

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Платформа для управления личными
финансами и планирования бюджета на основе
микросервисной архитектуры**

Студент

(подпись, дата)

Родионов Иван Викторович

(ФИО полностью)

Руководитель

(подпись, дата)

Балашова И.Ю.

(фамилия, инициалы)

Нормоконтролер

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Родионов Иван Викторович гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Платформа для управления личными финансами и планирования бюджета на основе микросервисной архитектуры

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Балашова И.Ю.

4. Задание на работу (назначение разработки, исходные данные и т. п.)

1) *Назначение: веб-платформа для управления финансами и планирования бюджета, обеспечивающая учет доходов и расходов, контроль бюджетов и целей.*

2) *Среда выполнения: Windows 11.*

Языковые средства разработки: Java, TypeScript, Python, SQL, UML.

Инструментальные средства: IntelliJ IDEA 2024, WebStorm 2024, pgAdmin, Visual Paradigm 17.3, Docker.

Технологии разработки: микросервисная архитектура, ООП, REST API, JWT.

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) учет транзакций (доходы, расходы, переводы между кошельками);

в) управление категориями операций и аналитикой по категориям;

г) формирование бюджетов по периодам и контроль лимитов;

д) постановка финансовых целей и отслеживание прогресса;

е) просмотр сводной аналитики (баланс, динамика, структура расходов);

ж) импорт транзакций из квитанций в формате PDF.

5. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области и постановка задачи;
- 2) Проектирование;
- 3) Реализация;
- 4) Контроль качества программного обеспечения;
- 5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 116 листов, 32 рисунка, 28 таблиц, 1 листинг, 21 использованный источник и 6 приложений.

ПЛАТФОРМА ДЛЯ УПРАВЛЕНИЯ ЛИЧНЫМИ ФИНАНСАМИ, ПЛАНИРОВАНИЕ БЮДЖЕТА, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, JAVA, SPRING BOOT, REACT, KAFKA, REDIS, POSTGRESQL, NGINX, JWT.

Объект исследования – программные средства управления личными финансами, учета транзакций и планирования бюджета.

Цель выпускной квалификационной работы – спроектировать и разработать платформу для управления личными финансами и планирования бюджета.

Технологии разработки – серверная часть приложения реализована на языке Java с использованием фреймворка Spring Boot, клиентская часть разработана на React с использованием TypeScript. Для хранения данных использована СУБД PostgreSQL, для межсервисного взаимодействия применяется технология Apache Kafka, для кэширования и управления сессиями используется Redis. Развертывание выполнено с применением Docker, маршрутизация запросов реализована через Nginx.

Среда исполнения – Windows 11, Docker Desktop, веб-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы была спроектирована и реализована платформа для управления личными финансами, включающая модули авторизации, пользователей, кошельков, транзакций, категорий, бюджетов, финансовых целей и аналитики. Реализована обработка финансовых операций, контроль лимитов, формирование аналитических отчетов. Все требования к разработке приложения выполнены.

					ПГУ 09.03.04 – 08BP222.25 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Платформа для управления личными финансами и планирования бюджета на основе микросервисной архитектуры. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Родионов И.В.						
Пров.		Балашова И.Ю.					4	116
Реценз.						Группа 22ВП2		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и выявление требований на разработку платформы для управления личными финансами и планирования бюджета	9
1.1 Исследование предметной области и анализ процессов управления финансами.....	9
1.2 Сравнительный анализ приложений для управления личными финансами и планирования бюджета.....	10
1.3 Анализ функциональных требований	16
1.4 Анализ нефункциональных требований	20
1.5 Планирование разработки и оценки бюджета	22
2 Проектирование платформы для управления личными финансами и планирования бюджета.....	24
2.1 Архитектура приложения.....	24
2.2 Проектирование структур данных	26
2.2.1 Концептуальная модель базы данных.....	26
2.2.2 Логическая модель базы данных	28
2.3 Проектирование пользовательского интерфейса.....	29
2.4 Проектирование классов программных средств.....	30
3 Реализация платформы для управления личными финансами и планирования бюджета.....	33
3.1 Реализация базы данных.....	33
3.2 Реализация программных компонентов	40
3.3 Реализация пользовательского интерфейса	43
4 Контроль качества программного обеспечения.....	56
4.1 Выбор режима тестирования и метрик кода	56
4.2 Тестирование платформы для управления личными финансами и планирования бюджета.....	57

4.2.1 Тестирование пользовательской части приложения для управления личными финансами и планирования бюджета.....	57
4.2.2 Тестирование серверной части приложения для управления личными финансами и планирования бюджета	60
4.3 Оценка качества программного кода	66
4.3.1 Метрика SLOC.....	66
4.3.2 Метрика Холстеда.....	68
Заключение	74
Список использованных источников	76
Приложение А (справочное) Глоссарий	79
Приложение Б (обязательное) Листинг приложения	82
Приложение В (рекомендуемое) Графическая часть	86
Приложение Г (рекомендуемое) Результаты функционального тестирования..	92
Приложение Д (рекомендуемое) Листинг модульных тестов.....	98
Приложение Е (рекомендуемое) Результаты модульного тестирования.....	111

Введение

Цифровизация финансовых сервисов привела к тому, что у пользователя одновременно появляются банковские карты, электронные кошельки, накопительные счета и регулярные платежи в различных сервисах. При большом объеме операций ручной учет перестает быть надежным: повышается риск возникновения ошибок, ухудшается качество финансового планирования. Именно поэтому задача создания единой платформы для управления личными финансами и планирования бюджета является актуальной.

