МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации
информационных технологий

Отчет по практическим работам №5-8

по дисциплине «Системная и программная инженерия»

Выполнили:
Студенты группы ИКБО-11-22 Берчик А.С.
Андрусенко Л.Д.
Гришин А.В.
Малкин Г.Д.
Гоппен С.Д.

Проверил: Преподаватель Михайлова Е.К.

Москва 2025

СОДЕРЖАНИЕ

1. СОЗДАНИЕ СТРУКТУРНОЙ ДИАГРАММЫ СИСТЕМЫ

1.1. Структурные диаграммы

Основное назначение структурных диаграмм заключается в графическом представлении состава статистических совокупностей, характеризующихся как соотношение различных частей каждой из совокупностей.

Структурная диаграмма – это инструмент модульного дизайна сверху вниз, построенный из квадратов, представляющих различные модули в системе и соединяющие их линии. Линии представляют связь и / или право собственности между видами деятельности и вспомогательными видами деятельности, как они используются в организационных диаграммах.

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их кооперации, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

Диаграмма объектов в языке моделирования UML предназначена для демонстрации совокупности моделируемых объектов и связей между ними в фиксированный момент времени.

1.2. Построение диаграммы процессов в нотации IDEF0

Нотации – графические модели, которые используются, чтобы фиксировать бизнес-процессы, анализировать их и оптимизировать. По сравнению с текстовыми описаниям, графические модели занимают меньше места, помогают увидеть алгоритм наглядно, представить, как он проходит от начала до конца. Однако, в отличие от текстового описания, графическая модель хуже передает детали.

Нотации применяются, чтобы сотрудники могли понять и запомнить схему, по которой они должны, к примеру, обрабатывать заявку на поставку партии товара. А руководителю схема будет полезна, чтобы он мог найти проблемные или избыточные элементы (этапы, сотрудников), внести нужные корректировки. Часто это помогает ускорить или удешевить работу компании.

Данная нотация позволяет создать модель, которая будет отражать:

- структуру системы;
- функции;
- потоки ресурсов, информации.

Модель IDEF0 разворачивается одновременно слева направо и сверху вниз, по диагонали. Объекты, расположенные левее/выше, доминируют над теми, которые находятся правее/ниже. Доминирующие объекты могут включать в себя зависимые: например, доставка заказа – это элемент, входящий в состав более масштабного процесса управления заказами. Также доминирующие объекты могут являться предшествующими этапами для зависимых: получение заявки – согласование заявки.

Графические элементы:

- прямоугольники – действия или этапы;

- стрелки – ресурсы, исполнители, необходимые для совершения действия или прохождения этапа.

Главное достоинство IDEF0 – крайне высокая степень детализации, можно создать модель, которая будет учитывать на каждом этапе практически все ресурсы, сотрудников, которые потребуются даже для самых сложных алгоритмов. Недостатком является то, что графическая модель занимает очень много места, её тяжело читать, не имея специальных навыков.

1.3. Ход работы

Диаграмма классов на Рисунке 1.1 описывает структуру финансового приложения, предназначенного для персонального учета доходов, расходов, кредитов и лимитов. Центральным элементом системы является класс User, представляющий пользователя и содержащий идентификаторы, контактные данные и методы взаимодействия с системой, такие как регистрация, вход в систему, восстановление пароля и получение отчетов. С ним напрямую связаны классы Income, Expense, Credit и Limit, каждый из которых содержит информацию о финансовых операциях и предоставляет базовые методы управления данными: добавление, редактирование и удаление записей.

Классы Income, Expense, Credit и Limit включают поля суммы, даты, категории и ссылки на пользователя. Категории представлены отдельным классом Category, определяющим имя, тип и связь с владельцем —

пользователем. Каждая операция имеет привязку к определённой категории, что позволяет структурировать и анализировать данные. Система отчетности реализована через классы Report и ReportEntry: первый агрегирует записи по заданному диапазону дат и позволяет фильтрацию по категориям или дате, второй содержит информацию о категории и общей сумме операций.

Поддержка бизнес-логики обеспечивается через класс System, который содержит список пользователей и реализует методы прогнозирования, анализа тенденций и отправки уведомлений. Дополнительно в системе присутствует класс Admin, предназначенный для управления пользователями и настройки интеграций. Связи между классами иллюстрируют агрегацию, композицию и ассоциации, отражающие отношения между пользователями, их операциями и отчетностью. Диаграмма предоставляет полное представление о логике взаимодействия ключевых компонентов приложения.

