

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

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

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
Ассистент кафедры ЭИ
_____. А.П. Лыщик
_____._____.2022

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

БГУИР КП 1-40 01 02-08 011 ПЗ

Выполнил студент группы 073601
Леванович Максим Александрович

(подпись студента)

Курсовой проект представлен на
проверку _____._____.2022

(подпись студента)

Минск 2022

СОДЕРЖАНИЕ

Введение	5
1 Описание функционирования банковской системы.....	6
1.1 Основные понятия и термины	6
1.2 Расчет процентов по вкладу.....	6
2 Постановка задачи и обзор методов её решения	9
3 Моделирование программного средства.....	11
4 Информационная модель системы	17
5 Модели представления системы.....	21
5.1 Описание диаграммы вариантов использования	21
5.2 Описание диаграммы классов	23
5.3 Описание диаграммы состояний	24
5.4 Описание диаграммы последовательности.....	25
5.5 Описание диаграммы деятельности сравнения простых и сложных процентов при помощи графика.....	27
5.6 Описание диаграммы развёртывания системы.....	28
6 Обоснование оригинальных решений по использованию технических и программных средств.....	29
7 Описание алгоритмов.....	31
7.1 Алгоритм работы администратора	31
7.2 Алгоритм добавления строки в базу данных клиентов	32
8 Руководство пользователя	33
9 Результаты тестирования разработанной системы.....	44
Заключение	47
Список использованных источников	49
Приложение А (обязательное) Код программы	50

ВВЕДЕНИЕ

Сбережения населения образуют инвестиционный потенциал страны, необходимый для её экономического роста. Главной формой сбережений населения с точки зрения возможности их привлечения для финансирования экономики республики являются вклады (депозиты) населения в банках. Большую актуальность вопросы вкладов приобретают в свете решения проблем укрепления банковской системы республики, формирование её ресурсной базы. Вложение средств посредством депозитных операций должно осуществляться таким образом, чтобы по истечении определённого периода не только вернуть вложенные средства, но и получить желаемый эффект от этих вложений, который обеспечивается заявленным уровнем процентной ставки.

Целью данного курсового проекта является оптимизация процесса привлечения денежных средств во вклады, посредством создания специализированного программного приложения, обеспечивающего эффективное взаимодействие вкладчиков с работниками банка. Стоит отметить, что под эффективным взаимодействием понимается то, что пользователь сможет получить все данные и информацию, которая ему необходима в кратчайшие сроки при минимальных временных затратах и взаимодействиях с сотрудниками банка.

Поставленная цель потребовала решения следующих задач:

- изучить предметную область: понятие банковского вклада и его видов;
- проанализировать деятельность коммерческих банков в Республике Беларусь в части разработки приложений, используемых банками при работе с клиентами;
- спроектировать информационную модель системы, построить модели представления системы с использованием UML диаграмм, а также разработать необходимую модель базы данных;
- разработать программное приложение;
- провести тестирование разработанного приложения.

1 ОПИСАНИЕ ФУНКЦИОНИРОВАНИЯ БАНКОВСКОЙ СИСТЕМЫ

1.1 Основные понятия и термины

Понятие «банковская система» содержит в себе два элемента: банк и система. Термин «система» чаще всего рассматривается как форма организации, объединение множества элементов, взаимосвязанных друг с другом. Определение понятия «банк» находится в законодательных актах. В соответствии со ст. 8 Банковского кодекса Республики Беларусь: «Банк является юридическим лицом, имеющим исключительное право осуществлять в совокупности определенный набор банковских операций» [1].

Банковская система является частью финансовой системы государства и регулируется его законодательством. Исходя из 136 статьи Конституции Республики Беларусь, банковская система Республики Беларусь состоит из Национального банка Республики Беларусь и коммерческих банков. Национальный банк регулирует кредитные отношения, денежное обращение, определяет порядок расчетов и обладает исключительным правом эмиссии денег.

Банк является неким юридическим лицом, которое имеет исключительное право осуществлять следующие банковские операции:

1. Привлечение денежных средств и юридических лиц на счета и во вклады (депозиты).

2. Размещение привлеченных денежных средств физических/юридических лиц от своего имени и за свой счет на условиях возвратности, платности, срочности.

3. Открытие и ведение банковских счетов физических/юридических лиц.

В данной курсовой работе будет рассматриваться реализация первой из указанных выше операций – привлечение денежных средств путем реализации депозитов.

Рассмотрим понятие депозита. Депозит (вклад) – это фиксированная денежная сумма, которую клиент передает на хранение банку и получает от этого доход в виде начисленных процентов [2].

1.2 Расчет процентов по вкладу

Проценты - сумма доходов, полученных от размещения денег во вклад в различных формах и на различных условиях. Сумма процентов зависит от

суммы вклада, срока выплаты и процентной ставки, характеризующей интенсивность начисления процентов. Увеличение суммы вклада за счёт присоединения начисленных процентов называют наращиванием (ростом) первоначальной суммы долга. Интервал времени, за который начисляют проценты, называют периодом начисления.

Существует 2 методики начисления процентов: простые проценты и сложные проценты. При использовании простых процентов сумма процентов в течении всего срока вклада определяют исходя из первоначальной суммы вклада независимо от количества периодов начисления и их длительности. Формула расчета простых процентов представлена ниже:

$$S_n = S \cdot \left(1 + \frac{nr}{100}\right), \quad (1.1)$$

где S – начальная сумма вклада,

S_n – сумма вклада через n лет (месяцев),

r – годовая (месячная) процентная ставка.

При начислении сложных процентов денежные средства, начисленные после первого периода начисления, являются частью общего срока хранения вклада, не выплачиваются, а присоединяются к сумме вклада. Во время второго срока проценты вычисляются уже от новой суммы денежных средств, которая была получена в результате сложения первоначального вклада и процентов, начисленных за время нового периода, и так далее на каждом последующем периоде начисления. Сложные проценты рассчитываются по формуле, указанной ниже.

$$S_n = S \cdot \left(1 + \frac{r/m}{100}\right)^{n \cdot m}, \quad (1.2)$$

где S – начальная сумма вклада,

S_n – сумма вклада через n лет (месяцев),

r – годовая (месячная) процентная ставка,

n – количество лет/месяцев размещения вклада

m – количество капитализаций в год.

Таким образом, исходя из исследования, проведенного в данной главе, можно сделать следующие выводы:

1. Банк, как юридическое лицо, может осуществлять такую операцию, как привлечение денежных средств во вклады.

2. В зависимости от типа расчета процентов по вкладам, по истечении срока депозита, вкладчик получает различный доход.

2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ

В наше время почти все системы подразумевают под собой хранение и обработку многочисленных объемов данных, которые необходимы для работы сферы. Банковская система также не является исключением в этом вопросе. С каждым годом потоки данных, поступающих в коммерческие банки, становится все больше, в связи с чем увеличились и требования к хранению и скорости обработки данных.

Сотрудникам банка все сложнее справляться с такими объёмами работы, поэтому требуется разработка и применение удобных компьютерных технологий, которые позволили бы максимально оптимизировать и облегчить процессы, связанные с функционированием банковской системы.

Разработанная система должна обеспечивать хранение корректной информации о вкладах, а также содействовать ускоренной обработке этих данных. В том числе разработанная система должна устранять ряд недостатков, таких как обеспечение клиентов непроверенной или некорректной информацией о событиях, небольшой скорости обработки и предоставления информации, а в результате повысить оперативность и качество работы банка и устранить недостатки при выполнении всех необходимых вычислений и операций.

Таким образом, чтобы создать программное приложение для банковской системы и достичь цели, поставленной в рамках данного курсового проекта, необходимо реализовать следующие задачи:

Создать базу данных, которая содержала бы все данные физических лиц (вкладчиков и сотрудников банка), а также информацию обо всех имеющихся вкладах и их типах.

Подготовить все необходимые схемы и диаграммы, при помощи которых с наибольшей корректностью можно было бы воссоздать логику всех процессов, которые будут реализовываться в приложении.

Создать удобный и понятный пользователям интерфейс приложения.

Осуществить клиент-серверное взаимодействие в приложении, которое смогло бы объединить результаты выполненных выше целей вместе и гарантировать отлаженную работу системы.

Провести программное тестирование полученного программного приложения, а затем, при необходимости, исправить имеющиеся ошибки в работе системы.

Таким образом, можно сделать вывод, что при качественном выполнении всех вышеперечисленных задач, можно достигнуть главной цели проекта и создать приложение, которое поможет оптимизировать работу банковской системы.

3 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Описание системы с помощью IDEF0 называется функциональной моделью. Функциональная модель предназначена для описания существующих бизнес-процессов, в котором используются как естественный, так и графический языки.

Для передачи информации о конкретной системе источником графического языка является сама методология IDEF0. Сначала проводится описание системы в целом и ее взаимодействия с окружающим миром (контекстная диаграмма), после чего проводится функциональная декомпозиция – система разбивается на подсистемы, и каждая подсистема описывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности.

Одной из важнейших функций разрабатываемой системы является анализ возможного вклада – так называемое «прогнозирование» вклада, поэтому важно понять, по какому принципу оно осуществляется в реальной предметной области, то есть какое место оно занимает в функционировании банка.

На контекстной диаграмме (верхний уровень), представленной на рисунке 3.1, изображена функциональная модель «Анализ планируемого вклада», включающей в себя следующие информационные потоки:

- входные (левые стрелки): клиент и начальная сумма планируемого вклада;
- механизмы (верхние стрелки): законы Республики Беларусь, Устав банка, типы вкладов;
- управление (нижние стрелки): отдел работы с клиентами, отдел управления вкладами, юридический отдел;
- выходные (правые стрелки): оформленные (графически) данные по планируемому вкладу.

Диаграмма декомпозиции является своего рода дочерней диаграммой по отношению к контекстной: она уточняет (детализирует) функциональный блок.

Первый уровень декомпозиции удобно разбить на три блока – «Обратиться в отдел работы с клиентами», «Выбрать тип вклада» и «Получить графическое представление данных о вкладе». Эти три процесса позволяют обобщенно представить процесс анализа вклада. Кроме уже имеющихся стрелок, появляется ряд новых:

- на выходе первого блока и, соответственно, на входе второго – стрелка «Доступ к банковской информации», который может быть

предоставлен только после прохождения определенных формальных процедур в банке;

– на выходе второго блока и, соответственно, на входе третьего – стрелка «Выбранный тип вклада», который необходим для расчета финансовых показателей вклада и представления графической информации.

Декомпозиция первого уровня представлена на рисунке 3.2.

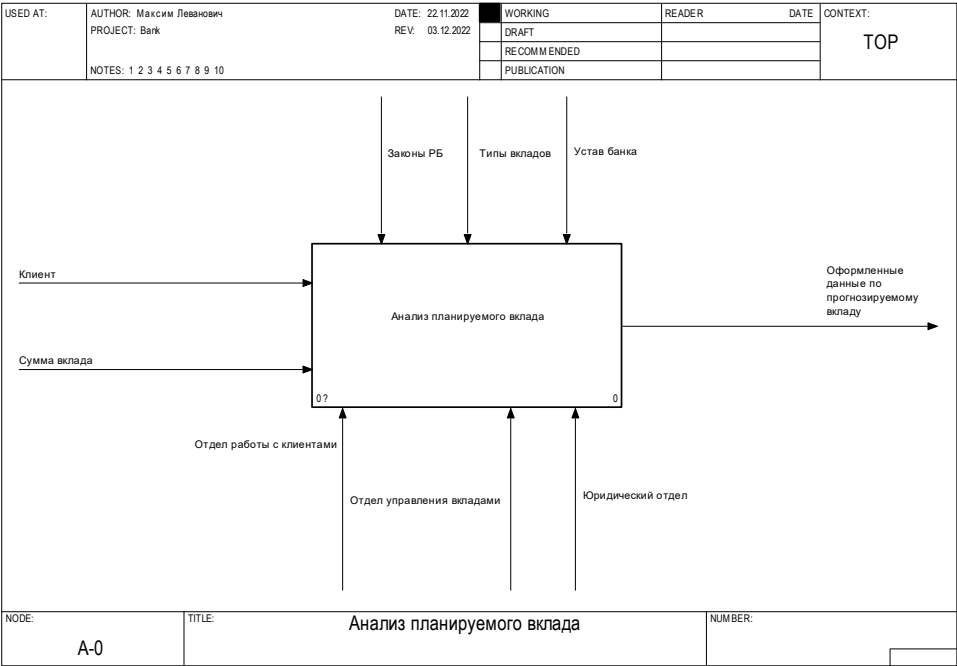


Рисунок 3.1 – Контекстная диаграмма

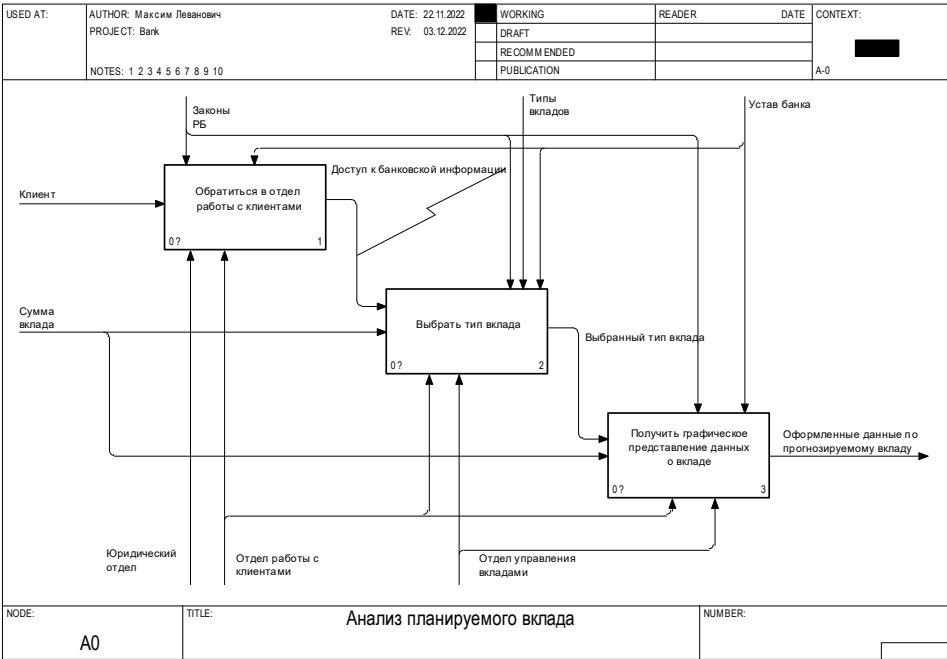


Рисунок 3.2 – Диаграмма декомпозиции первого уровня

Второй уровень декомпозиции будет состоять из разбиения каждого из представленных на рисунке 3.2 блоков на еще несколько.