Существующие решения для персонального финансового учета часто способны закрыть лишь часть потребностей пользователя: одни решения ориентированы на базовый учет операций, другие удобны для планирования бюджета, но ограничены в архитектурной гибкости, интеграции или масштабировании. Для практического применения в профессиональной среде требуется система, которая сочетает удобный интерфейс, расширяемую серверную часть и устойчивую обработку финансовых данных.

Объектом исследования являются процессы управления личными финансами пользователя. Предметом исследования являются методы и программные средства автоматизации учета операций, бюджетирования, финансовой аналитики, формирования целей в веб-платформе.

Целью выпускной квалификационной работы является разработка платформы, которая способна обеспечить учет личных финансов, планирование бюджета, контроль финансовых целей и аналитическую поддержку принятия решений.

Для достижения цели были поставлены следующие задачи:

- исследовать предметную область управления личными финансами и определить ключевые пользовательские сценарии;
- выполнить сравнительный анализ существующих программных продуктов и выделить их ограничения;

- сформировать функциональные и нефункциональные требования к разрабатываемой платформе;
- реализовать программное решение на основе микросервисной архитектуры и оценить качество результата.

Решение поставленных задач позволит создать современную и удобную платформу для управления финансами и автоматизации финансовых операций.

1 Анализ предметной области и выявление требований на разработку платформы для управления личными финансами и планирования бюджета

1.1 Исследование предметной области и анализ процессов управления финансами

Перед разработкой платформы необходимо произвести формализацию задач предметной области, а также анализ существующих аналогов, чтобы определить требования к системе.

Управление личными финансами включает несколько взаимосвязанных процессов: учет доходов и расходов, распределение затрат по категориям, контроль лимитов бюджета, формирование накоплений и анализ динамики финансового состояния. Ключевая проблема пользователя заключается не в отсутствии данных, а в их разрозненности и сложности интерпретации.

Для пользователя финансовых платформ являются приоритетными следующие возможности системы:

- наблюдать полную и понятную картину личных финансов в текущий момент;
- быстро определять категории, в которых расходы превышают ожидаемый уровень;
- контролировать выполнение месячного бюджета и лимитов;
- отслеживать продвижение к финансовым целям (накопления, снижение долга);
- получать наглядные отчеты и прогнозы для планирования дальнейших расходов.

Выделены основные информационные сущности системы: пользователь, кошелек, категория, транзакция, бюджет, лимит категории, финансовая цель, аналитика в графическом представлении. Данные сущности формируют базовый контур системы и отражают фактические действия пользователя: от фиксации операции до контроля достижения финансовой цели.

Ключевое значение имеют правила валидации финансовых операций. Так, для операций типа «расход» и «перевод» необходим контроль достаточности баланса, а для перевода между кошельками необходимо обязательно указывать оба существующих кошелька, тем самым исключив перевод на исходящий кошелек. Таким образом, система должна обеспечить не только хранение данных, но и строгое соблюдение прикладных ограничений.

Таким образом, при разработке платформы важно обеспечить несколько ключевых условий: корректную обработку транзакций, удобный ввод данных и быстрое получение аналитики. Именно эти требования определяют архитектурные решения и состав функциональных возможностей системы.

1.2 Сравнительный анализ приложений для управления личными финансами и планирования бюджета

Для выявления сильных и слабых сторон существующих решений были рассмотрены следующие продукты: «КМуMoney», «Firefly III», «Goodbudget» и «CoinKeeper». Сравнение проводилось по критериям, которые влияют на практическую применимость системы, а именно: архитектурная масштабируемость, кроссплатформенность, синхронизация данных, интеграция с банками, поддержка бюджетирования и аналитики, а также удобство пользования интерфейсом системы.

Приложение «КМуMoney» представляет собой десктопное приложение, которое реализовано на языке C++. Архитектура приложения представлена единой кодовой базой, где компоненты тесно связаны между собой, что существенно затрудняет масштабирование приложения и интеграцию с внешними сервисами [1]. При таком подходе в случае ухудшении производительности необходимо масштабировать приложение целиком, что характерно неэффективным использованием вычислительных ресурсов. Интерфейс приложения «КМуMoney» представлен на рисунке 1.

