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

\_\_\_\_\_\_\_\_\_\_\_\_\_Н.Д. Рязанцев

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Курсового проекта

На тему

**РАЗРАБОТКА ПОДСИСТЕМЫ УЧЁТА И ВЕДЕНИЯ СЕМЕЙНОГО БЮДЖЕТА**

БГУИР КП 1-40 05 01-12 048 ПЗ

Выполнила студентка группы 324404

Лазицкая Евгения Игоревна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

Курсовой проект представлен на

Проверку \_\_\_.\_\_\_\_.2025

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

Минск 2025

**РЕФЕРАТ**

БГУИР КП 1-40 05 01-12 048 ПЗ

РАЗРАБОТКА ПОДСИСТЕМЫ УЧЁТА И ВЕДЕНИЯ СЕМЕЙНОГО БЮДЖЕТА: курсовая работа / Е.И. Лазицкая. – Минск : БГУИР, 2025, – п.з. – 42 с.

Пояснительная записка 44 с., 30 рис., 10 источников, 3 приложения.

Данная курсовая работа посвящена разработке подсистемы учёта и ведения семейного бюджета. Цель работы – оптимизация процесса ведения семейного бюджета и разработка приложения, обеспечивающего удобный и эффективный контроль финансов, что способствует увеличению производительности и улучшению финансовой грамотности населения.

Объектом исследования являются существующие приложения и подходы к учёту семейных финансов. Предметом исследования выступают методы управления и отслеживания финансов.

В ходе работы применялись современные методы обработки данных, проведен функциональный анализ процессов, а также выполнено моделирование системы с использованием UML-диаграмм. В результате изучены методы управления и отслеживания финансов, смоделирована архитектура подсистемы и реализовано программное обеспечение для учёта и ведения семейного бюджета.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc199778892)

[1. Организация деятельности подсистемы учёта семейного бюджета 7](#_Toc199778893)

[1.1 Обзор предметной области 7](#_Toc199778894)

[1.2 Обзор программных аналогов 8](#_Toc199778895)

[2. Постановка задачи для разработки подсистемы учёта семейного бюджета 10](#_Toc199778896)

[3. Функциональная модель процесса учёта и ведения семейного бюджета 11](#_Toc199778897)

[4. Информационная модель процесса учёта и ведения семейного бюджета 14](#_Toc199778898)

[5. Модели представления подсистемы на основе UML дианрамм 16](#_Toc199778899)

[5.1. Спецификация вариантов использования подсистемы учёта семейного бюджета 16](#_Toc199778900)

[5.2. Описание диаграммы последовательности подсистемы учёта семейного бюджета 17](#_Toc199778901)

[5.3. Описание диаграммы состояний 18](#_Toc199778902)

[5.4. Описание диаграммы развёртывания и компонентов 19](#_Toc199778903)

[5.5. Описание диаграммы классов 20](#_Toc199778904)

[6. Описание алгоритмов, реализующих бзнес ̶ логику подсистемы 22](#_Toc199778905)

[6.1. Алгоритма взаимодействия клиента и сервера 22](#_Toc199778906)

[6.2. Алгоритм работы пользователя 23](#_Toc199778907)

[6.3. Алгоритм работы администратора 24](#_Toc199778908)

[7. Руководство пользователя 25](#_Toc199778909)

[8. Результаты тестирования 26](#_Toc199778910)

[Заключение 32](#_Toc199778911)

[Список использованных источников 33](#_Toc199778912)

[Приложение А (обязательное) Отчёт о проверке на заимствования в системе «Антиплагиат» 34](#_Toc199778913)

[Приложение Б (обязательное) Листинг кода алгоритмов, реализующих бизнес-логику 35](#_Toc199778914)

[Приложение В (обязательное) Скрипт создания базы данных 41](#_Toc199778915)

[Ведомость документов 44](#_Toc199778916)

# 

# **ВВЕДЕНИЕ**

Сегодня практически невозможно представить мир без информационных технологий. Почти в каждой семье есть компьютер или ноутбук, а иногда даже несколько устройств. Информационные технологии стали неотъемлемой частью повседневной жизни, охватывая все сферы — от образования до управления финансами.

Современные технологии дают возможность создавать разнообразные приложения, направленные на решение повседневных задач и автоматизацию рутинных процессов. Одной из актуальных задач в настоящее время является эффективное управление личными и семейными финансами. В этом контексте всё больше семей приходят к использованию специализированных программных решений — подсистем учёта и ведения семейного бюджета.

Ведение бюджета позволяет грамотно планировать доходы и расходы, анализировать финансовые потоки и вырабатывать более осознанный подход к расходованию средств. Автоматизированные подсистемы для ведения бюджета упрощают этот процесс, обеспечивая структурированное хранение данных, гибкие возможности анализа и визуализации, а также разделение доступа между членами семьи, что особенно важно при совместном управлении финансами.

Эффективное управление бюджетом — это не просто учёт трат, а важный ресурс, который влияет на благополучие всей семьи. Повышение эффективности финансового планирования является важным направлением совершенствования жизни домохозяйства.

Целью курсового проекта является оптимизация процесса ведения семейного бюджета и разработка приложения, обеспечивающего удобный и эффективный контроль финансов.

Для достижения поставленной цели необходимо было решить следующие задачи [1]:

– исследовать существующие приложения и подходы к учёту семейных финансов;

– описать основные процессы, характерные для управления бюджетом семьи;

– смоделировать информационные структуры для хранения данных;

– спроектировать пользовательский интерфейс и представление данных;

– выбрать подходящие программные средства для разработки ПО;

– описать бизнес-логику подсистемы;

– сформировать сценарии работы системы;

– протестировать приложение и устранить выявленные ошибки.

Разработка такого приложения позволит сократить временные и интеллектуальные затраты на управление семейными финансами, пересмотреть необходимость расходов, улучшить коммуникацию между членами семьи и повысить общий уровень финансовой грамотности.

1. **ОРГАНИЗАЦИЯ ДЕЯТЕЛЬНОСТИ ПОДСИСТЕМЫ УЧЁТА СЕМЕЙНОГО БЮДЖЕТА**
   1. **Обзор предметной области**

Вопросы финансовой грамотности и управления личными средствами всегда занимали важное место в жизни каждого человека. Всё больше семей осознают необходимость контроля над своими расходами, планирования бюджета и достижения финансовой стабильности. Эффективное ведение семейного бюджета становится неотъемлемой частью повседневной жизни, позволяя грамотно управлять доходами, контролировать траты и достигать поставленных целей.

Семейный бюджет охватывает множество аспектов: повседневные расходы, коммунальные платежи, обучение, медицинские услуги, отдых, накопления и инвестиции. Для учета этих операций требуется внимательное планирование и системный подход. Каждая из этих категорий имеет свои особенности, и их эффективное отслеживание требует применения современных инструментов и решений.

Финансовое планирование в семье включает в себя учет регулярных доходов (зарплата, пенсии, стипендии, пособия и т. д.) и обязательных расходов. Это требует использования вспомогательных средств: таблиц, блокнотов, а всё чаще — специализированных цифровых приложений и информационных систем. Современные решения для учета семейного бюджета предлагают автоматизированные механизмы ведения записей, анализа трат, построения отчетов и планирования будущих расходов.

Планирование расходов помогает распределить бюджет так, чтобы избежать избыточных трат и накопить средства на крупные цели. К ним могут относиться покупка недвижимости, автомобиля, отпуск, обучение детей или создание финансовой «подушки безопасности». Для этого необходимо грамотно организовать сбор данных и анализ текущего финансового положения семьи.

Инвентаризация ресурсов в контексте бюджета предполагает учет текущих остатков по счетам, наличных средств, долговых обязательств и активов. Своевременное обновление этой информации позволяет видеть реальную картину и принимать обоснованные финансовые решения. Регулярные проверки бюджета помогают корректировать отклонения и избегать критических ситуаций.

Финансовое управление в рамках семейного бюджета включает отслеживание всех операций, построение статистики, анализ источников расходов, а также корректировку планов в зависимости от текущей экономической ситуации. Системы учёта позволяют точно учитывать вложения средств, обеспечивать прозрачность финансов и участвовать всем членам семьи в формировании бюджета.

Сегодня всё больше семей обращают внимание на электронные системы ведения бюджета, благодаря их удобству, доступности и функциональности. Современные приложения позволяют создавать категории доходов и расходов, устанавливать лимиты, получать уведомления о превышении бюджета, формировать графики и отчеты. Эти системы становятся незаменимыми помощниками в процессе контроля финансов.

Подсистемы учёта и ведения семейного бюджета играют значимую роль в формировании ответственного отношения к деньгам и укреплении финансовой дисциплины. Они упрощают процесс учета, минимизируют вероятность ошибок, помогают отслеживать цели и добиваться их реализации. Благодаря автоматизации снижается нагрузка на каждого члена семьи, а сама система становится информативной и удобной для совместного использования.

* 1. **Обзор программных аналогов**

Для эффективного управления семейным бюджетом существует множество программных решений, ориентированных на различные категории пользователей — от новичков до продвинутых пользователей финансовых инструментов. Ниже приведён обзор наиболее популярных и функциональных программ, предназначенных для учёта доходов и расходов в рамках домашней бухгалтерии:

1. Zen-Money — популярное облачное приложение для учёта личных финансов [2] (рисунок 1.1). Поддерживает синхронизацию с банковскими счетами, автоматическое распознавание операций и распределение по категориям. Пользователи могут устанавливать бюджеты, отслеживать статистику и планировать расходы. Интерфейс интуитивно понятен, а аналитика позволяет быстро выявлять «финансовые дыры».

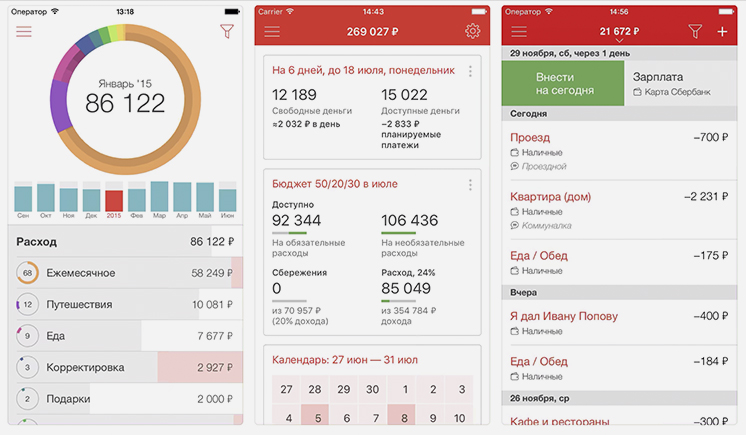


Рисунок1.1 – Вид приложения Zen-Money

1. Дребеденьги – программа, разработанная специально для семейного учёта [3] (рисунок 1.2). Поддерживает совместное использование с разделением прав доступа, ведение бюджета, анализ трат, составление отчётов и напоминания о регулярных платежах.