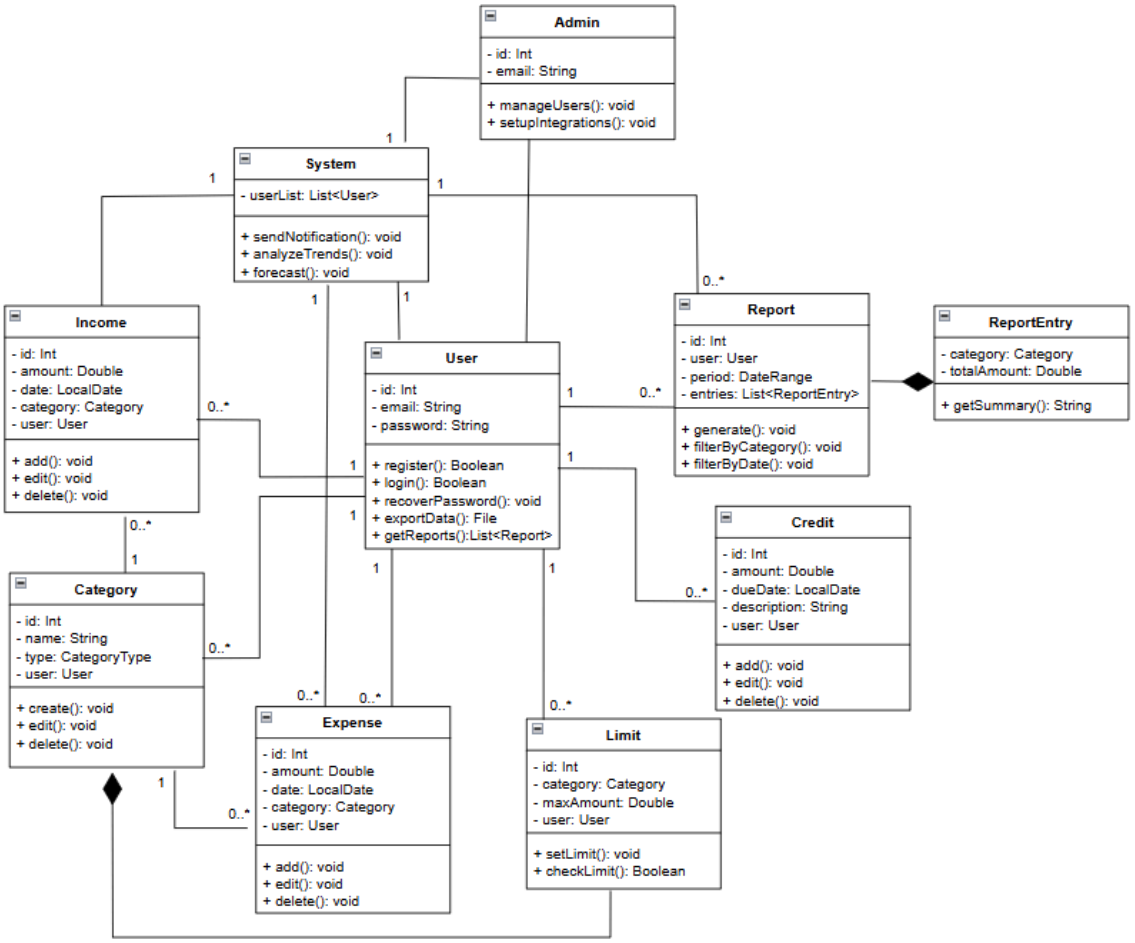

Рисунок 1.1 – Диаграмма классов

Диаграмма объектов на Рисунке 1.2 иллюстрирует на данном этапе работы приложения активность пользователя user1: User, вошедший в систему под своим логином и просматривающий свою финансовую активность. Объектная диаграмма отражает конкретную конфигурацию объектов, находящихся в оперативной памяти приложения.

Объект user1: User

Содержит личные данные пользователя (email, пароль), а также связан с рядом объектов, отражающих его деятельность: расходы, доходы,

кредиты, лимиты, категории и отчёты.

Доходы (income1: Income**,** income2: Income**)**

Два объекта доходов отображают конкретные поступления средств. Они ссылаются на user1 как владельца и на объекты category1: Category и category2: Category, которые определяют тип дохода (например, «зарплата», «подарок»).

Расходы (expense1: Expense**)**

Отражает одну траты пользователя. Содержит ссылку на категорию category3: Category (например, «еда») и принадлежит пользователю user1.

Кредит (credit1: Credit**)**

Объект, представляющий заём. Имеет описание и дату, до которой нужно погасить долг. Также ссылается на user1.

Лимит (limit1: Limit**)**

Ограничение по сумме, установленное пользователем на определённую категорию (например, максимум на развлечения). Ссылается на category3 и пользователя user1.

Категории (category1**,** category2**,** category3**)**

Объекты категорий, определяющие тип расходов или доходов. Все связаны с user1.

Отчёт (report1: Report**)**

Объект, формирующий агрегированные данные по финансам пользователя за определённый период. Внутри него содержится список reportEntry1, reportEntry2, каждый из которых описывает категорию и общую сумму по ней.

Объект system1: System

Содержит список пользователей, включая user1, и отвечает за обработку уведомлений, анализ трендов и прогнозирование.

Админ (admin1: Admin**)**

Управляет пользователями в системе, включая user1, через объект system1.

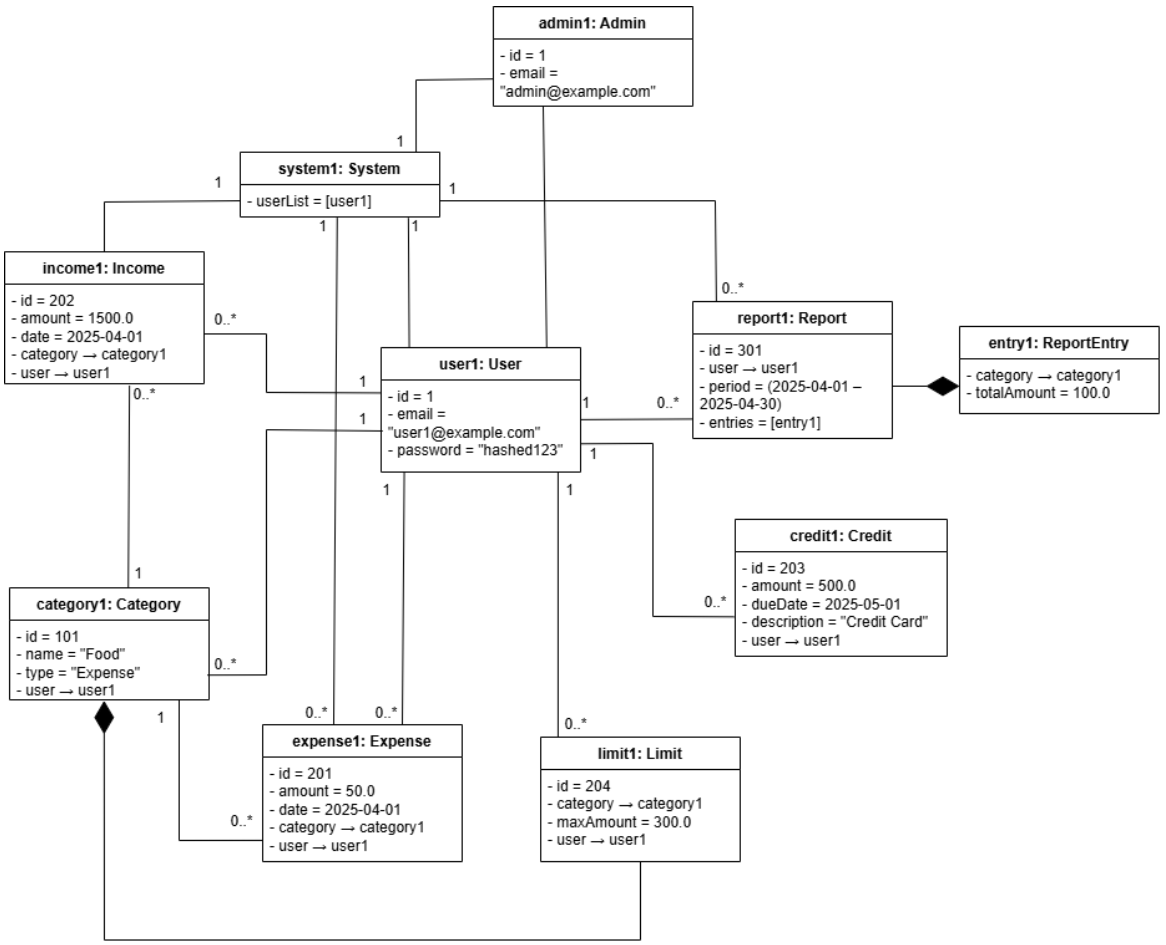

Рисунок 1.2 – Диаграмма объектов

Контекстная диаграмма A0 на Рисунке 1.3 отражает основную цель системы управления личными финансами — предоставление пользователю инструментов для эффективного контроля и анализа своих финансовых данных. Система принимает на вход данные пользователя, финансовые транзакции, категории, а также пользовательские настройки. На выходе система предоставляет отчеты и аналитику, уведомления, прогнозы и экспортированные данные, что позволяет пользователю принимать обоснованные финансовые решения.

