лет выдача вклада.

Сущность Работник содержит такие атрибуты как:

– worker_id – атрибут унаследованный от сущности Пользователь;
– worker_position – должность работника.

Сущность Сумма выдачи содержит такие атрибуты как:

– total_deposit_id – атрибут унаследованный от сущности Вклад;
– total_deposit_amount – сумма после выдачи вклада.

Сущность История вкладов содержит такие атрибуты как:

– history_contribution_id – атрибут унаследованный от сущности Вклад.

Рассмотрим разработанную логическую модель системы (см. рисунок 4.1).

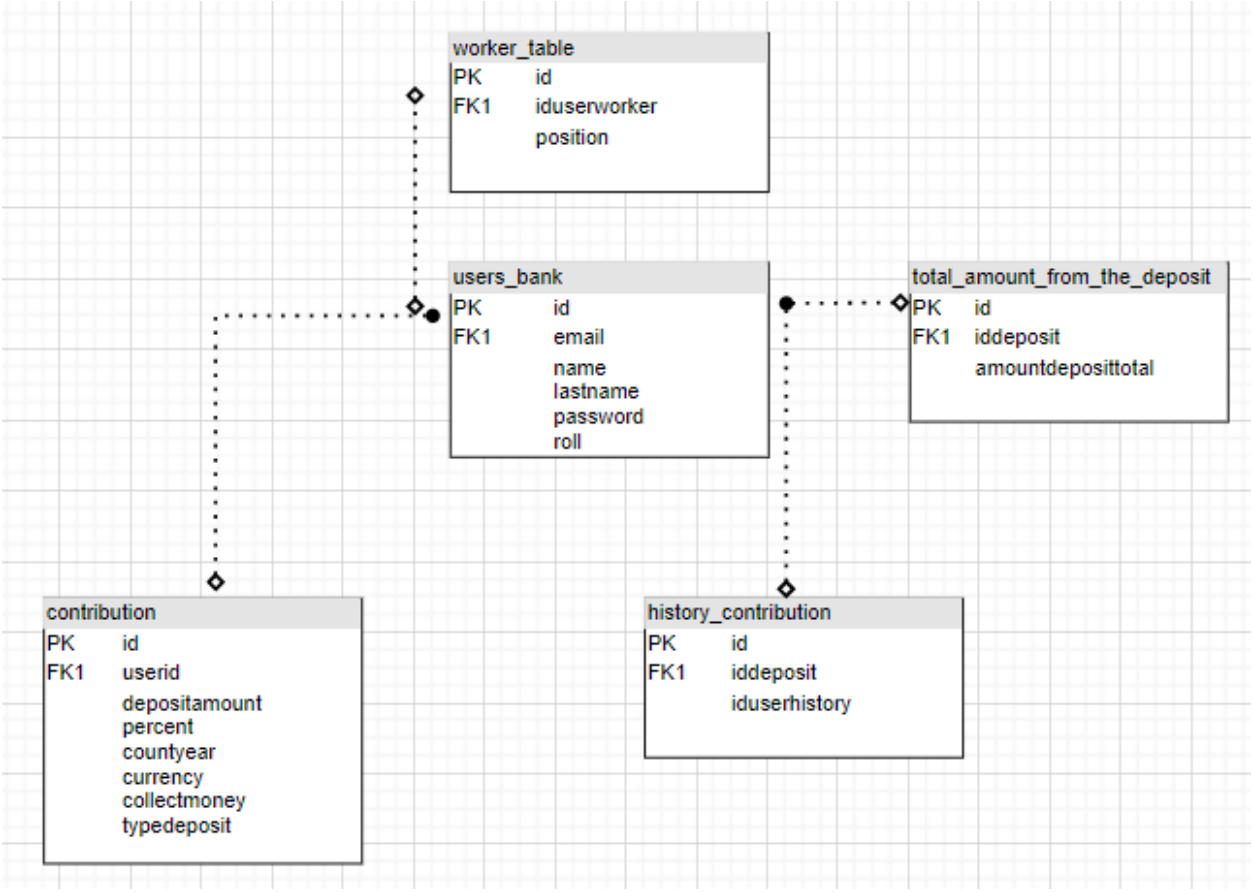



Рисунок 4.1 – Модель системы

Установленные стандарты позволяют избежать различной трактовки построенной модели. Данная информационная модель была успешно приведена к третьей нормальной форме, где каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

# 5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ

## 5.1 Диаграмма вариантов использования (use case diagram)

UML — это инструментарий моделирования, который используется для построения диаграмм.

Диаграммы прецедентов UML идеально подходят для:

- представление целей взаимодействия системы с пользователем;
- определения и организации функциональных требований в системе;
- указания контекста и требований системы;
- моделирования основного потока событий в сценарии использования.

Сценарий — это чёткая последовательность действий, которая показывает поведение. При разработке пользовательского интерфейса он описывает взаимодействие между пользователем (или категорией пользователей, например, администраторами системы, конечными пользователями) и системой. Такой сценарий состоит из последовательного описания комбинаций отдельных действий и задач (например, нажатий клавиш, щелчков по элементам управления, ввода данных в соответствующие поля и т. д.).

На диаграммах UML актёры изображаются в виде стилизованных человечков. Актеры — это пользователи (может быть человек, организация или внешняя система), которые взаимодействуют с системой.

Варианты использования представлены с помеченной овальной формой. Рисунки-палочки представляют актеров в процессе, а участие актера в системе моделируется линией между актером и сценарием использования. Граница системы рисуется с помощью рамки вокруг самого варианта использования.

Диаграмма прецедентов используется для просмотра поведения системы таким образом, чтобы:

- пользователь мог понять, как использовать каждый элемент;
- разработчик мог реализовать эти элементы.

Графическое представление диаграммы состояний представлено на рисунке 5.1.

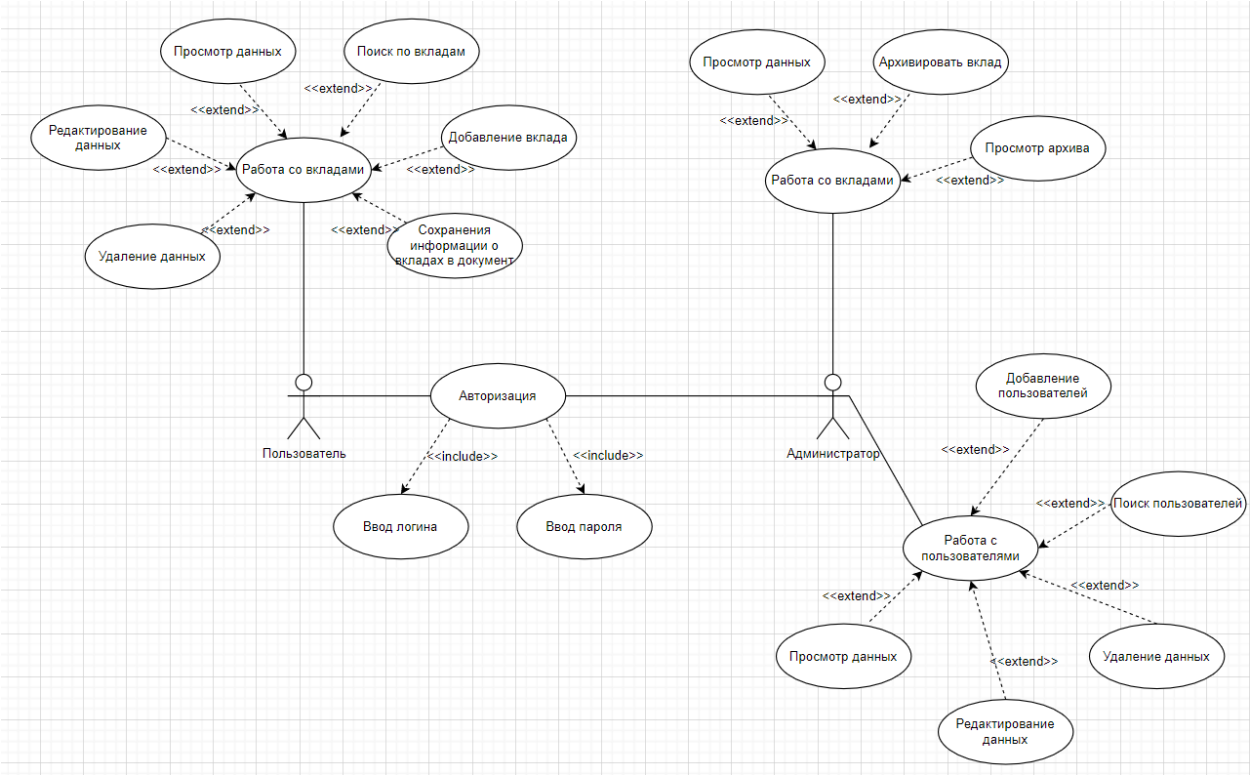



Рисунок 5.1 – Диаграмма вариантов использования

## 5.2 Диаграмма состояний (statechart diagram)

Диаграмма состояний — это диаграмма, которая используется для описания поведения системы с учетом всех возможных состояний объекта при возникновении события. Это поведение представлено и проанализировано в серии событий, которые происходят в одном или нескольких возможных состояниях. Каждая диаграмма представляет объекты и отслеживает различные состояния этих объектов по всей системе.

Каждая диаграмма состояний обычно имеет начало — темный круг, который обозначает начальное состояние, и конец— круг с рамкой, который обозначает конечное состояние. Но несмотря на наличие четких начальной и конечной точек, диаграммы состояний не обязательно являются лучшим инструментом для отслеживания общего развития событий. Скорее, они иллюстрируют конкретные виды поведения - в частности, переходы из одного состояния в другое.

Диаграммы состояний в основном изображают состояния и переходы. Состояния представлены прямоугольниками с закругленными углами. Переходы отмечены стрелками, которые переходят из одного состояния в другое, показывая, как изменяются состояния.

Графическое представление диаграммы состояний представлено на рисунке 5.2.