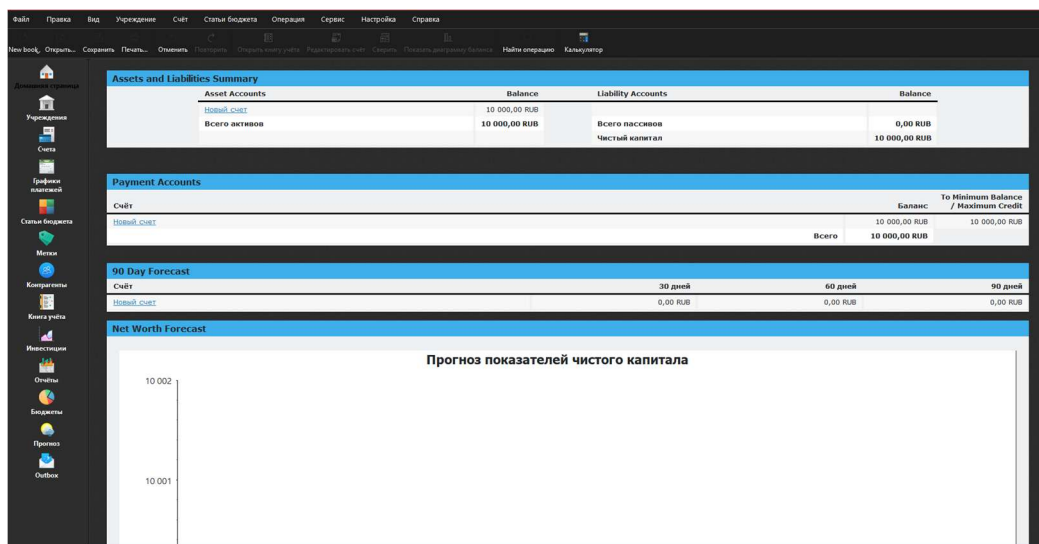


Рисунок 1 – Интерфейс приложения «КМуMoney»

Преимуществом приложения «КМуMoney» является стабильная базовая функциональность для персонального учета, а также независимость от облачной инфраструктуры при локальном хранении данных. Недостатки связаны с архитектурой и UX: приложение ориентировано на использование в качестве десктопного, где интерфейс не соответствует современным стандартам, а также требует времени на освоение [2].

«Firefly III» представляет собой веб-приложение, которое реализовано с помощью языковых средств PHP, в архитектуре используется паттерн MVC. Интерфейс приложения «Firefly III» представлен на рисунке 2.

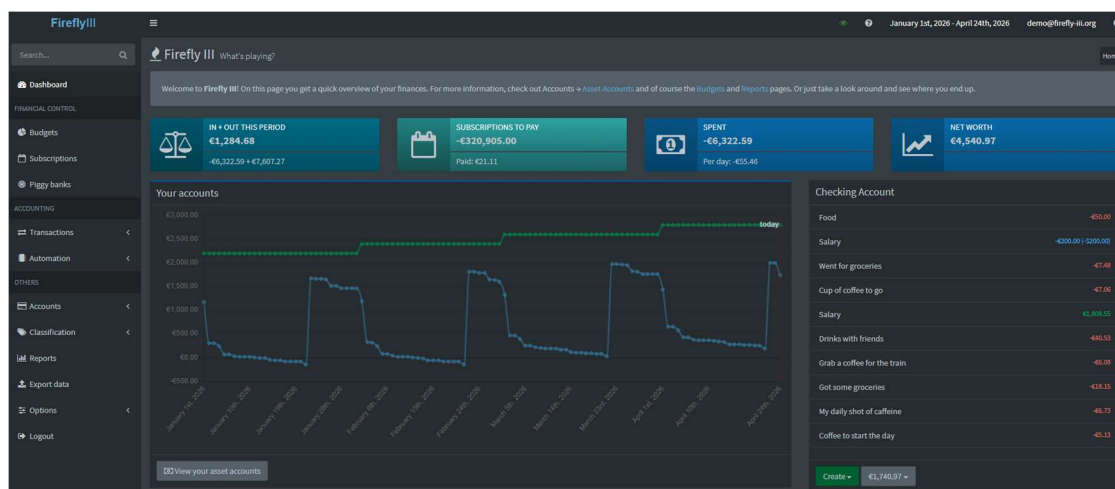


Рисунок 2 – Интерфейс приложения Firefly III

Приложение имеет открытый исходный код и гибкую настройку категорий и счетов. Однако сравнительно высокий порог входа для неподготовленного пользователя является недостатком данной платформы. Монолитная архитектура приложения усложняет независимое масштабирование отдельных подсистем при росте нагрузки.

«YNAB (You Need A Budget)» представляет собой онлайн-сервис, где ключевая идея заключается в планировании денег по категориям до нуля. Интерфейс приложения «YNAB (You Need A Budget)» представлен на рисунке 3.