Управление процессом осуществляется через политики и правила системы, а также законодательные требования, обеспечивающие безопасность и конфиденциальность данных. Механизмы реализации включают программное обеспечение, пользовательские устройства и сетевые ресурсы, которые обеспечивают функционирование системы и взаимодействие с пользователями и внешними системами.

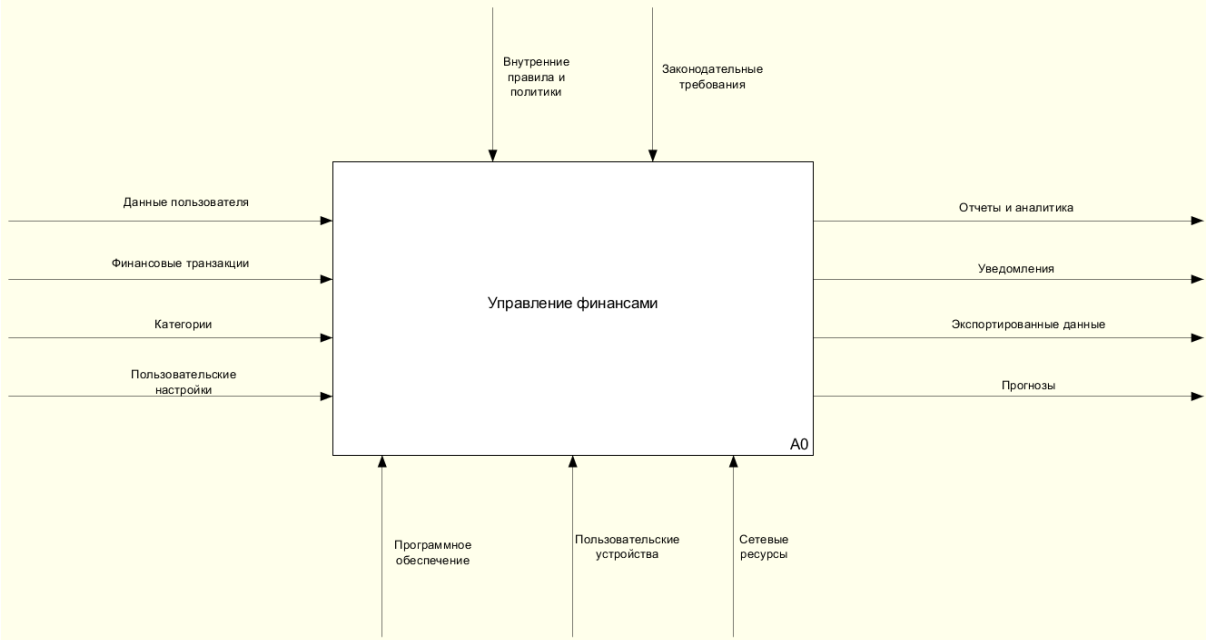

Рисунок 1.3 – Контекстная диаграмма

Функциональный блок A0 был декомпозирован на уровень вниз (Рисунок 1.4).

Блок A1: Регистрация и авторизация обеспечивает регистрацию и авторизацию пользователей, предоставляя доступ к системе. Пользователь вводит свои данные для регистрации или входа. Политики безопасности и законодательные требования используются для защиты данных.

Блок A2: Управление финансовыми данными обрабатывает

финансовые транзакции, категории и настройки, которые вводятся пользователем. Эти данные обновляются в финансовой базе данных. Управляется политиками и правилами системы для корректной обработки данных.

Блок A3: Анализ и отчетность генерирует отчеты и аналитику на основе обновленных финансовых данных. Пользователь может экспортировать данные для дальнейшего использования. Политики и правила системы определяют формат и содержание отчетов.

Блок A4: Уведомления и прогнозирование создает уведомления и прогнозы на основе текущих финансовых данных. Помогает пользователю планировать свои финансы и принимать обоснованные решения. Уведомления и прогнозы отправляются пользователю для информирования о финансовом состоянии.

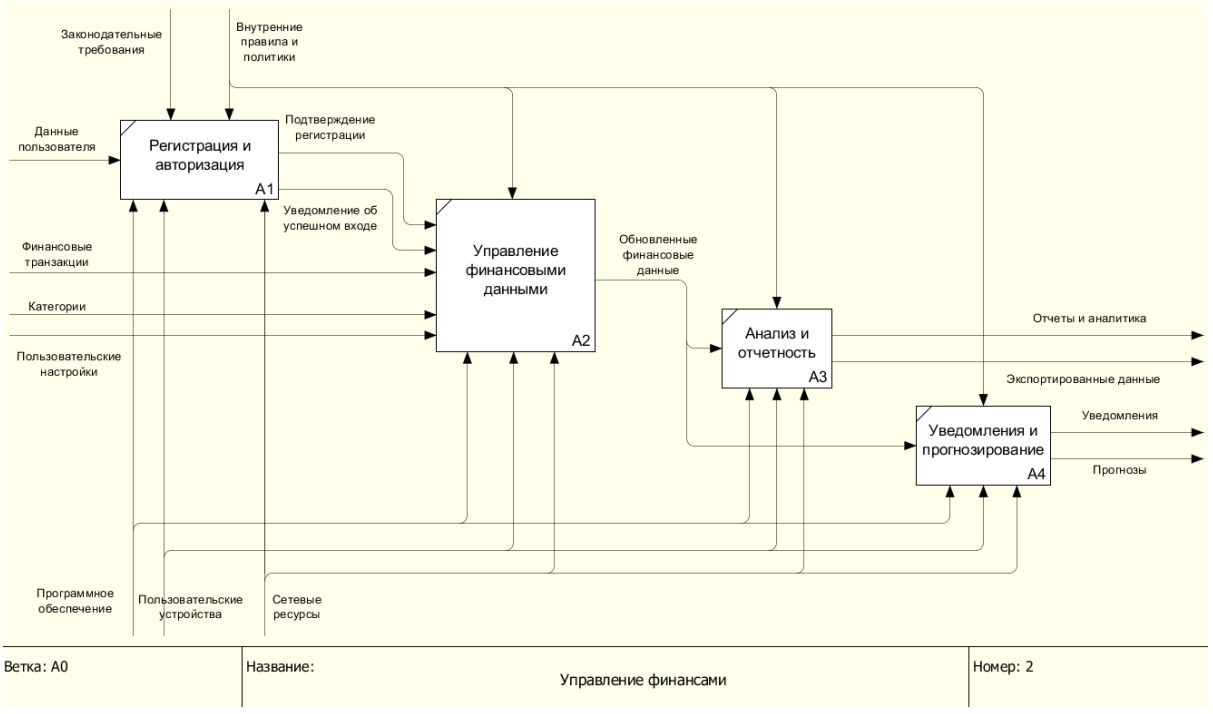

Рисунок 1.4 – Декомпозиция контекстной диаграммы

2. СОЗДАНИЕ ИНФОРМАЦИОННОЙ ДИАГРАММЫ СИСТЕМЫ И ЛОГИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ

2.1. Диаграмма в нотации DFD

DFD — общепринятое сокращение от англ. data flow diagrams — диаграммы потоков данных. Так называется методология графического структурного анализа, описывающая внешние по отношению к системе, источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. Диаграмма потоков данных (data flow diagram, DFD) — один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML.