Блок «Обратиться в отдел работы с клиентами» удобно разбить еще на три – «Предоставить паспорт», «Заполнить данные клиента» и «Подписать договор с банком», – которые в полной мере отражают работу с банковского работника клиентом до момента начала консультации по вкладу. Появляются две новые стрелки:

- на выходе первого блока и, соответственно, на входе второго – «Паспортные данные», необходимые для заполнения и подписания договора;
- на выходе второго блока и, соответственно, на входе третьего – «Договор», без которого предоставление доступа к информации банка считается недопустимым.

Поскольку декомпозиция данного блока покрывает основные бизнес-процессы и не требует большей детализации, разбивать их до третьего уровня нет необходимости.

Диаграмма декомпозиции блока «Обратиться в отдел работы с клиентами» изображена на рисунке 3.3

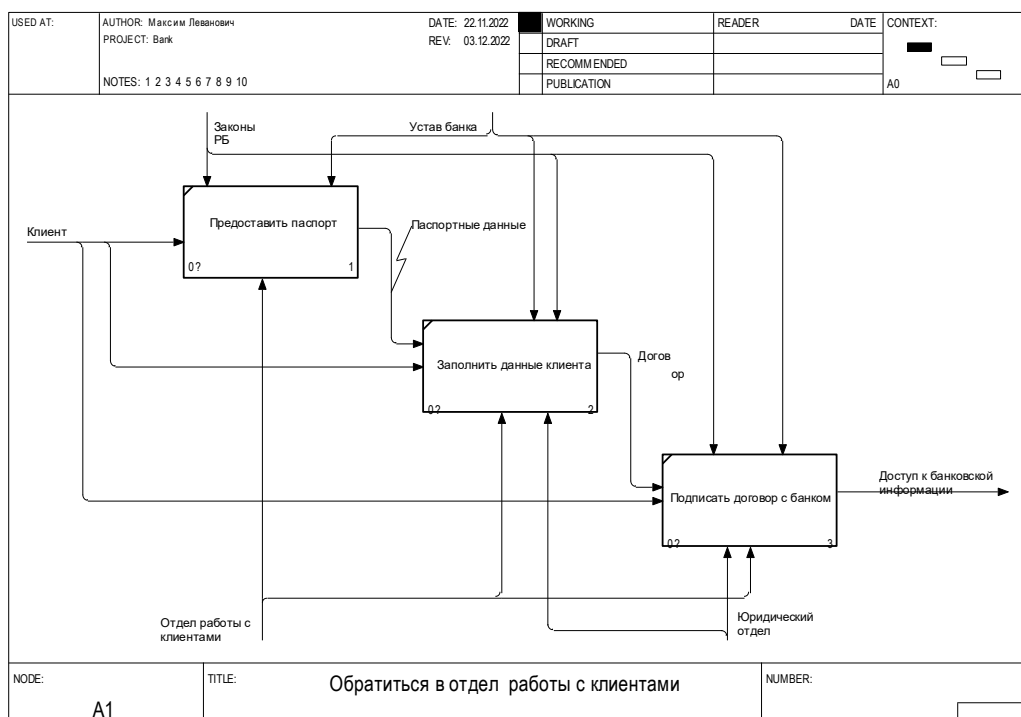


Рисунок 3.3 – Декомпозиция блока «Обратиться в отдел работы с клиентами»

Вернувшись к декомпозиции первого уровня, выбираем следующий блок – «Выбрать тип вклада». Его также имеет смысл разбить на три

дополнительных блока, а именно: «Выбрать категорию начальной суммы» (во многих банках возможность выбора того или иного типа вклада зависит от величины вложения), «Выбрать процентную ставку», «Выбрать срок вклада».

Категория начальной суммы передается из первого блока во второй, а выбранная процентная ставка – из второго в третий.

Декомпозиция данного блока представлена на рисунке 3.4.

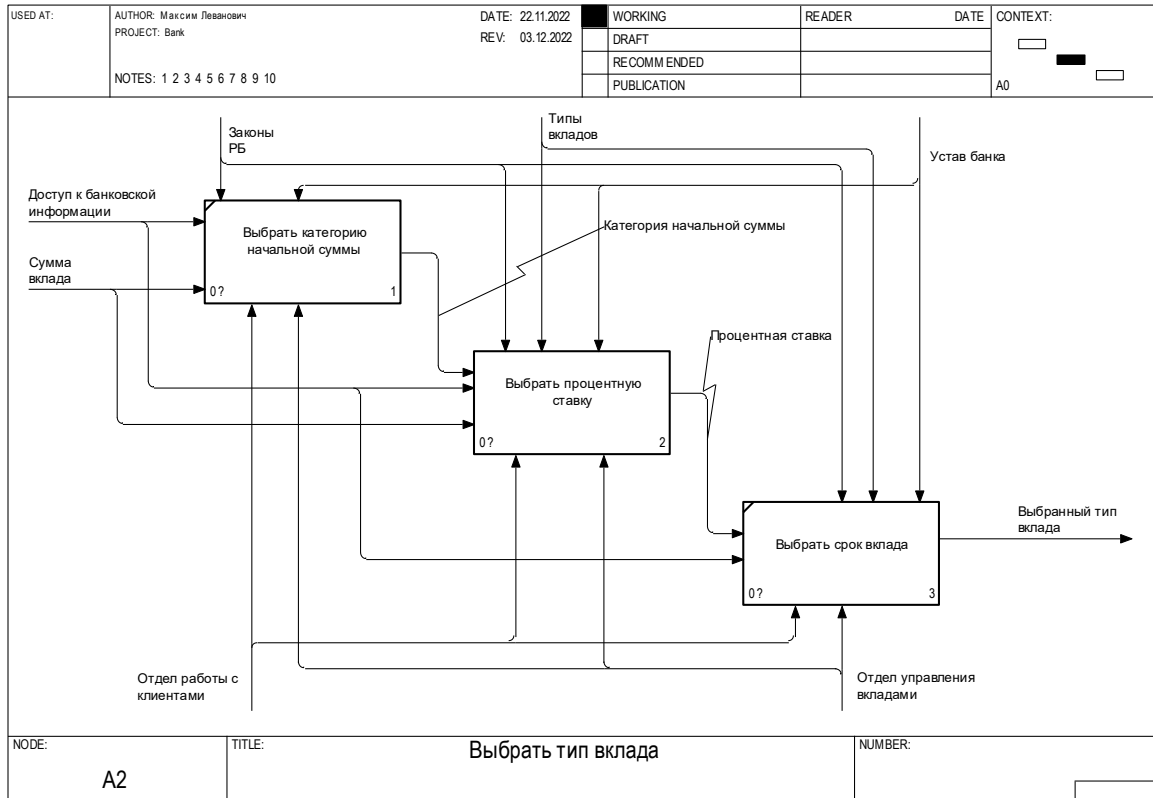


Рисунок 3.4 – Декомпозиция блока «Выбрать тип вклада»

Поскольку выбор процентной ставки – составной, комплексный процесс, его необходимо декомпозировать на третий и четвертый уровни для более глубокого понимания.

Удобно декомпозировать данный блок сначала на три – «Определить тип начисления процента» (простые или сложные проценты), «Определить период капитализации», «Определить величину процентной ставки».

Для окончательного определения величины процентной ставки необходимо знать категорию начальной суммы, а также тип начисления процента и период капитализации, которые появляются в ходе протекания указанных бизнес-процессов.

Декомпозиция третьего уровня представлена на рисунке 3.5.

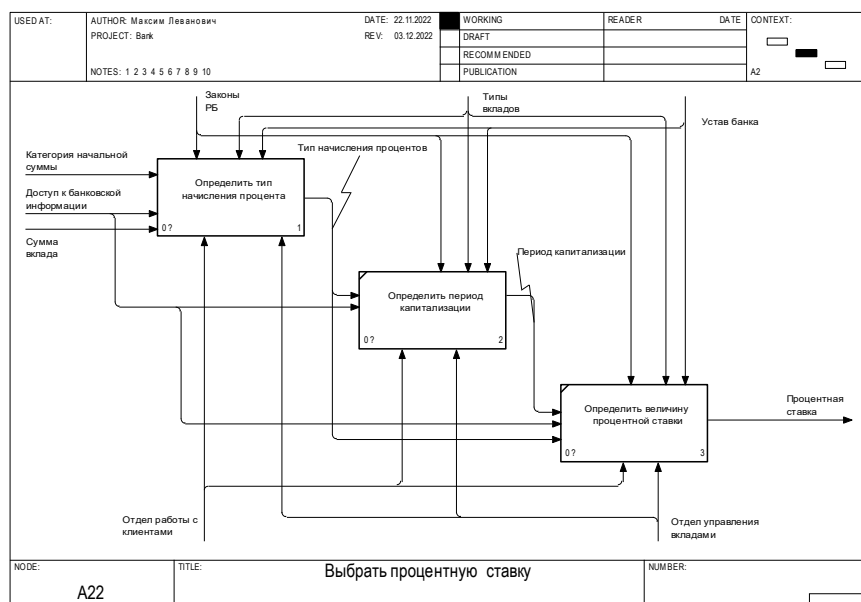


Рисунок 3.5 – Декомпозиция блока «Выбрать процентную ставку»

Несмотря на то что данные функциональные блоки могут показаться довольно простыми, первый из них можно также декомпозировать.

Разбиение блока «Определить тип начисления процента», относящаяся к четвертому уровню декомпозиции, произведено на два функциональных блока – «Предоставить сравнительный анализ типов начисления процента» и «Сравнить рост суммы вклада», которая включает графическое представление прогноза роста вклада по нескольким видам начисления процента.

Данная диаграмма является простой для понимания из-за наличия лишь двух блоков разбиения. Она представлена на рисунке 3.6.

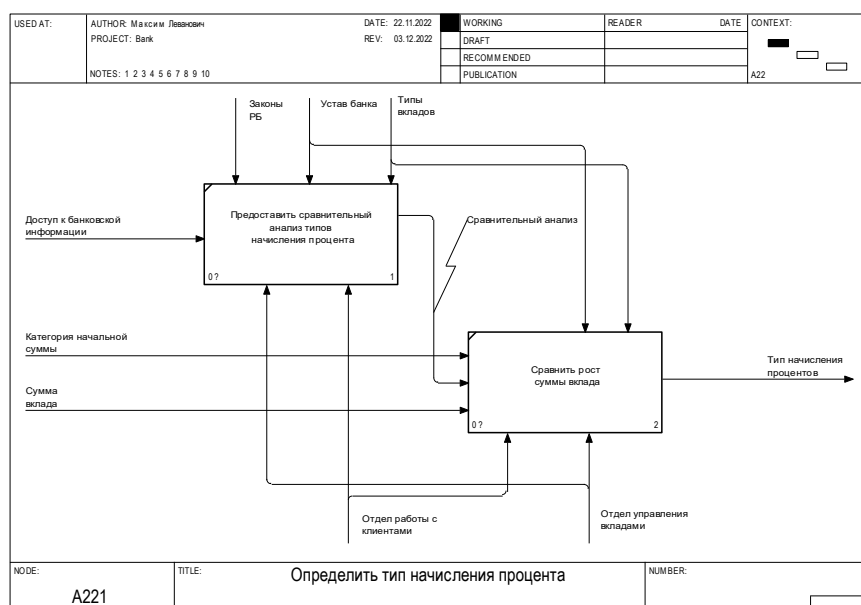


Рисунок 3.6 – Декомпозиция блока «Определить тип начисления процента»

Разбив в итоге блок «Выбрать тип вклада» первого уровня диаграммы декомпозиции на второй, третий и четвертый уровни и вернувшись к первому, заметим, что осталось декомпозировать лишь блок «Получить графическое представление данных о вкладе».

Его удобно разбить на два процесса – «Рассчитать основные показатели» и, получив на выходе ряд численных показателей, «Предоставить инфографику». Как и предыдущая диаграмма, она является достаточно простой для понимания и не требует дополнительных объяснений.

Диаграмма декомпозиции данного блока представлена на рисунке 3.7.

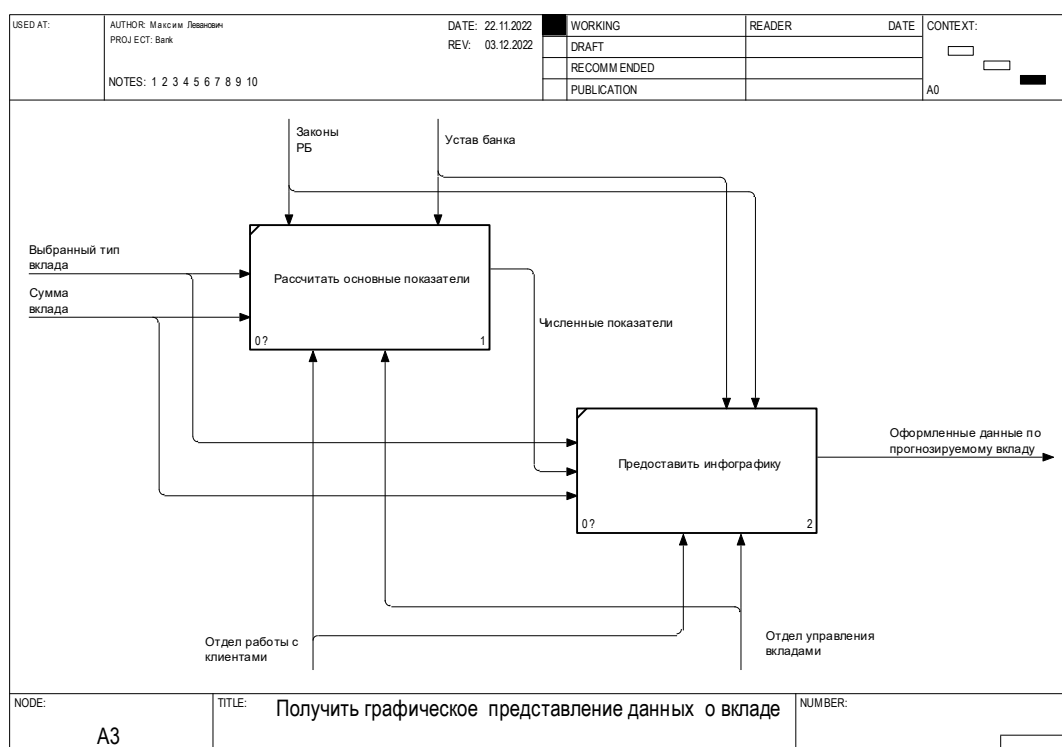


Рисунок 3.7 – Декомпозиция блока «Получить графическое представление данных о вкладе»

Таким образом, была полностью завершена работа с диаграммами нотации *IDEF0*, позволивших в наглядной форме представить бизнес-процессы банка, связанные с анализом открытия вклада и мы, изучив предметную область, можем перейти к постановке задачи на разработку необходимого программного средства.