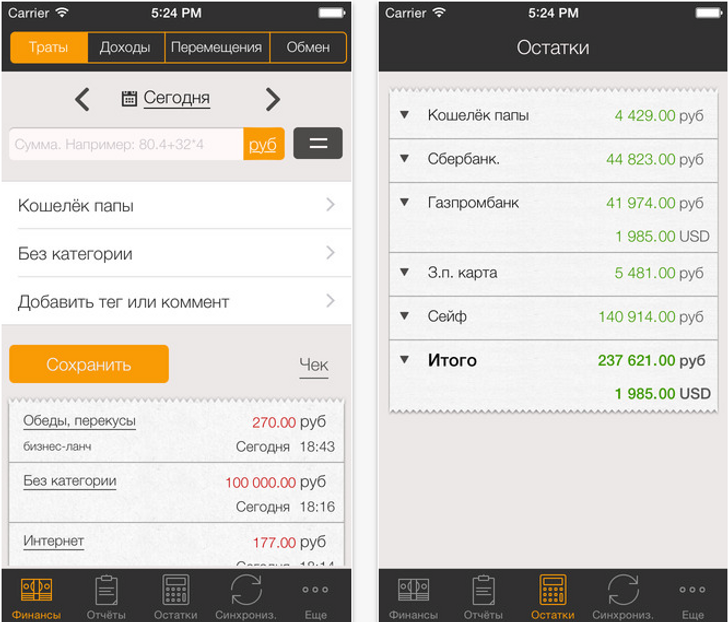


Рисунок 1.2 – Вид приложения Дребеденьги

1. HomeBank – бесплатное кроссплатформенное ПО с открытым исходным кодом [4] (рисунок 1.3). Предназначено для более продвинутых пользователей.

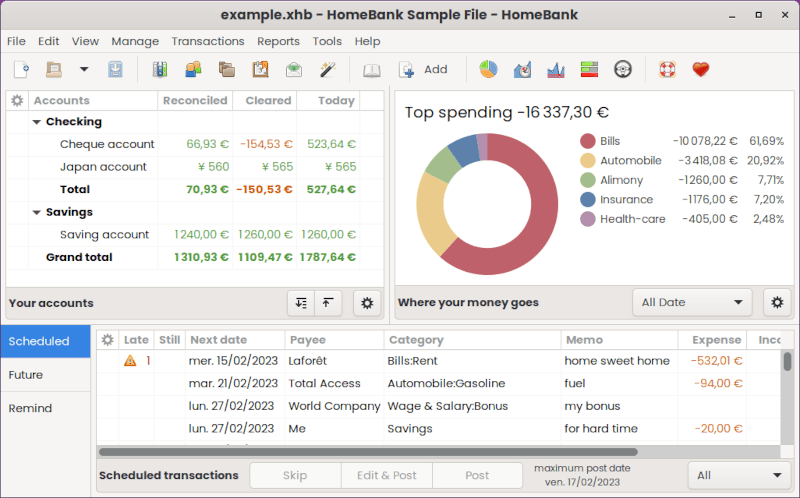


Рисунок 1.3 – Вид приложения HomeBank

Оно поддерживает импорт банковских выписок, отчеты, графики и кастомизацию категорий.

1. **ПОСТАНОВКА ЗАДАЧИ ДЛЯ РАЗРАБОТКИ ПОДСИСТЕМЫ УЧЁТА СЕМЕЙНОГО БЮДЖЕТА**

Целью данного курсового проекта является создание клиент-серверного приложения, предназначенного для автоматизации процессов, связанных с учётом и ведением семейного бюджета. Основная задача системы — обеспечить удобное и безопасное управление доходами, расходами, долговыми обязательствами и финансовыми целями семьи. Разрабатываемая подсистема должна способствовать упрощению ведения финансовой отчетности, повышению финансовой дисциплины и осознанному планированию расходов.

Для выполнения цели курсового проекта требуется выполнить ряд важных задач:

– спроектировать базы данных;

– спроектировать серверной и клиентской частей;

– связать БД и сервер;

– разработать удобный интерфейс приложения.

Приложение предусмотрено для трёх ролей: администратор, пользователь и пользователь-владелец (пользователь, который создаёт таблицу).

Со стороны администратора программа должна обеспечивать следующие возможности:

– просмотр данных о пользователях и таблицах;

– добавление, удаление, редактирования пользователей и таблиц;

– просмотр статистики таблиц;

– создание текстового отчёта о пользователях и таблицах;

Со стороны пользователя программа должна предоставлять следующие возможности:

– редактирование собственных данных (пароль, имя, номер телефона);

– просмотр, добавление, удаление, редактирование данных в таблицах;

– создание таблиц;

– просмотр статистики таблиц.

Со стороны пользователя-владельца программа должна предоставлять следующие возможности:

– редактирование собственных данных (пароль, имя, номер телефона);

– просмотр, добавление, удаление, редактирование данных в таблицах;

– создание таблиц;

– удаление, созданных этим пользователем таблиц;

– добавление, удаление участников созданных этим пользователем таблиц;

– просмотр статистики таблиц;

Данная глава служит основой для дальнейшей разработки курсового проекта.

1. **ФУНКЦИОНАЛЬНАЯ МОДЕЛЬ ПРОЦЕССА УЧЁТА И ВЕДЕНИЯ СЕМЕЙНОГО БЮДЖЕТА**

Для получения более полного представления о том, как можно автоматизировать подсистему учёта и ведения семейного бюджета с помощью программного продукта, была создана функциональная модель IDEF0 [5]. Ниже (на рисунке 3.1) представлена контекстная диаграмма процесса «Управлять семейным бюджетом».

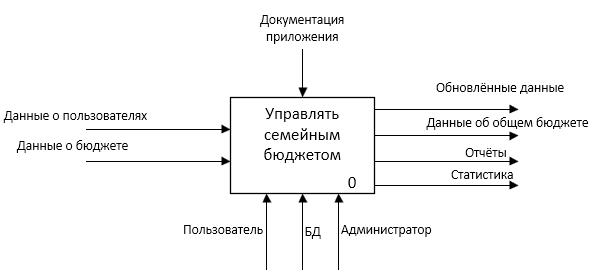


Рисунок 3.1 - Контекстная диаграмма

На рисунке 3.2 предоставлена декомпозиция верхнего процесса, состоящая из 2 основных подпроцессов: войти в приложение и заниматься таблицами.

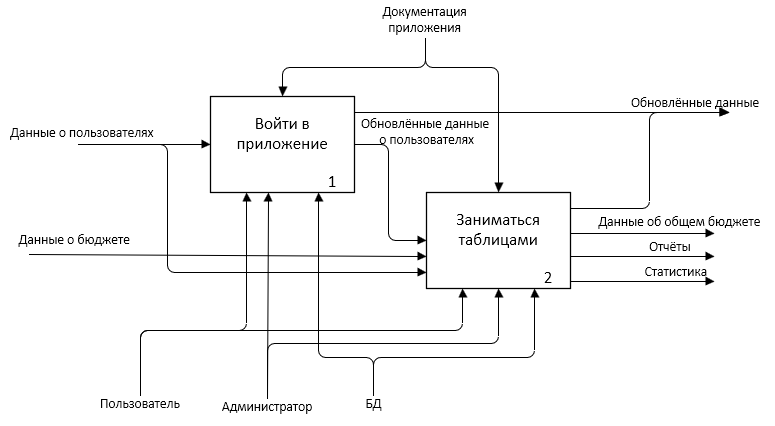


Рисунок 3.2 – Декомпозиция верхнего процесса

На рисунке 3.3 декомпозирован процесс Войти в приложение. Он состоит из Создать аккаунт и Войти в аккаунт. Процесс Заниматься таблицами декомпозирован на рисунке 3.4 и состоит из Заниматься пользователями и Заниматься таблицами бюджетов.

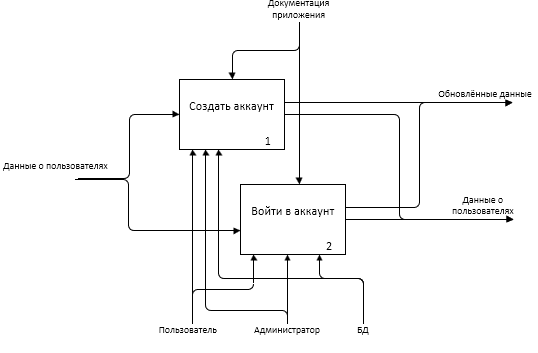


Рисунок 3.3 – Декомпозиция процесса Войти в приложение

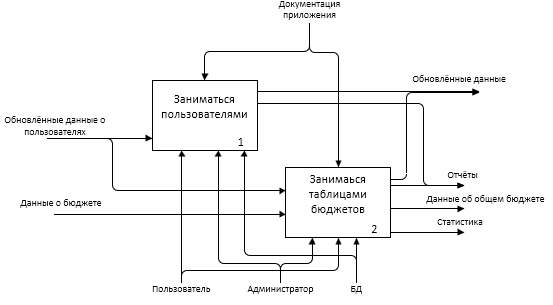


Рисунок 3.4 – Декомпозиция процесса Заниматься таблицами

На рисунке 3.5 показана декомпозиция процесса Заниматься таблицами бюджетов, которая влючает в себя процессы Настроить таблицу бюджета, Настроить пользователей таблицы и Настроить записи таблицы. А на рисунке 3.6 показана декомпозиция процесса Заниматься пользователями. Она содержит процессы Заниматься собственным аккаунтом и Заниматься таблицей с пользователями.

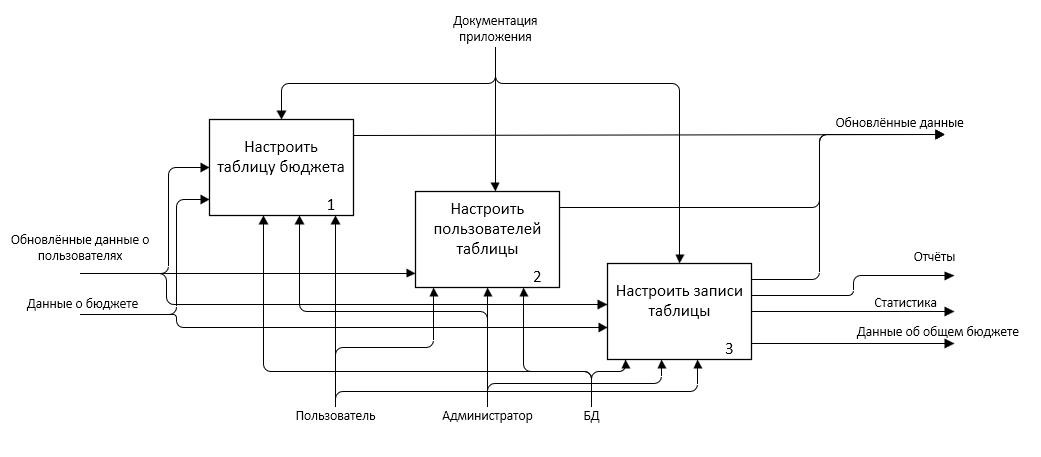


Рисунок 3.5 – Декомпозиция процесса Заниматься таблицами бюджетов

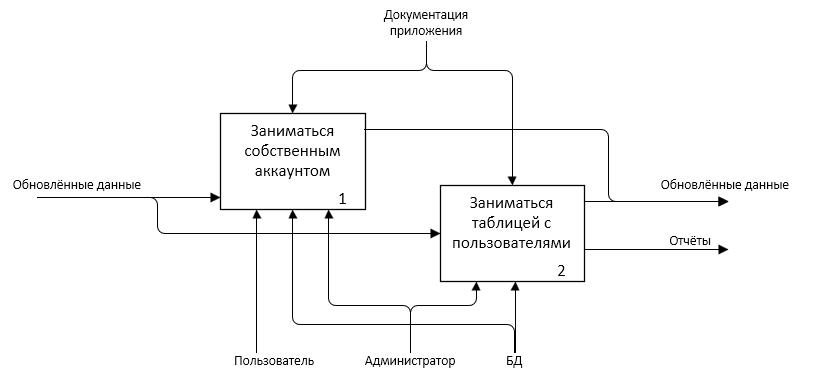


Рисунок 3.6 – Декомпозиция процесса Заниматься пользователями

На рисунке 3.7 показана декомпозиция процесса Заниматься таблицей с пользователями, которая содержит процессы Удалить пользователя, Редактировать пользователя и Получить отчёт о пользователях.