2.2 Ход работы

На рисунках 2.1 и 2.2 показаны диаграммы в нотации DFD, описывающие потоки данных в ходе работы системы.

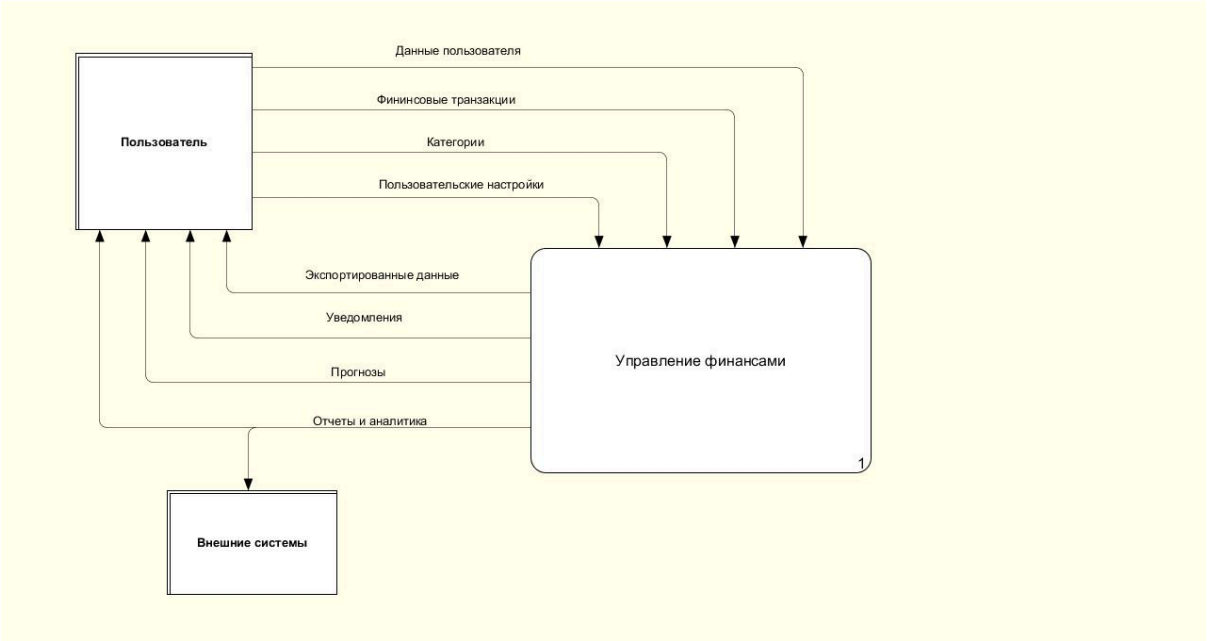

Рисунок 2.1 – Контекстная диаграмма

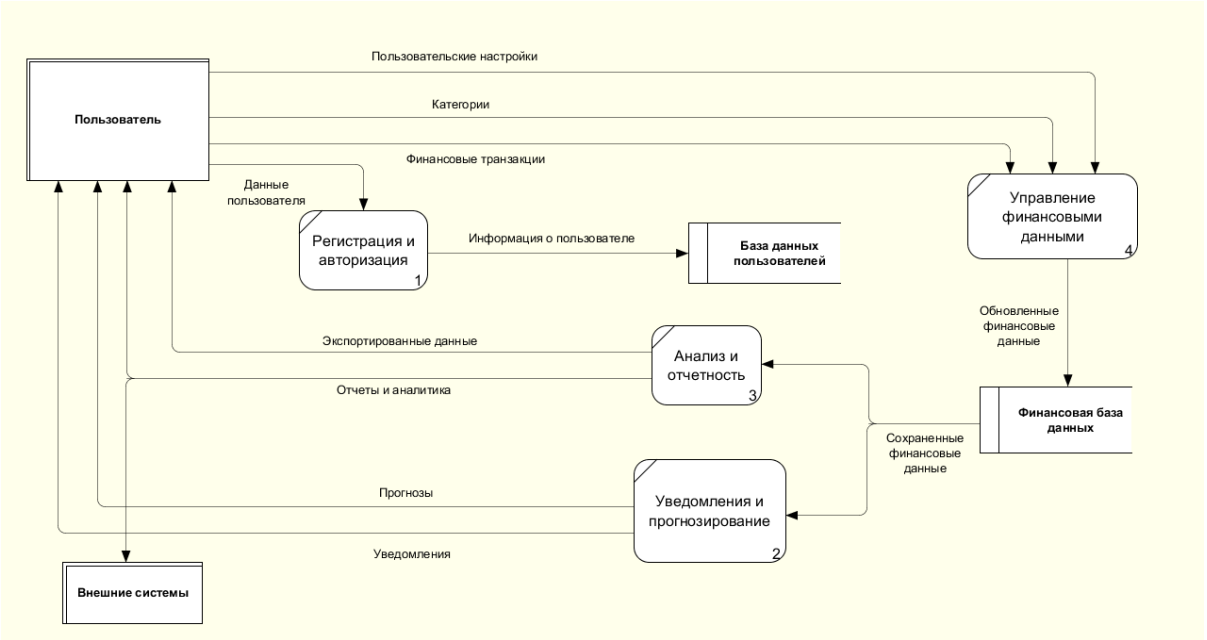

Рисунок 2.2 – Декомпозиция контекстной диаграммы

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Информационное взаимодействие компонентов системы

Первый уровень – уровень представления – отвечает за визуальное отображение финансовой информации и взаимодействие с пользователем. Здесь размещены экраны входа в систему, регистрации пользователя, основного меню приложения, а также интерфейсы добавления и редактирования финансовых операций. На этом уровне происходит обработка событий пользовательского ввода (нажатия кнопок, заполнение форм), валидация введенных данных и отображение ответных реакций системы через уведомления, диалоги и обновление визуальных компонентов. Модули представления передают запросы пользователя на следующий уровень и обновляют интерфейс на основе полученных результатов.