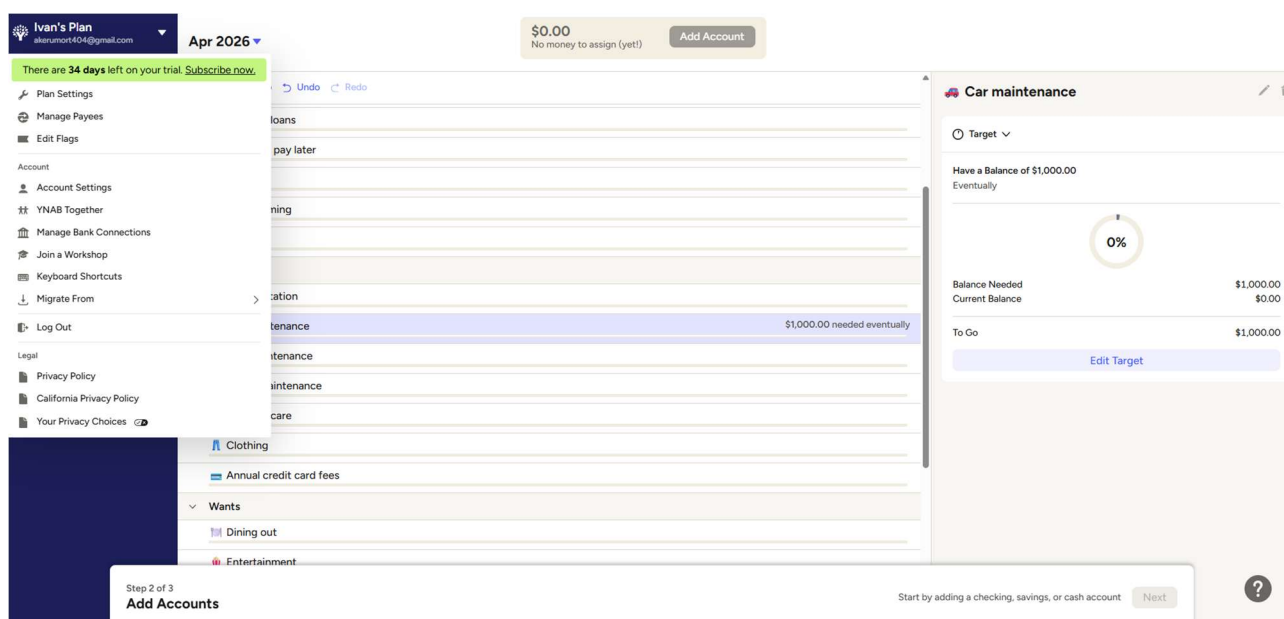


Рисунок 3 – Интерфейс приложения «YNAB (You Need A Budget)»

Среди преимуществ данного сервиса можно выделить следующие функции: автоподгрузка банковских операций и синхронизация между устройствами, цели с визуальным прогрессом, планировщик для погашения долга.

Из недостатков стоит выделить отсутствие расширенной аналитики на уровне отдельного сервиса, а также отсутствие выделенного встроенного аудит-журнала изменений транзакций. Также платформа «YNAB (You Need A Budget)» предоставляется на платной основе и на английском языке, что существенно усложняет процесс свободного планирования личных финансов.

«Goodbudget» представляет собой веб-сервис, в основе которого лежит концепция «конвертного бюджетирования», которая подходит для ежедневного контроля расходов и семейного использования. Интерфейс «Goodbudget» представлен на рисунке 4.