Рисунок 3.7 – Декомпозиция процесса Заниматься таблицей с пользователями

Функциональная модель на основе IDEF0 позволила детально описать процессы подсистему учёта и ведения семейного бюджета, начиная от обработки заявок клиентов до формирования отчетности [6].

Каждый подпроцесс имеет четкие входы, выходы и управляющие факторы, что обеспечивает прозрачность и эффективность работы системы. Модель может быть использована для дальнейшей автоматизации и оптимизации бизнес-процессов.

1. **ИНФОРМАЦИОННАЯ МОДЕЛЬ ПРОЦЕССА УЧЁТА И ВЕДЕНИЯ СЕМЕЙНОГО БЮДЖЕТА**

Сущностями в модели БД курсовой работе являются: sex, persons, users, place, roles, records, table\_member, tables.

Sex отражает пол пользователя и содержит поля: id – первичный ключ, по которому будет определятся пол; name – название пола.

Persons хранит информацию пользователя, не требующуюся при входе в приложение: id – первичный ключ, по которому будет определятся к какому пользователю относится информация; phone\_num – номер телефона пользователя; sex\_id – внешний ключ, содержащий id пола пользователя и соединяющий таблицы persons и sex.

Roles отражает роль пользователя и содержит поля: id – первичный ключ, по которому будет определятся роль; name – название роли.

Users содержит информацию о пользователе: id – первичный ключ, по которому определяется пользователь; username – имя пользователя (логин); password – пароль пользователя; role\_id – внешний ключ, содержащий id роли пользователя; person\_id – внешний ключ, содержащий id сущности persons.

Place отражает причину изменения бюджета (продукты, обучение и т.п.) и содержит поля: id – первичный ключ, по которому будет определятся причина; name – название причины.

Tables отражает таблицы созданные пользователями и содержит поля: id – первичный ключ, по которому определяется таблица, созданная пользователем; title – название пользовательской таблицы; author\_id – внешний ключ, содержащий id пользователя, который создал эту таблицу.

Table\_member содержит поля: id – первичный ключ, по которому определяется строка таблицы, table\_id – внешний ключ, содержащий id таблицы, к которой относится пользователь с id, равным member\_id; member\_id – внешний ключ, содержащий id пользователя который является участником таблицы с id, равным table\_id.

Records отражает запись в таблице и содержит поля: id – первичный ключ, по которому определяется запись; datas – дата, в которую изменился баланс; cash – сумма, на которую баланс изменился; place\_id – внешний ключ, содержащий id причины изменения бюджета; table\_member\_id – внешний ключ, содержащий id сущности table\_member, с помощью которого определяется таблица и пользователь для записи.

Все сущности представлены на рисунке 4.1 в схеме IDEF1.X.

Сущности описывают объекты, являющиеся предметом деятельности предметной области, и субъекты, осуществляющие деятельность в рамках предметной области. Свойства объектов и субъектов реально мира описываются с помощью атрибутов.

Все эти сущности необходимы для решения задачи, поставленной данным курсовым проектом.

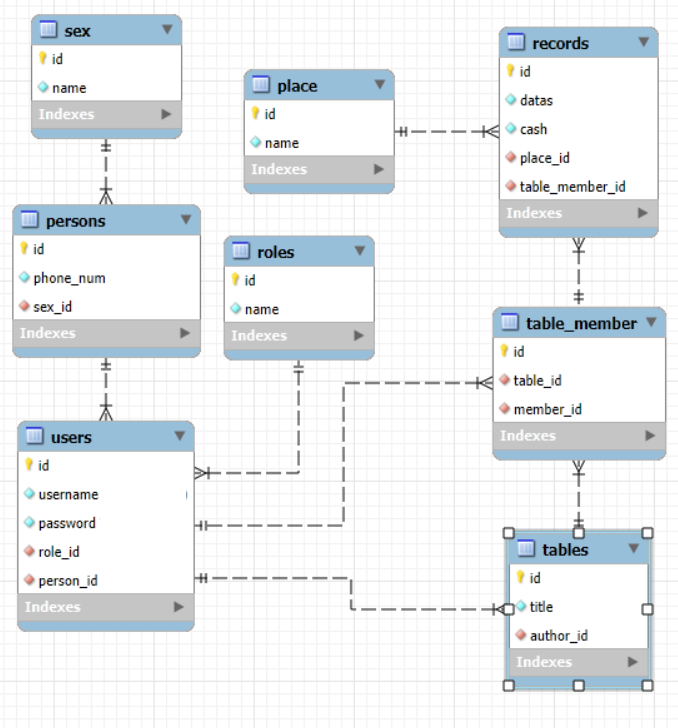


Рисунок 4.1 – Схема IDEF1.X

База данных должна находиться в третьей нормальной форме. Это необходимо для удобства и эффективности работы с базой данных. Нахождение в третьей нормальной форме подразумевает нахождение таблиц базы данных в первых двух нормальных формах.

Таблица находится в первой нормальной форме, если каждый столбец содержит только одно значение (атомарное значение) и все записи в таблице уникальны, то есть нет повторяющихся строк.

Таблица находится во второй нормальной форме, если она находится в первой нормальной форме, а каждое не ключевое поле функционально зависит от всего первичного ключа.

1. **МОДЕЛИ ПРЕДСТАВЛЕНИЯ ПОДСИСТЕМЫ НА ОСНОВЕ UML ДИАГРАММ**

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

* 1. **Спецификация вариантов использования подсистемы учёта семейного бюджета**

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе [7].

В данном проекте представлены две роли: пользователь и администратор.

На рисунке 5.1 представлена диаграмма вариантов использования для администратора и пользователя:

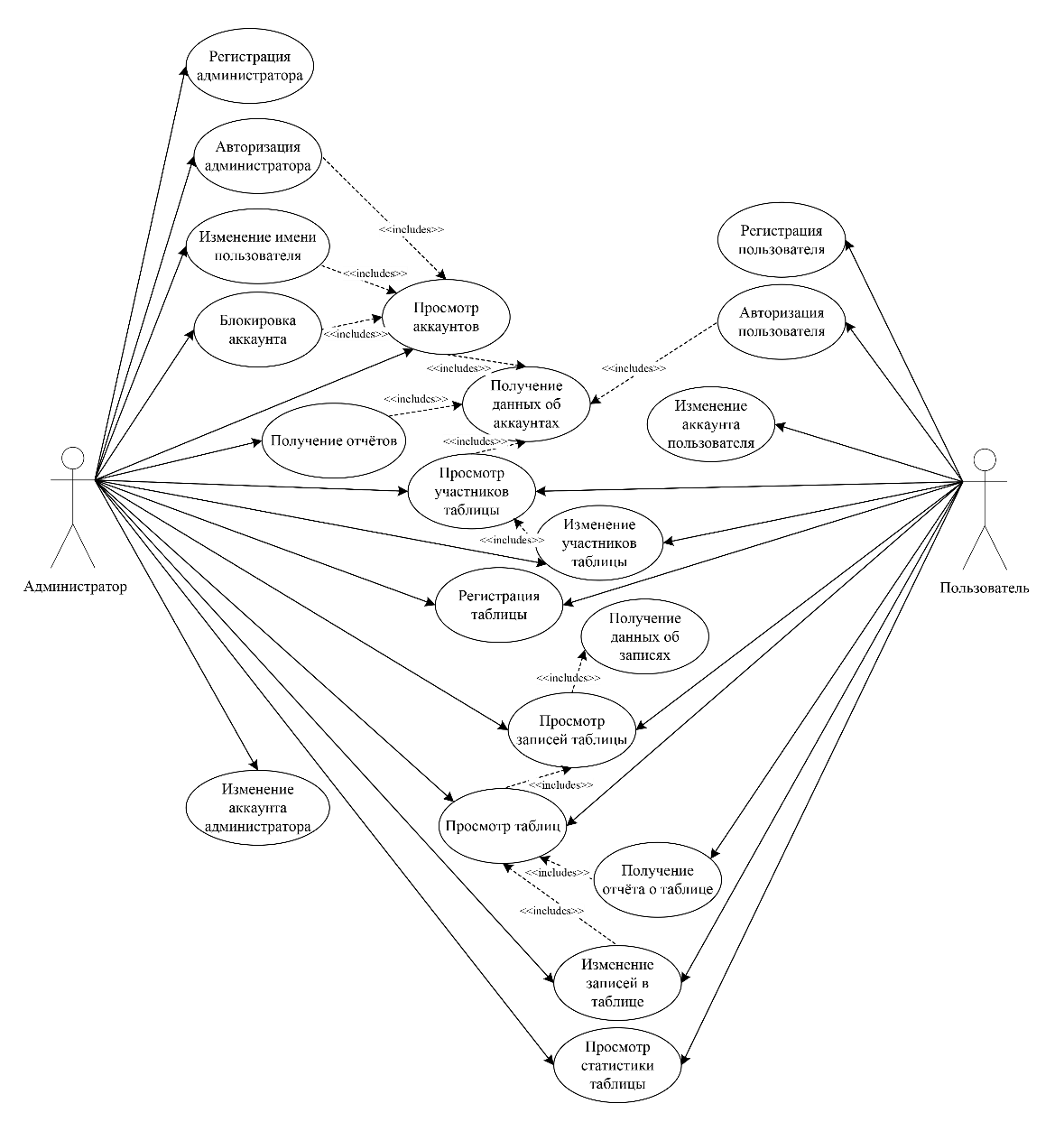


Рисунок 5.1 – Диаграмма вариантов использования для администратора и пользователя

Варианты использования для пользователя и администратора:

– Регистрация администратора;

– Регистрация пользователя;

– Авторизация администратора;

– Авторизация пользователя;

– Изменение имени пользователя;

– Блокировка аккаунта;

– Получение отчётов;

– Просмотр аккаунтов;

– Просмотр участников таблицы;

– Изменение участников таблицы;

– Изменение аккаунта пользователя;

– Регистрация таблицы;

– Просмотр записей таблицы;

– Просмотр таблиц;

– Получение отчёта о таблице;

– Изменение аккаунта администратора;

– Изменение записей в таблице;

– Просмотр статистики таблицы.