Второй уровень – уровень бизнес-логики – содержит основные алгоритмы обработки финансовых данных. Именно здесь выполняются операции сортировки, фильтрации и группировки транзакций, производятся расчеты итоговых сумм по категориям, вычисляются статистические показатели и формируются прогнозы будущих расходов с использованием методов линейной регрессии. Бизнес-логика обрабатывает команды, полученные от уровня представления, применяет необходимые трансформации к данным и возвращает результаты для отображения. Также этот уровень контролирует соблюдение установленных лимитов расходов и генерирует уведомления при приближении к пороговым значениям.

Третий уровень – уровень доступа к данным – отвечает за взаимодействие с удаленным API и локальным хранилищем. Данный уровень инкапсулирует все детали сетевого взаимодействия, включая

формирование HTTP-запросов, обработку ответов сервера и трансформацию между сетевыми форматами и объектами приложения. Он также обеспечивает кэширование данных для работы в автономном режиме и синхронизацию локальных изменений с сервером при восстановлении подключения. Этот уровень абстрагирует источник данных от остальной части приложения, что позволяет легко заменить сервер или способ хранения без изменения бизнес-логики.

Четвертый уровень – уровень моделей данных – представляет собой структуры и классы, описывающие основные сущности системы: пользователей, финансовые операции, категории доходов и расходов. Этот уровень определяет формат хранения информации и правила ее проверки. Модели данных используются всеми остальными уровнями как общий язык взаимодействия, что обеспечивает согласованность и целостность информации в системе.

Взаимодействие между уровнями выглядит следующим образом. Пользователь взаимодействует с интерфейсом на уровне представления, например, вводит сумму и выбирает категорию для нового расхода. Уровень представления собирает необходимые данные и передает команду на уровень бизнес-логики. Бизнес-логика проверяет соответствие операции установленным правилам (например, не превышает ли она лимит для выбранной категории) и формирует запрос к уровню доступа к данным для сохранения информации. Уровень доступа к данным преобразует запрос в формат, понятный серверу, отправляет его и обрабатывает ответ. После успешного сохранения данных обновленная информация проходит обратный путь: она загружается из уровня доступа к данным, обрабатывается бизнес-логикой и отображается пользователю через уровень представления. Таким образом, каждый уровень выполняет свою специализированную функцию, взаимодействуя с соседними уровнями через четко определенные интерфейсы.

3.2 Логическая модель базы данных

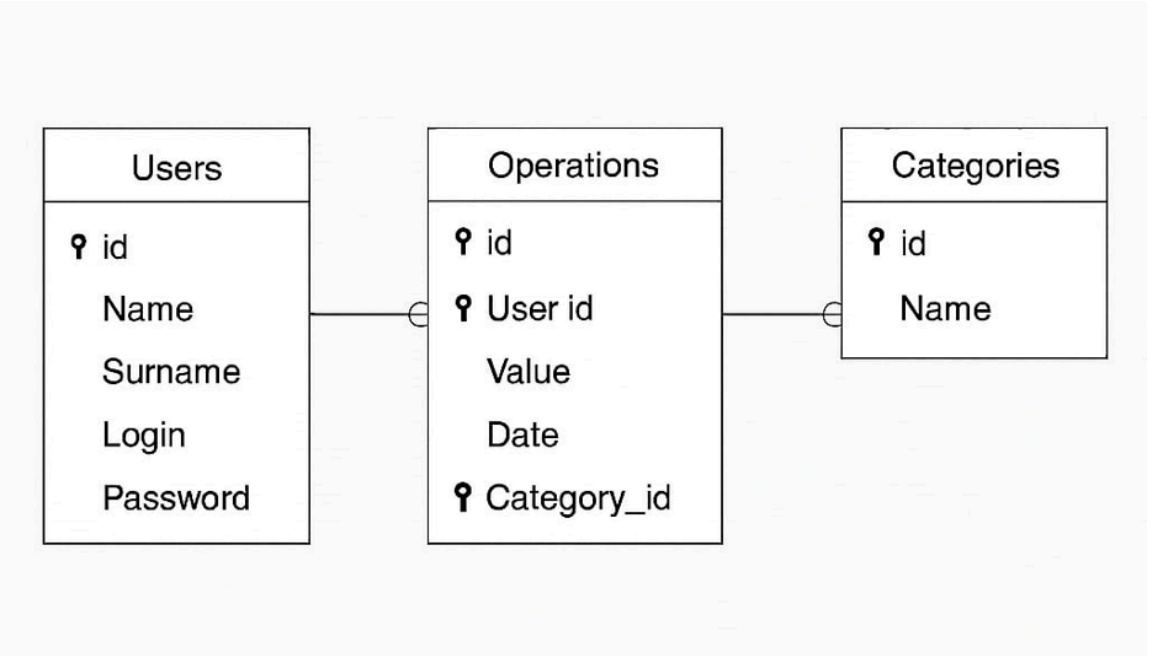

Рисунок 3 - Логическая модель базы данных

3.3 Архитектура системы

Общее описание архитектуры

Предлагаемая архитектура системы представляет собой мобильное приложение для управления личными финансами, построенное на основе клиент-серверной модели. Система состоит из следующих основных компонентов:

1. Мобильное приложение (клиентская часть) - Android-приложение с современным пользовательским интерфейсом
2. Серверная часть - RESTful API для обработки запросов и работы с базой данных
3. База данных - реляционное хранилище для пользовательских данных и финансовых операций

Архитектура построена по принципу многоуровневого разделения ответственности, что обеспечивает гибкость, масштабируемость и удобство сопровождения системы.

Программные решения и обоснование выбора

Язык программирования: Kotlin

Обоснование: Kotlin является современным, безопасным и лаконичным языком, официально поддерживаемым Google для разработки Android-приложений. По сравнению с Java, Kotlin обеспечивает более высокую производительность разработки и меньшее количество ошибок благодаря нулевой безопасности (null safety) и функциональным возможностям языка.

Архитектурный паттерн: MVVM (Model-View-ViewModel)

- **Обоснование**: MVVM обеспечивает четкое разделение логики представления от бизнес-логики, что делает код более тестируемым и поддерживаемым. Использование ViewModel из библиотеки Android Architecture Components позволяет сохранять состояние приложения при изменениях конфигурации устройства (например, при повороте экрана).

Библиотеки и фреймворки:

- **AndroidX** - набор современных библиотек для разработки Android-приложений

Обоснование: Обеспечивает совместимость с новыми версиями Android и предоставляет готовые решения для типичных задач разработки.

- **Material Design Components** - библиотека для создания интерфейсов в стиле Material Design

Обоснование: Позволяет создать современный, консистентный и интуитивно понятный интерфейс, соответствующий рекомендациям Google.

- **Retrofit + OkHttp** - библиотеки для работы с REST API

Обоснование: Retrofit обеспечивает типобезопасный клиент для HTTP-запросов, а OkHttp предоставляет эффективный HTTP-клиент с поддержкой кэширования, перехвата запросов и других полезных функций.