4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ

Под информационной моделью в широком смысле можно понимать всю совокупность информации, которая характеризует свойства и состояния объекта и его взаимосвязь с внешней по отношению к системе средой. Суть информационной модели могут отражать логическая и физическая модели используемой базы данных [10].

Центральным понятием при проектировании базы данных является термин «сущность». Сущностью могут называть любой конкретный или абстрактный объект, сведения о котором планируется хранить в создаваемой базе данных. Это позволяет классифицировать данные и отслеживать их связи и взаимодействия друг с другом. Свойства и характеристики сущности носят название «атрибут» и также являются ключевыми при моделировании базы данных.

Для реализации функций системы удобно использовать пять основных сущностей: «Сотрудники», «Клиенты», «Договоры», «Типы вкладов» и, наконец, «Вклады».

Сущность «Сотрудники» содержит информацию о банковских рабочих. Она характеризуется следующими полями:

- «ID сотрудника» – ключевое поле, содержащее уникальный номер-идентификатор работника;
- «ФИО»;
- «Номер контракта», определяющее документ, заключенный при найме сотрудника на работу;
- «Дата приема»;
- «Логин», необходимый для авторизации в системе;
- «Пароль», необходимый для авторизации в системе;
- «Позиция» – поле, определяющее права в системе.

Следует отметить, что поле «Позиция» при рассмотрении его на физическом уровне будет отмечено целочисленным типом. Это удобно при дальнейшем кодировании: цифра «1» будет обозначать рядового сотрудника-консультанта, работающего с договорами, которые он оформил; цифра «2» – администратора базы данных.

Сущность «Клиенты» содержит информацию о клиентуре банка и определяется следующими полями:

- «ID клиента» – ключевое поле, содержащее уникальный идентификатор клиента;
- «Логин» и «Пароль», необходимые для авторизации в системе;

- «ФИО»;
- «Серия и номер паспорта»;
- «Телефон» и «Адрес», необходимые для связи работника и клиента.

Сущность «Договоры» содержит два основных атрибута – ключевое поле «Номер договора» и «Дата», определяющее время заключения договора. Эта сущность является дочерней по отношению к сущностям «Сотрудники» и «Клиенты» и в обоих случаях характеризуется связью «один ко многим» (грубо говоря, один сотрудник может заключать множество договоров и один клиент может иметь (подписывать) множество договоров). Связь с двумя таблицами добавляет в сущность «Договоры» еще два атрибута – «ID Сотрудника» и «ID Клиента».

Сущность «Типы вкладов» содержит информацию о параметрах вкладов, которые можно открыть в банке, и состоит из ряда атрибутов: ключевой атрибут «ID типа вкладов», «Название», «Минимальная начальная сумма», «Максимальная начальная сумма», «Срок вклада» (количество месяцев), «Капитализация» (сколько раз в год капитализируются проценты) и, наконец, «Процентная ставка».

Последняя сущность моделируемой базы данных – «Вклады» – отражает реально сделанные в данном банке вклады. Она состоит из следующих атрибутов: ключевое поле «Номер вклада», являющееся уникальным для каждого отдельно взятого вклада, «Начальная сумма», «Дата открытия», «Планируемая конечная сумма». Кроме того, данная сущность связана со следующими тремя.

1 «Клиент». Клиент открывает вклад, причем предполагается, что один клиент может открыть более одного вклада, поэтому устанавливается связь «один ко многим».

2 «Договоры». Один договор свидетельствует об открытии одного вклада, поэтому устанавливается связь «один к одному».

3 «Типы вкладов». Одному типу вкладов может соответствовать множество сделанных вкладов, поэтому выбранный тип связи – «один ко многим».

Во всех трех случаях сущность «Вклады» является дочерней.

Представим модель базы данных на двух уровнях – логическом и физическом.

Логический уровень является своего рода обобщающим и описывает, какие данные будут храниться в базе данных и каким образом сущности связаны друг с другом. На этом уровне важно отразить такие понятия теории проектирования баз данных, как таблицы, ключи и индексы, но без учета их

внутренней организации. Модель базы данных на логическом уровне представлена на рисунке 4.1.

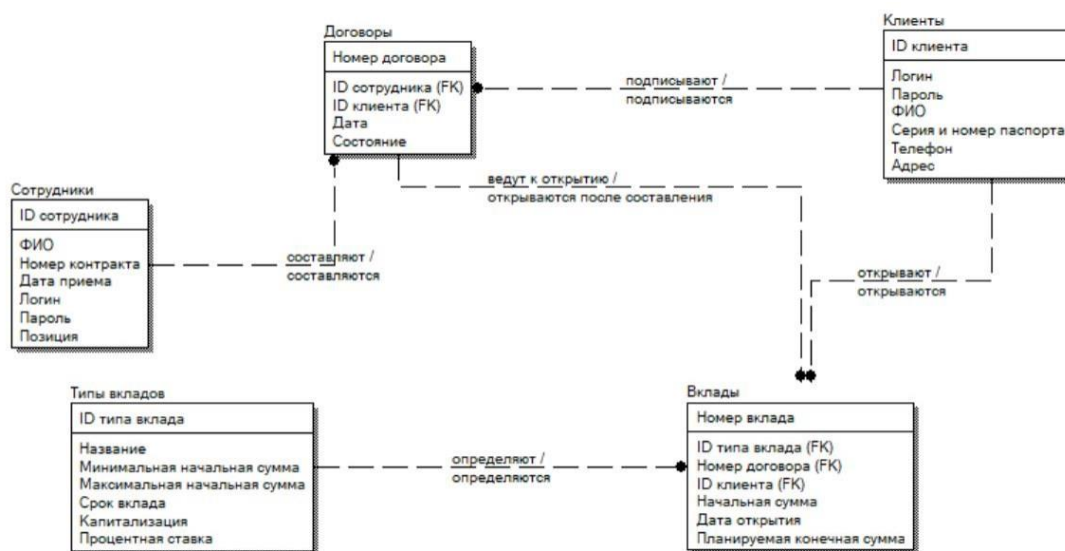


Рисунок 4.1 – Логический уровень базы данных

Физический (внутренний) уровень в свою очередь отражает представление базы данных в ЭВМ в контексте конкретной СУБД. Этот уровень учитывает способ организации данных на электронном носителе. Например, существенную роль играют типы данных, соответствующие каждому из заявленных атрибутов.

Физическая модель банковской системы вкладов физлиц представлена на рисунке 4.2.

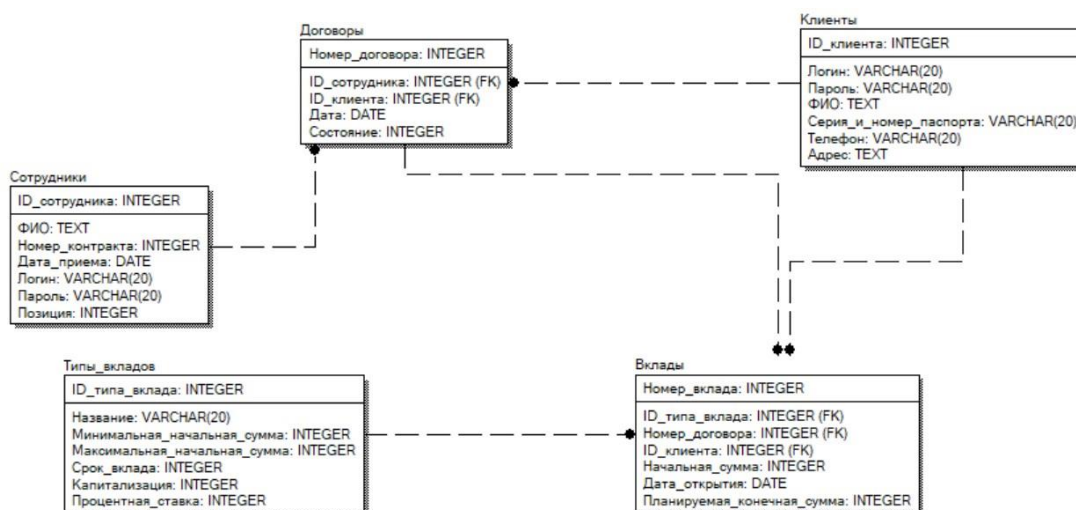


Рисунок 4.2 – Физический уровень базы данных

Существует также понятие «нормальной формы», которое важно при создании базы данных.

Говорят, что таблица находится в первой нормальной форме, когда каждый его кортеж (строка таблицы) имеет только одно значение для каждого атрибута. Стоит отметить, что если речь идет о реляционной модели баз данных, к которой относится и проектируемая база, то они всегда находятся в первой нормальной форме.

Таблица находится во второй нормальной форме, если она уже находится в первой и каждое ее поле, не являющееся ключевым, неприводимо зависит от ключевого. Важно заметить, что если сущность имеет лишь один атрибут в качестве ключа, то это условие выполняется автоматически, что свидетельствует о том, что проектируемая база находится во второй нормальной форме.

Таблица находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме и каждое поле, не являющееся ключевым, нетранзитивно зависит от ключевого. Проще говоря, не должно быть полей, содержимое которых может относиться сразу к нескольким записям. Поскольку в проектируемой базе данных таких полей нет, можно сказать, что она действительно находится в третьей нормальной форме.

Таким образом, можно сделать вывод, что модель IDEF1X при помощи построения логической и физической модели базы данных дает нам структуру, которая поможет нам создать качественное приложение.

5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ

Для описания модели представления разрабатываемой системы удобно использовать широко распространенный язык моделирования *UML*. *UML* – унифицированный язык, что означает, что диаграммы, разработанные на нем, поймет любой человек, использующий его и знающий назначения тех или иных элементов на конкретной диаграмме.

Применение данной нотации на этапе анализа и дизайна обеспечивает лучшее понимание структуры разрабатываемого программного обеспечения, дает возможность взглянуть на предмет проектирования с разных точек зрения, а также помогает другим программистам в команде быстрее и легче понимать код, написанный вами. Кроме всего прочего, эти диаграммы интуитивно понятны и достаточно просты в чтении даже при минимальном понимании нотации [7].

5.1 Описание диаграммы вариантов использования

Диаграмма вариантов использования, или диаграмма прецедентов, ставит перед собой цель дать действующим в процессе разработки ПО лицам, а именно: заказчику, аналитику, проектировщику, разработчику и тестировщику – понимать, каким функционалам должна обладать разрабатываемая система, каким должно быть ее поведение в тех или иных ситуациях.

При разработке такой диаграммы необходимо четко отделить рамки системы (от англ. *boundary* – граница), выделить действующих лиц, которых называют акторами, или актерами, а также определить сценарии их взаимодействия непосредственно с проектируемой системой.

Сами варианты использования (прецеденты) изображаются в виде эллипсов и могут иметь между собой различные виды отношений:

1 Отношение включения (*include*) показывает связь базового прецедента с другим, «поведение которого всегда задействуется базовым вариантом использования». Такое отношение изображается пунктирной линией со стрелкой, направленной из базового варианта во включенный.

2 Отношение расширения (*extend*) отображает специфический случай развития базового варианта и не является обязательным при вызове базового. Изображается пунктирной стрелкой, направленной в сторону расширяемого прецедента.

3 Отношение обобщения (*generalization*) является своего рода противоположностью предыдущего отношения. Изображается в виде стрелки

с треугольным наконечником, направленным в стороны более общего варианта. Часто используется не только в качестве отношения между прецедентами, но и для отображения отношений между актерами.

4 Отношение актора и варианта использования называют отношением ассоциации (*association*) и изображают в виде прямой без стрелки.

На рисунке 5.1 представлена диаграмма вариантов использования для разрабатываемой системы.

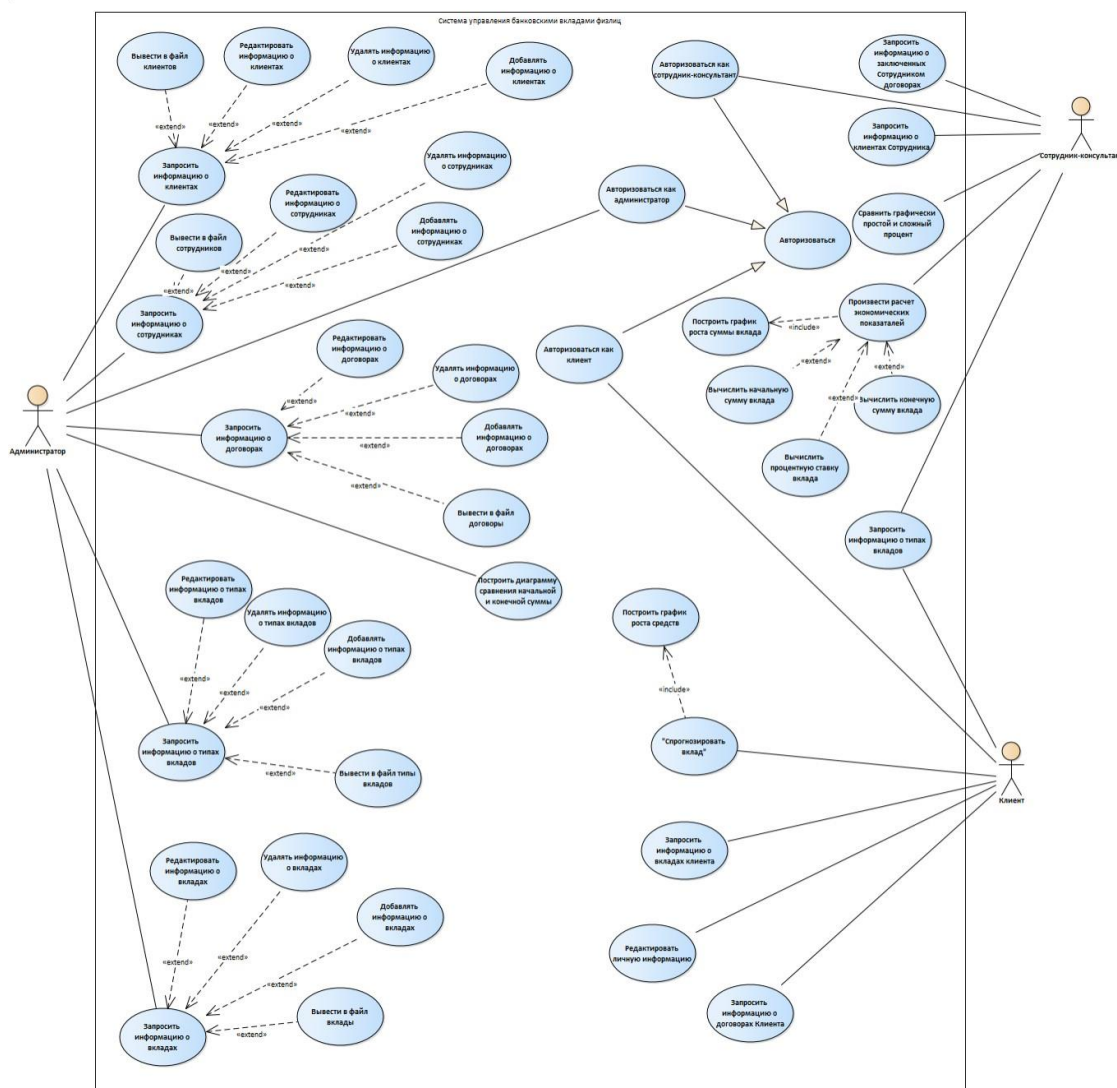


Рисунок 5.1 – Диаграмма вариантов использования

Как видно из диаграммы, данная система задействует трех актеров, а именно: администратора системы, сотрудника-консультанта и клиента. Общим для всех актеров прецедентом является возможность авторизоваться в системе, но так как авторизация для них является средством получения различных прав доступа к функционалу системы, то необходимо разбить ее на

три варианта – «...как администратор», «...как сотрудник-консультант», «...как клиент». Вариант «Авторизоваться» является обобщением для вышеуказанных прецедентов.