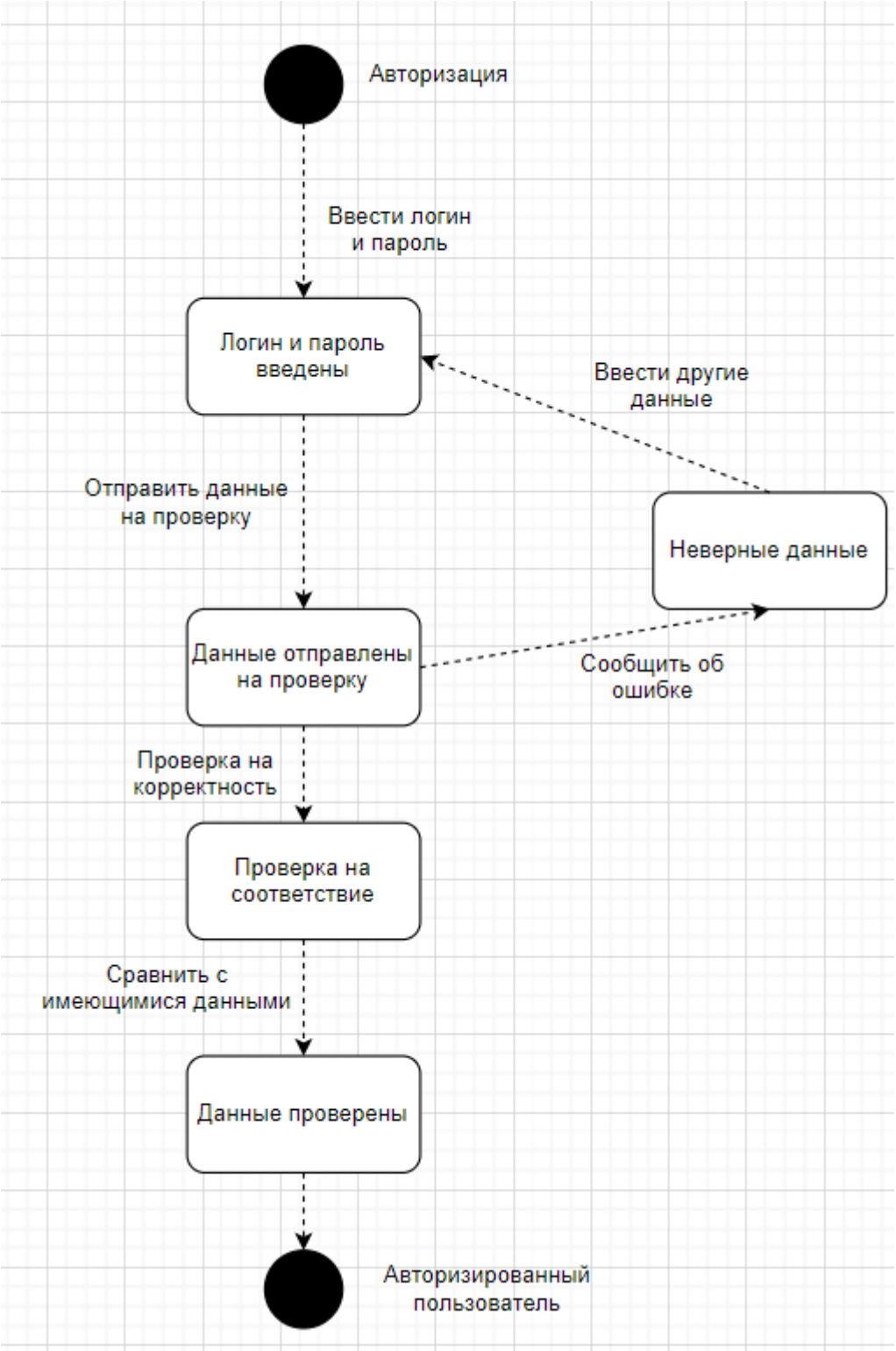



Рисунок 5.2 – Диаграмма состояний

## 5.3 Диаграмма последовательностей (Sequence diagram)

Диаграммы последовательности иногда называют диаграммами событий или сценариями событий, они описывают, как и в каком порядке группа объектов взаимодействуют. Эти диаграммы используются разработчиками программного обеспечения и бизнес-профессионалами для понимания требований к новой системе или для документирования существующего процесса.

Символ объекта представляет класс или объект в UML и демонстрирует, как объект будет вести себя в контексте системы. Атрибуты класса не должны быть перечислены в этой форме.

Коробка активации представляет время, необходимое объекту для выполнения задачи. Чем дольше будет выполняться задание, тем дольше будет окно активации.

Символ актера показывает объекты, которые взаимодействуют или являются внешними по отношению к системе.

С помощью стрелок и символов сообщения информация передается между объектами. Эти символы могут отражать начало и выполнение операции или отправку и прием сигнала.

Синхронный символ сообщения — это сплошная линия со сплошной стрелкой. Этот символ используется, когда отправитель должен дождаться ответа на сообщение, прежде чем оно продолжится. На диаграмме должны отображаться как звонок, так и ответ.

Асинхронный символ сообщения — это сплошная линия с подкладкой стрелки. Асинхронные сообщения не требуют ответа перед продолжением отправителя. Только диаграмма должна быть включена в диаграмму.

Символ асинхронного обратного сообщения представлен пунктирной линией с подкладкой стрелки.

Асинхронный символ создания сообщения представлен пунктирной линией с подкладкой стрелки. Это сообщение создает новый объект.

Символ ответного сообщения — это пунктирная линия со стрелкой на линии.

Удалить символ сообщения представлен сплошной линией со сплошной стрелкой, за которой следует X. Это сообщение уничтожает объект.

Графическое представление диаграммы последовательностей представлено на рисунке 5.3.

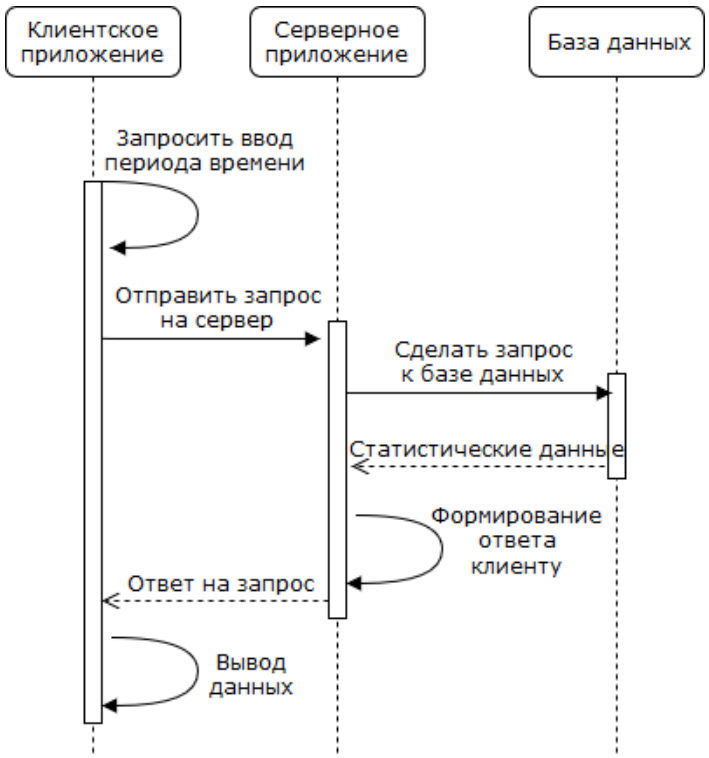



Рисунок А.3 – Диаграмма последовательности

## 5.4 Диаграмма компонентов (component diagram)

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Компонент (component) — элемент модели, представляющий некоторую модульную часть системы с инкапсулированным содержимым, спецификация которого является взаимозаменяемой в его окружении.

Графическое представление диаграммы последовательностей представлено на рисунке 5.4.

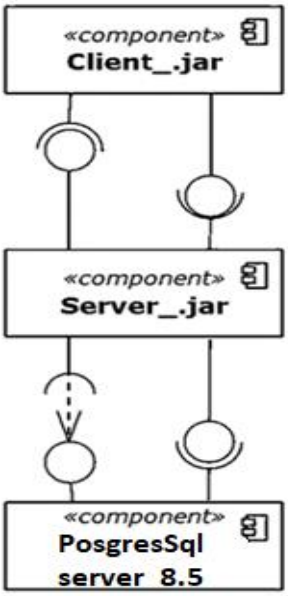



Рисунок 5.4 – Диаграмма компонентов

## 5.5 Диаграмма развертывания (deployment diagram)

Диаграмма развёртывания - один из доступных видов диаграмм, поддерживаемых Flexberry.

Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах компании, вызывают веб-сервисы, используют общие ресурсы и т. д. В таких случаях полезно иметь графическое представление инфраструктуры, на которую будет развернуто приложение. Для этого и нужны диаграммы развёртывания, которые иногда называют диаграммами размещения.

Графическое представление диаграммы последовательностей представлено на рисунке 5.5.

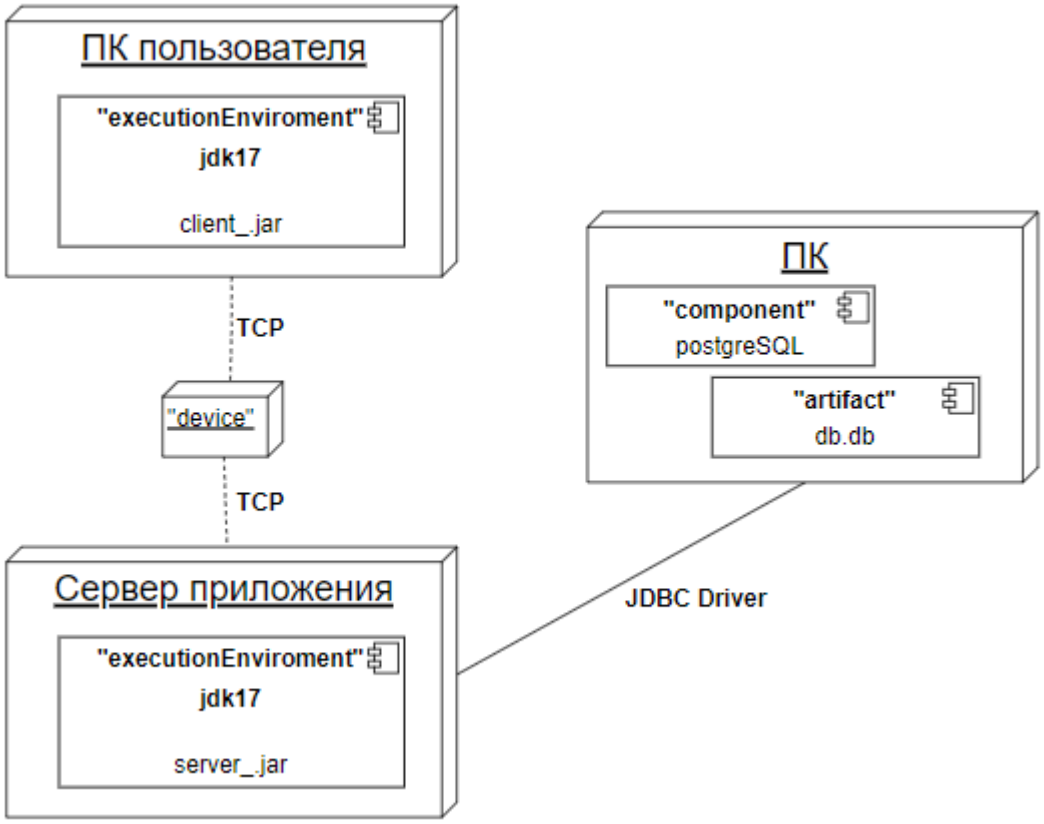



Рисунок 5.5 – Диаграмма развертывания

## 5.6 Диаграмма классов (static structure diagram)

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Целью создания диаграммы классов является графическое представление статической структуры декларативных элементов системы (классов, типов и т. п.) Она содержит в себе также некоторые элементы поведения (например — операции), однако их динамика должна быть отражена на диаграммах других видов (диаграммах коммуникации, диаграммах состояний). Для удобства восприятия диаграмму классов можно также дополнить представлением пакетов, включая вложенные.