- **RxJava/RxAndroid** - библиотеки для реактивного программирования

Обоснование: Позволяют эффективно работать с асинхронными операциями и событиями, упрощают управление многопоточностью и обработку ошибок.

- **MPAndroidChart** - библиотека для визуализации данных

Обоснование: Предоставляет богатый набор инструментов для создания интерактивных графиков и диаграмм, что критически важно для финансового приложения.

- **Kotlin Statistics** - библиотека для статистического анализа

Обоснование: Упрощает реализацию алгоритмов прогнозирования и анализа финансовых данных, в частности, линейной регрессии для предсказания будущих расходов.

Серверная часть

Язык программирования: C# (.NET Core)

Обоснование: C# в сочетании с .NET Core обеспечивает высокую производительность, кроссплатформенность и широкий набор инструментов для создания веб-сервисов. Язык имеет строгую типизацию, что уменьшает количество ошибок при разработке.

Фреймворки:

- **ASP.NET Core** - фреймворк для создания веб-приложений и API

Обоснование: Предоставляет высокопроизводительную и модульную платформу для создания RESTful API с поддержкой различных форматов данных и аутентификации.

- **Entity Framework Core** - ORM для работы с базой данных

Обоснование: Упрощает доступ к данным и обеспечивает абстракцию от конкретной СУБД, поддерживает миграции и различные подходы к моделированию данных.

База данных

СУБД: PostgreSQL

Обоснование: PostgreSQL является мощной, свободной и открытой СУБД с большим сообществом поддержки. Она обеспечивает высокую надежность, соответствие стандарту SQL и хорошую производительность. В отличие от более простых решений (например, SQLite), PostgreSQL легко масштабируется и поддерживает более сложные типы данных и операции, что может быть полезно при развитии системы.

Дополнительные технологии

Docker - платформа для контейнеризации приложений

Обоснование: Упрощает развертывание серверной части и базы данных, обеспечивает изоляцию компонентов и упрощает масштабирование.

JWT (JSON Web Tokens) - метод для безопасной передачи утверждений между сторонами

Обоснование: Обеспечивает статусную аутентификацию и

авторизацию между клиентом и сервером, что важно для личного финансового приложения.

HTTPS - протокол для безопасной передачи данных

Обоснование: Необходим для защиты персональных и финансовых данных пользователей при передаче между клиентом и сервером.

Преимущества предлагаемой архитектуры

1. **Модульность и разделение ответственности** - каждый компонент системы отвечает за свою часть функциональности, что упрощает тестирование, отладку и поддержку.
2. **Масштабируемость** - архитектура позволяет легко расширять функциональность как клиентской, так и серверной части.
3. **Безопасность** - использование современных протоколов и методов аутентификации обеспечивает надежную защиту пользовательских данных.
4. **Отказоустойчивость** - кэширование данных на стороне клиента позволяет приложению частично функционировать даже при отсутствии подключения к серверу.
5. **Удобство разработки** - выбранные технологии имеют хорошую документацию, активное сообщество и широкий набор инструментов разработки.
6. **Производительность** - используемые фреймворки и библиотеки оптимизированы для высокой производительности и низкого потребления ресурсов.

Предложенная архитектура и технологический стек полностью соответствуют требованиям системы личного финансового учета.

Клиентская часть обеспечивает удобный и отзывчивый пользовательский интерфейс, а серверная часть гарантирует безопасное хранение и обработку данных. Выбранные технологии являются современными, хорошо поддерживаемыми и обладают обширной документацией, что упростит разработку и сопровождение системы.