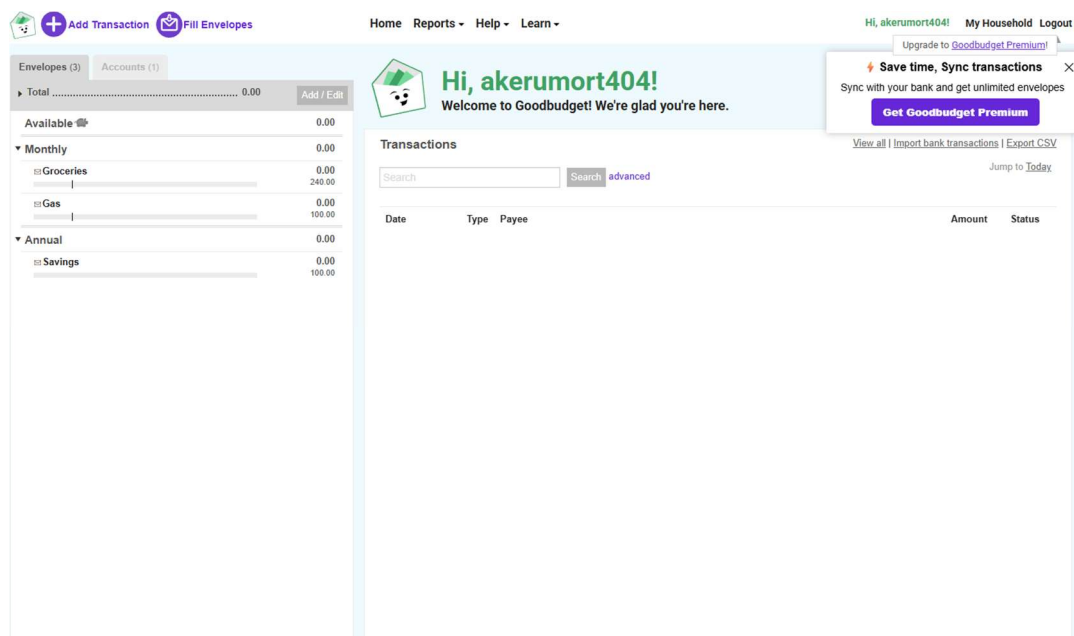


Рисунок 4 – Интерфейс приложения «Goodbudget»

Основным преимуществом сервиса «Goodbudget» является интуитивно понятный веб-интерфейс, а также гибкая настройка на старте использования. Ограничения проявляются в глубине аналитики и расширяемости: часть функций доступна только в платной модели, а архитектурные детали системы публично раскрыты достаточно ограниченно, из-за чего сложно оценить потенциал интеграций и масштабирования.

Мобильное приложение «CoinKeeper» поддерживает совместное ведение бюджета, работу с несколькими валютами, установку лимитов по категориям. Интерфейс главного меню приложения на iOS представлен на рисунке 5.

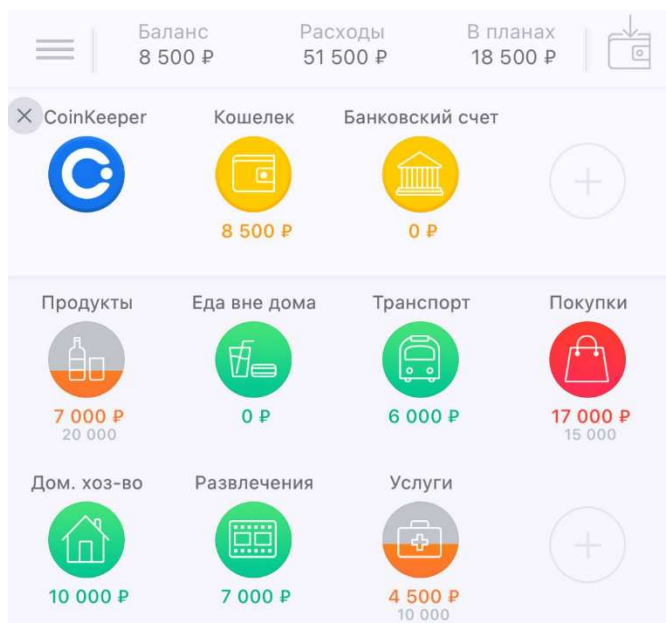


Рисунок 5 – Интерфейс главного меню приложения «CoinKeeper»

Также в приложении реализована наглядная визуализация расходов, что делает приложение удобным для повседневного финансового контроля. Аналитика доходов и расходов представлена на рисунке 6.

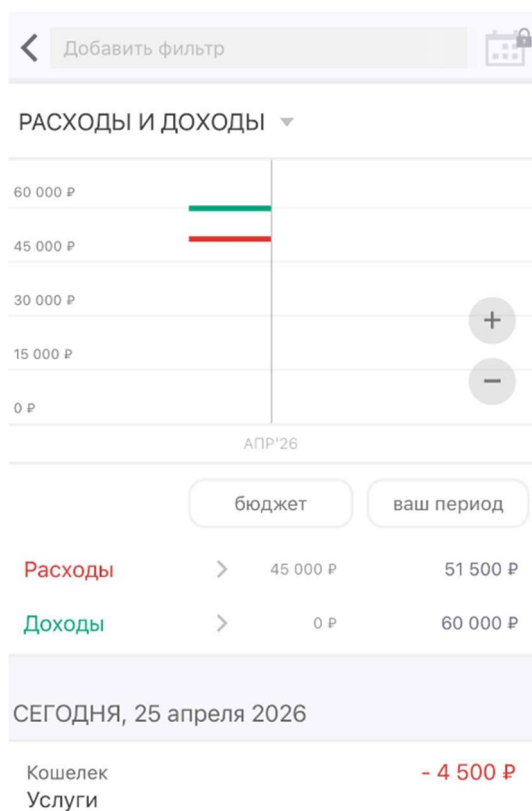


Рисунок 6 – Интерфейс окна с аналитикой приложения «CoinKeeper»

Основным недостатком приложения «CoinKeeper» является небольшая перегруженность интерфейса с навязчивыми промо-баннерами. Также некоторая часть полезных функций вынесена в платную подписку. Архитектурное решение является закрытым.

Для сравнения и итоговой оценки рассмотренных продуктов была использована следующая шкала критериев с оценкой:

«да» – функционал реализован;

«нет» – функционал отсутствует или не реализован в базовой версии.