Администратор системы фактически выполняет роль администратора базы данных, поэтому доступный ему функционал системы – это реализация *CRUD*-функций работы с базой данных: создание, чтение, редактирования и удаление записей в базе данных. Он же может выводить данные всех таблиц в файл.

Роль сотрудника-консультанта предполагает управления заключенными непосредственно этим сотрудником договорами. Он может просматривать информацию о различных типах вкладов для дальнейшего консультирования, а также редактировать информацию. Кроме того, оба актера могут «прогнозировать» планируемый вклад, что описывалось ранее.

Уникальные возможности клиента сводятся к просмотру информации о заключенных им договорах и вкладах.

Зачастую варианты использования, представленные на диаграмме прецедентов, в дальнейшем подробно описываются в ходе создания документации, разрабатываемой бизнес-аналитиком, и данные помещаются в таблицу.

Таким образом, диаграмма прецедентов является также мощным инструментом для документирования проекта.

5.2 Описание диаграммы классов

Данная диаграмма широко применяется проектировщиками и разработчиками для демонстрации обобщенной иерархии и структуры классов, используемых для работы программного средства.

Класс изображается прямоугольником с указанием его свойств и методов. Классы могут быть в различных отношениях друг с другом. Основные из них:

1. Ассоциация – отношение между классами, когда объекты одного класса связаны с объектами другого. Изображается прямой линией или линией со стрелкой. Иногда на диаграмме указывается множественность ассоциации (связи «один к одному», «один ко многим» или «многие ко многим»).

2. Агрегация – подвид ассоциации, который демонстрирует связь целого с частью. Изображается линией с не закрашенным ромбом на конце.

3. Композиция – «жесткая агрегация». В этом случае уничтожение класса-контейнера приведет к уничтожению всего содержимого. Изображается линией с закрашенным ромбом на конце.

4. Наследование – отношение, показывающее, что один класс является подвидом другого (например, классы «Консультант» и «Администратор» являются наследниками класса «Сотрудник»).

Создавая диаграмму классов для нашего приложения, примем во внимание, что вклад не может существовать без клиента-вкладчика и типа вклада, контракт не может существовать без клиента и сотрудника (при этом, например, при увольнении сотрудника контракт не исчезает). Если учесть вышеуказанные требования, получим следующего вида диаграмму (рис. 5.2).

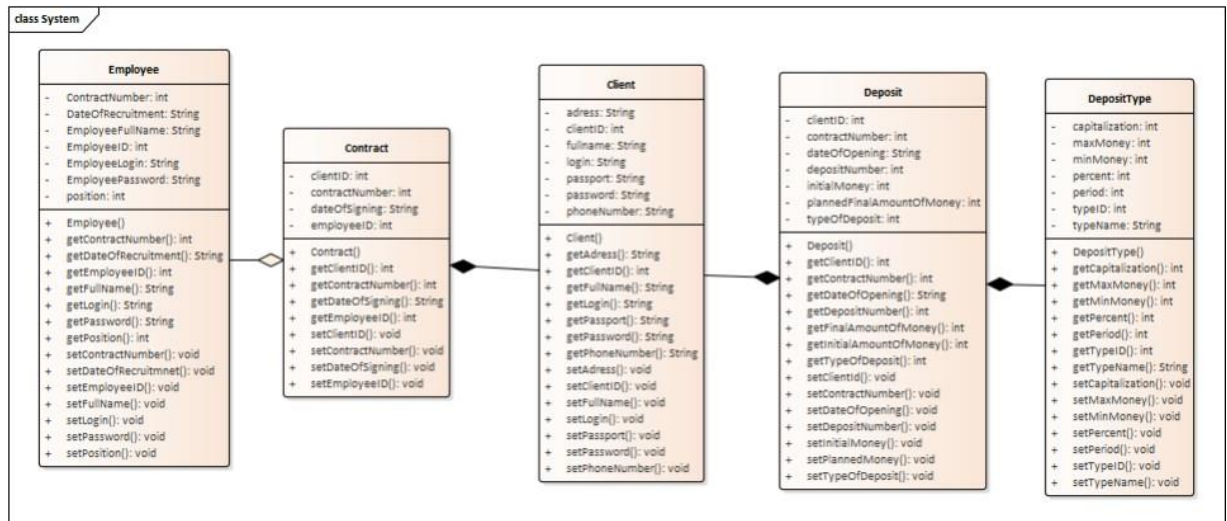


Рисунок 5.2 – Диаграмма классов

5.3 Описание диаграммы состояний

Состоянием принято считать некоторый период, в течение которого объект удовлетворяет какому-то определенному условию, «выполняет ожидаемую деятельность или ожидает некоторого события».

Состояние в языке *UML* изображается прямоугольником с сопряженными сторонами (иногда фигуру называют прямоугольником со скругленными сторонами), начальное состояние (инициализацию) – закрашенным черным кругом, конечное – окружностью с таким кругом внутри.

Диаграмма состояний для разрабатываемой системы представлена на рисунке 5.3.

Сущность «Договоры», как отмечалось в предыдущей главе, имеет атрибут «Состояние», которое отвечает за статус договора. Таких статусов может быть пять:

- «Оформляется»: клиенту уже назначен сотрудник-консультант, но договор еще не подписан – он находится на стадии юридического оформления и открытия вклада;
- «Активен»: вклад по договору открыт, но срок по нему еще не настал;
- «Приостановлен»: данные, представленные клиентом при оформлении договора, потеряли свою релевантность; клиент должен внести изменения в установленный срок;
- «В отключении»: срок по договору настал, но клиент еще не получил вклад;
- «Закрит»: либо клиент забрал деньги, либо он не внес изменения в данные в установленный срок.

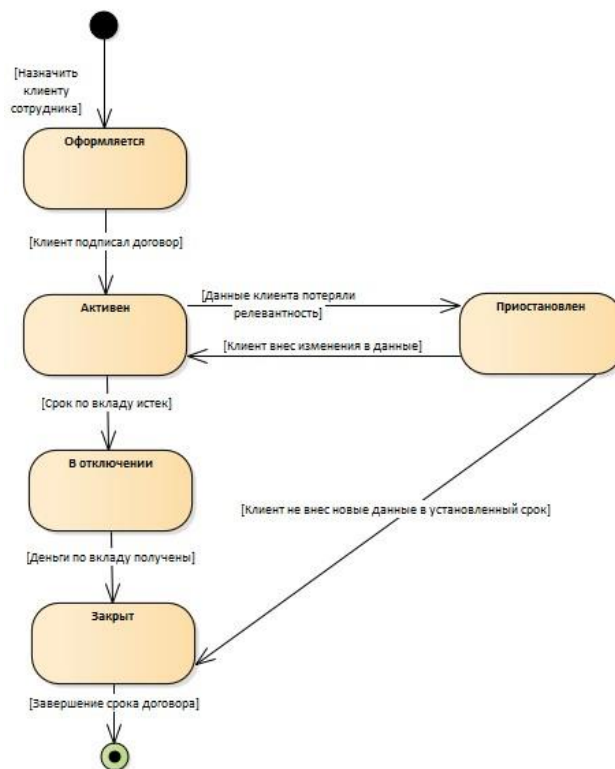


Рисунок 5.3 – Диаграмма состояний

Таким образом, диаграмма состояний отражает возможные состояния определенной сущности и все переходы между ними, которые возможны в течение жизненного цикла.

5.4 Описание диаграммы последовательности

Диаграмма последовательности отображает взаимодействия различных программных объектов – актеров и сущностей. Под взаимодействием в данном

случае понимается обмен различного рода сообщениями и то, как действия актера влияют на систему в целом.

На рисунке 5.4 представлена диаграмма последовательности, выполненная в рамках варианта использования «Спрогнозировать вклад».

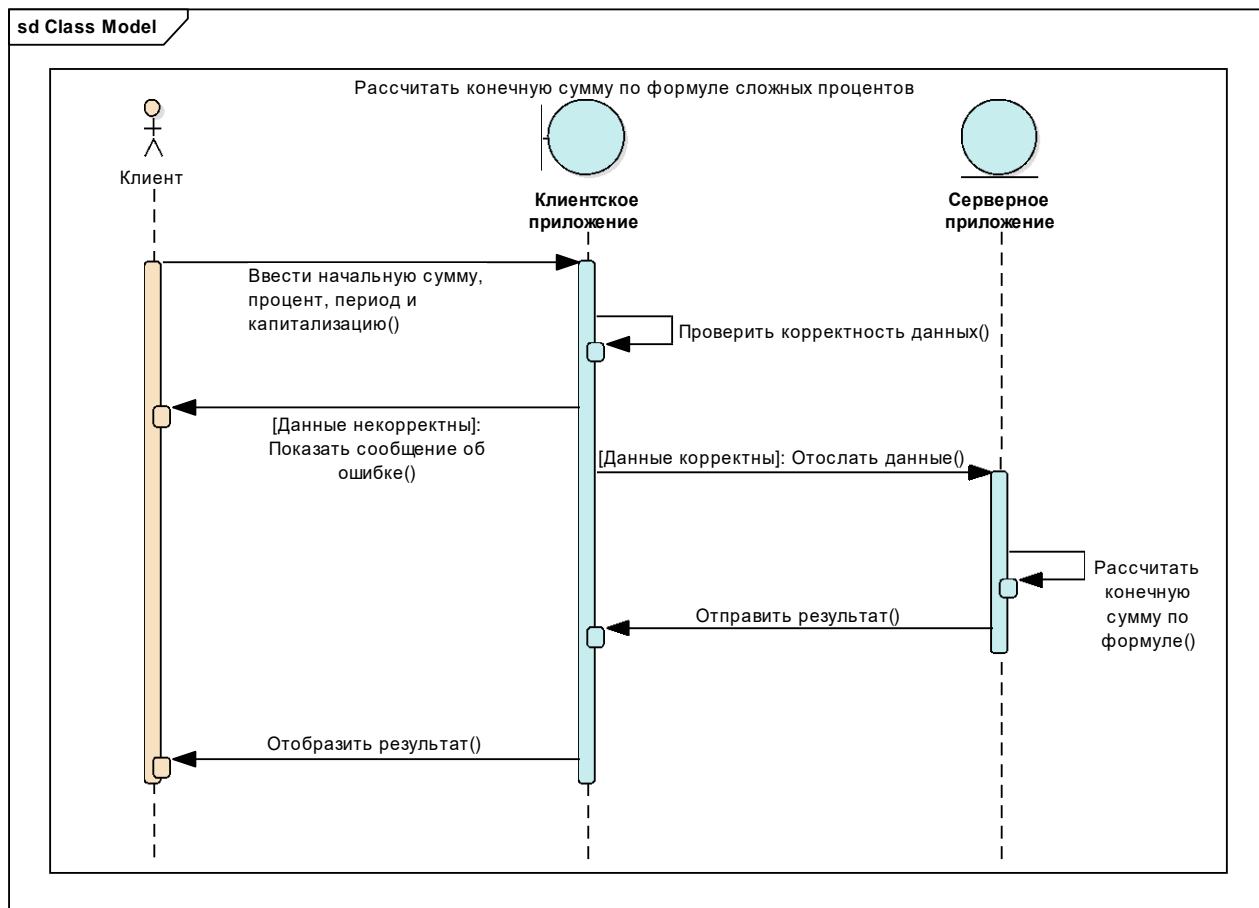


Рисунок 5.4 – Диаграмма последовательности

В окне, предоставляемом клиентской частью системы, клиент вводит начальную сумму, процентную ставку, капитализацию и период вклада. Эта информация проходит валидацию и, в случае корректности введенной информации (все поля представляют собой числа в допустимых диапазонах), отправляется серверу.

Сервер, используя формулу сложных процентов, подсчитывает конечную денежную сумму вклада, отправляет результат клиентскому приложению, которое отображает его пользователю.

Диаграмма последовательно является часто используемой при проектировании информационных систем, поскольку позволяет описать процессы, происходящие в программном средстве, четко и ясно.

5.5 Описание диаграммы деятельности сравнения простых и сложных процентов при помощи графика

Диаграммы деятельности (Activity diagram), называемые также диаграммами активности или диаграммами видов деятельности, были введены в язык UML сравнительно недавно. Диаграмма деятельности - это, по существу, блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Результат может привести к изменению состояния системы или возвращению некоторого значения [9].

Диаграмма деятельности отличается от традиционной блок-схемы

- более высоким уровнем абстракции;
- возможностью представления с помощью диаграмм деятельности управления параллельными потоками наряду с последовательным управлением.