В данной подсистеме администратор может выполнять большинство функций пользователя. Чёткое различие администратора и пользователя заключается в том, что администратор может работать с аккаунтами всех пользователей, изменяя или удаляя их. Пользователь может изменить только собственный аккаунт. Работа с таблицами у пользователя и администратора реализуется одинаково.

Многие операции происходят пошагово, с помощью скрытых операций, с которыми роли не взаимодействуют напрямую, но вызывают их с помощью других операций.

Представленная диаграмма охватывает ключевые сценарии работы с подсистемой, отражая ее основное назначение - автоматизацию учёта и ведения семейного бюджета. Разделение прав доступа и четкая структура взаимодействий обеспечивают безопасность данных и эффективность работы с ними.

* 1. **Описание диаграммы последовательности подсистемы учёта семейного бюджета**

Диаграмма последовательности авторизации детализирует процесс взаимодействия между пользователем, клиентским приложением, сервером и базой данных при выполнении сценария входа в систему (рисунок 5.2).

Процесс начинается с авторизации пользователя в интерфейсе приложения:

– ввод логина;

– ввод пароля;

– нажатие кнопки "войти" для инициирования процедуры проверки.

Клиентское приложение, получив данные от пользователя, выполняет первичную валидацию:

– проверку заполненности полей;

– соответствие длины введенных значений требованиям системы;

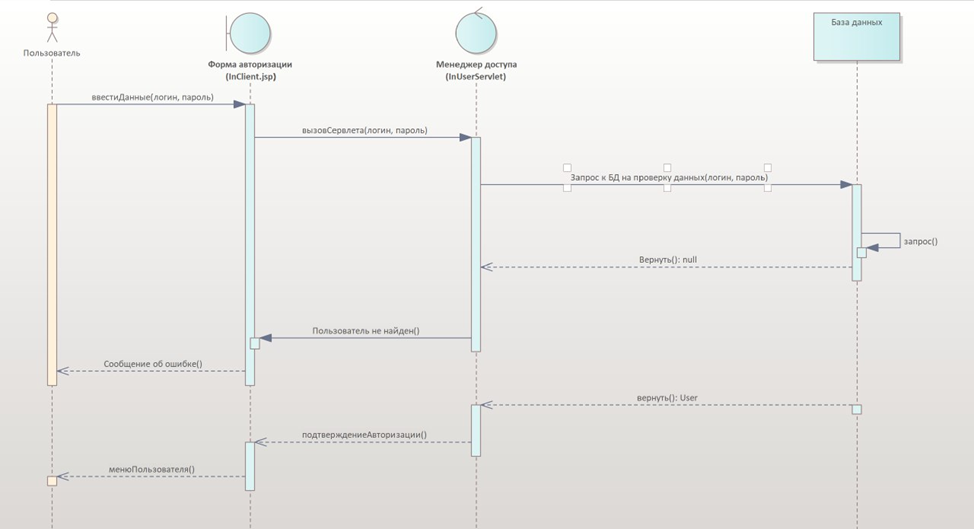


Рисунок 5.2 – Диаграмма последовательности авторизации

При успешной первичной проверке клиентское приложение отправляет учетные данные на сервер через защищенное соединение. Серверная часть, получив запрос, выполняет следующие действия:

– создает новый экземпляр объекта для обработки запроса авторизации;

– формирует параметризованный sql-запрос к базе данных;

– отправляет запрос в субд через пул соединений;

База данных, получив запрос, выполняет поиск пользователя по комбинации логина и пароля. В зависимости от результата проверки возможны два сценария:

1. Сценарий успешной авторизации:

– база данных возвращает серверу данные пользователя;

– сервер создает сессию и генерирует токен доступа;

– клиентскому приложению отправляется положительный ответ;

– происходит перенаправление в меню системы.

1. Сценарий неудачной авторизации:

– база данных сообщает об отсутствии совпадений;

– клиенту возвращается сообщение об ошибке;

Диаграмма наглядно демонстрирует четкое разделение ответственности между компонентами системы и последовательность их взаимодействия.

* 1. **Описание диаграммы состояний**

Диаграмма состояний помогает описать поведение отдельно взятого объекта помогает. Также зачастую диаграмма состояний используется аналитиками для описания последовательности переходов объекта из одного состояния в другое.

Ниже предоставлена диаграмма состояний при регистрации (рисунок 5.3).

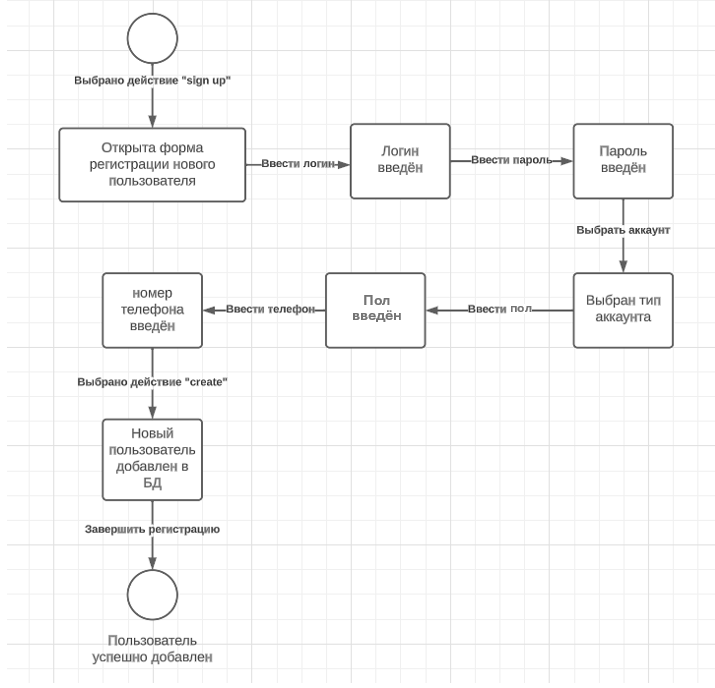


Рисунок 5.3 – Диаграмма состояний при регистрации

При регистрации пользователя необходимо перейти в интерфейсе в меню создания аккаунта, затем ввести запрашиваемые данные: логин, пароль, тип аккаунта (пользователь или администратор), пол, номер телефона.

Эти данные последовательно считываются, проверяются, после чего создаётся пользователь с указанными данными и добавляется в базу данных, завершая процесс регистрации.

Диаграмма демонстрирует этапы обработки и сохранения данных при регистрации.

* 1. **Описание диаграммы развёртывания и компонентов**

Диаграмма развертывания отображает физическую архитектуру подсистемы учёта семейного бюджета и взаимосвязи между ее основными компонентами. Как показано на рисунке 5.4, расположенном ниже, система построена по классической трехзвенной архитектуре, включающей клиентские рабочие станции, сервер приложений и сервер базы данных.

Серверная часть системы представлена компонентом ServerMain.jar, который содержит основной исполняемый код серверного приложения. Сервер работает под управлением *Java Runtime Environment* и использует *TCP*/*IP*-порт 6666 для взаимодействия с клиентами. В его состав входит специализированный модуль администратора, обеспечивающий дополнительные функции управления системой. Для хранения данных используется база данных PostgreSQL.

Клиентская часть приложения реализована с помощью JavaFX, которая обеспечивает графический интерфейс приложения [10].

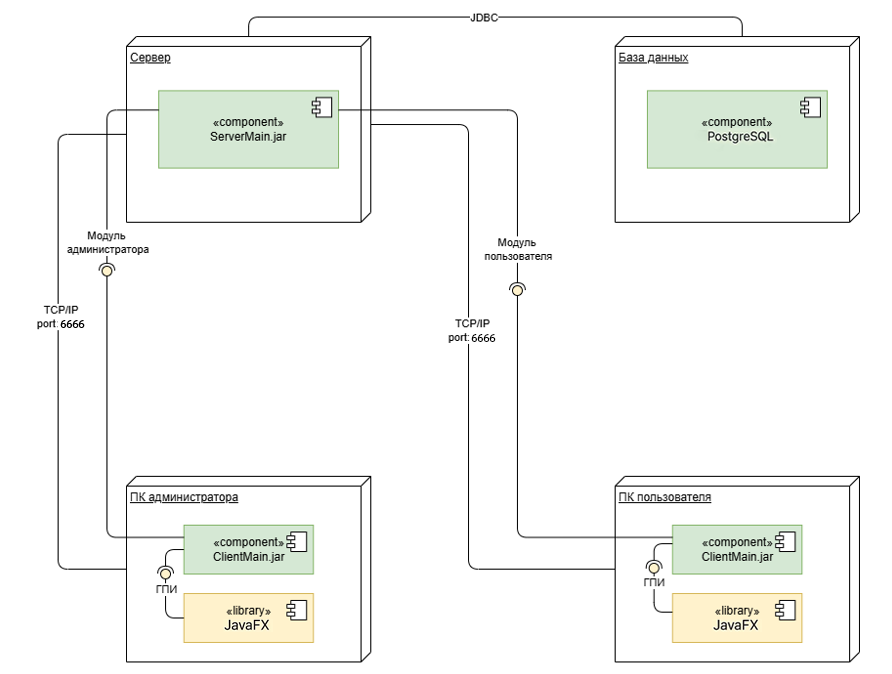


Рисунок 5.4 – Диаграмма развёртывания и компонентов приложения

Особенностью данной архитектуры является:

– четкое разделение функций между компонентами;

– использование стандартных протоколов и технологий;

– возможность масштабирования серверной части;

– единая точка доступа к бизнес-логике;

– централизованное хранение данных.

Представленная конфигурация обеспечивает надежную работу системы при умеренной нагрузке и может быть масштабирована для поддержки большего количества пользователей за счет увеличения ресурсов серверного оборудования.

* 1. **Описание диаграммы классов**

Диаграмма классов представляет объектную модель подсистемы учёта и ведения семейного бюджета, отражая ключевые сущности и их взаимосвязи (рисунок 5.5). Основу системы составляют восемь центральных классов, каждый из которых соответствует важному бизнес-концепту предметной области.

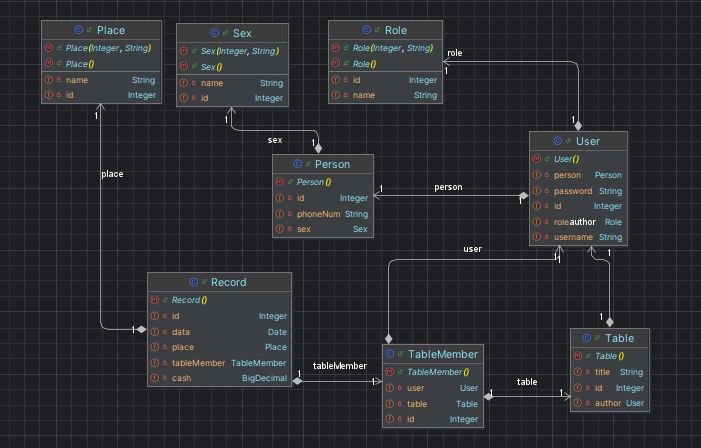


Рисунок 5.5 – Диаграмма классов приложения

Класс Sex описывает пол пользователя, Role – роль, и Place – сферу расходов/доходов. Эти классы содержат уникальный идентификатор и название (User, Admin, Man, Woman, Продукты, Образование и т.д.).