3.4 Архитектурная диаграмма разработки

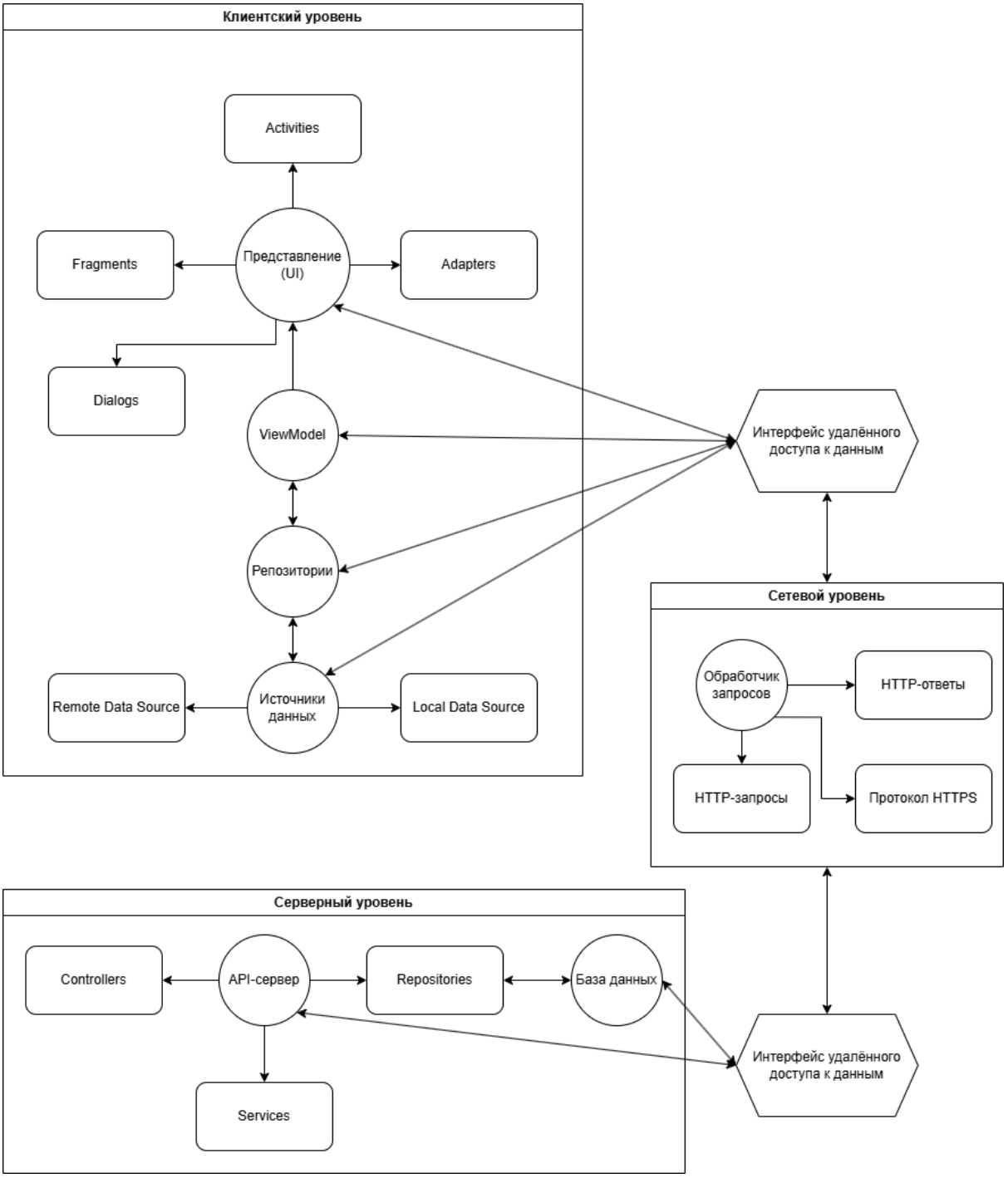

Рисунок 4 - Архитектурная диаграмма разработки

3.5 Матрица требований

Таблица 1 – Дополненная матрица требований

Составляющая	**Требования**	**Суть**	**Компоненты архитектуры**
Регистрация и авторизация	Регистрация пользователей	Пользователь должен иметь возможность регистрироваться с использованием email и пароля.	Модуль представления (UI-фрагменты авторизации), Модуль доступа к данным (API-клиент), Серверный компонент (Контроллер пользователей)
	Авторизация	Пользователь должен иметь возможность входить в систему.	Модуль представления (UI-фрагменты авторизации), Модуль доступа к данным (API-клиент), Серверный компонент (Контроллер аутентификации)
	Восстановление пароля	Система должна поддерживать восстановление пароля.	Модуль представления (UI-диалоги восстановления), Модуль доступа к данным (API-клиент), Серверный компонент (Сервис пользователей)
Управление доходами	Добавление доходов	Пользователь должен иметь возможность добавлять доходы.	Модуль представления (UI-фрагменты доходов), Модуль бизнес-логики (Менеджер операций), Модуль доступа к данным (Репозиторий операций)
	Редактирование доходов	Пользователь должен иметь возможность редактировать доходы.	Модуль представления (UI-фрагменты доходов), Модуль бизнес-логики (Менеджер операций), Модуль доступа к данным (Репозиторий операций)
	Удаление доходов	Пользователь должен иметь возможность удалять доходы.	Модуль представления (UI-фрагменты доходов), Модуль бизнес-логики (Менеджер операций), Модуль

			доступа к данным (Репозиторий операций)
Управление расходами	Добавление расходов	Пользователь должен иметь возможность добавлять расходы.	Модуль представления (UI-фрагменты расходов), Модуль бизнес-логики (Менеджер операций), Модуль доступа к данным (Репозиторий операций)
	Редактирование расходов	Пользователь должен иметь возможность редактировать расходы.	Модуль представления (UI-фрагменты расходов), Модуль бизнес-логики (Менеджер операций), Модуль доступа к данным (Репозиторий операций)
	Удаление расходов	Пользователь должен иметь возможность удалять расходы.	Модуль представления (UI-фрагменты расходов), Модуль бизнес-логики (Менеджер операций), Модуль доступа к данным (Репозиторий операций)
Категоризация	Создание категорий	Пользователь должен иметь возможность создавать категории.	Модуль представления (UI-фрагменты категорий), Модуль бизнес-логики (Менеджер категорий), Модуль доступа к данным (Хранилище категорий)
	Редактирование категорий	Пользователь должен иметь возможность редактировать категории.	Модуль представления (UI-фрагменты категорий), Модуль бизнес-логики (Менеджер категорий), Модуль доступа к данным (Хранилище категорий)
	Удаление категорий	Пользователь должен иметь возможность удалять категории.	Модуль представления (UI-фрагменты категорий), Модуль бизнес-логики (Менеджер категорий), Модуль доступа к данным (Хранилище категорий)

Отчеты и аналитика	Генерация отчетов	Система должна генерировать отчеты по доходам и расходам.	Модуль бизнес-логики (Сервис аналитики), Модуль представления (UI-компоненты визуализации), Модуль доступа к данным (Репозиторий операций)
	Фильтрация данных	Пользователь должен иметь возможность фильтровать данные.	Модуль бизнес-логики (Сервис фильтрации), Модуль представления (UI-компоненты фильтров), Модуль доступа к данным (Репозиторий операций)
Управление лимитами	Установка лимитов	Пользователь должен иметь возможность устанавливать лимиты.	Модуль представления (UI-фрагменты лимитов), Модуль бизнес-логики (Сервис лимитов), Модуль доступа к данным (Хранилище настроек)
	Уведомления о лимитах	Система должна уведомлять о приближении к лимитам.	Модуль бизнес-логики (Сервис уведомлений), Модуль представления (Система уведомлений), Инфраструктурный компонент (Сервис уведомлений)
Управление кредитами и ссудами	Добавление кредитов/ссуд	Пользователь должен иметь возможность добавлять кредиты/ссуды.	Модуль представления (UI-фрагменты кредитов), Модуль бизнес-логики (Менеджер кредитов), Модуль доступа к данным (Репозиторий кредитов)
	Редактирование кредитов/ссуд	Пользователь должен иметь возможность редактировать кредиты/ссуды.	Модуль представления (UI-фрагменты кредитов), Модуль бизнес-логики (Менеджер кредитов), Модуль доступа к данным (Репозиторий кредитов)

	Удаление кредитов/ссуд	Пользователь должен иметь возможность удалять кредиты/ссуды.	Модуль представления (UI-фрагменты кредитов), Модуль бизнес-логики (Менеджер кредитов), Модуль доступа к данным (Репозиторий кредитов)
Экспорт данных	Экспорт в CSV	Пользователь должен иметь возможность экспортировать данные в CSV.	Модуль бизнес-логики (Сервис экспорта), Модуль представления (UI-компоненты экспорта), Инфраструктурный компонент (Файловая система)
Прогнозирование	Прогнозирование доходов/расходов	Система должна предоставлять инструменты для прогнозирования.	Модуль бизнес-логики (Сервис прогнозирования), Модуль представления (UI-компоненты визуализации), Алгоритмический компонент (Расчетный модуль)
Уведомления	Уведомления о лимитах	Система должна отправлять уведомления о лимитах.	Модуль бизнес-логики (Сервис уведомлений), Системный компонент (Менеджер уведомлений), Модуль представления (UI-компоненты уведомлений)
Поддержка платформ	Мобильные устройства	Система должна работать на Android.	Инфраструктурный компонент (Android SDK), Модуль представления (Адаптивный UI), Системный компонент (Оптимизация ресурсов) |
Производительность	Время отклика	Время отклика не должно превышать 2 секунд.	Инфраструктурный компонент (Асинхронная обработка), Модуль бизнес-логики (Оптимизация алгоритмов), Модуль доступа к данным (Кэширование)