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния [6].

Диаграмма деятельности, используемая для сравнения простых и сложных процентов посредством графика представлена на рисунке 5.5.

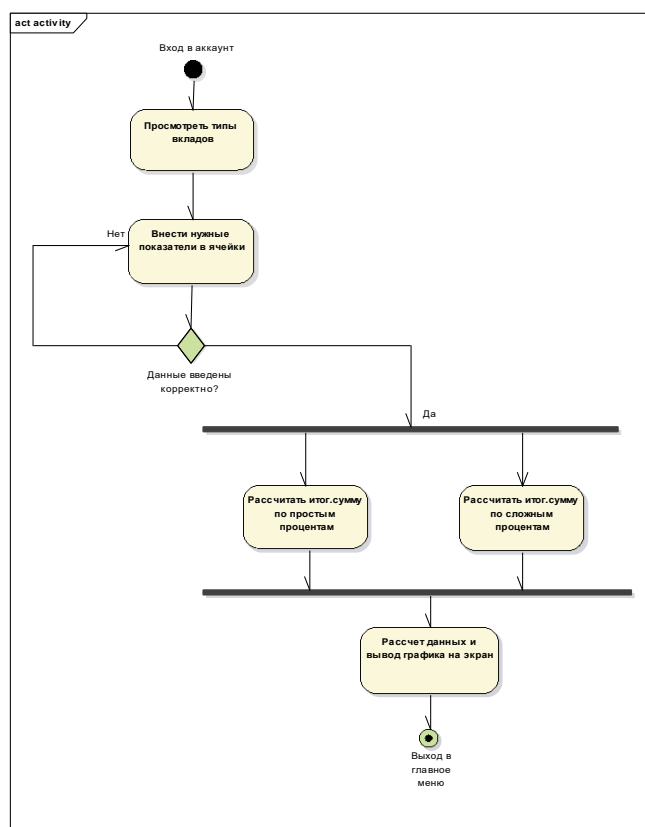


Рисунок 5.5 – Диаграмма деятельности

5.6 Описание диаграммы развёртывания системы

Диаграмма развёртывания дополняет собой представления о физическом уровне системы, описывая топологию и необходимые аппаратные средства. При проектировании больших систем такая схема помогает понять, какие недостатки имеет физический уровень и какая аппаратура необходима для полноценного функционирования системы в нормальном и аварийных режимах.

Диаграмма развёртывания для проектируемой системы изображена на рисунке 5.6.

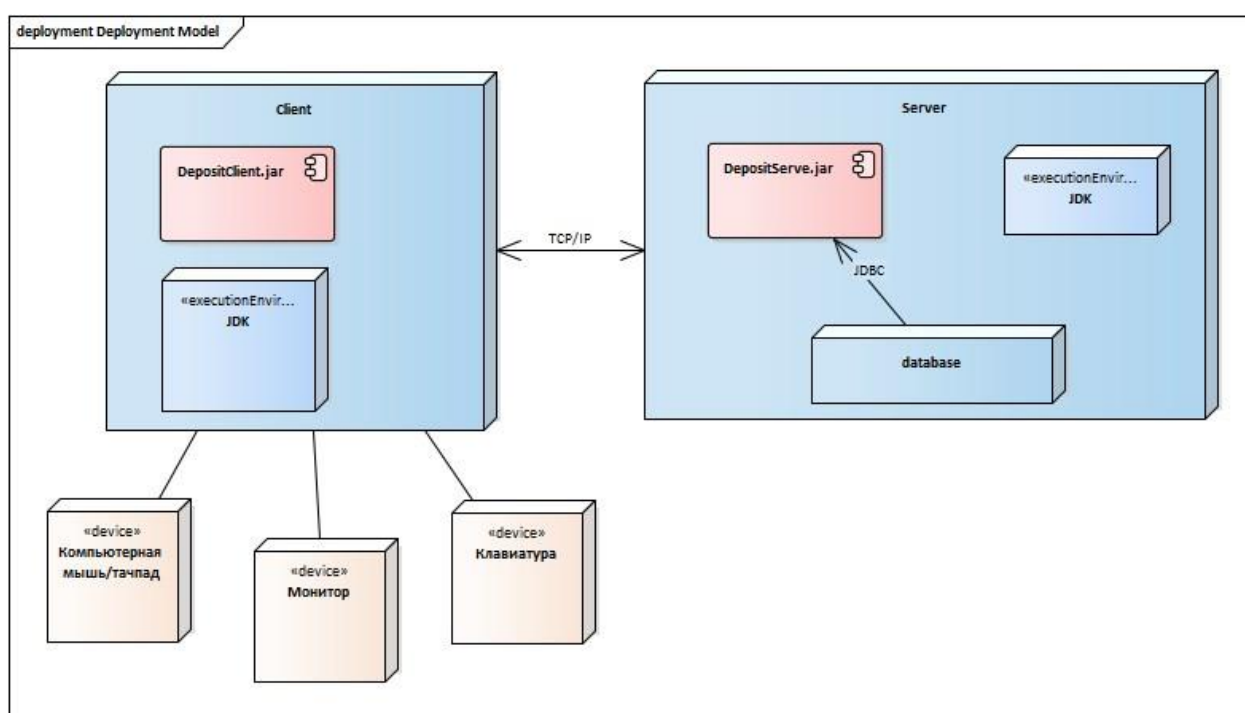


Рисунок 5.6 – Диаграмма развёртывания

Как видно, данная диаграмма не только показывают аппаратное обеспечение, но и отражает, как именно осуществляется связь между компонентами (например, связь между клиентом и сервером осуществляется по протоколу *TCP/IP*, а между сервером и базой данных – с помощью java-стандарта взаимодействия с базой данных *JDBC*).

6 ОБОСНОВАНИЕ ОРИГИНАЛЬНЫХ РЕШЕНИЙ ПО ИСПОЛЬЗОВАНИЮ ТЕХНИЧЕСКИХ И ПРОГРАММНЫХ СРЕДСТВ

Разработка любых приложений и программных обеспечений подразумевает под собой использование различных средств, которые помогут облегчить и усовершенствовать работу, а также создать оригинальное решение по использованию различных средств.

В данном приложении были применены некоторые паттерны, которые помогли усовершенствовать его работу и оптимизировать написание кода.

В сфере разработки программного обеспечения паттерн проектирования – это повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Паттерны проектирования представляют собой обобщение опыта профессиональных разработчиков ПО. Паттерн проектирования можно рассматривать как некий шаблон, в соответствии с которым пишут программы [3].

На рисунке 6.1 представлены 3 типа паттернов, чаще всего используемых для написания программного кода.

-  — поведенческие (behavioral);
-  — порождающие (creational);
-  — структурные (structural).

Рисунок 6.1 – Виды паттернов

Первая группа – поведенческие шаблоны (behavioral). Они определяют эффективные способы взаимодействия различных объектов в системе. Вторая группа – это creational, т.е. порождающие паттерны. Они в той или иной степени работают с механизмами создания объектов. И, наконец, третья

группа – структурные паттерны (structural). Они описывают создание более сложных объектов, либо упрощают работу с другими объектами системы [8].

В данном приложении были использованы два порождающих паттерна: одиночка и строитель.

Синглтон (одиночка) – это шаблон (паттерн) проектирования, который выполняет две главные роли:

1. Дает гарантию, что у класса будет всего один экземпляр.
2. Предоставляет глобальную точку доступа к экземпляру данного класса.

Отсюда две особенности, характерные для практически каждой реализации паттерна синглтон:

1. Приватный конструктор. Ограничивает возможность создания объектов класса за пределами самого класса.
2. Публичный статический метод, который возвращает экземпляр класса. Данный метод называют `getInstance()`. Это глобальная точка доступа к экземпляру класса [4].

Паттерн Builder позволяет создавать различные варианты объекта, избегая загрязнения конструктора. Полезно, когда может быть несколько вариантов объекта. Или, когда есть много шагов, вовлеченных в создание объекта.

Цель паттерна Builder – отделить строительство сложного объекта от его представления, так что один и тот же процесс строительства может создать разные представления.

Строитель чаще всего используется в случаях, когда:

- алгоритм создания сложного объекта должен быть независим от частей, составляющих объект, и от того, как они собираются
- процесс строительства должен позволять различные представления для объекта, который построен [5].

Таким образом, используя паттерны, перечисленные выше, нам удастся достичь усовершенствования работы нашего приложения, а также облегчить написание программного кода.

7 ОПИСАНИЕ АЛГОРИТМОВ

Рассмотрим основные алгоритмы функционирования разрабатываемой системы, а именно: авторизацию, работу администратора, консультанта и клиента, а также алгоритм добавления записи в таблицу базы данных (добавление в другие таблицы реализуется аналогично) [7].

7.1 Алгоритм работы администратора

Блок-схема, показанная на рисунке 7.1, отображает основные функции администратора системы. Блоки, начинающиеся со слов «Работа с...», представляют собой основные функции работы с базами данных (*CRUD*), реализованные для конкретных таблиц. Кроме того, администратор способен сравнивать имеющиеся в распоряжении банка деньги, полученные от вкладчиков, и конечные суммы, которые будут им выданы, в виде диаграммы.



Рисунок 7.1 – Блок-схема алгоритма работы администратора

7.2 Алгоритм добавления строки в базу данных клиентов

Как и все представленные алгоритмы, данная последовательность действий интуитивно понятно: чтобы добавить строку в базу данных клиентов, необходимо удостовериться, что пользователя с таким же логином нет в системе, в случае чего передать sql-запрос серверу и выполнить непосредственно добавление.

Блок-схема данного алгоритма представлена на рисунке 7.2.

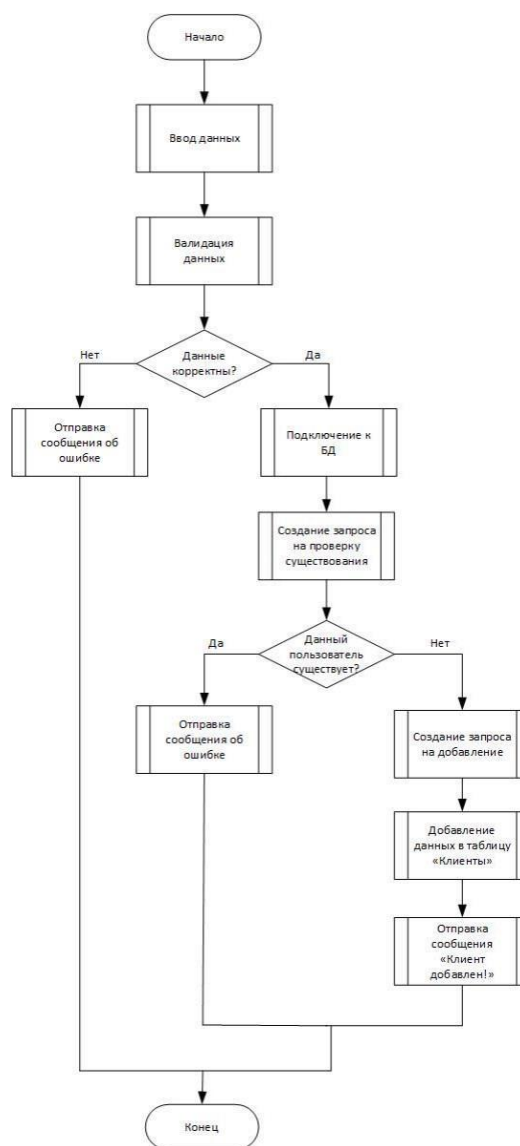


Рисунок 7.2 – Блок-схема алгоритма добавления клиента в базу данных

Таким образом, в главе проиллюстрированы алгоритмы, реализующие фундаментальные требования системы.

8 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Глава «Руководство пользователя» поможет нам продемонстрировать работу созданного приложения. Также, стоит отметить, что для удобства работы с программы и ее запуска были созданы jar-файлы, при помощи которых мы сможем запустить работу нашего сервера и, соответственно, клиента.

Для авторизации пользователя необходимо ввести логин и пароль, а также отметить, является ли данный пользователь сотрудником банка (рис. 8.1).

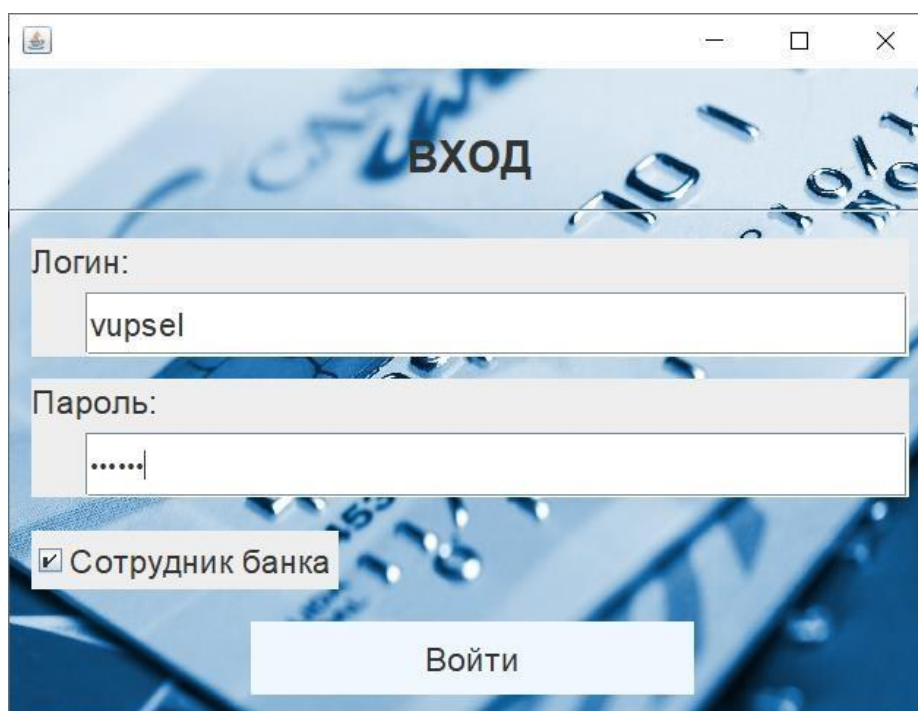
The image shows a web browser window with a login form. The title bar of the window is not visible. The form has a blue background with a faint image of a calculator. The word 'ВХОД' (Login) is centered at the top in a large, bold, black font. Below it, there are three input fields. The first is labeled 'Логин:' (Login) and contains the text 'vupsel'. The second is labeled 'Пароль:' (Password) and contains seven dots '.....'. The third is a checkbox labeled 'Сотрудник банка' (Bank employee), which is checked. Below these fields is a large, light blue button labeled 'Войти' (Login).