Класс Person описывает пользователя и содержит параметры, которые не используются для входа в приложение (пол, номер телефона).

Класс User является одним из основных классов приложения. Он описывает пользователя и используется для его индентификации, т.к. содержит уникальные поля username(имя пользователя или логин) и password(пароль), он также содержит экземпляр класса Person, который хранит остальную информацию о пользователе.

Класс Table также является основным классом программы. Он хранит информацию о пользовательской таблице (название и автора этой таблицы).

Класс TableMember нужен для соотношения пользователей и таблиц. Он является вспомогательным классом, с помощью которого можно узнать: является ли пользователь участником таблицы или нет. Хранит таблицу и пользователя, принадлежащего ей.

Класс Record представляет запись в таблице. Он содержит информацию о дате изменения баланса, изменившейся сумме, сферу расходов/доходов, а также соотношение пользователя и таблицы, которое определяет к какой таблице принадлежит запись и какому пользователю.

Данная модель демонстрирует сбалансированный подход к проектированию, учитывающий как текущие требования к подсистеме, так и возможные направления развития в будущем. На данной диаграмме отчётливо видно связи между классами. Каждый класс имеет чётко определенную зону ответственности, что способствует поддержанию целостности данных и упрощает дальнейшую разработку системы.

1. **ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС ̶ ЛОГИКУ ПОДСИСТЕМЫ**
   1. **Алгоритма взаимодействия клиента и сервера**

На рисунке 6.1 представлен принцип взаимодействия клиента и сервера, через запросы.

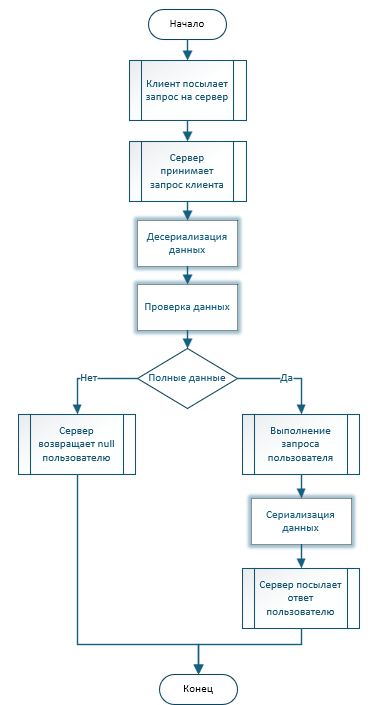


Рисунок 6.1 – Блок-схема алгоритма взаимодействия клиента и сервера

Клиент посылает серверу запрос (Request), сервер принимает запрос, проверяет данные, которые прислал клиент, и посылает ответ (Response). Если данные прошли проверку, сервер выполняет запрос, возвращает необходимые данные и сообщение об успешной операции, если не прошли – возвращает null и сообщение о неудаче.

* 1. **Алгоритм работы пользователя**

На рисунке 6.2 представлен алгоритм работы пользователя.

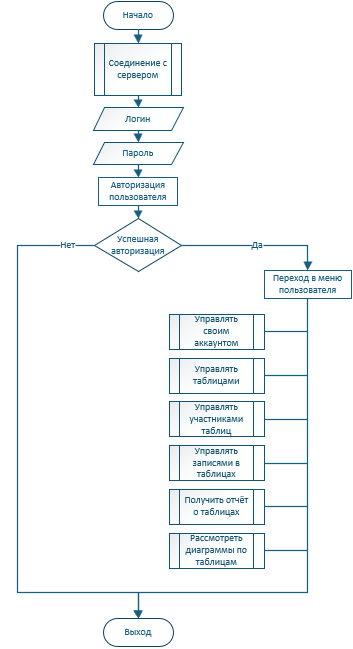


Рисунок 6.2 – Алгоритм работы пользователя

После соединения с сервером пользователь вводит свои данные и авторизируется, если данные правильные. Далее пользователь входит в меню и может произвести операции: редактирование или удаление собственного аккаунта; создание, удаление, редактирование таблиц; добавление, удаление участников таблиц; добавление, удаление, редактирование записей в таблицах; сохранение отчёта с информацией о таблицах, записях и участников таблиц; просмотр диаграмм по таблицам.

* 1. **Алгоритм работы администратора**

На рисунке 6.3 представлен алгоритм работы администратора.

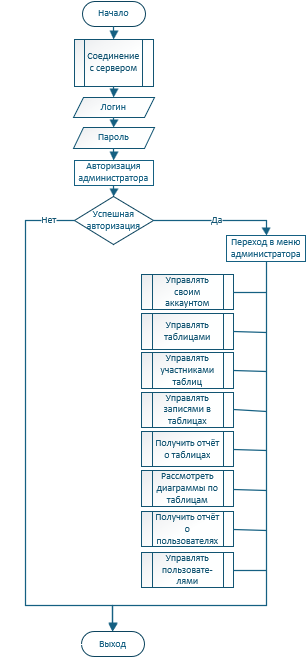


Рисунок 6.3 – Алгоритм работы администратора

Администратор, как и пользователь, после соединения с сервером вводит свои данные и авторизируется (если данные правильные). Далее входит в меню администратора и может произвести такие же операции, как и пользователь: редактирование или удаление собственного аккаунта; создание, удаление, редактирование таблиц; добавление, удаление участников таблиц; добавление, удаление, редактирование записей в таблицах; сохранение отчёта с информацией о таблицах, записях и участников таблиц; просмотр диаграмм по таблицам.

Помимо этого, администратор может ещё получать отчёт о всех пользователях подсистемы, а также удалять и редактировать аккаунты других пользователей, а не только себя.

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для корректной работы приложения необходимо:

– Установить *JDK* 24 для запуска *Java*-приложения [8].

– Установить СУБД *PostgreSQL* версии 9.0+. Для удобства работы с базой данных рекомендуется использовать *pgAdmin 4* или аналогичный клиент [9].

– Выполнить *SQL*-скрипт, который создаст структуру базы данных, включая таблицы товаров, заказов, клиентов и пользователей, а также заполнит их тестовыми данными.

– Сервер по умолчанию использует *TCP*-порт 5432. Для изменения порта отредактируйте параметр в классе *ServerMain.java*:

try (ServerSocket serverSocket = new ServerSocket(6666)) { ... }

– Настроить подключение к БД:

*hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect  
hibernate.connection.driver\_class=org.postgresql.Driver  
hibernate.connection.url=jdbc:postgresql://localhost:5432/familywollet  
hibernate.connection.username=postgres  
hibernate.connection.password=Root*

Измените эти значения, если ваша БД использует другие параметры.

– Собрать проект в *JAR*-файл или запустите класс *ServerMain* напрямую из *IDE*.

Ошибки подключения к БД или занятого порта будут отображены в консоли.

Сервер должен оставаться запущенным во время работы клиентских приложений.

Если требуется изменить параметры после первого запуска, остановите сервер, внесите правки в код и перезапустите приложение.

1. **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

Первое что видит пользователь при запуске приложения – это окно (рисунок 8.1), через которое осуществляется вход, здесь пользователь должен ввести своё имя (логин) и пароль. При попытке входа с незаполненными полями или неверных данных, над кнопкой войти появляется надпись, сообщающая об ошибке.

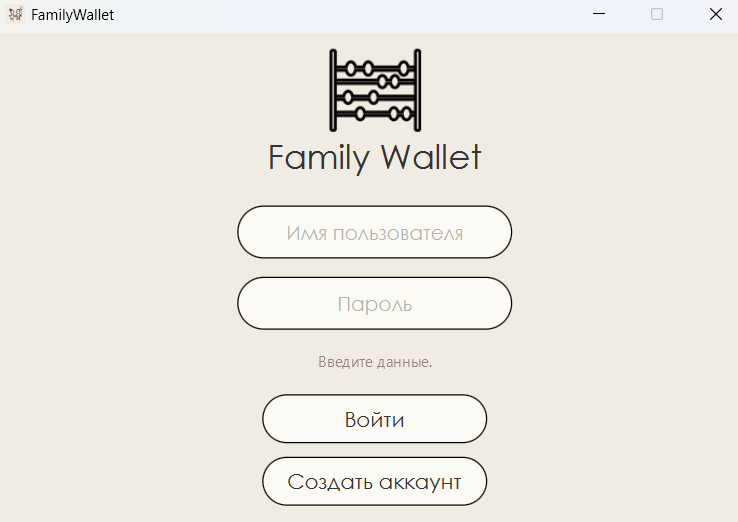


Рисунок 8.1 – Вход в приложение

Чтобы создать аккаунт, нужно нажать на кнопку Создать аккаунт. Появится окно регистрации (рисунок 8.2), где надо ввести все необходимые данные, а также выбрать роль: администратор или обычный пользователь.



Рисунок 8.2 - Регистрация

При нажатии на кнопку в левом верхнем углу, пользователь выходит из окна регистрации и возвращается к окну входа в приложение, а при нажатии кнопки создать, если все данные введены корректно, создаётся новый аккаунт с введёнными данными, в противном случае – появляется сообщение об ошибке.

При входе в приложение в роли администратора, появляется окно меню администратора (рисунок 8.3).

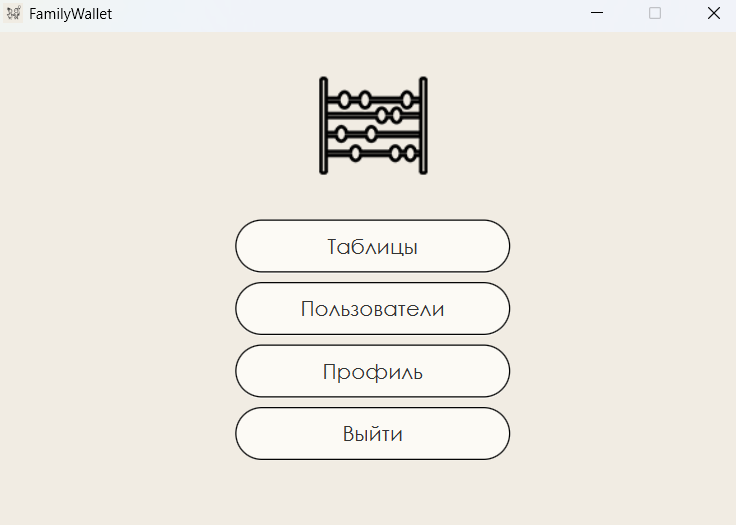


Рисунок 8.3 – Меню администратора

При входе в приложение в роли пользователя, появляется окно меню пользователя (рисунок 8.4).



Рисунок 8.4 – Меню пользователя

При нажатии на кнопку выйти, пользователь возвращается к окну входа в приложение.

При нажатии на кнопку профиль, пользователь попадает в личный кабинет (рисунок 8.5), где отображена часть его данные. В этом окне он может изменить часть своих данных или удалить аккаунт, через который происходил вход в приложение. С помощью кнопки в левом верхнем углу, можно вернуться в меню.