При представлении сущностей реального мира разработчику требуется отразить их текущее состояние, их поведение и их взаимные отношения. На каждом этапе осуществляется абстрагирование от маловажных деталей и концепций, которые не относятся к реальности (производительность, инкапсуляция, видимость и т. п.).

Графическое представление диаграммы последовательностей представлено на рисунке 5.6.

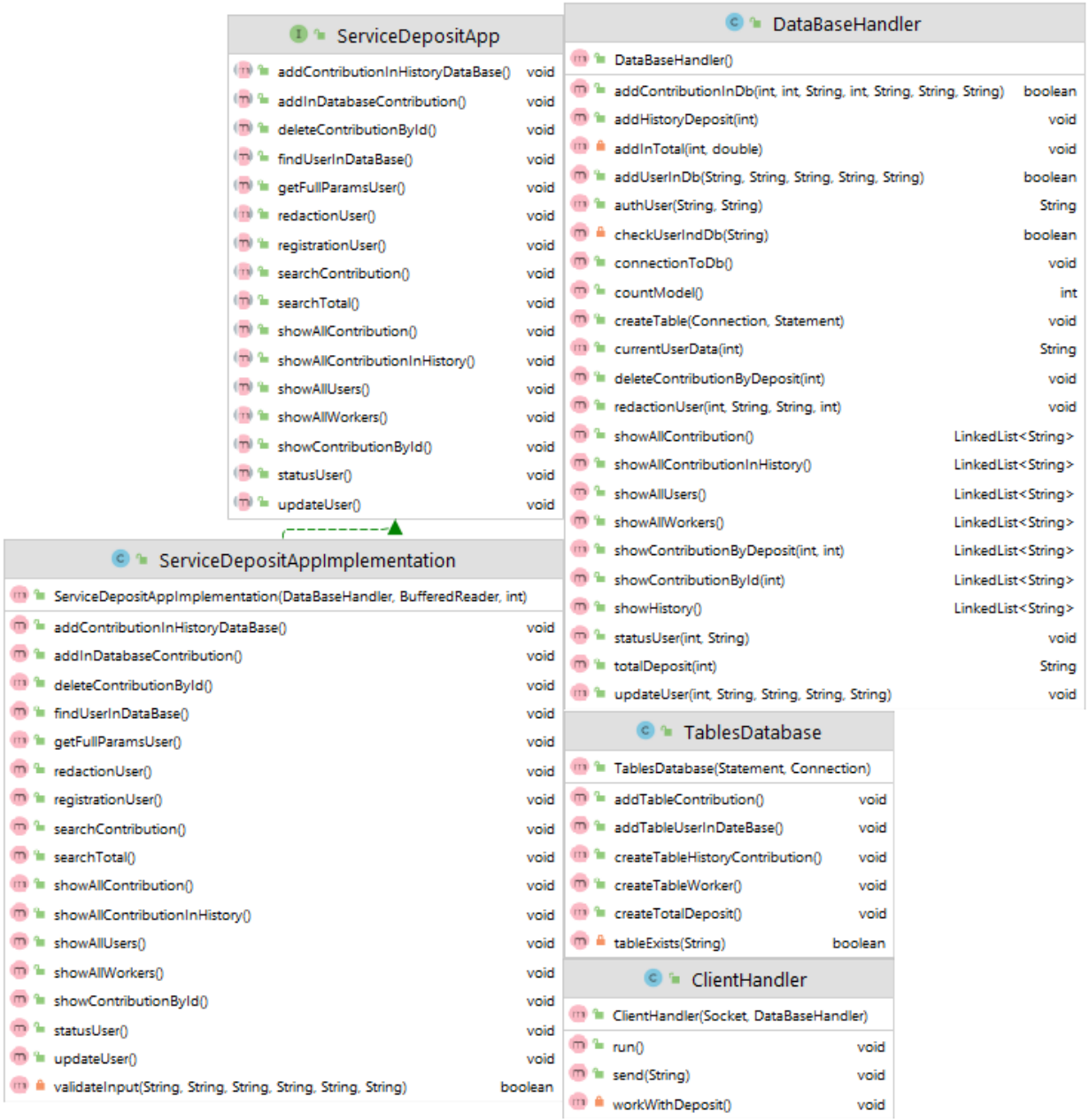



Рисунок А.6 – Диаграмма классов

# 6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ УПРАВЛЕНИЯ БАНКОВСКИМИ ВКЛАДАМИ ФИЗИЧЕСКИХ ЛИЦ

## 6.1 Алгоритм работы функции обработки входящего запроса

На сервере обработка входящих запросов реализована следующим образом: имеется функция, получающая входящие запросы, которые представляют из себя название команды. На основе данных запроса получает экземпляр класса, обрабатывающего запрос. Далее вызывается метод, обрабатывающий класс. Внутри данного метода производятся необходимые действия на основе запроса и возвращает результат, который далее передаётся клиенту.

Графическое представление алгоритма представлено на рисунке 6.1.


Рисунок 6.1 – Алгоритм работы функции обработки входящего запроса

### 6.2 Алгоритм работы функции поиска по вкладам

Имеется возможность осуществлять поиск по следующим сущностям: вклад. Для этого необходимо в клиентском приложении в соответствующей сущности нажать кнопку «Поиск». В окне необходимо ввести данные, на основе которых будет осуществляться поиск. Далее, по нажатию на кнопку «Поиск», будет отправлен соответствующий запрос на сервер, который в свою очередь сделает запрос к базе данных, получит ID сущностей, удовлетворяющих поисковому запросу, и вернёт их клиентскому приложению. Найденные сущности будут выведены в таблице.

Графическое представление алгоритма представлено на рисунке 6.2.


Рисунок 6.2 – Алгоритм работы функции поиска по вкладам

# 7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ЧАСТИ СИСТЕМЫ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ

Данное программное обеспечение представляет собой автоматизированную систему управления вкладами физических лиц. Чтобы войти в систему нужно воспользоваться окном авторизации пользователей, которое продемонстрировано на рисунке 7.1.

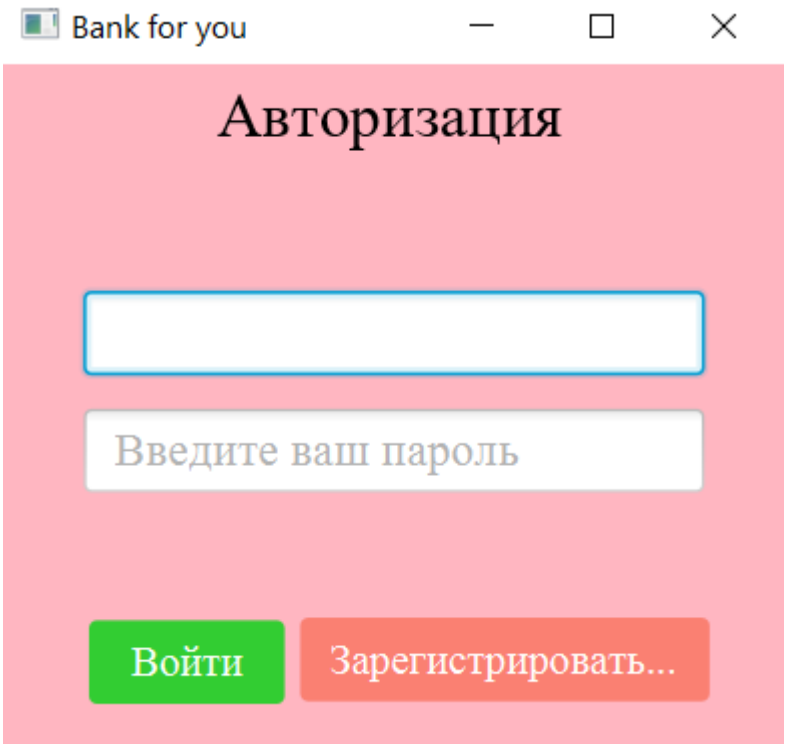



Рисунок 7.1 – Окно авторизации пользователей системы

После авторизации открывается главное окно выбранного пользователя для дальнейшей работы в системе. Рассмотрим сначала вариант работы программы для администратора. Он имеет доступ ко всем вкладкам.

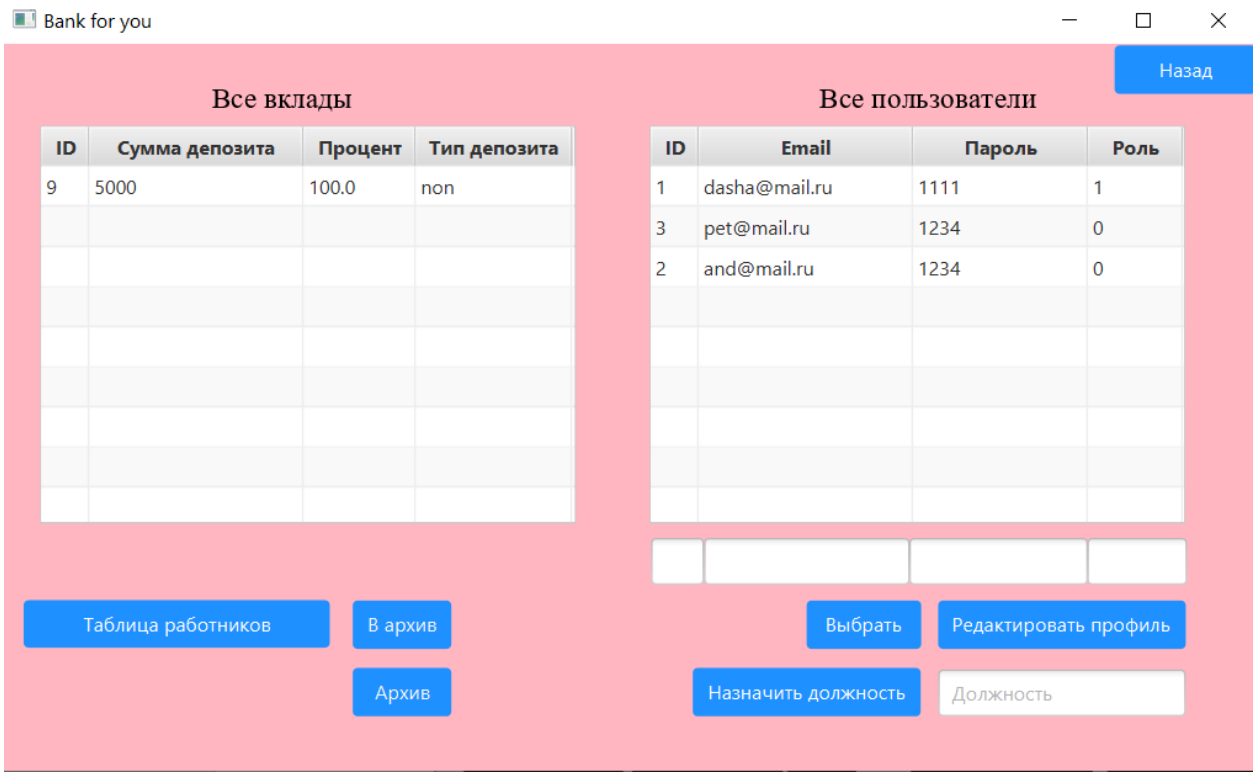



Рисунок 7.2 – Главное окно администратора системы

При переходе на вкладку «Работники» администратор видит информацию о том или ином работнике. Так же имеется возможность добавления, редактирования и удаления работника. Данное окно представлено на рисунке 7.3.