Рисунок 8.1 – Форма авторизации пользователя

Если введенные данные корректны и пользователь с такими данными существует, ему будет предоставлен соответствующий доступ.

Главное меню администратора выглядит следующим образом (рис. 8.2).

Вкладка «Клиенты» переводит администратора на форму работы с клиентами. В окне отображается таблица клиентов и кнопки, отвечающие за работу с данной таблицей, причем кнопки редактирования и удаления записей активируются только в том случае, если администратор выбрал определенную строку таблицы.

Также стоит отметить, что в данном окне сортирует сортировка по ФИО, а также поиск по таким критериям, как: логин, паспорт и телефон. Окно изображено на рисунке 8.3.

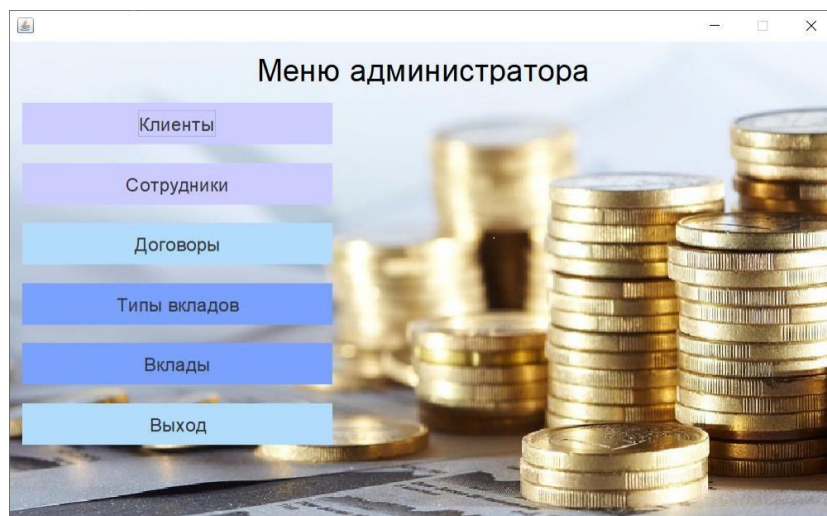


Рисунок 8.2 – Меню администратора

Вкладка «Клиенты» переводит администратора на форму работы с клиентами. В окне отображается таблица клиентов и кнопки, отвечающие за работу с данной таблицей, причем кнопки редактирования и удаления записей активируются только в том случае, если администратор выбрал определенную строку таблицы.

Также стоит отметить, что в данном окне сортирует сортировка по ФИО, а также поиск по таким критериям, как: логин, паспорт и телефон. Окно изображено на рисунке 8.3.

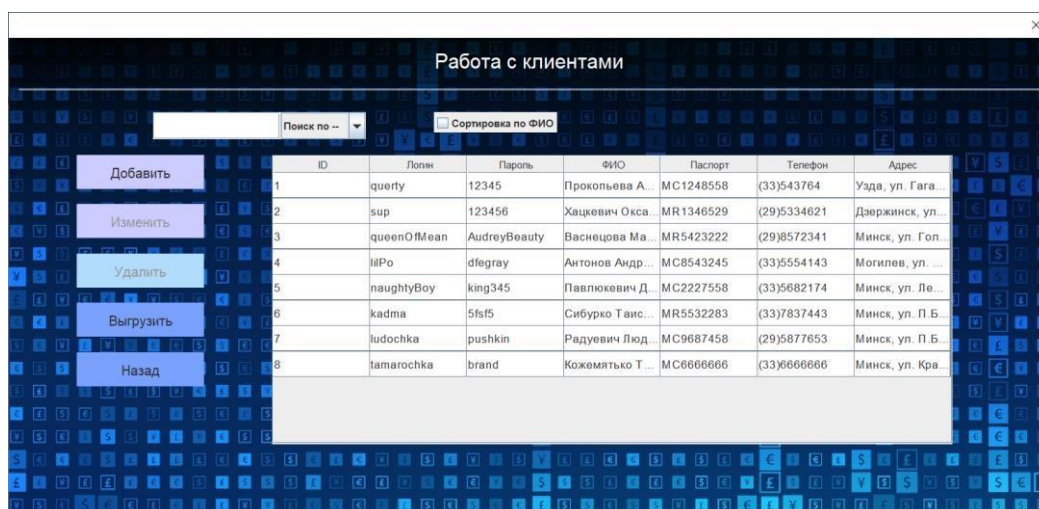


Рисунок 8.3 – Работа с клиентами

Для добавления клиента необходимо ввести все данные, причем важно отметить, что *id* клиента формируется автоматически (рис. 8.4).

Добавленный клиент сразу же отобразится в имеющейся таблице благодаря обновлению базы данных.

Добавление клиента

ID клиента: 9

Логин: vadim

Пароль:

ФИО: Гайдук Вадим Иванович

Паспорт: MC2790564

Номер телефона: (33)3667549

Адрес: г. Дзержинск, улица Минская, д.17

Добавить

Рисунок 8.4 – Окно добавления клиента

Отображение добавления нового клиента представлено на рисунке 8.5.

Работа с клиентами

Поиск по -- Сортировка по ФИО

ID	Логин	Пароль	ФИО	Паспорт	Телефон	Адрес
1	querty	12345	Прокопьева А...	MC1248558	(33)543764	Узда, ул. Гага...
2	sup	123456	Хацкевич Ока...	MR1346529	(29)5334621	Дзержинск, ул...
3	queenOfMean	AudreyBeauty	Васнецова Ма...	MR5423222	(29)8572341	Минск, ул. Гол...
4	HiPo	dfegray	Антонов Андр...	MC8543245	(33)5554143	Могилев, ул. ...
5	naughtyBoy	king345	Павлюкевич Д...	MC2227558	(33)5682174	Минск, ул. Ле...
6	kadma	5fsf5	Сибурко Таис...	MR5532283	(33)7837443	Минск, ул. П.Б...
7	ludochka	pushkin	Радужевич Люд...	MC9687458	(29)5877653	Минск, ул. П.Б...
8	tamarochka	brand	Кожемятко Т...	MC6666666	(33)6666666	Минск, ул. Кра...
9	vadim	password	Гайдук Вадим ...	MC2790564	(33)3667549	г. Дзержинск, ...

Добавить

Изменить

Удалить

Выгрузить

Назад

Рисунок 8.5 – Результат добавления нового клиента

Чтобы отредактировать какую-либо информацию о пользователе, нужно выбрать в списке данного пользователя и нажать кнопку «Изменить». В нашем случае, я внесу изменения в запись клиента с ID номер 4(изменим его адрес проживания).

Также мы можем удалить какую-либо запись о пользователе также выбрав этого пользователя и нажав кнопку «Удалить». Удалим созданного нами ранее пользователя с ID номер 9.

Результаты выполнения редактирования и удаления представлены на рисунке 8.6.

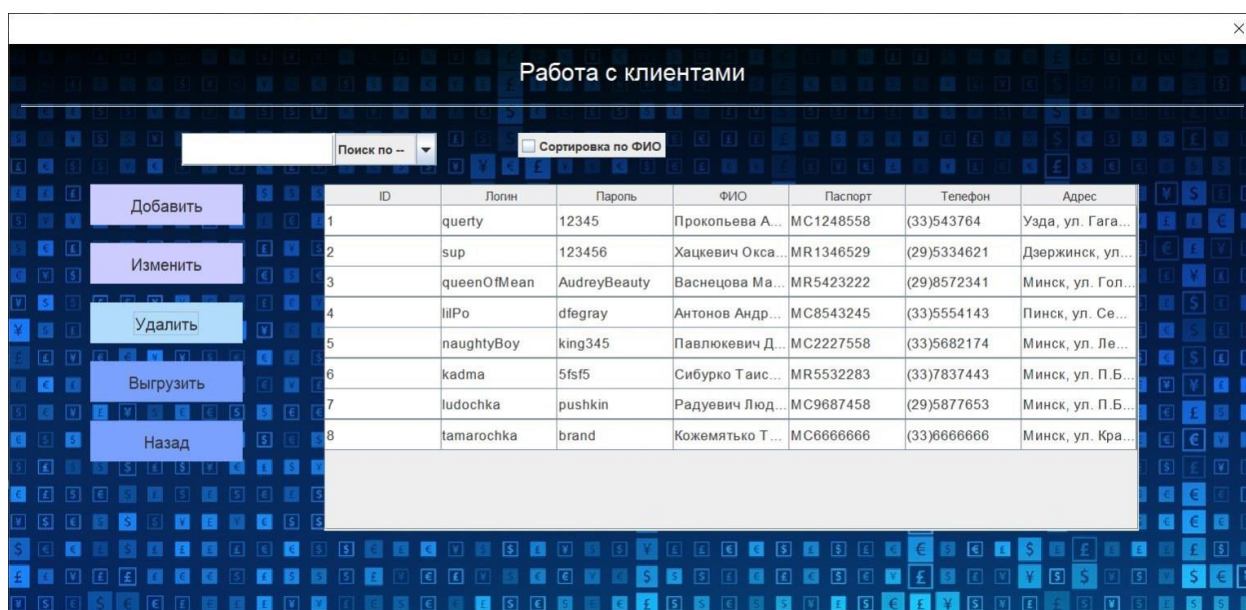


Рисунок 8.6 – Результат редактирования и удаления

Кроме того, мы можем вывести в файл все имеющиеся записи о клиентах, нажав кнопку «Выгрузить».

При совершении какого-либо действия пользователю показывается окно, информирующего его о том, что действие совершено (рис. 8.7).

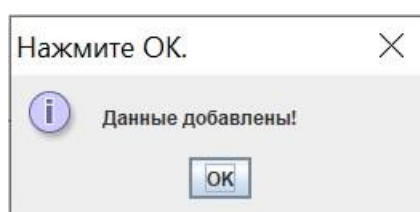


Рисунок. 8.7 – Добавление данных в файл

Работа с остальными таблицами проходит аналогичным способом, однако в окне «Вклады» есть кнопка «Диаграмма», позволяющая графически сравнить имеющиеся в банке средства, предоставленные в пользование вкладчиками, и итоговые суммы, которые банк в итоге выплатит клиенту (рис. 8.8).

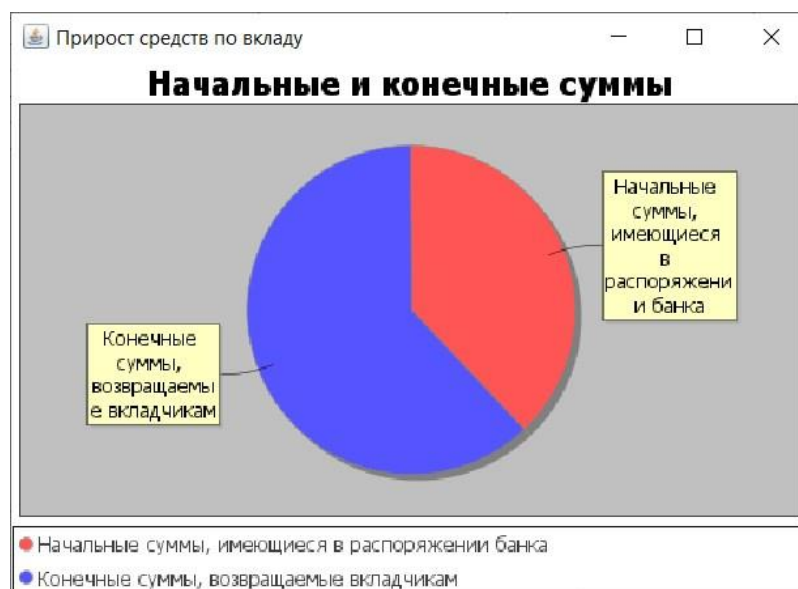


Рисунок 8.8 – Диаграмма средств в банке

После авторизации в режиме консультанта высвечивается меню следующего вида (рис. 8.9).

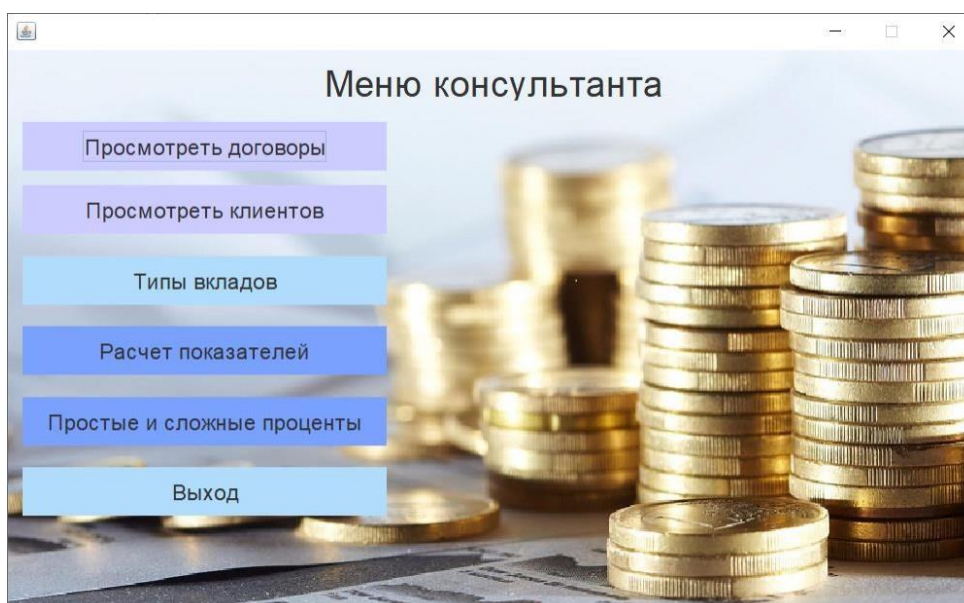


Рисунок 8.9 – Меню консультанта

Как уже было отмечено ранее, консультант не может просматривать все договоры и всех клиентов, однако у него есть доступ к тем, с кем/которыми он работал напрямую (рис. 8.10).