Результаты сравнения приложений для управления личными финансами и планирования бюджета приведены в таблице 1.

Таблица 1 – Сравнительный анализ продуктов-аналогов

Критерий	KMyMoney	Firefly III	YNAB	Goodbudget	CoinKeeper
Кроссплатформенность	Да	Да	Да	Да	Да
Синхронизация между устройствами	Нет	Да	Да	Да	Да
Поддержка бюджетирования	Да	Да	Да	Да	Да
Аналитика и визуализация	Да	Да	Да	Да	Да
Архитектурная масштабируемость, расширяемость	Да	Да	Нет	Нет	Нет
Удобство использования	Нет	Нет	Да	Да	Да

Проведенный анализ показывает, что даже функционально развитые продукты, как правило, имеют зависимость от внешней облачной инфраструктуры и платных тарифов, а также неполную прозрачность внутренних механизмов или перегруженность интерфейса. Также, несмотря на наличие аналитики и визуализации, глубина их реализации в рассматриваемых продуктах сильно различается.

На основании сравнения сформирован вывод о целесообразности разработки собственной платформы для управления финансами. Проект будет ориентирован на устранение ограничений, обнаруженных у рассмотренных аналогов: недостаточной архитектурной гибкости, зависимости от закрытых экосистем, ограниченной адаптируемости под пользовательские сценарии и неоднородности аналитических возможностей. В качестве базового подхода выбрана микросервисная архитектура, позволяющая независимо развивать доменные области (авторизация, пользователи, кошельки, транзакции, бюджеты, цели, аналитика) и масштабировать их по мере роста нагрузки.

Разрабатываемая платформа будет объединять единый веб-интерфейс повседневного учета финансов, инструменты бюджетного планирования и контроля лимитов, механизмы постановки и отслеживания финансовых целей, а также аналитические отчеты и оценки с прогнозированием на основе накопленных данных. Дополнительно планируется предусмотреть автоматизацию ввода данных за счет транзакций из PDF-чеков. Такой подход формирует единый инструмент, который не только фиксирует финансовые операции, но и поддерживает пользователя в принятии обоснованных решений, повышая прозрачность личного бюджета и удобство ежедневного финансового управления.

1.3 Анализ функциональных требований

На основе анализа предметной области и анализа существующих решений были определены следующие функциональные требования:

- регистрация, аутентификация и выход пользователя из системы;
- поддержка access/refresh-сессий и управление ролями доступа;
- просмотр и редактирование профиля пользователя;
- создание, изменение, деактивация и просмотр кошельков;
- ведение пользовательских и системных категорий операций;
- выполнение базовых CRUD-операций (просмотр, создание, изменение, удаление) над транзакциями;

- поддержка типов транзакций: доход, расход, перевод между кошельками;
- проверка бизнес-ограничений транзакций (валидность полей, контроль баланса, корректность реквизитов перевода);
- хранение и просмотр истории изменений транзакций;
- проверка бизнес-ограничений транзакций (валидность полей, контроль баланса, корректность реквизитов перевода);
- хранение и просмотр истории изменений транзакций;
- формирование бюджетов по периоду и кошельку;
- установка лимитов по категориям в рамках бюджета;
- автоматический пересчет освоения бюджета по фактическим операциям;
- создание и сопровождение финансовых целей;
- расчет прогресса и статуса цели на основе фактических данных;
- получение сводной аналитики (доходы, расходы, баланс);
- формирование отчетов по динамике, долям категорий и трендам;
- получение прогнозной аналитики по историческим данным;
- импорт транзакционных данных из PDF-чеков;
- фильтрация, поиск и постраничный вывод списков данных;
- обеспечение изоляции данных разных пользователей.

Диаграмма вариантов использования в UML – это графическая модель, которая показывает, какие функции предоставляет система и как пользователь взаимодействует с приложением [3]. Такая диаграмма позволяет определить границы системы, роли и ключевые сценарии использования.

На рисунке 7 представлена диаграмма вариантов использования на основе выявленных требований.