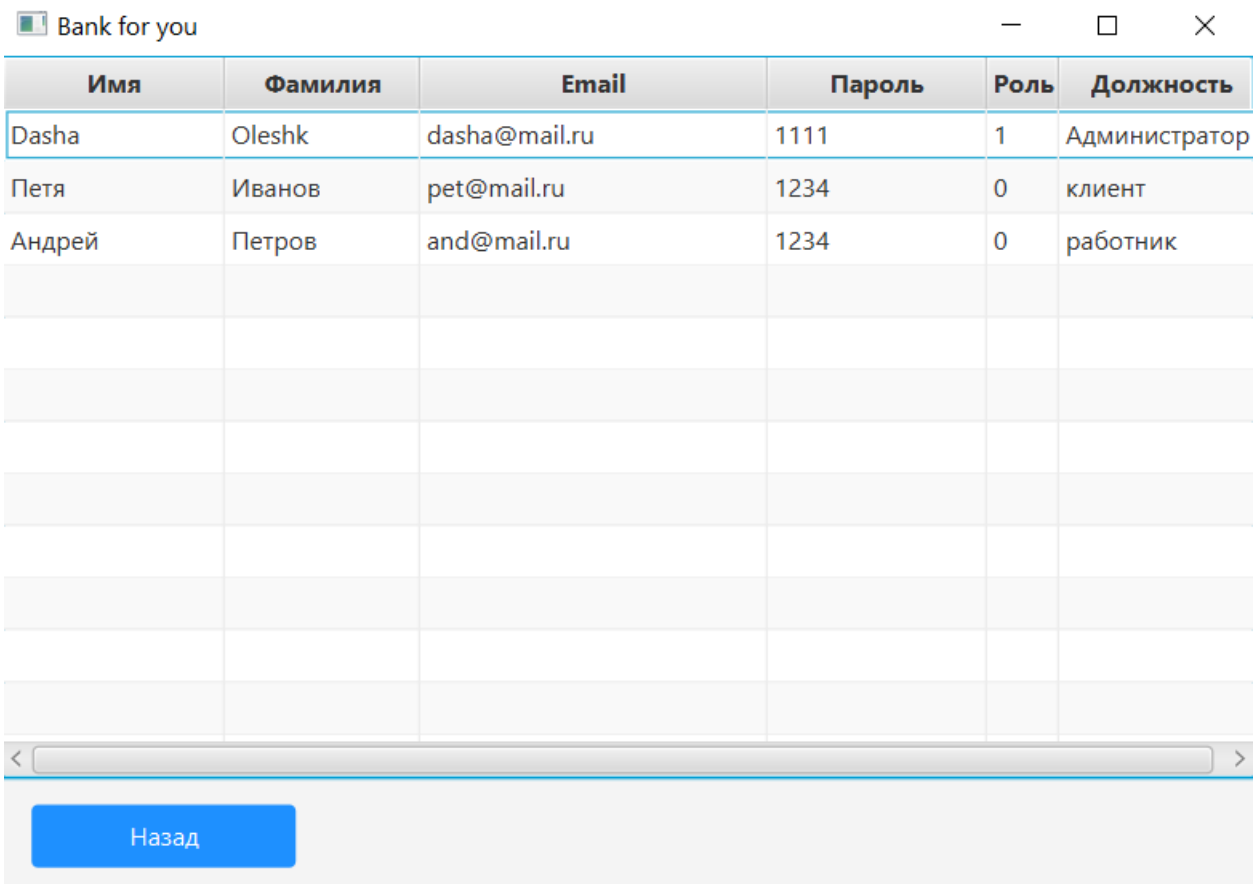



Рисунок 7.3 – Вкладка «Работники»

Администратор имеет возможность редактировать всех пользователей (рисунок 7.4).

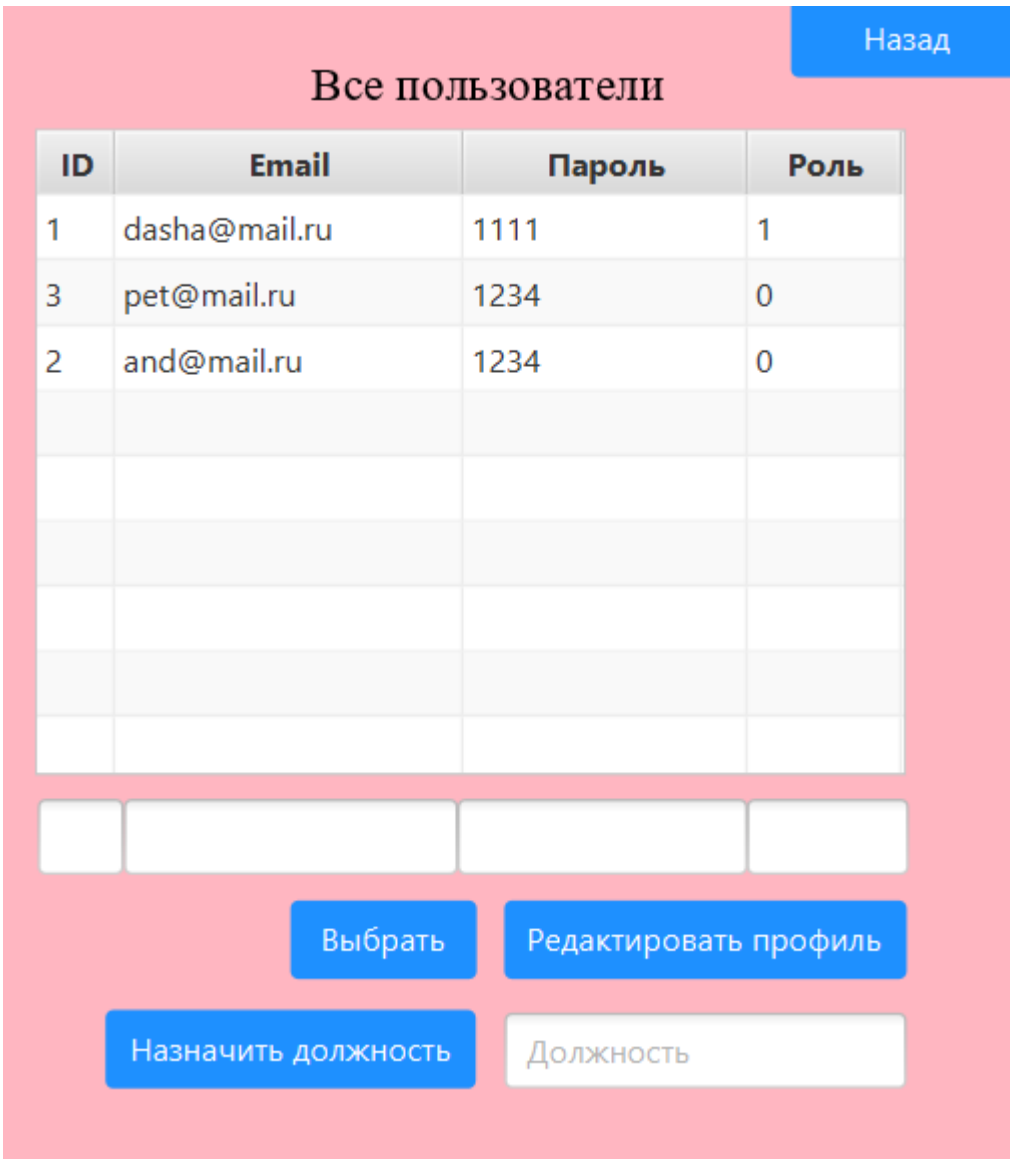



Рисунок 7.4 – Редактор всех пользователей

Администратор может добавлять в архив вклад любого пользователя. Таким образом, у пользователя не будет отображаться заархивированный администратором вклад. Администратор выбирает вклад и нажимает на «В архив». Все заархивированные вклады администратор может посмотреть, перейдя по кнопке «Архив». (рисунок 7.5).

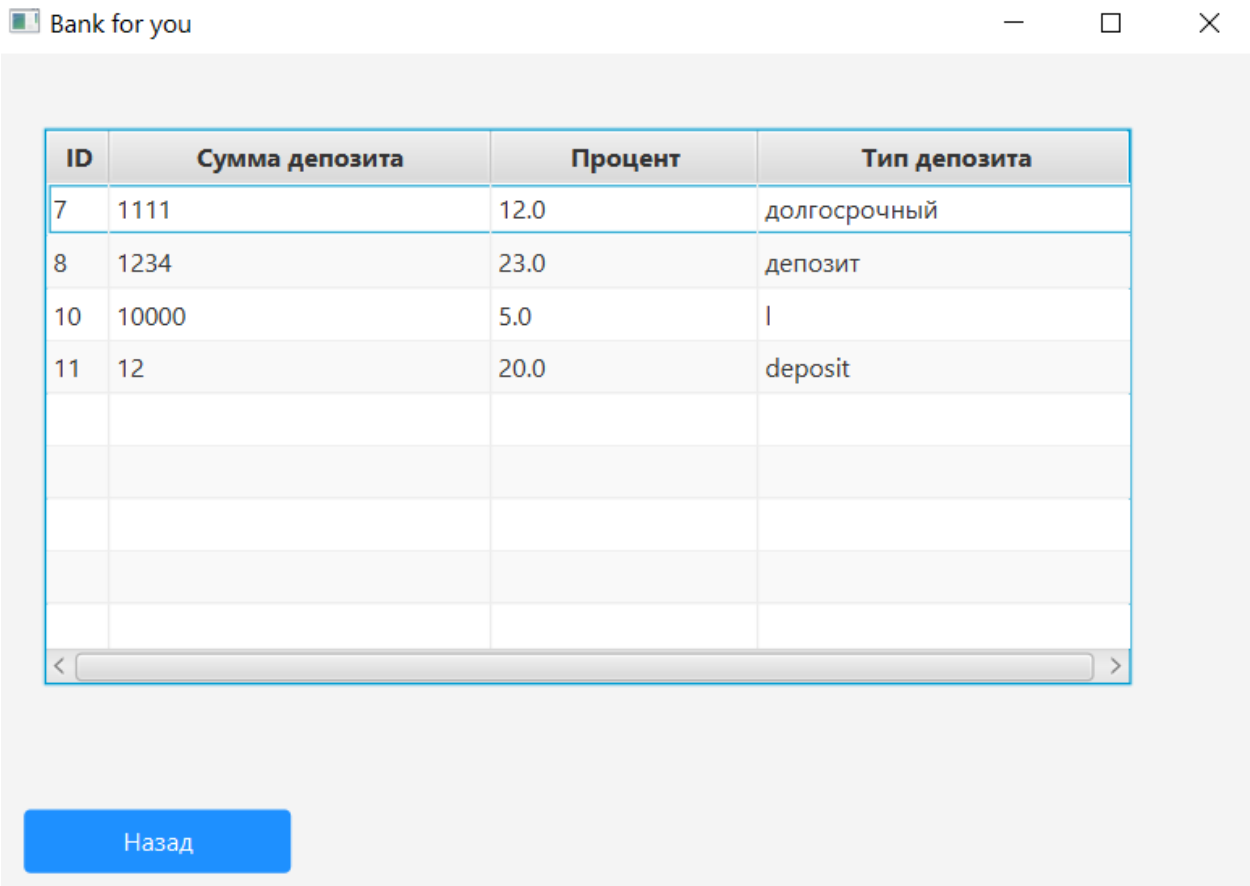



Рисунок 7.5 – Архив вкладов

Далее рассмотрим функционал программы при авторизации обычного пользователя.