ID	Логин	Пароль	ФИО	Паспорт	Телефон	Адрес
3	queenOfMean	AudreyBeauty	Васнецова Мария Серг	MR5423222	(29)8572341	Минск, ул. Голубева, 1
6	kadmia	9fat5	Сибурко Татьяна Карен	JMR5532283	(33)7837443	Минск, ул. П. Бровки, 6
7	ludochka	pushkin	Радзевич Людмила Мх.	MC9687458	(29)5877653	Минск, ул. П. Бровки, 6

Рисунок 8.10 – Договоры, заключенные данным консультантом

Рассмотрим вкладку «Расчет экономических показателей». Данный расчет можно производить тремя различными способами:

- рассчитать конечную сумму по начальной, капитализации, периоду и процентной ставке;
- рассчитать начальную сумму по конечной, капитализации, периоду и процентной ставке;
- рассчитать процентную ставку по начальной и конечной суммам, капитализации и периоду.

После расчета будет также построен график, отражающий рост средств вкладчика (рис. 8.11).



Рисунок 8.11 – Расчет экономических показателей

Как видно из графика, расчет идет согласно формуле сложных процентов.

Однако консультант имеет возможность сравнить вклады, реализуемые по простому и сложному процентам (рис. 8.12).

По данному графику видно, что сложные проценты гораздо выгоднее простых для большого периода вклада, хотя при маленьких сроках они примерно равнозначны.

На этом уникальные возможности консультанта заканчиваются.

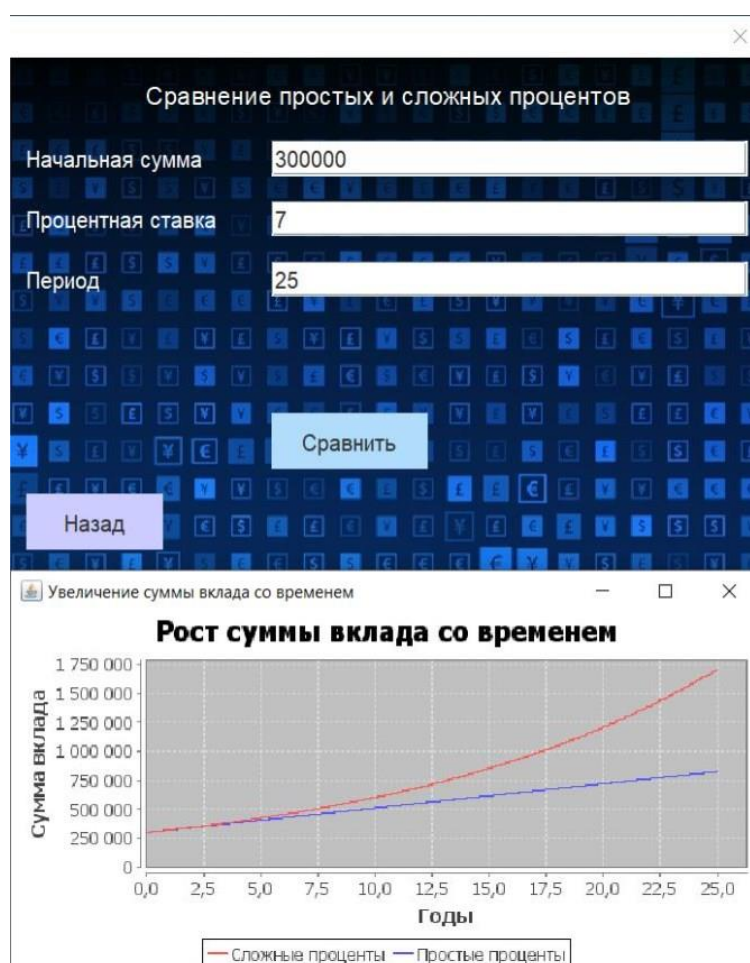


Рисунок 8.12 – Сравнение простых и сложных процентов

После авторизации в режиме клиента всплывает меню следующего вида (рис. 8.13).

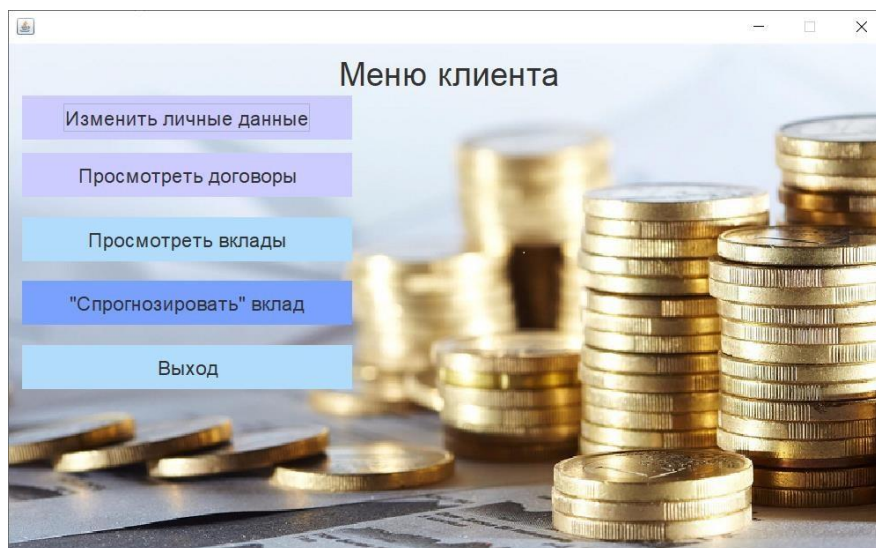


Рисунок 8.13 – Меню клиента банка

Как уже отмечалось ранее, клиент имеет возможность редактировать персональные данные, включая логин, пароль и номер телефона (рис. 8.14).

Рисунок 8.14 – Изменение персональных данных клиента

Как и консультант, клиент имеет возможность просматривать договоры, которые он заключил, и вклады, которые он открыл. Кроме того, он может «спрогнозировать» будущий вклад, введя начальную сумму, капитализацию, срок и период вклада. Система посчитает конечную сумму и начертит график роста средств потенциального вкладчика. (рис. 8.15).

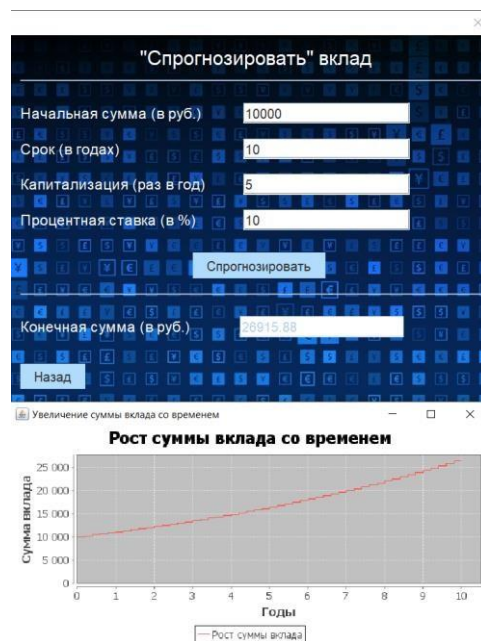


Рисунок 8.15 – Прогнозирование вклада клиента

На этом уникальные возможности клиента закончены.

Стоит также отметить, что в данной программе существует обработка исключительных ситуаций, результаты работы которой будут представлены ниже.

На этапе авторизации, если введенные данные некорректны (пользователь с таким логином и паролем не найдены), будет выведено соответствующее сообщение (рис. 8.16).

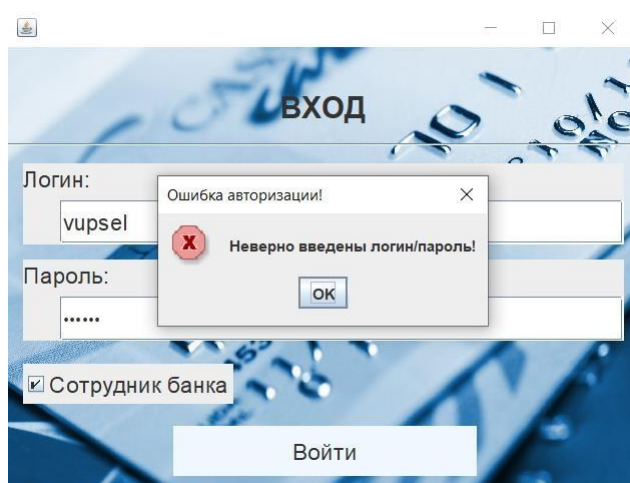


Рисунок 8.16 – Неверно введенные логин и пароль

Корректность данных проверяется также при добавлении клиента или сотрудника. Если пользователь с таким логином уже существует,

администратору придет уведомление (рис. 8.17). Аналогичная ситуация произойдет, если клиент попытается изменить логин на уже использующийся в системе (за одной особенностью: по вышеуказанной логике клиент не сможет изменить логин на собственный, поэтому эту особенность надо учитывать; для возможности оставить при изменении данных логин неисправленным создана отдельная функция).

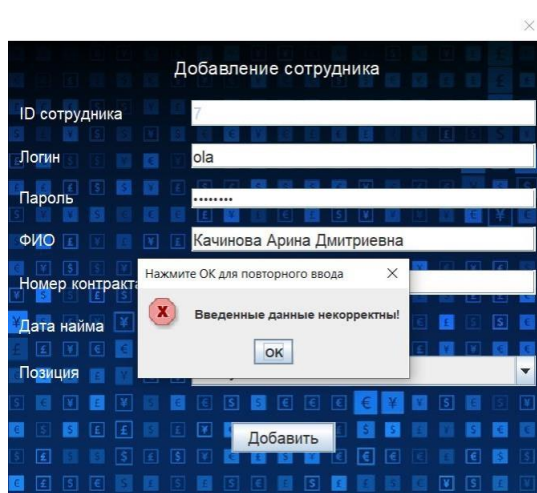


Рисунок 8.17 – Пользователь с таким логином уже существует в системе

Всячески валидируется и другие данные. Во-первых, те текстовые данные, которые имеют в базе данных тип varchar, не должны превышать по длине двадцать символов, а числовые поля не должны содержать никаких знаков, кроме цифр и разделителя целой и дробной частей. Отметим также, что числа не должны быть неположительными, поскольку они используются только для отображения положительных величин (id, денежная сумма, процентная ставка и так далее). Валидация данного типа представлена на рисунке 8.18.

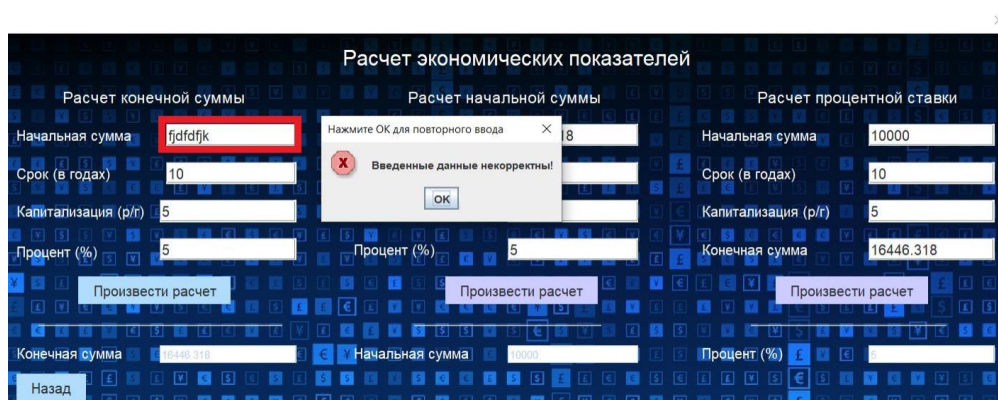


Рисунок 8.18 – Некорректное число

Поскольку дата в системе представляется строкой типа String, важно установить ей соответствующий стандарт. В разработанной системе придерживаются стандарта «YYYY-MM-DD», где YYYY – год, MM – месяц (восьмой месяц указывается как «08», а не просто «8»), DD – день. Месяц не превышает двенадцати, а день – тридцати одного, год не может быть больше 2020. Валидация даты представлена на рисунке 8191.

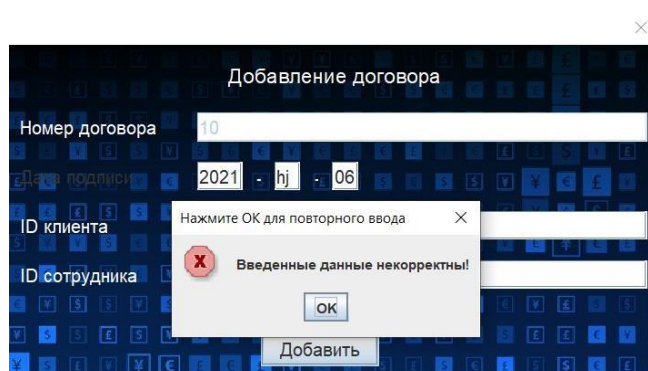


Рисунок 8.19 – Валидация даты

Валидация происходит и в других случаях. Например, при расчете процентной ставки консультантом сообщение о некорректности данных будет выведено и в случае, если конечная сумма вклада меньше начальной (рис. 8.20). Несмотря на то что даже в этом случае процентную ставку можно вычислить (она будет отрицательной), это противоречит бизнес-логике и не имеет смысла.

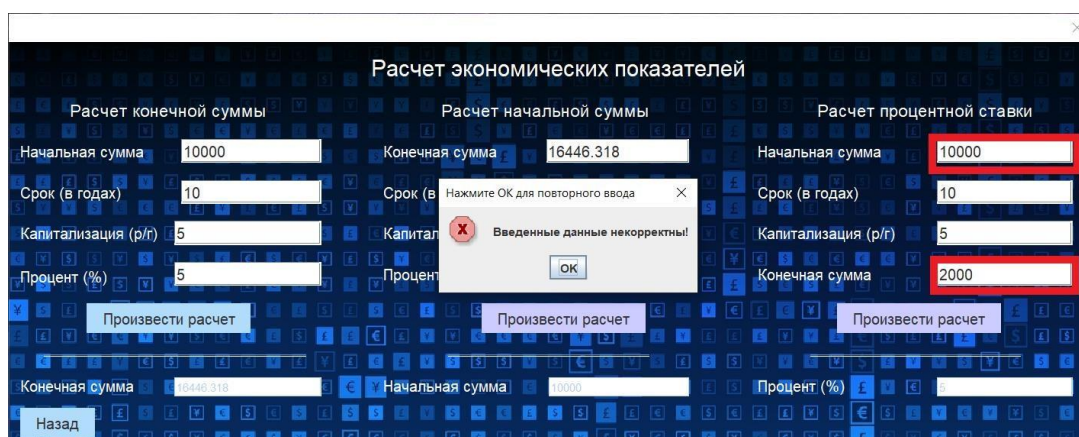


Рисунок 8.20 – Неверно введенные начальная и конечные суммы

9 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

Любая разрабатываемая система должна подвергаться тестированию со стороны разработчика либо специально обученного человека. Для достижения отличной работы данной программы также были написаны некоторые тесты.