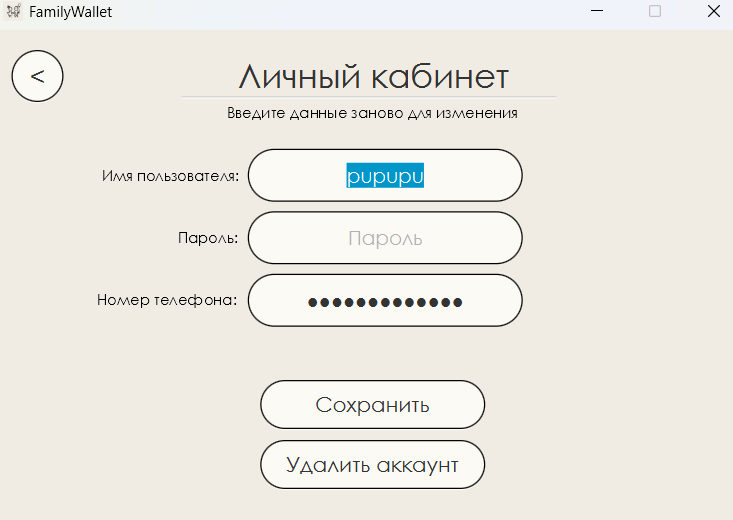


Рисунок 8.5 – Личный кабинет

В меню администратора есть кнопка пользователи, при нажатии на которую, загружается окно (рисунок 8.6) с таблицей пользователей и несколькими кнопками: удаление, редактирование пользователя и сохранения отчёта по пользователям (текстовый файл с информаций о пользователях).

В левом нижнем углу есть кнопка, с помощью которой можно вернуться в меню администратора.

В таблице пользователей отображены данные о пользователях. Для удаления или редактирования пользователя, необходимо выделить строку с нужным для изменения или удаления пользователем. При попытке произвести операцию без выделения пользователя или ввода некорректного значения в текстовый файл (новое имя), выводится сообщение о соответствующей ошибке.

При нажатии кнопки получить отчёт по таблице, создаёт или перезаписывается текстовый файл (users.txt), где сохраняется информация о пользователях из таблицы (рисунок 8.7).

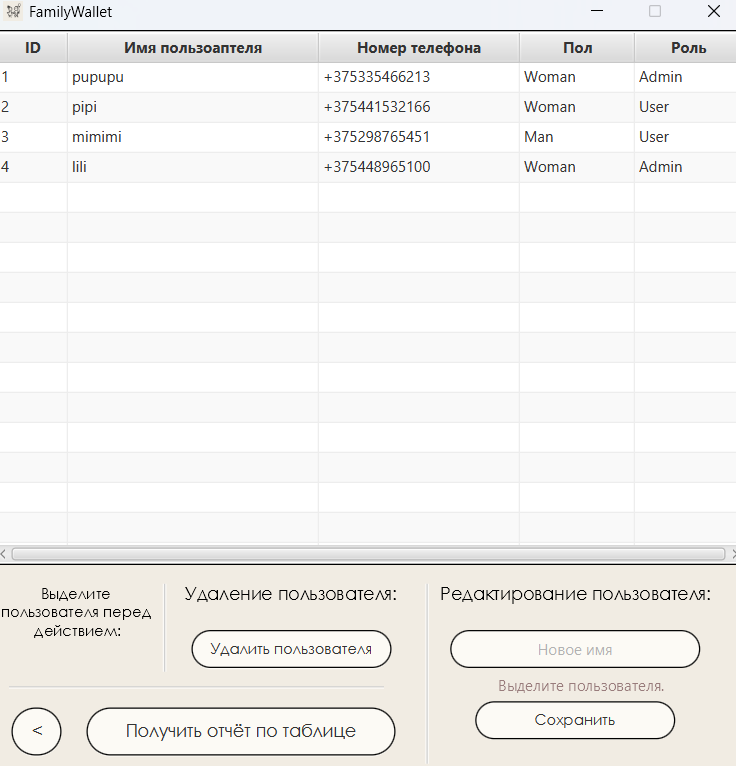


Рисунок 8.6 – Таблица с пользователями

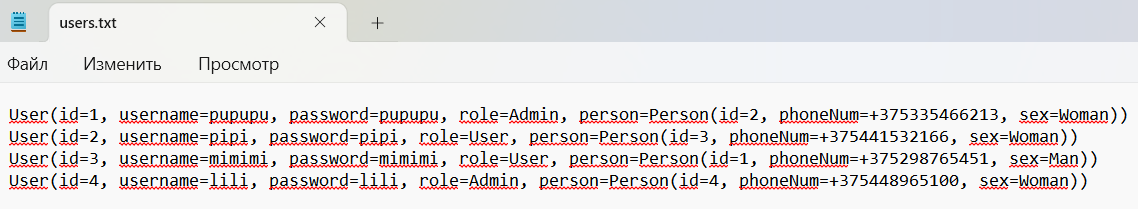


Рисунок 8.7 – Отчёт по пользователям

В меню у пользователя и у администратора есть кнопка таблицы. При нажатии на неё, загружается окно с таблицами пользователя (рисунок 8.7).

В этом окне у пользователя (или администратора) есть возможность переключаться между разными таблицами, в которых он участвует, с помощью вкладок. На вкладках отображаются названия таблиц, а во вкладке отображаются записи (данные об изменении бюджета), принадлежащие этой таблице.

Под таблицами расположены кнопки, для которых надо выделить запись (строку) или таблицу (вкладку): для того, чтобы удалить или переименовать таблицу или для получения итогов таблицы, нужно выделить вкладку; для изменения или удаления записи – выделить строку в таблице.

Для того, чтобы создать или переименовать таблицу, нужно ввести (новое) название таблицы.

Для добавления или удаления участника таблицы необходимо выбрать таблицу и указать пользователя. Для добавления записи указать пользователя, таблицу, сумма изменения бюджета, дату и описание.

Для получения отчёта необходимо выделить таблицу (вкладку) и нажать на кнопку получить отчёт по таблице, после чего создастся текстовый файл с названием идентичному названию вкладки, там будет вся информация о записях выбранной таблицы.

В текстовых полях (на фоне) написано какому формату запись должна соответствовать, а также при неправильном вводе значений или отсутствии введённых значений выводятся соответствующие оповещения.

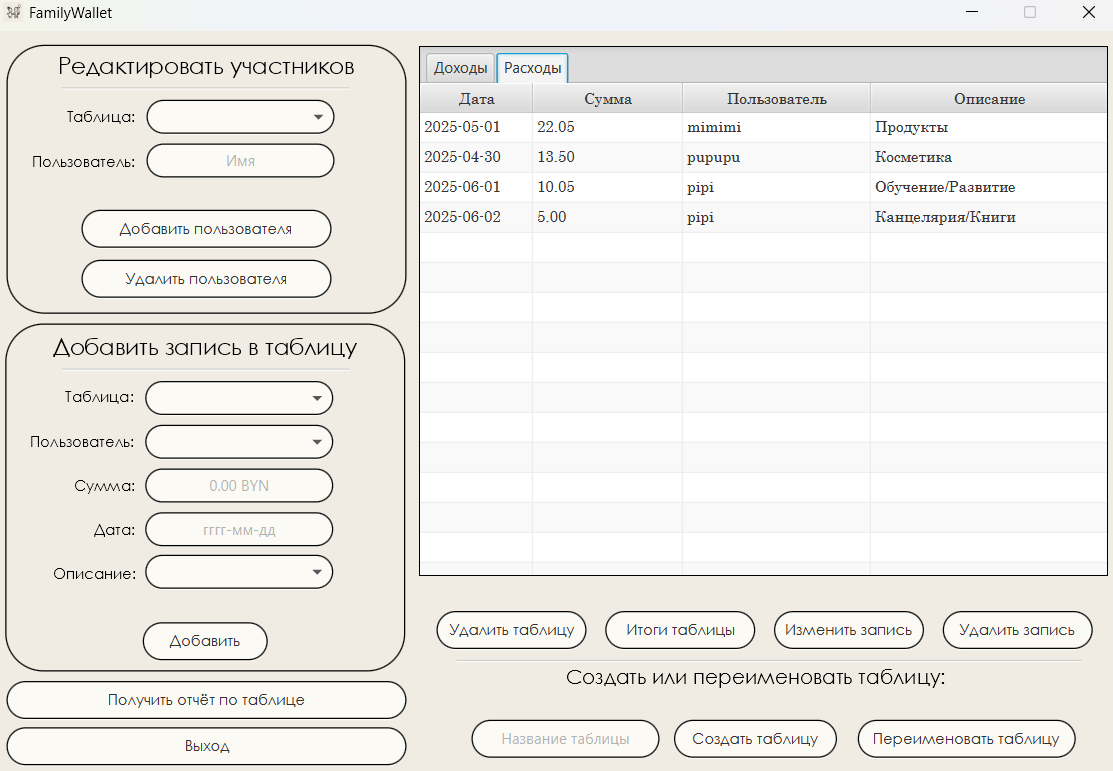


Рисунок 8.7 – Таблицы пользователя

На рисунке 8.8 представлен отчёт по показанной выше таблице.

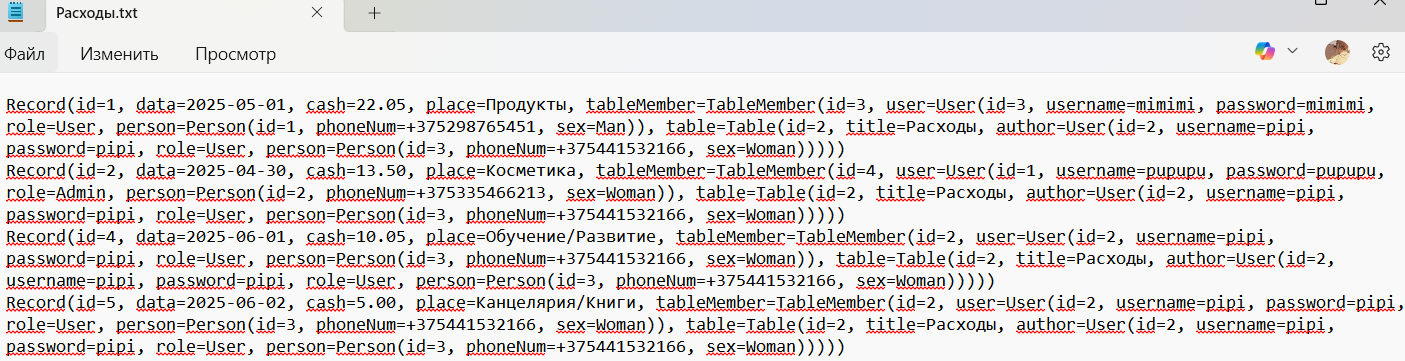
****

Рисунок 8.8 – Отчёт по таблице Расходы

При нажатии на кнопку итоги таблицы, загружается окно с диаграммой (рисунок 8.9), показывающей, что повлияло на изменение бюджета. При нажатии на кнопку изменить запись, появляется окно для ввода информации о записи заново (рисунок 8.10).