Вкладка «Вклады» содержит в себе информацию о сумме вклада, процентной ставке, количество лет, валюта вклада. Имеется возможность редактирования, удаления, добавления новой информации, а также просмотр статистики (рисунок 7.6).

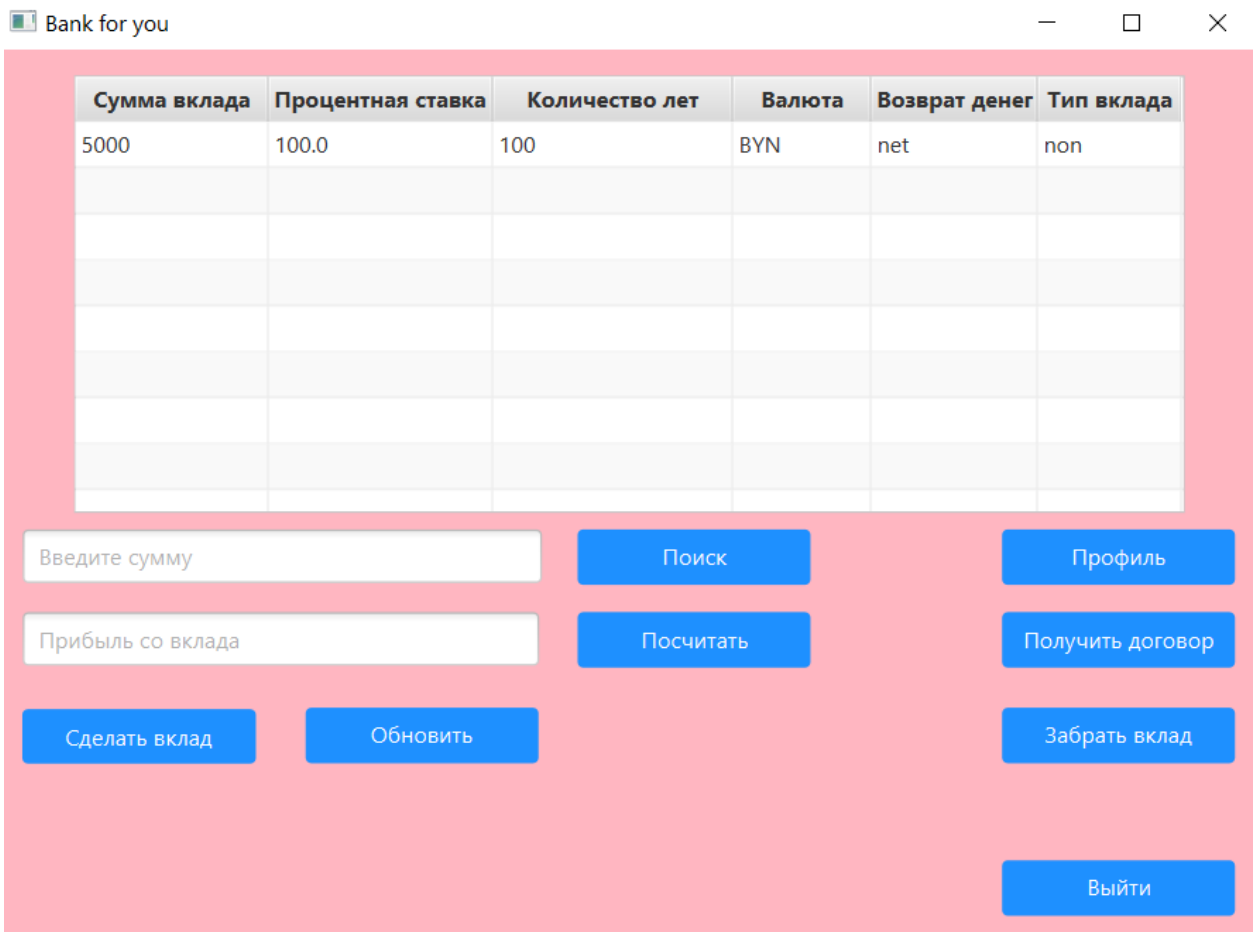



Рисунок 7.6 – Вкладка «Вклады»

Вкладка «Посчитать» подразумевает в себе информацию о вкладе. Так же имеется возможность создания нового вклада (рисунок 7.7).

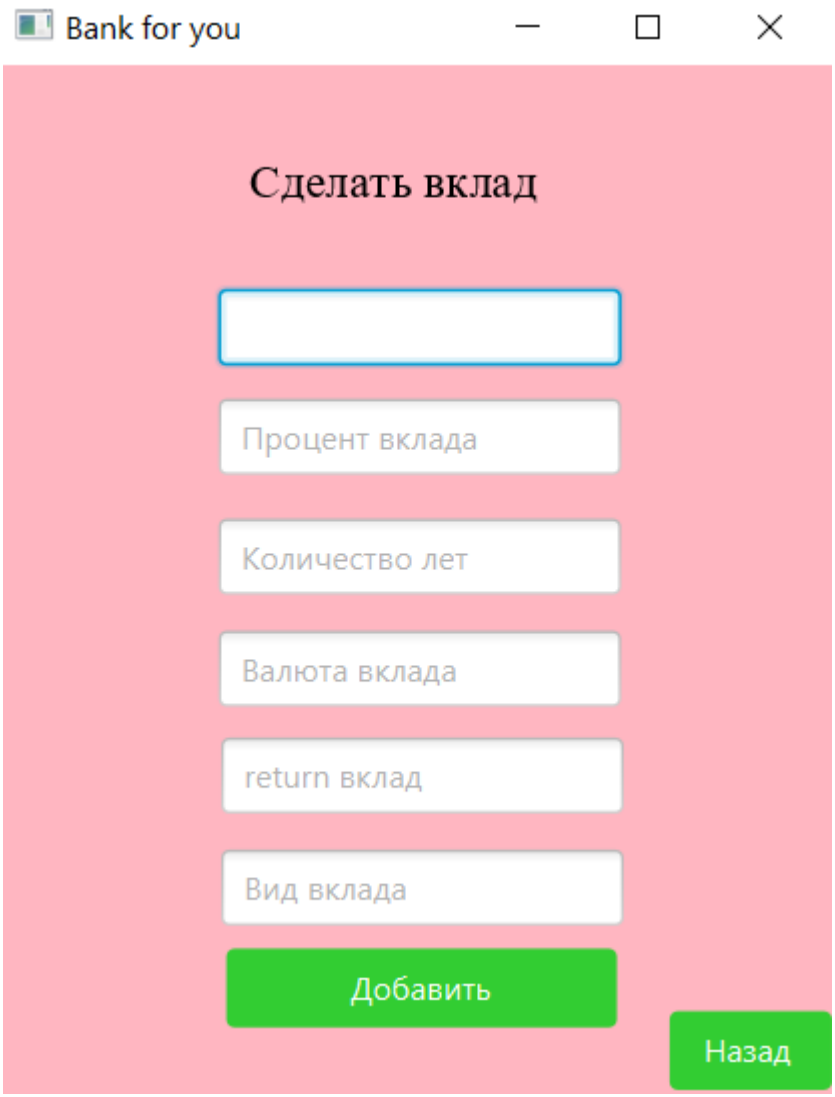



Рисунок 7.7 – Вкладка «Вклады»

Так же в данном окне есть возможность редактировать учетную запись текущего пользователя (рисунок 7.8)

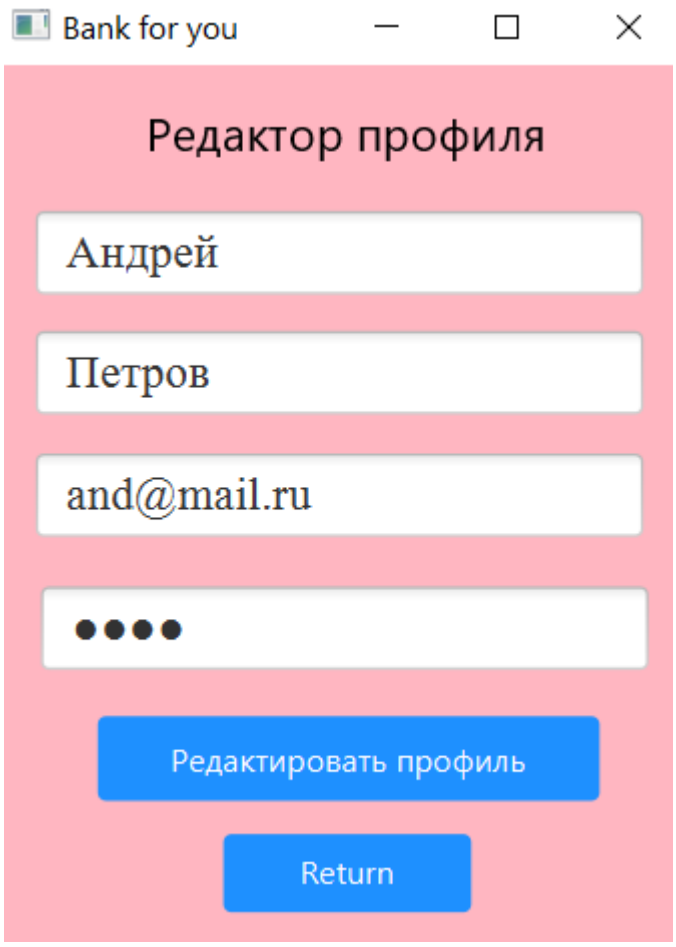



Рисунок 7.8 – Окно изменения учетной записи

# 8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВИЕМ С КЛИЕНТАМИ В БАНКОВСКОЙ СФЕРЕ

Тестирование систем – важный этап производства ПО, направленный на детальное изучение программного кода и выявление багов в работе продукта. Тестирование необходимо для того, чтобы пользователь при работе с системой не мог ввести данные, которые не может обработать система, так как это приведет к сбоям в ПО. Так же, при вводе неверных данных, пользователя необходимо уведомить о том, что его данные введены не верно и, желательно, точно указать на его ошибку, чтобы пользователю было понятно, что ему следует исправить.

Моделируя различные ситуации, было выполнено тестирование созданного программного обеспечения с целью:

– того, чтобы приложение в будущем работало правильно при любых обстоятельствах;

– предоставления актуальной информации о состоянии продукта на данный момент;

– того, чтобы конечный продукт соответствовал всем описанным требованиям.