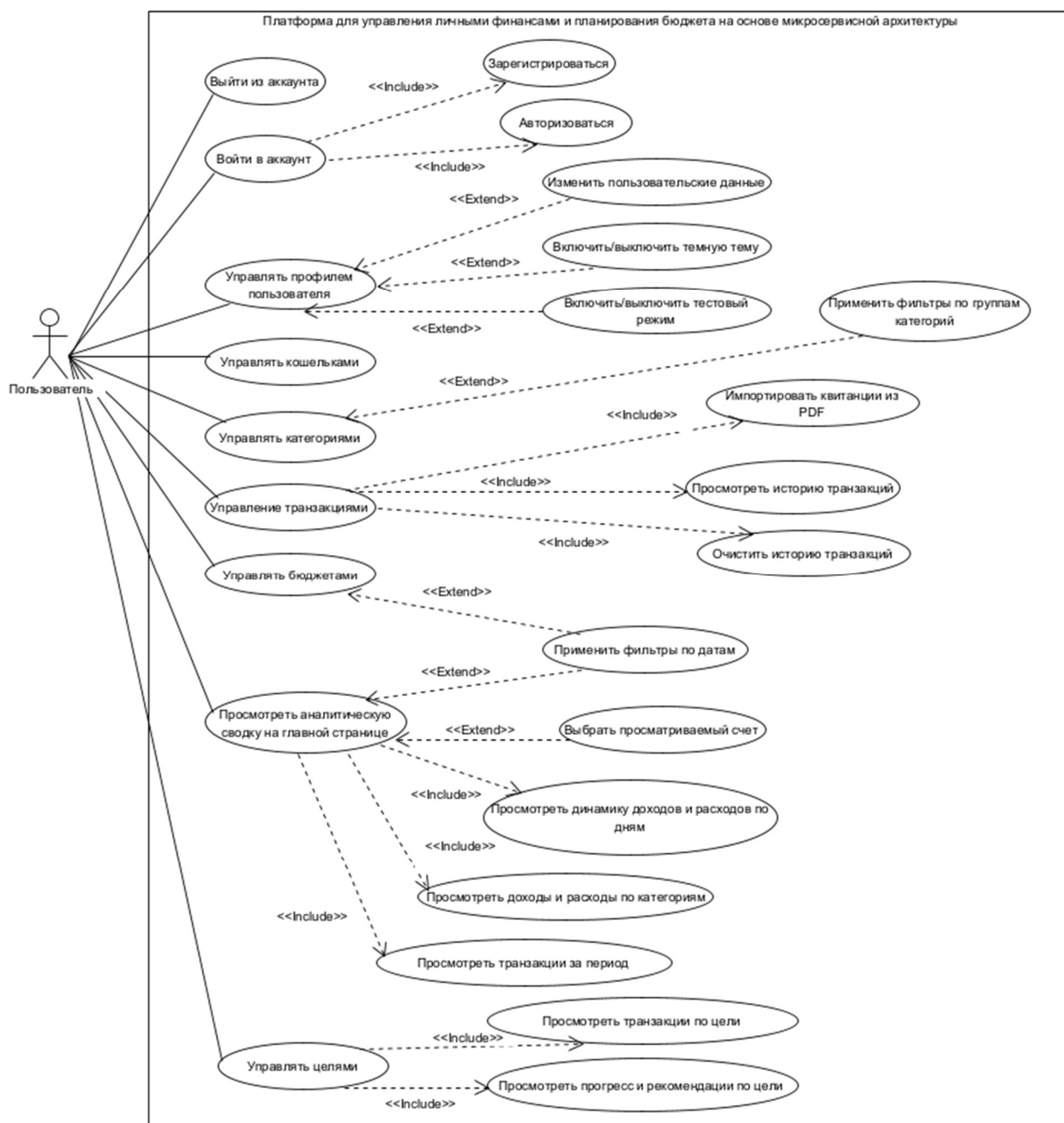


Рисунок 7 – Диаграмма вариантов использования

Диаграмма позволяет формализовать функциональные требования, выделить ключевые пользовательские сценарии и проверить полноту реализуемого функционала. Среди основных вариантов использования системы необходимо выделить сценарии создания транзакции, формирования бюджета с лимитами, создание финансовой цели с отслеживанием прогресса. Сценарий варианта использования «Создать транзакцию» представлен на рисунке 8.

Препедент:
Создать транзакцию
ID: 1
Краткое описание: Пользователь добавляет финансовую операцию (доход, расход или перевод между кошельками)
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: 1. Пользователь авторизован в системе 2. В системе создан хотя бы один активный кошелек 3. Для операции перевода доступны два разных кошелька
Основной поток: 1. Пользователь открывает форму создания транзакции 2. Пользователь выбирает тип операции (INCOME, EXPENSE, TRANSFER) 3. Пользователь заполняет обязательные поля (сумма, кошелек, дата, категория или описание) 4. Для перевода пользователь выбирает кошелек, на который будет осуществлен перевод 5. Система проверяет корректность данных и бизнес-ограничения 6. Система сохраняет транзакцию 7. Система обновляет связанные финансовые показатели
Постусловие: 1. Транзакция сохранена и доступна в списке операций 2. Актуальные балансы и агрегированные данные обновлены

Рисунок 8 – Сценарий варианта использования «Создать транзакцию»

Сценарий варианта использования «Сформировать бюджет с лимитами» представлен на рисунке 9.