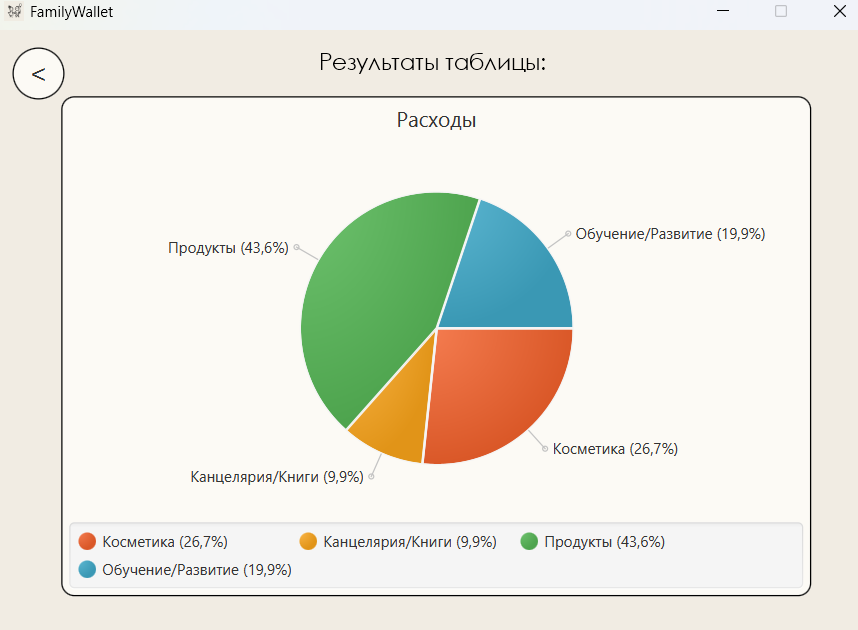


Рисунок 8.9 – Итоги таблицы

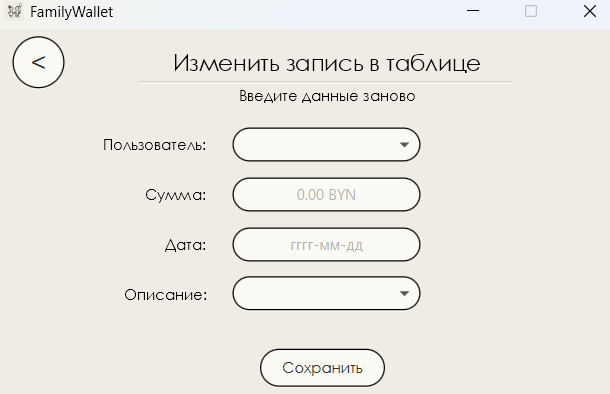


Рисунок 8.10 – Редактирование записи

После введения всех данных, при нажатии на кнопку сохранить, запись изменяется и пользователя перебрасывает в окно с таблицами. Также, в окно с таблицами можно вернуться при нажатии на кнопку в верхнем правом углу.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была успешно разработана подсистема учёта и ведения семейного бюджета, в полной мере соответствующая поставленным во введении целям и задачам. Проведённый анализ подтвердил актуальность создания такой системы в современных условиях, когда финансовая грамотность и эффективное управление личными средствами становятся всё более востребованными.

Разработанная система решает ключевые проблемы ручного ведения семейного бюджета, предлагая удобный инструмент для регистрации доходов и расходов, планирования финансов и контроля над движением денежных средств.

Архитектура построена на основе клиент-серверной модели, что обеспечивает надёжное хранение данных, возможность масштабирования и безопасный доступ для нескольких пользователей. Использование современных технологий Java, PostgreSQL и JavaFX позволило создать стабильное и понятное приложение, адаптированное для пользователей с разным уровнем компьютерной подготовки.

Внедрение данной подсистемы в повседневную жизнь семьи позволит:

– повысить осознанность в распределении бюджета;

– своевременно отслеживать и анализировать расходы;

– формировать финансовые цели и контролировать их достижение;

– вести учет долговых обязательств и сбережений;

– повысить общую финансовую дисциплину и снизить риск перерасхода.

Проведённое тестирование системы подтвердило её работоспособность и соответствие заявленным функциональным требованиям. Интерфейс интуитивно понятен, а все модули взаимодействуют корректно, обеспечивая целостность данных и удобство использования.

Таким образом, данный курсовой проект демонстрирует действенный подход к автоматизации ведения семейного бюджета и может быть использован в качестве основы для последующего развития полноценной финансовой платформы, ориентированной на широкий круг пользователей.

Курсовая работа выполнена самостоятельно, проверена в системе «Антиплагиат». Процент оригинальности составляет 95%. Цитирования оформлены в соответствии с требованиями и отражены в разделе «Список использованных источников». Скриншот приведён в приложении A.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Стандарт предприятия дипломные проекты (работы) общие требования СТП 01–2024 [Электронный ресурс]. — Режим доступа: https://www.bsuir.by/m/12\_100229\_1\_185586.pdf

[2] Zen-Money [Электронный ресурс]. – Режим доступа: https://zenmoney.ru/. – Дата доступа: 25.05.2025.

[3] Дребеденьги [Электронный ресурс]. – Режим доступа: https://www.drebedengi.ru/. – Дата доступа: 25.05.2025.

[4] HomeBank [Электронный ресурс]. – Режим доступа: https://homebank.kz/. – Дата доступа: 25.05.2025.

[5] Создание схемы IDEF0 [Электронный ресурс]. – Режим доступа: https://support.microsoft.com/ru-ru/office/создание-схемы-idef0-ea7a9289-96e0- 4df8-bb26-a62ea86417fc . – Дата доступа: 25.05.2025.

[6] Методология IDEF0: Система моделирования процессов [Электронный ресурс]. – Режим доступа: https://www.idef0.org – Дата доступа: 25.05.2025.

[7] UML 2.5 Specification [Электронный ресурс]. – Режим доступа: https://www.omg.org/spec/UML/2.5.1/. – Дата доступа: 25.05.2025.

[8] Васильев А.Ю. Java на примерах. Практика, практика и только практика. СПб.: Наука и Техника, 2023. 480 с. ISBN 978-5-94387-876-2.

[9] PostgreSQL: Документация [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/. – Дата доступа: 25.05.2025.

[10] JavaFX: Официальная документация [Электронный ресурс]. – Режим доступа: https://openjfx.io/. – Дата доступа: 25.05.2025.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Отчёт о проверке на заимствования в системе «Антиплагиат»**

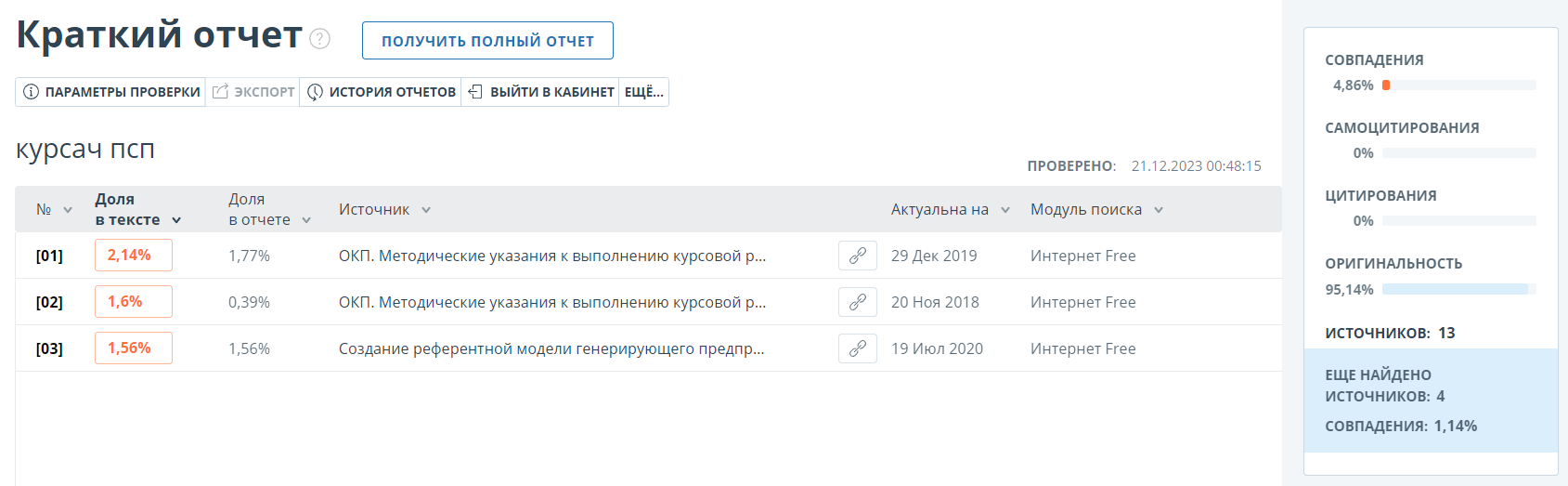


Рисунок А.1 – Результат проверки на антиплагиат

# **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Листинг кода алгоритмов, реализующих бизнес-логику**

package server.controllers;  
  
import server.exceptions.ResponseException;  
import server.entities.User;  
import server.network.Request;  
import server.network.Response;  
import server.serializer.Deserializer;  
import server.serializer.Serializer;  
import server.services.PersonService;  
import server.services.RoleService;  
import server.services.SexService;  
import server.services.UserService;  
import server.utils.Pair;  
  
import java.util.Objects;  
  
public class UserController {  
 private final UserService userService;  
 private final PersonService personService;  
 private final RoleService roleService;  
 private final SexService sexService;  
  
 public UserController(UserService *userService*, PersonService *personService*, RoleService *roleService*, SexService *sexService*) {  
 this.userService = *userService*;  
 this.personService = *personService*;  
 this.roleService = *roleService*;  
 this.sexService = *sexService*;  
 }  
  
 public UserController() {  
 this.userService = new UserService();  
 this.personService = new PersonService();  
 this.roleService = new RoleService();  
 this.sexService = new SexService();  
 }  
  
 public Response login(Request *request*) {  
 Deserializer deserializer = new Deserializer();  
 User user = (User) deserializer.extractData(*request*);  
  
 try {  
 User existingUser = userService.login(user);  
 String loggedInUser = Serializer.**toJson**(existingUser);  
  
 return new Response(true, "Login Successful",

Продолжение приложения Б

loggedInUser);  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);  
 } catch (Exception *e*) {  
 return new Response(false, "An unexpected error occurred", null);  
 }  
 }  
  
  
 public Response register(Request *request*) {  
 Deserializer deserializer = new Deserializer();  
 Object extractedData;  
  
 try {  
 extractedData = deserializer.extractData(*request*);  
 } catch (IllegalArgumentException *e*) {  
 return new Response(false, "Invalid user data", null);  
 }  
  
 if (!(extractedData instanceof User *user*)) {  
 return new Response(false, "Invalid user data", null);  
 }  
  
 try {  
 String registeredUserJson = Serializer.**toJson**(  
 userService.register(*user*, personService, roleService, sexService)  
 );  
 return new Response(true, "Registration Successful", registeredUserJson);  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);  
 }  
 }  
  
 public Response getAllUsers() {  
 try {  
 String users = Serializer.**toJson**(userService.findAllEntities());  
 return new Response(true, "Users retrieved successfully", users);  
 } catch (Exception *e*) {  
 *e*.printStackTrace();  
 return new Response(false, "Failed to retrieve users", null);  
 }  
 }