Были выявлены различные исключительные ситуации и предусмотрена их обработка.

В случае если пользователь не заполнил одно или несколько полей окна авторизации или ввёл неверные логин и/или пароль, ему будет представлено сообщение об ошибке (рисунок 8.1).

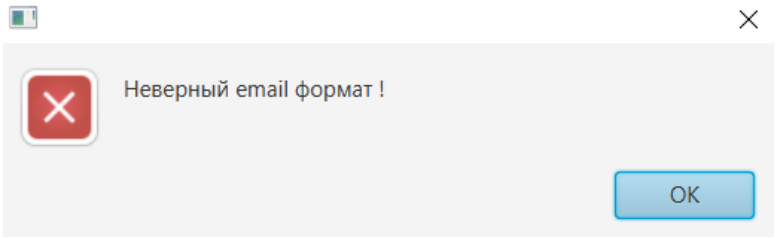



Рисунок 8.1 – Окно уведомления об ошибке «Не удалось авторизоваться»

Также предусмотрена обработка такой ошибки, как удаления вклада или добавления (рисунок 8.2, рисунок 8.3).

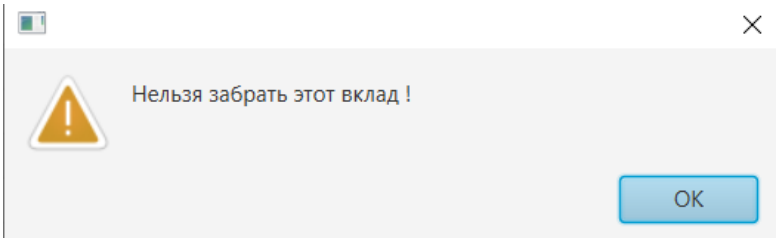



Рисунок 8.2 – Окно уведомления об ошибке «Нельзя удалить этот вклад»

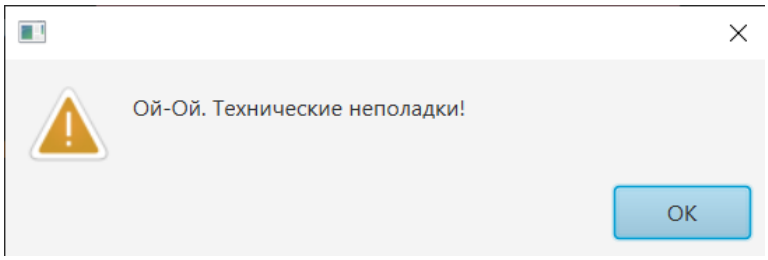



Рисунок 8.3 – Окно уведомления об ошибке «Ой-Ой. Технические неполадки!»

Итог: весь желаемый и описанный в предыдущих главах функционал приложения работает без сбоев.

# ЗАКЛЮЧЕНИЕ

Таким образом, была построена модель базы данных “Bank for you” по вкладчикам и вкладам. Метод сущность-связь был рассмотрен в конкретной предметной области, учтены возникающие спорные моменты, детально описаны степени связи и классы принадлежности, подробно расписаны атрибуты, их типы данных, английские эквиваленты и накладываемые ограничения.

Был разработан интуитивно-понятный пользователю интерфейс взаимодействия с системой банковских вкладов. Для удобства пользователя кнопки в интерфейсе программы расположены по блокам, относящимся к разным разделам работы программы: работа со вкладами и работа с аккаунтом пользователя.

Были включены функции, позволяющие пользователю быстро определить размер его прибыли со вклада. Пользователям, при работе с банковскими системами, важнее получить важную им информацию. В курсовом проекте были предусмотрены эти моменты, так что пользователь может в один клик получить нужную ему информацию: отчёт по всем его действующим вкладам, а также выгоду от любого вклада, выбранного из таблицы.

Для администратора так же был разработан и разбит по блокам удобный интерфейс. У администратора присутствует доступ к базе данных всех активных проектов, а также доступ ко всем зарегистрированным в системе пользователям. Администратор может присваивать должности клиентам, что позволит ему понимать «важность» пользователя при возникновении спорных ситуаций, связанных со вкладами пользователя. Администратор может архивировать нужный ему вклад, а так же просматривать архив. Это позволит в дальнейшем сформировать годовой, к примеру, отчёт по проделанной работе банка. Он видит, какие вклады были сделаны, какой плюс от этих вкладов получил банк и сколько клиентов остались довольными во время взаимодействия с банковским программным обеспечением и с самим банком. Так же администратор имеет возможность редактировать данные клиента, назначать нового администратора. Данная система разработана таким образом, что администратор имеет полный контроль над всей системой.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Эккель, Б. Философия JAVA : учеб. пособие / Б. Эккель – Санкт-Петербург: Питер, 2016. – 1168 с.

[2] Шилдт, Герберт Java 8. Руководство для начинающих: учеб. пособие / Герберт Шилдт – М.: ООО «И.Д. Вильямс», 2015. – 720 с.

[3] Иванов, Д. Моделирование на UML / Д. Иванов, Ф. Новиков – Санкт-Петербург: СПбГУ ИТМО, 2010. – 200 с.

[4] Методология IDEF0 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.expeducation.ru/ru/article/view?id=11003.

[5] Блок-схемы алгоритмов [Электронный ресурс]. – Электронные данные. – Режим доступа: https://pro-prof.com/archives/1462.

[6] MySQL Functions [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.w3schools.com/sql/sql_ref_mysql.asp.

[7] Паттерны [Электронный ресурс]. – Электронные данные. – Режим доступа: https://studfiles.net/preview/3640855/.

[8] Java [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/java/.

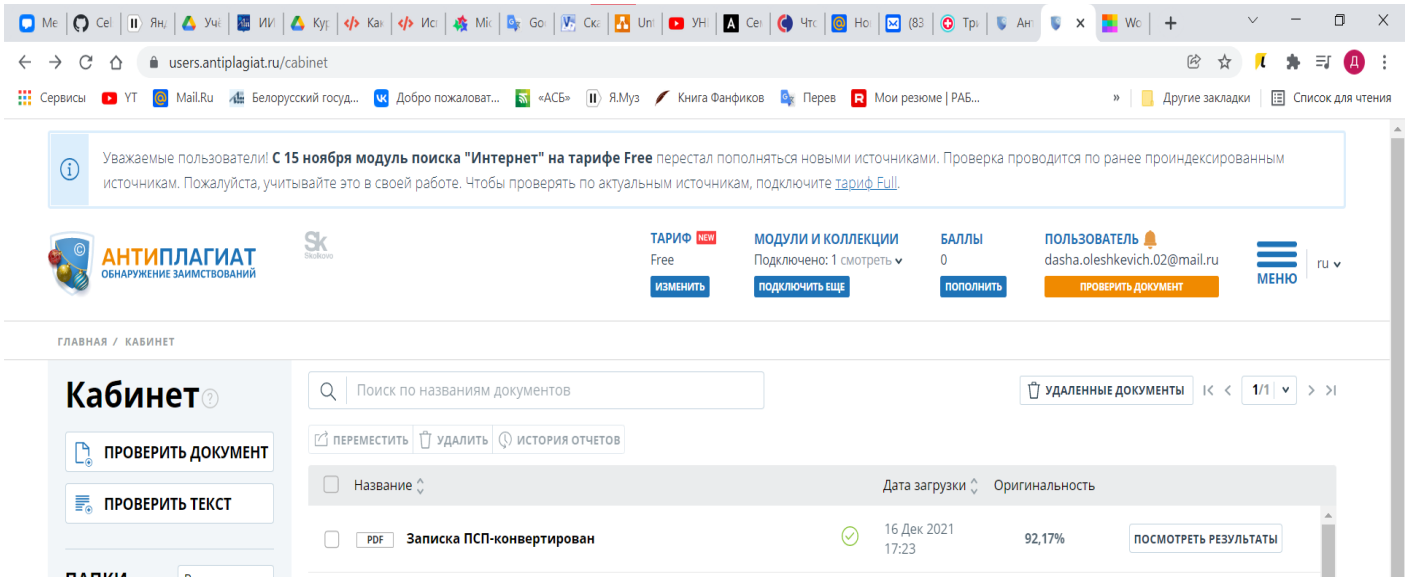
[9] Java [Электронный ресурс]. – Электронные данные. – Режим доступа: https://javarush.ru/.

[10] Вклады [Электронный ресурс]. – Электронные данные. – Режим доступа: https://myfin.by/wiki/term/vklad-depozit.

[11] Вклады [Электронный ресурс]. – Электронные данные. – Режим доступа: https://sovcombank.ru/blog/sberezheniya/chto-takoe-vklad-v-banke-kak-ego-otkrit.

# ПРИЛОЖЕНИЕ А
# (рекомендуемое)

## Антиплагиат



## ПРИЛОЖЕНИЕ Б
## (обязательное)
## Листинг алгоритмов, реализующих бизнес-логику

Элементы класса Contribution

```
package sample.cours.model;

public class Contribution {


    private int id;
    private int depositAmount;
    private double percent;
    private int countYear;
    private String currency;
    private String collectMoney;
    private String typeDeposit;

    public Contribution(int id,int depositAmount, double percent, int
countYear, String currency, String collectMoney, String typeDeposit) {
        this.id = id;
        this.depositAmount = depositAmount;
        this.percent = percent;
        this.countYear = countYear;
        this.currency = currency;
        this.collectMoney = collectMoney;
        this.typeDeposit = typeDeposit;
    }

    public Contribution(int id, int depositAmount, double
percent,String typeDeposit) {
        this.id = id;
        this.depositAmount = depositAmount;
        this.percent = percent;
        this.typeDeposit = typeDeposit;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCollectMoney() {
        return collectMoney;
    }

    public void setCollectMoney(String collectMoney) {
        this.collectMoney = collectMoney;
    }

    public String getTypeDeposit() {
        return typeDeposit;
```

```
    }

    public void setTypeDeposit(String typeDeposit) {
        this.typeDeposit = typeDeposit;
    }

    public int getDepositAmount() {
        return depositAmount;
    }

    public void setDepositAmount(int depositAmount) {
        this.depositAmount = depositAmount;
    }

    public double getPercent() {
        return percent;
    }

    public void setPercent(double percent) {
        this.percent = percent;
    }

    public int getCountYear() {
        return countYear;
    }

    public void setCountYear(int countYear) {
        this.countYear = countYear;
    }

    public String getCurrency() {
        return currency;
    }

    public void setCurrency(String currency) {
        this.currency = currency;
    }

    @Override
    public String toString() {
        return "Contribution : " +
                "depositAmount = " + depositAmount +
                ", percent = " + percent +
                ", countYear = " + countYear +
                ", currency = '" + currency + '\''+"\n";
    }
}
```