	Время загрузки	Время загрузки страниц не должно превышать 3 секунд.	Инфраструктурный компонент (Управление ресурсами), Модуль представления (Ленивая загрузка UI), Модуль доступа к данным (Эффективные запросы)
	Обработка данных	Система должна обрабатывать до 10,000 транзакций за 5 секунд.	Модуль бизнес-логики (Оптимизированные алгоритмы), Инфраструктурный компонент (Многопоточная обработка), Модуль доступа к данным (Индексирование)
Масштабируемость	Поддержка роста пользователей	Система должна поддерживать до 100,000 пользователей.	Серверный компонент (Балансировка нагрузки), Модуль доступа к данным (Оптимизация БД), Инфраструктурный компонент (Кэширование)
	Поддержка больших объемов данных	Система должна обрабатывать большие объемы данных.	Модуль доступа к данным (Пагинация), Серверный компонент (Оптимизация запросов), Инфраструктурный компонент (Управление памятью)
Надежность	Восстановление данных	В случае сбоя данные должны восстанавливаться автоматически.	Серверный компонент (Резервное копирование), Модуль доступа к данным (Журналирование), Инфраструктурный компонент (Обработка ошибок)
	Обработка ошибок	Система должна уведомлять о проблемах.	Модуль представления (UI обработки ошибок), Инфраструктурный компонент (Перехват исключений), Модуль доступа к данным (Транзакции)

Безопасность	Защита данных	Все данные должны быть зашифрованы.	Инфраструктурный компонент (Шифрование), Модуль доступа к данным (Безопасное хранение), Серверный компонент (Защита в транзите)
	Конфиденциальность	Система должна соответствовать требованиям регуляторов.	Инфраструктурный компонент (Управление доступом), Серверный компонент (Соответствие стандартам), Модуль бизнес-логики (Аудит действий)

4. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

4.1 Обоснование выбора ГОСТ для разработки технического задания

Для разработки технического задания (ТЗ) проекта мобильного приложения управления личными финансами, описанного в отчетах №1-4 и №5-8, выбран ГОСТ 34.602-2020 «Техническое задание на создание автоматизированной системы».

Причины выбора:

1. Комплексность проекта:

Проект включает не только разработку клиентского мобильного приложения (Android) и серверной части (REST API на .NET Core), но и взаимодействие с базой данных (PostgreSQL), обеспечение безопасности (шифрование, HTTPS, JWT), а также интеграцию с внешними системами. ГОСТ 34.602-2020 охватывает все аспекты автоматизированных систем (АС), включая требования к структуре, надежности, безопасности, документации и взаимодействию компонентов.

2. Соответствие разделам ТЗ:

Общие сведения: Включает описание системы, участников разработки, сроки и источники финансирования (соответствует разделам отчетов о команде и этапах разработки).

Назначение и цели: Совпадает с функциональными и нефункциональными требованиями из матрицы требований (управление финансами, аналитика, прогнозирование).

Требования к системе:

Надежность: Требования к восстановлению данных, обработке ошибок, времени отклика (см. нефункциональные требования).

Безопасность: Шифрование данных, соответствие ФЗ-152, использование HTTPS и JWT.

Эргономика: Требования к интерфейсу (Material Design, адаптивность).

Документирование: Учитывает необходимость подготовки технической и пользовательской документации.

3. Учет автоматизации процессов:

Система автоматизирует финансовый учет, анализ и прогнозирование, что соответствует определению автоматизированной системы в ГОСТ 34.602-2020.

4. Гибкость стандарта:

ГОСТ 34.602-2020 допускает адаптацию разделов под специфику проекта, что важно для мобильного приложения с клиент-серверной архитектурой. Например, раздел «Требования к видам обеспечения» может быть дополнен описанием использования Kotlin, .NET Core, Retrofit, RxJava и других технологий.

ГОСТ 34.602-2020 выбран как основной стандарт, так как он полностью охватывает требования к автоматизированной системе, включая программные, аппаратные и организационные аспекты.

4.2 Техническое задание на создание автоматизированной системы "UWasting" (по ГОСТ 34.602-2020)

1. Общие сведения

1.1. Наименование системы

- Полное наименование: Мобильное приложение для управления личными и семейными финансами

- Краткое наименование: UWasting

1.2. Основание для разработки

Работа выполняется на основании задания по дисциплине «Системная и программная инженерия» от кафедры Математического обеспечения и стандартизации информационных технологий РТУ МИРЭА

1.3. Заказчик и разработчик

Заказчик: РТУ МИРЭА

Разработчик: Студенты группы ИКБО-11-22 (Берчик А.С., Андрусенко Л.Д., Гришин А.В., Малкин Г.Д., Гоппен С.Д.)

1.4. Сроки разработки

Начало работ: 14.02.2025

Окончание работ: 30.05.2025

2. Назначение и цели создания системы

2.1. Назначение

Автоматизация процессов:

- Учёт доходов, расходов, кредитов и ссуд.
- Генерация аналитических отчетов.
- Прогнозирование финансовых потоков.

2.2. Цели

- Повышение финансовой грамотности пользователей.
- Сокращение времени на ручной учет.

3. Требования к системе

3.1. Функциональные требования

- Регистрация/авторизация (с восстановлением пароля).
- Управление доходами, расходами, категориями.
- Генерация отчетов (графики, CSV-экспорт).
- Уведомления о приближении к лимитам.

3.2. Нефункциональные требования

Производительность:

- Время отклика ≤ 2 сек.
- Обработка 10,000 транзакций за 5 сек.

Безопасность:

- Шифрование данных (AES-256, TLS/SSL).
- Соответствие ФЗ-152 "О персональных данных".

Надежность:

- Восстановление данных при сбоях.

Платформы: Android 12 и выше.

4. Состав и содержание работ

1. Анализ требований.
2. Прототипирование.
3. Разработка:
   - 3.1. Клиентская часть (Kotlin, RxJava).
   - 3.2. Серверная часть (.NET Core, Entity Framework).
4. Тестирование (нагрузочное, юзабилити-тесты).

5. Порядок приемки

1. Этапы тестирования:
   - 1.1. Модульное (покрытие кода ≥ 85%).
   - 1.2. Интеграционное (проверка API).
   - 1.3. Приёмочное (соответствие ТЗ).

2. Критерии приемки:
    2.1. Выполнение всех функциональных требований.
    2.2. Прохождение нагрузочных тестов.

6. Источники разработки

1. ГОСТ 34.602-2020.
2. Анализ аналогов (приложения MoneyFlow, CoinKeeper).

ЗАКЛЮЧЕНИЕ

В ходе выполнения практических работ были завершены этапы проектирования и моделирования системы управления личными финансами.

Разработаны структурные диаграммы (классов, объектов, IDEF0) и информационные модели (DFD), отражающие логику работы системы.

Определена клиент-серверная архитектура с использованием технологий Kotlin (клиент), .NET Core (сервер) и PostgreSQL (СУБД).

Дополнена матрица требований, подтверждающая соответствие функциональных и нефункциональных характеристик проекту.

Полученные результаты являются основой для дальнейшей разработки, тестирования и внедрения системы.