Продолжение приложения Б

public Response deleteUser(Request *request*) {  
 Deserializer deserializer = new Deserializer();  
 String login = (String) deserializer.extractData(*request*);  
  
 try {  
 User foundUser = userService.findByUsername(login);  
 if (foundUser != null) {  
 personService.deleteEntity(foundUser.getPerson());  
  
 return new Response(true, "User deleted successfully", null);  
 } else {  
 return new Response(false, "User not found", null);  
 }  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);  
 }  
 }  
  
  
 public Response updateEntity(Request *request*) {  
 Object extractData = new Deserializer().extractData(*request*);  
  
 if (extractData instanceof Pair<?,?> *pair* &&  
 *pair*.getKey() instanceof User *userToUpdate* &&  
 *pair*.getValue() instanceof User *userThatOperate*) {  
 User existingUserToUpdate = userService.findEntity(*userToUpdate*.getId());  
 User existingUserThatOperate = userService.findEntity(*userThatOperate*.getId());  
  
 if (existingUserToUpdate == null || existingUserThatOperate == null) {  
 return new Response(false, "Some of the users don't exist", null);  
 }  
  
// if (Objects.equals(userToUpdate.getId(), userThatOperate.getId())) {  
// return new Response(false, "You can't edit your profile", null);  
// }  
  
 try {  
 userService.updateEntity(*userToUpdate*, personService);  
 return new Response(true, "User and associated Person updated successfully", null);  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);

Продолжение приложения Б

}  
 } else {  
 return new Response(false, "Bad information from client!", null);  
  
 }  
 }  
  
 public Response readEntity(Request *request*) {  
 Deserializer deserializer = new Deserializer();  
 String username = (String) deserializer.extractData(*request*);  
  
 try {  
 User user = userService.findByUsername(username);  
 String userJson = Serializer.**toJson**(user);  
 if (user != null) {  
 return new Response(true, "User retrieved successfully", userJson);  
 } else {  
 return new Response(false, "User not found", null);  
 }  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);  
 }  
 }  
}

package server.controllers;  
  
import server.entities.BaseTable;  
import server.entities.User;  
import server.exceptions.ResponseException;  
import server.network.Request;  
import server.network.Response;  
import server.serializer.Deserializer;  
import server.serializer.Serializer;  
import server.services.TableService;  
import server.utils.Pair;  
  
public class TableController {  
 private final TableService tableService;  
 //private final UserService userService;  
  
 public TableController(TableService *tableService*) {  
 this.tableService = *tableService*;  
 }  
  
 public TableController(){  
 this.tableService = new TableService();

Продолжение приложения Б

}  
  
 public Response getAllTables() {  
 try {  
 String tables = Serializer.**toJson**(tableService.findAllEntities());  
 return new Response(true, "Table retrieved successfully", tables);  
 } catch (Exception *e*) {  
 *e*.printStackTrace();  
 return new Response(false, "Failed to retrieve table", null);  
 }  
 }  
  
 public Response updateEntity(Request *request*) {  
 Object extractData = new Deserializer().extractData(*request*);  
  
 if (extractData instanceof Pair<?,?> *pair* &&  
 *pair*.getKey() instanceof BaseTable *tableToUpdate* &&  
 *pair*.getValue() instanceof BaseTable *tableThatOperate*) {  
 BaseTable existingUserToUpdate = tableService.findEntity(*tableToUpdate*.getId());  
 BaseTable existingUserThatOperate = tableService.findEntity(*tableThatOperate*.getId());  
  
 if (existingUserToUpdate == null || existingUserThatOperate == null) {  
 return new Response(false, "Some of the tables don't exist", null);  
 }  
  
// if (Objects.equals(userToUpdate.getId(), userThatOperate.getId())) {  
// return new Response(false, "You can't edit your profile", null);  
// }  
  
 try {  
 tableService.updateEntity(*tableToUpdate*);  
 return new Response(true, "Table updated successfully", null);  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);  
 }  
 } else {  
 return new Response(false, "Bad information from client!", null);

Продолжение приложения Б

}  
 }  
  
 public Response createTable(Request *request*) {  
 Object extractData = new Deserializer().extractData(*request*);  
  
 if(extractData instanceof BaseTable *newTable*){  
 try {  
 tableService.saveEntity(*newTable*);  
 return new Response(true, "Table saved successfully", null);  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);  
 }  
 }  
 return new Response(false, "Table not found or invalid type", null);  
 }  
  
 public Response deleteTable(Request *request*) {  
 Object extractData = new Deserializer().extractData(*request*);  
  
 if(extractData instanceof BaseTable *delTable*) {  
  
 try {  
 tableService.deleteEntity(*delTable*);  
 return new Response(true, "Table deleted successfully", null);  
 } catch (ResponseException *e*) {  
 return new Response(false, *e*.getMessage(), null);  
 }  
 }  
 return new Response(false, "Table not found or invalid type", null);  
 }  
}

package server.controllers;  
  
import server.network.Response;  
import server.serializer.Serializer;  
import server.services.RoleService;  
  
public class RoleController {  
 private final RoleService roleService;  
  
 public RoleController(final RoleService *roleService*) {  
 this.roleService = *roleService*;  
 }

# **ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Скрипт создания базы данных**

-- Roles Table

CREATE TABLE Roles (

id SERIAL PRIMARY KEY,

name VARCHAR(10) NOT NULL UNIQUE

);

-- Sex Table

CREATE TABLE Sex (

id SERIAL PRIMARY KEY,

name VARCHAR(10) NOT NULL UNIQUE

);

-- Place Table

CREATE TABLE Place (

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL UNIQUE

);

-- Persons Table

CREATE TABLE Persons (

id SERIAL PRIMARY KEY,

phone\_num VARCHAR(20) NOT NULL UNIQUE,

sex\_id INT NOT NULL,

FOREIGN KEY (sex\_id) REFERENCES Sex(id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Users Table

CREATE TABLE Users (

id SERIAL PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(50) NOT NULL,

role\_id INT NOT NULL,

person\_id INT NOT NULL UNIQUE,

FOREIGN KEY (role\_id) REFERENCES Roles(id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (person\_id) REFERENCES Persons(id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Tables Table

CREATE TABLE Tables (

id SERIAL PRIMARY KEY,

title VARCHAR(50) NOT NULL,

author\_id INT NOT NULL,

Продолжение приложения В

FOREIGN KEY (author\_id) REFERENCES Users(id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Table\_Мember Table

CREATE TABLE Table\_Member (

id SERIAL PRIMARY KEY,

table\_id INT NOT NULL,

member\_id INT NOT NULL,

FOREIGN KEY (table\_id) REFERENCES Tables(id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (member\_id) REFERENCES Users(id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Records Table

CREATE TABLE Records (

id SERIAL PRIMARY KEY,

datas DATE NOT NULL,

cash DECIMAL(10,2) NOT NULL,

place\_id INT NOT NULL,

table\_member\_id INT NOT NULL,

FOREIGN KEY (table\_member\_id) REFERENCES Table\_Member(id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (place\_id) REFERENCES Place(id) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Insert Roles

INSERT INTO Roles (name) VALUES

('Admin'),('User');

-- Insert Sex

INSERT INTO Sex (name) VALUES

('Man'),('Woman');

-- Insert Place

INSERT INTO Place (name) VALUES

('Продукты'),('Косметика'),('Одежда'),

('Техника'),('Развлечения'),('Обучение/Развитие'),

('Жильё'),('Мебель'),('Кафе/Ресторан'),

('Канцелярия/Книги'),('Здоровье'),('Подарки'),

('Подписки'),('Другое'), ('Работа');

-- Insert Persons

INSERT INTO Persons (phone\_num, sex\_id) VALUES

('+375298765451', 1),('+375335466213', 2), ('+375441532166', 2);

-- Insert Users

Продолжение приложения В

INSERT INTO Users (username, password, role\_id, person\_id) VALUES

('pupupu', 'pupupu', 1, (SELECT id FROM Persons WHERE phone\_num = '+375335466213' LIMIT 1)), -- Role Admin

('pipi', 'pipi', 2, (SELECT id FROM Persons WHERE phone\_num = '+375441532166' LIMIT 1)),

('mimimi', 'mimimi', 2, (SELECT id FROM Persons WHERE phone\_num = '+375298765451' LIMIT 1));

-- Insert Tables

INSERT INTO Tables (title, author\_id) VALUES

('Доходы', (SELECT id FROM Users WHERE username = 'pipi' LIMIT 1)),

('Расходы', (SELECT id FROM Users WHERE username = 'pipi' LIMIT 1));

-- Insert Table\_Member

INSERT INTO Table\_Member (table\_id, member\_id) VALUES

((SELECT id FROM Tables WHERE title = 'Доходы' LIMIT 1), (SELECT author\_id FROM Tables WHERE title = 'Доходы' LIMIT 1)), -- Author 'Доходы'

((SELECT id FROM Tables WHERE title = 'Расходы' LIMIT 1), (SELECT author\_id FROM Tables WHERE title = 'Расходы' LIMIT 1)), -- Author 'Расходы'

((SELECT id FROM Tables WHERE title = 'Расходы' LIMIT 1), (SELECT id FROM Users WHERE username = 'mimimi' LIMIT 1)),

((SELECT id FROM Tables WHERE title = 'Расходы' LIMIT 1), (SELECT id FROM Users WHERE username = 'pupupu' LIMIT 1));

-- Insert Records

INSERT INTO Records (datas, cash, place\_id, table\_member\_id) VALUES

('2025-05-01', 22.05, (SELECT id FROM Place WHERE name = 'Продукты' LIMIT 1),

(SELECT id FROM Table\_Member WHERE

table\_id = (SELECT id FROM Tables WHERE title = 'Расходы' LIMIT 1) AND member\_id = (SELECT id FROM Users WHERE username = 'mimimi' LIMIT 1) LIMIT 1)),

('2025-04-30', 13.50, (SELECT id FROM Place WHERE name = 'Косметика' LIMIT 1),

(SELECT id FROM Table\_Member WHERE

table\_id = (SELECT id FROM Tables WHERE title = 'Расходы' LIMIT 1) AND member\_id = (SELECT id FROM Users WHERE username = 'pupupu' LIMIT 1) LIMIT 1)),

('2025-05-14', 5.00, (SELECT id FROM Place WHERE name = 'Работа' LIMIT 1),

(SELECT id FROM Table\_Member WHERE

table\_id = (SELECT id FROM Tables WHERE title = 'Доходы' LIMIT 1) AND member\_id = (SELECT id FROM Users WHERE username = 'pipi' LIMIT 1) LIMIT 1));

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | | Наименование | | | Дополнитель- ные сведения | | | |
|  | | | | | Текстовые документы | | |  | | | |
|  | | | | |  | | |  | | | |
| БГУИР КП 1-40 05 01-12 048 ПЗ | | | | | Пояснительная записка | | | 44 с. | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | | Графические документы | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 05 01-12 107 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Разработка подсистемы учёта и ведения семейного бюджета  Ведомость курсового проекта | Лит | | | | Лист | Листов |
| Разраб. | | Лазицкая |  |  |  | Т | |  | 1 | 1 |
| Пров. | | Рязанцев |  |  | Кафедра ЭИ гр. 324403 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |
| . | |  |  |  |
|  | |  |  |  |

# **ВЕДОМОСТЬ ДОКУМЕНТОВ**