Элементы класса User

```
package sample.cours.model;

public class User {

    private String name;
    private String lastName;
```

```
    private String password;
    private String email;
    private int roll;
    private String position;



   private int idUser;
   public static User currentUser;


    public User(int idUser,String email, String password, int roll) {
        this.password = password;
        this.email = email;
        this.roll = roll;
        this.idUser = idUser;
    }

    public User(String name, String lastName, String password, String
email, int roll, int idUser, String position) {
        this.name = name;
        this.lastName = lastName;
        this.password = password;
        this.email = email;
        this.roll = roll;
        this.position = position;
        this.idUser = idUser;
    }

    public User(String email, int idUser, int roll) {
     this.email = email;
     this.idUser = idUser;
     this.roll = roll;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
```

```
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getRoll() {
        return roll;
    }

    public void setRoll(int roll) {
        this.roll = roll;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }


    public int getIdUser() {
        return idUser;
    }

    public void setIdUser(int idUser) {
        this.idUser = idUser;
    }
}
```

## ПРИЛОЖЕНИЕ В
## (обязательное)
## Листинг основных элементов программы

Подключение к базе данных и запросы

```
package database;

import constParams.Const;


import java.sql.*;
import java.util.LinkedList;

public class DataBaseHandler   {
    private  Connection connection;
    private  Statement statement;


    public DataBaseHandler() {
        connectionToDb();
        createTable(connection,statement);
    }



    public void connectionToDb(){
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        try {
            connection =
DriverManager.getConnection(Const.HOST_DATABASE+Const.NAME_DATABASE,
                    Const.USER_DATABASE,
                    Const.PASSWORD_DATABASE);
            statement= connection.createStatement();

            System.out.println("Database connection is done");

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


    public void createTable(Connection connection, Statement
statement){
        new TablesDatabase(statement,connection);
    }
```

```
    public boolean addUserInDb(String emailDb, String passwordDb,
String rollDb,String name,String lastName){
          if(!checkUserIndDb(emailDb)) {
              return false;
          }

            try {
                String query = " insert into users_bank (email,
password,roll,name,lastName )"
                        + " values ( ?, ?,?,?,?)";

                PreparedStatement preparedStmt =
connection.prepareStatement(query);
                preparedStmt.setString (1, emailDb);
                preparedStmt.setString (2, passwordDb);
                preparedStmt.setString (3, rollDb);
                preparedStmt.setString (4, name);
                preparedStmt.setString (5, lastName);


                preparedStmt.executeUpdate();

            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        return true;
    }

    private boolean checkUserIndDb(String emailUserDb){
        String query = "SELECT " + Const.ID  + " FROM " +
Const.USERS_TABLE +
                " WHERE " + Const.EMAIL + " = " + "'" + emailUserDb +
"'";
        ResultSet rs = null;
        int idInDb=0;
        try {
            rs = statement.executeQuery(query);
            while (rs.next()) {
               idInDb =  rs.getInt(Const.ID);

            }
            if(idInDb==0){
                return true;
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return false;
    }

    public String authUser(String email,String password) {
        String currentUser="";
        try {
            String query = "SELECT " + Const.EMAIL + "," +
Const.PASSWORD+","+Const.ID+","+Const.ROLL + " FROM " +
```

```
Const.USERS_TABLE +
                    " WHERE " + Const.EMAIL + " = " + "'" + email +
"'" + " AND " + Const.PASSWORD + " = " + "'" + password + "'";

            ResultSet rs = statement.executeQuery(query);
            while (rs.next()) {
                if (!rs.getString(Const.EMAIL).equals("") &&
                        !rs.getString(Const.PASSWORD).equals("")) {
                    currentUser+=rs.getString(Const.ID)+" ";
                    currentUser+=rs.getString(Const.ROLL);
                }
            }

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

        if(currentUser.equals("")) {
            return "false";
        }
        else {
            return currentUser;
        }
    }

    public boolean addContributionInDb(int userIdDb, int depositDb ,
String percent, int countYearDb ,String currencyDb
    ,String collectMoneyDb,String typeDepositDb){

        try {
            String query = " insert into " +Const.CONTRIBUTION_TABLE +
" (userId, depositAmount,percent
,countYEAR,currency,collectMoney,typeDeposit )"
                    + " values ( ?, ?,?,?,?,?,?)";

            PreparedStatement preparedStmt =
connection.prepareStatement(query);

            preparedStmt.setInt (1, userIdDb);
            preparedStmt.setInt (2, depositDb);
            preparedStmt.setString (3, percent);
            preparedStmt.setInt (4, countYearDb);
            preparedStmt.setString (5, currencyDb);
            preparedStmt.setString (6, collectMoneyDb);
            preparedStmt.setString (7, typeDepositDb);

            preparedStmt.executeUpdate();

            System.out.println("Данные в "+Const.CONTRIBUTION_TABLE);
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        double sumInOneYears =((Double.parseDouble(percent) *
depositDb) /100)*countYearDb;

        addInTotal(countModel(),sumInOneYears);
```

```
        return true;
    }

    private void addInTotal(int id,double amountDepositTotal){
        try {
            String query = " insert into " +Const.TOTAL_DEPOSIT + "
(idDeposit, amountDepositTotal)"
                    + " values ( ?, ?)";

            PreparedStatement preparedStmt =
connection.prepareStatement(query);

            preparedStmt.setInt (1, id);
            preparedStmt.setDouble (2, amountDepositTotal);
            preparedStmt.executeUpdate();

            System.out.println("Данные в "+Const.TOTAL_DEPOSIT);
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
    public LinkedList<String> showContributionById(int idUser){

        LinkedList<String> list = new LinkedList<>();
        String query = "SELECT "+Const.ID_CONTRIBUTION+" , " +
Const.DEPOSIT_AMOUNT+" , "+Const.PERCENT+" , "
                +Const.COUNT_YEAR+" ," +Const.CURRENCY+" , "
+Const.COLLECT_MONEY+" , " +Const.TYPE_DEPOSIT+ " FROM " +
Const.CONTRIBUTION_TABLE +
                " WHERE " + Const.USER_ID + " = " + "'" + idUser + "'"
+ " AND " + Const.ID_CONTRIBUTION + " NOT IN  (select " +
Const.ID_CONTRIBUTION_TOTAL + " from " + Const.HISTORY_CONTRIBUTION +
")";
        ResultSet rs = null;
        String contribution="";
        try {
            rs = statement.executeQuery(query);

            while (rs.next()) {

                contribution+=rs.getString(Const.ID_CONTRIBUTION)+" ";
                contribution+=rs.getString(Const.DEPOSIT_AMOUNT)+" ";
                contribution+=rs.getString(Const.PERCENT)+" ";
                contribution+=rs.getString(Const.COUNT_YEAR)+" ";
                contribution+=rs.getString(Const.CURRENCY)+" ";
                contribution+=rs.getString(Const.COLLECT_MONEY)+" ";
                contribution+=rs.getString(Const.TYPE_DEPOSIT)+" ";
                list.add(contribution);

                contribution="";
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return list;
    }
```

```
    public LinkedList<String> showContributionByDeposit(int
depositDb,int idUser){
        LinkedList<String> list = new LinkedList<>();
        String query = "SELECT "+Const.ID_CONTRIBUTION+" , " +
Const.DEPOSIT_AMOUNT+" , "+Const.PERCENT+" , "
                +Const.COUNT_YEAR+" ," +Const.CURRENCY+" , "
+Const.COLLECT_MONEY+" , " +Const.TYPE_DEPOSIT+ " FROM " +
Const.CONTRIBUTION_TABLE +
                " WHERE " + Const.DEPOSIT_AMOUNT + " = " + "'" +
depositDb + "'" +" AND "+ Const.USER_ID + " = " + "'" + idUser + "'" +
" AND " + Const.ID_CONTRIBUTION + " NOT IN  (select " +
Const.ID_CONTRIBUTION_TOTAL + " from " + Const.HISTORY_CONTRIBUTION +
")";
        ResultSet rs = null;
        String contribution="";
        try {
            rs = statement.executeQuery(query);

            while (rs.next()) {

                contribution+=rs.getString(Const.ID_CONTRIBUTION)+" ";
                contribution+=rs.getString(Const.DEPOSIT_AMOUNT)+" ";
                contribution+=rs.getString(Const.PERCENT)+" ";
                contribution+=rs.getString(Const.COUNT_YEAR)+" ";
                contribution+=rs.getString(Const.CURRENCY)+" ";
                contribution+=rs.getString(Const.COLLECT_MONEY)+" ";
                contribution+=rs.getString(Const.TYPE_DEPOSIT)+" ";
                list.add(contribution);
                contribution="";
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return list;
    }

    public int countModel()  {
     LinkedList<String> listContribution =showAllContribution();
        String[] subStr;
        subStr = listContribution.get(listContribution.size()-
1).split(" ");
     return Integer.parseInt(subStr[0]);
    }

    public void deleteContributionByDeposit(int id)  {

        String selectSQL = "DELETE FROM "+Const.CONTRIBUTION_TABLE +
" WHERE id = ?";
        try {
            connection.prepareStatement(selectSQL);
            PreparedStatement preparedStmt =
connection.prepareStatement(selectSQL);
            preparedStmt.setInt(1, id);
            preparedStmt.executeUpdate();
```

```
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
    public String totalDeposit(int id){
        String selectSQL = "SELECT "+Const.AMOUNT_TOTAL_DEPOSIT +"
FROM "+Const.TOTAL_DEPOSIT  +  " WHERE idDeposit = "+ "'" + id + "'" ;
        ResultSet rs = null;
        String contribution="";
        try {
            rs = statement.executeQuery(selectSQL);

            while (rs.next()) {

contribution+=rs.getString(Const.AMOUNT_TOTAL_DEPOSIT);

            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return contribution;

    }

    public String currentUserData(int id){
          Connection connection = null;
          Statement statement = null;

        try {
            connection = this.connection;
            statement= connection.createStatement();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        String currentUser="";
        try {
            String query = "SELECT " + Const.NAME_USER + " , " +
Const.LAST_NAME_USER+" , "+Const.PASSWORD +" , "+Const.EMAIL+" ,
"+Const.ROLL+ " FROM " + Const.USERS_TABLE +
                    " WHERE " + Const.ID + " = " + "'" + id + "'";

            ResultSet rs = statement.executeQuery(query);
            while (rs.next()) {

                    currentUser+=rs.getString(Const.NAME_USER)+" ";
                    currentUser+=rs.getString(Const.LAST_NAME_USER)+"
";
                    currentUser+=rs.getString(Const.PASSWORD)+" ";
                    currentUser+=rs.getString(Const.EMAIL)+" ";
                    currentUser+=rs.getString(Const.ROLL)+" ";
                    currentUser+=id;
            }

        } catch (SQLException e) {
            e.printStackTrace();
```

```
        }
        return currentUser;

    }

    public  void updateUser(int id,String name ,String lastName,String
email ,String password)  {

        String query = "UPDATE users_bank SET name  = ?, lastName = ?
,email = ?,password = ? WHERE id = ?";
        PreparedStatement preparedStmt = null;
        try {
            preparedStmt = connection.prepareStatement(query);
            preparedStmt.setString   (1, name);
            preparedStmt.setString(2, lastName);
            preparedStmt.setString(3, email);
            preparedStmt.setString(4, password);
            preparedStmt.setInt(5, id);
            preparedStmt.executeUpdate();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }

    }

    public void statusUser(int id ,String position){
        try {
            String query = " insert into " +Const.WORKER_TABLE + "
(idUserWorker,position )"
                    + " values ( ?, ?)";

            PreparedStatement preparedStmt =
connection.prepareStatement(query);

            preparedStmt.setInt (1, id);
            preparedStmt.setString (2, position);


            preparedStmt.executeUpdate();

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }

    }
    public LinkedList<String>  showAllWorkers(){
        LinkedList<String> list = new LinkedList<>();
        String query = "SELECT " + Const.ID_USER_WORKER+" ,
"+Const.POSITION+ " FROM " + Const.WORKER_TABLE ;

        String contribution="";
        try {
            ResultSet rs  = statement.executeQuery(query);

            while (rs.next()) {

contribution+=currentUserData(Integer.parseInt(rs.getString(Const.ID_U
```

```
SER_WORKER)))+" ";
                contribution+=rs.getString(Const.POSITION);
                System.out.println(contribution);
                list.add(contribution);

                contribution="";
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return list;
    }
    public LinkedList<String> showAllContribution() {
        LinkedList<String> list = new LinkedList<>();
  /*      String query = "SELECT "+Const.ID_CONTRIBUTION+" , " +
Const.DEPOSIT_AMOUNT+" , "+Const.PERCENT+" , "
                 +Const.TYPE_DEPOSIT+ " FROM " +
Const.CONTRIBUTION_TABLE ;*/
        String query = "SELECT "+Const.ID_CONTRIBUTION+" , " +
Const.DEPOSIT_AMOUNT+" , "+Const.PERCENT+" , "
                +Const.TYPE_DEPOSIT+ " FROM " +
Const.CONTRIBUTION_TABLE + " WHERE " + Const.ID_CONTRIBUTION +
                " NOT IN  (select " + Const.ID_CONTRIBUTION_TOTAL + "
from " + Const.HISTORY_CONTRIBUTION + ")";
        ResultSet rs = null;
        String contribution="";
        try {
            rs = statement.executeQuery(query);

            while (rs.next()) {

                contribution+=rs.getString(Const.ID_CONTRIBUTION)+" ";
                contribution+=rs.getString(Const.DEPOSIT_AMOUNT)+" ";
                contribution+=rs.getString(Const.PERCENT)+" ";
                contribution+=rs.getString(Const.TYPE_DEPOSIT)+" ";
                list.add(contribution);
                contribution="";
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return list;
    }

    public LinkedList<String> showAllContributionInHistory() {
        LinkedList<String> list = new LinkedList<>();

        String query = "SELECT "+Const.ID_CONTRIBUTION+" , " +
Const.DEPOSIT_AMOUNT+" , "+Const.PERCENT+" , "
                +Const.TYPE_DEPOSIT+ " FROM " +
Const.CONTRIBUTION_TABLE + " WHERE " + Const.ID_CONTRIBUTION +
                " IN  (select " + Const.ID_CONTRIBUTION_TOTAL + " from
" + Const.HISTORY_CONTRIBUTION + ")";
```

```
        ResultSet rs = null;
        String contribution="";
        try {
            rs = statement.executeQuery(query);

            while (rs.next()) {

                contribution+=rs.getString(Const.ID_CONTRIBUTION)+" ";
                contribution+=rs.getString(Const.DEPOSIT_AMOUNT)+" ";
                contribution+=rs.getString(Const.PERCENT)+" ";
                contribution+=rs.getString(Const.TYPE_DEPOSIT)+" ";
                list.add(contribution);
                contribution="";
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return list;

    }


    public LinkedList<String> showAllUsers() {

        LinkedList<String> list = new LinkedList<>();
        String query = "SELECT "+Const.ID+" , " + Const.EMAIL+" ,
"+Const.PASSWORD+" , "
                +Const.ROLL+ " FROM " + Const.USERS_TABLE ;
        ResultSet rs = null;
        String contribution="";
        try {
            rs = statement.executeQuery(query);

            while (rs.next()) {

                contribution+=rs.getString(Const.ID)+" ";
                contribution+=rs.getString(Const.EMAIL)+" ";
                contribution+=rs.getString(Const.PASSWORD)+" ";
                contribution+=rs.getString(Const.ROLL)+" ";
                list.add(contribution);
                contribution="";
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return list;
    }

    public void redactionUser(int idUser, String email,String
password,int roll) {
        String query = "update "+Const.USERS_TABLE+" SET   email =?,
password =?, roll=? WHERE " + Const.ID + " = " + "'" + idUser + "'";
        PreparedStatement preparedStmt = null;
        try {
            preparedStmt = connection.prepareStatement(query);
```

```
            preparedStmt.setString(1, email);
            preparedStmt.setString(2, password);
            preparedStmt.setInt   (3, roll);
            // execute the java preparedstatement
            preparedStmt.executeUpdate();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
    public void addHistoryDeposit(int id){
        try{
        String query = " insert into " +Const.HISTORY_CONTRIBUTION + "
(idDeposit )"
                + " values (?)" /* + "delete from " +
Const.CONTRIBUTION_TABLE
                    + " WHERE id IN (select idDeposit from " +
Const.HISTORY_CONTRIBUTION + ")"*/;
        PreparedStatement preparedStmt =
connection.prepareStatement(query);
        preparedStmt.setInt (1, id);
        preparedStmt.executeUpdate();

    } catch (SQLException throwables) {
        throwables.printStackTrace();
     }
    }

    public LinkedList<String> showHistory(){
        LinkedList<String> list = new LinkedList<>();
        String query = "SELECT "+Const.ID_CONTRIBUTION_HISTORY+ " FROM
" + Const.HISTORY_CONTRIBUTION ;
        ResultSet rs = null;
        String contribution="";
        try {
            rs = statement.executeQuery(query);

            while (rs.next()) {

contribution+=rs.getString(Const.ID_CONTRIBUTION_HISTORY);
                list.add(contribution);
                contribution="";
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return list;
    }

   }
```