Тестирование программного кода - процесс выполнения программного кода, направленный на выявление существующих в нем дефектов. Под дефектом здесь понимается участок программного кода, выполнение которого при определенных условиях приводит к неожиданному поведению системы (т.е. поведению, не соответствующему требованиям). Неожиданное поведение системы может приводить к сбоям в ее работе и отказам, в этом случае говорят о существенных дефектах программного кода. Некоторые дефекты вызывают незначительные проблемы, не нарушающие процесс функционирования системы, но несколько затрудняющие работу с ней. В этом случае говорят о средних или малозначительных дефектах.

Также стоит отметить, что для тестирования приложений используется такой фреймворк, как JUnit.

JUnit — это инфраструктура модульного тестирования для языка программирования Java. Он играет решающую роль при разработке на основе тестов и представляет собой семейство платформ модульного тестирования, известных под общим названием xUnit.

JUnit продвигает идею «сначала тестирование, а затем кодирование», которая делает упор на настройке тестовых данных для фрагмента кода, который можно сначала протестировать, а затем реализовать. Этот подход похож на «немного протестировать, немного кодировать, немного протестировать, немного кодировать». Это увеличивает производительность программиста и стабильность программного кода, что, в свою очередь, снижает нагрузку на программиста и время, затрачиваемое на отладку.

Реализация проведенных процессов в коде представлена в программном коде ниже.

```
public class TestServerWork {  
  
    @Before  
    public void setConnection() {  
        ThreadServer.setConnection();  
    }  
  
    @Test
```

```

public void testSelect() throws SQLException {
    int expectedAmount = 5;
    String query = "select * from deposits";
    ResultSet resultSet = DatabaseConnection.getInstance().select(query);
    int actualAmount = 0;
    while (resultSet.next()) {
        actualAmount++;
    }
    Assert.assertEquals(expectedAmount, actualAmount);
}

@Test
public void testInsert() throws SQLException {
    int expectedAmount = 9;
    String insertQuery = "insert into contacts VALUES(9,'2018-11-19', 4, 3)";
    DatabaseConnection.getInstance().insert(insertQuery);
    String selectQuery = "select * from contacts";
    ResultSet resultSet = DatabaseConnection.getInstance().select(selectQuery);
    int actualAmount = 0;
    while (resultSet.next()) {
        actualAmount++;
    }
    Assert.assertEquals(expectedAmount, actualAmount);
}
}

```

Как видно из приведённого кода программы, каждый тест в обязательном порядке помечается аннотацией `@Test`. Также стоит отметить, что часть кода, помеченная аннотацией `@Before`, выполняется всегда перед реализацией любого из тестов.

Реализация тестов представлена на рисунке 9.1.

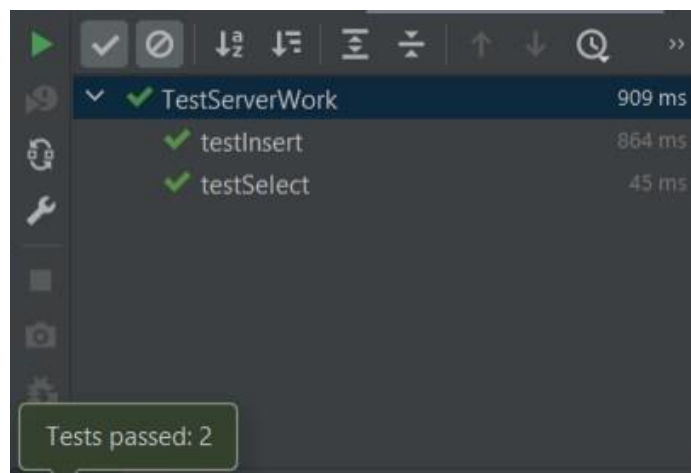


Рисунок 9.1 – Результат запуска тестов

Проведение множества тестирований каждой автоматизированной системы является важным этапом её разработки. Оценивая каждую часть кода изолированно и подтверждая корректность её работы, значительно проще установить проблему, чем если бы элемент был частью системы.

ЗАКЛЮЧЕНИЕ

В заключение отметим, что итогом написания курсового проекта является не только функционирующая информационная система, позволяющая оптимизировать работу банка, но и идущая с ней документация, включающая исходные требования к системе, подробное описание бизнес-логики, используемых типов данных, вариантов использования и множественные диаграммы, помогающие понять, как работает данное программное средство.

В ходе выполнения курсового проектирования было подробно изучена предметная область и связанные с ней математические модели (простые и сложные проценты). Для того чтобы упростить понимание предметной области и ускорить процесс проектирования системы, были построены диаграммы в нотации *IDEFO* и *UML*.

Разработанное приложение соответствует архитектуре «клиент – сервер», причем важно отметить, что оно также является многопользовательским. Это значит, что доступ к разработанной системе может быть предоставлен одновременно нескольким пользователям.

Приложение поддерживает три типа пользователей – администратора, консультанта и клиента.

Администратор и консультант являются сотрудниками банка. Администратор является своего рода связующим звеном между бизнес-процессами, происходящими в банке, и информацией, хранящейся в базе данных. Его функции – это стандартные возможности администратора баз данных, а именно: просмотр, добавление, редактирование, удаление информации. Он также может сравнивать количество денег, предоставленных банку вкладчиками, и средствами, которые в итоге будут выплачены клиентам.

Возможности консультанта более узкоспециализированные: он может иметь доступ к просмотру только тех данных, которые либо являются общедоступными для всех сотрудников (типы вкладов), либо относятся к нему напрямую, например, договоры, которые он заключал с клиентами, и эти клиенты. Поскольку главная его бизнес-задача – это консультирование потенциальных вкладчиков, он имеет возможность рассчитывать экономические показатели по будущему вкладу и графически ее интерпретировать, а также сравнивать разные методы начисления процентов – простой и сложный.

У клиента самый низкий уровень доступа. Он, как и консультант, имеет доступ только к той информации, которая относится к нему напрямую

(открытые им вклады, подписанные им договоры), а также может изменить личную информацию (логин, пароль, номер телефона). Для удобства он может «спрогнозировать» потенциальный вклад по начальной сумме и другим экономическим показателям.

В дальнейшем функционал системы можно расширять, сделать ее веб-приложением. В таком случае конкурентоспособность банка, пользующегося данной системой, вырастет в сравнении с другими.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Банковский кодекс Республики Беларусь от 25 октября 2000 г., № 441-З (с изменениями и дополнениями по состоянию на 17.07.2018) — ЭТАЛОН. Законодательство Республики Беларусь — Нац. центр правовой информ. Респ. Беларусь : Минск, 2019.

[2] Финам [Электронный ресурс]. Режим доступа: <http://surl.li/awzjd>. — Дата доступа: 23.09.2021.

[3] Habr [Электронный ресурс]. Режим доступа : <https://habr.com/ru/company/ruvds/blog/427293/>. — Дата доступа: 02.10.2021.

[4] JavaRush [Электронный ресурс]. Режим доступа : <https://javarush.ru/groups/posts/2365-patternih-proektirovanija-singleton>. — Дата доступа: 02.10.2021.

[5] Java Blog [Электронный ресурс]. Режим доступа : <https://java-ru-blog.blogspot.com/2019/10/builder-pattern-java.html>. — Дата доступа: 02.10.2021.

[6] Ami Nstu [Электронный ресурс]. Режим доступа : <https://ami.nstu.ru/~vms/method9/lab5.htm>. — Дата доступа: 18.10.2021.

[7] Репин В. В., Елиферов В. Г. Процессный подход к управлению. Моделирование бизнес- процессов / Владимир Репин, Виталий Елиферов. - М. : Манн, Иванов и Фербер, 2013.

[8] Шилдт, Герберт. Java. Полное руководство, 10-е изд. : Пер. с англ. - СПб. ООО "Альфа-книга"; 2018. - 934 с.

[9] Хорстманн, Кей С., Корнелл, Гари. Java. Библиотека профессионала, том 1. Основы. 9-е изд. : Пер. с англ. — М. : ОО «И.Д. Вильямс», 2014 — 864 с.

[10] Хорстманн, Кей С., Корнелл, Гари. Java. Библиотека профессионала, том 2. Основы. 9-е изд. : Пер. с англ. — М. : ОО «И.Д. Вильямс», 2014 — 1008 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Код программы

Запуск сервера и установка многопоточного соединения:

```
Properties properties = new Properties();
properties.load(DepositServer.class.getClassLoader().getResourceAsStream("server.properties"));
PORT = Integer.parseInt(properties.getProperty("port"));

try (ServerSocket servSocket = new ServerSocket(PORT)) {
    System.out.println("Сервер запущен");
    while (true) {
        Socket socket = servSocket.accept();
        System.out.println("Был подключен новый клиент");
        try {
            new ThreadServer(socket);
        } catch (IOException e) {
            socket.close();
        }
    }
}
```

Класс, отвечающий за установку соединения с базой данных (в данном классе также реализуется паттерн «Одиночка»):

```
public class DatabaseConnection {
    private static DatabaseConnection instance;
    private String driverName;
    private Connection connect;
    private Statement statement;

    private DatabaseConnection() {
    }

    public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }

    public void setDriverName(String driverName) throws ClassNotFoundException {
        this.driverName = driverName;
        Class.forName(this.driverName);
    }
}
```


Продолжение приложения А

```
public void initConnection(String url, String name, String pass) throws SQLException {
    this.connect = DriverManager.getConnection(url, name, pass);
    this.statement = this.connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
    this.statement.execute("set character set utf8");
    this.statement.execute("set names utf8");
}

public void insert(String sqlString) {
    try {
        statement.executeUpdate(sqlString);
    } catch (SQLException ex) {
        Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void delete(String sqlString) {
    try {
        statement.executeUpdate(sqlString);
    } catch (SQLException ex) {
        Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void update(String sqlString) {
    try {
        statement.executeUpdate(sqlString);
    } catch (SQLException ex) {
        Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public ResultSet select(String sqlString) {
    ResultSet rs = null;
    try {
        rs = statement.executeQuery(sqlString);
    } catch (SQLException ex) {
        Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
    return rs;
}

public void close() {
    try {
        connect.close();
        statement.close();
    } catch (SQLException ex) {
```

Продолжение приложения А

```
        Logger.getLogger(DatabaseConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}
```

Метод, отвечающий за запуск потока на выполнение:

```
@Override
public void run() {
    int idOperation;
    ServerWork obj = new ServerWork(in, out);
    try {
        while (true) {
            String bufString = in.readLine();
            if (bufString.equals("END")) {
                DatabaseConnection.getInstance().close();
                System.out.println("Сервер отсоединен от базы данных");
                break;
            }
            idOperation = Integer.parseInt(bufString);
            obj.getId(idOperation);
        }
        System.out.println("Клиент был отсоединен");
    } catch (IOException ex) {
        System.err.println("IO Exception");
    } catch (SQLException ex) {
        Logger.getLogger(ThreadServer.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            socket.close();
        } catch (IOException ex) {
            System.err.println("Socket not closed");
        }
    }
}
```

Класс, реализующий соединение клиента с сервером:

```
public class ClientSocket {

    private Socket clientSocket;
    private ObjectOutputStream outputStream;
    private ObjectInputStream inputStream;

    private String message;
```

Продолжение приложения А

```
public ClientSocket(String ipAddress, String port) {
    try {
        clientSocket = new Socket(ipAddress, (int)Double.parseDouble(port));
        outputStream = new ObjectOutputStream(clientSocket.getOutputStream());
        inputStream = new ObjectInputStream(clientSocket.getInputStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void sendMessage(String message) {
    try {
        outputStream.writeObject(message);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void sendObject(Object object) {
    try {
        outputStream.writeObject(object);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public String readMessage() {
    try {
        message = (String) inputStream.readObject();
    } catch (ClassNotFoundException | IOException e) {
        e.printStackTrace();
    }

    return message;
}

public Object readObject() {
    Object object = new Object();
    try {
        object = inputStream.readObject();
    } catch (ClassNotFoundException | IOException e) {
        e.printStackTrace();
    }
    return object;
}

public void close() {
```

Продолжение приложения А

```
try {
    clientSocket.close();
    inStream.close();
    outStream.close();
} catch (IOException e) {
    e.printStackTrace();
}
}
```

Реализация шаблона «Строитель» для класса депозитов:

```
public interface DepositBuilderInterface {
    DepositBuilderInterface buildNumber(int depositNumber);
    DepositBuilderInterface buildInitialMoney(int initialMoney);
    DepositBuilderInterface buildDateOfOpening(String dateOfOpening);
    DepositBuilderInterface buildPlannedAmountOfMoney(int plannedAmountOfMoney);
    DepositBuilderInterface buildTypeOfDeposit(int typeOfDeposit);
    DepositBuilderInterface buildContractNumber(int contractNumber);
    DepositBuilderInterface buildClientId(int clientID);
    Deposit build();
}

public class DepositBuilder implements DepositBuilderInterface {
    private Deposit deposit;

    public DepositBuilder() {
        deposit = new Deposit();
    }

    @Override
    public DepositBuilderInterface buildNumber(int depositNumber) {
        deposit.setDepositNumber(depositNumber);
        return this;
    }

    @Override
    public DepositBuilderInterface buildInitialMoney(int initialMoney) {
        deposit.setInitialMoney(initialMoney);
        return this;
    }

    @Override
    public DepositBuilderInterface buildDateOfOpening(String dateOfOpening) {
        deposit.setDateOfOpening(dateOfOpening);
        return this;
    }
}
```

```

@Override
public DepositBuilderInterface buildPlannedAmountOfMoney(int plannedAmountOfMoney) {
    deposit.setPlannedMoney(plannedAmountOfMoney);
    return this;
}

@Override
public DepositBuilderInterface buildTypeOfDeposit(int typeOfDeposit) {
    deposit.setTypeOfDeposit(typeOfDeposit);
    return this;
}

@Override
public DepositBuilderInterface buildContractNumber(int contractNumber) {
    deposit.setContractNumber(contractNumber);
    return this;
}

@Override
public DepositBuilderInterface buildClientId(int clientId) {
    deposit.setClientId(clientId);
    return this;
}

@Override
public Deposit build() {
    return deposit;
}
}

```