# ПРИЛОЖЕНИЕ Г
## Листинг скрипта генерации базы данных
## (обязательно)

```
package database;

import constParams.Const;

import java.sql.*;

public class TablesDatabase {
     private final Statement statement;
     private final Connection connection;

    public TablesDatabase(Statement statement, Connection connection)
{
        this.statement = statement;
        this.connection = connection;

        createTableWorker();
        createTotalDeposit();
        createTableHistoryContribution();
        addTableUserInDateBase();
        addTableContribution();
    }

    public void createTableWorker(){

        if(tableExists(Const.WORKER_TABLE)) {
            try {
                String SQL = "CREATE TABLE "+Const.WORKER_TABLE +
                        "( " +
                        " id  SERIAL PRIMARY KEY," +
                        " idUserWorker INTEGER, " +
                        " position VARCHAR (50) " +
                        ")";

                statement.executeUpdate(SQL);
                System.out.println("Таблица  была создана ! "
+Const.WORKER_TABLE);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void createTotalDeposit(){
        if(tableExists(Const.TOTAL_DEPOSIT)) {
            try {
                String SQL = "CREATE TABLE "+Const.TOTAL_DEPOSIT +
                        "( " +
                        " id  SERIAL PRIMARY KEY," +
                        " idDeposit INTEGER, " +
                        " amountDepositTotal INTEGER " +
                        ")";
```

```
                statement.executeUpdate(SQL);
                System.out.println("Таблица  была создана ! "
+Const.TOTAL_DEPOSIT);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void createTableHistoryContribution(){

        if(tableExists(Const.HISTORY_CONTRIBUTION)) {
            try {
                String SQL = "CREATE TABLE
"+Const.HISTORY_CONTRIBUTION +
                        "( " +
                        " id  SERIAL PRIMARY KEY," +
                        " idDeposit INTEGER, " +
                        " idUserHistory INTEGER " +

                        ")";

                statement.executeUpdate(SQL);
                System.out.println("Таблица  была создана ! "
+Const.HISTORY_CONTRIBUTION);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void addTableUserInDateBase(){
        if(tableExists(Const.USERS_TABLE)) {
            try {
                String SQL = "CREATE TABLE "+Const.USERS_TABLE +
                        "( " +
                        " id  SERIAL PRIMARY KEY," +
                        " email VARCHAR (50), " +
                        " name VARCHAR (50), " +
                        " lastName VARCHAR (50), " +
                        " password VARCHAR (50), " +
                        " roll VARCHAR (50)"+
                        ")";

                statement.executeUpdate(SQL);
                System.out.println("Таблица с users была создана !
"+Const.USERS_TABLE);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void addTableContribution(){
        if(tableExists(Const.CONTRIBUTION_TABLE)) {
            try {
```

```
                String SQL = "CREATE TABLE "+Const.CONTRIBUTION_TABLE
+
                        "( " +
                        " id  SERIAL PRIMARY KEY," +
                        " userId INTEGER ,"+
                        " depositAmount INTEGER , " +
                        " percent VARCHAR (50) , " +
                        " countYEAR INTEGER,"+
                        " currency VARCHAR (50), "+
                        " collectMoney VARCHAR (50), "+
                        " typeDeposit VARCHAR (50) "+

                        ")";

                statement.executeUpdate(SQL);
                System.out.println("Таблица с users была создана !
"+Const.CONTRIBUTION_TABLE);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    private boolean tableExists(String nameTable){

        try{
            DatabaseMetaData md = connection.getMetaData();
            ResultSet rs = md.getTables(null, null, nameTable, null);
            rs.last();
            return rs.getRow() <= 0;
        }catch(SQLException ignored){

        }
        return true;
    }
}
```

## ПРИЛОЖЕНИЕ Д

## Ведомость курсового проекта

## (обязательное)

Перв. примен. ГУИР. ГУИР.506411.025 Д1

Справочный №

| Зона | Обозначение | Наименование | Дополнительные, сведения |
|---|---|---|---|
| | | | |
| | | *Текстовые документы* | |
| | | | |
| | ГУИР КП 1–40 05 01 025 ПЗ | Пояснительная записка | 62 с. |
| | | | |
| | | *Графические документы* | |
| | | | |
| | ГУИР.506411.025 Д1 | Схема алгоритма | Формат А3 |
| | | | |
| | ГУИР.506411.025 Д2 | IDEF0-модель процессов предметной области | Формат А4 |
| | | | |
| | ГУИР.506411.025 Д3 | Схема алгоритма поиска | Формат А4 |
| | | | |
| | ГУИР.506411.025 Д4 | Диаграммы | Формат А4, 2 л. |
| | | | |
| | ГУИР.506411.025 Д5 | Пользовательский графический интерфейс | Формат А4 |
| | | | |

ПоКПи

Инв. №

Взам. Инв. №

ПоКПись и

Инв. № подл.

| Изм. | Лист | № докум. | Подп. | Дата | ГУИР КП 1–40 05 01 011 Д1 | | |
|---|---|---|---|---|---|---|---|
| Разраб. | | Олешкевич | | 20.11.21 | **Программная поддержка управления взаимодействием с клиентами в банке** | Лит. | Лист 62 / Листов 62 |
| Пров. | | Унучек | | 04.12.21 | | Т | |
| Н.контр. | | Унучек | | 04.12.21 | | Кафедра ПИКС, группа 914301 | |
| Утв. | | Хорошко | | | | | |