Препедент:
Сформировать бюджет с лимитами
ID: 2
Краткое описание: Пользователь создает бюджет на выбранный период и задает лимиты по категориям расходов
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: 1. Пользователь авторизован в системе 2. В системе созданы кошельки и категории расходов
Основной поток: 1. Пользователь переходит в раздел бюджетов 2. Пользователь задает период бюджета и выбирает кошелек 3. Пользователь добавляет категории и устанавливает лимиты 4. Система проверяет корректность параметров бюджета 5. Система сохраняет бюджет и лимиты 6. Система выполняет расчет текущего использования бюджета по фактическим операциям 7. Пользователь просматривает результат в интерфейсе
Постусловие: 1. Бюджет и лимиты сохранены в системе 2. Пользователю доступен контроль исполнения бюджета по категориям

Рисунок 9 – Сценарий варианта использования «Сформировать бюджет с лимитами»

Сценарий варианта использования «Создать финансовую цель и отслеживать прогресс» представлен на рисунке 10.

Прецедент: Сформировать финансовую цель и отслеживать прогресс
ID: 3
Краткое описание: Пользователь задает финансовую цель и контролирует степень её достижения
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: 1. Пользователь авторизован в системе 2. В системе созданы необходимые кошельки и категории (при необходимости)
Основной поток: 1. Пользователь переходит в раздел финансовых целей 2. Пользователь вводит параметры цели (название, сумма, срок, тип цели, привязка к кошельку или категории) 3. Система выполняет проверку введенных данных 4. Система сохраняет цель 5. Система рассчитывает текущий прогресс на основе финансовых операций 6. Пользователь просматривает статус цели в интерфейсе
Постусловие: 1. Финансовая цель сохранена 2. Прогресс и статус цели доступны для последующего контроля

Рисунок 10 – Сценарий варианта использования «Сформировать финансовую цель и отслеживать прогресс»

Проведенный анализ функциональных требований позволил определить ключевые варианты использования платформы для управления финансами и планирования бюджета и формализовать основные пользовательские сценарии взаимодействия с системой. Описанные прецеденты отражают базовые бизнес-процессы приложения и показывают, каким образом пользователь выполняет учет операций, планирование бюджета и контроль финансовых целей. Это подтверждает полноту функционального контура разрабатываемой платформы и служит основой для последующего проектирования программных компонентов.

1.4 Анализ нефункциональных требований

Нефункциональные требования определяют качество работы системы и условия ее практической эксплуатации.

Были сформированы следующие нефункциональные требования:

– безопасность: защита API через JWT, разграничение доступа по ролям, контроль активных сессий, отсутствие доступа к данным других пользователей;

- производительность;
- надежность;
- масштабируемость: возможность независимого горизонтального масштабирования доменных сервисов;
- согласованность данных: соблюдение ограничений на уровне базы данных и сервисной логики, а также корректная межсервисная синхронизация через события в Kafka;
- сопровождаемость: разделение ответственности между сервисами;
- разворачиваемость: контейнеризация и запуск через Docker Compose;
- удобство использования: минимизация числа шагов в ключевых пользовательских сценариях (ввод транзакции, проверка бюджета, контроль целей);
- тестируемость и контролируемость качества: поддержка модульных и функциональных тестов, применение метрик кода для количественной оценки качества реализации.

Разработка платформы для управления личными финансами и планирования бюджета должна быть реализована с применением микросервисного подхода на языке Java (версия 17) с использованием фреймворка Spring Boot (Spring Web, Spring Security, Spring Data JPA). Пользовательская часть должна реализовываться как веб-приложение на React с использованием TypeScript. Для сервиса распознавания чеков должен быть написан отдельный сервис на языке Python с помощью технологии FastAPI. В качестве СУБД должна выступать PostgreSQL. Управление структурой данных и версиями схемы должно осуществляться с помощью Flyway. Межсервисное взаимодействие должно быть организовано через REST и Apache Kafka, кэширование и управление сессиями необходимо обеспечить с помощью Redis. Маршрутизация внешних запросов должна обеспечиваться через Nginx.

Система должна разворачиваться в контейнерной среде Docker и корректно исполняться на ОС семейства Windows (10 и выше) и Linux при

наличии Docker Runtime. Клиентская часть должна быть оптимизирована для современных браузеров: Google Chrome, Mozilla Firefox, Microsoft Edge.

1.5 Планирование разработки и оценки бюджета

Для реализации платформы управления личными финансами и планирования бюджета необходимо определить состав работ, их последовательность, а также установить сроки и рассчитать финансовые затраты проекта.

В качестве инструмента планирования использована среда Microsoft Office Project 2007, в которой сформирована структура проекта с основными этапами выполнения проекта. Сведения по основным этапам проекта представлено на рисунке 11.

Название задачи	Длительность	Начало
Начало проекта	0 дней	Пн 09.02.26
▢ Анализ предметной области и выявление требований на разработку платформы для управления личными финансами	5 дней	Пн 09.02.26
Анализ условий применения объектов предметной области	1 день	Пн 09.02.26
Анализ приложений-аналогов для управления финансами	1 день	Вт 10.02.26
Анализ функциональных требований	1 день	Ср 11.02.26
Анализ нефункциональных требований	1 день	Чт 12.02.26
Планирование процесса разработки приложения и оценка затрат на разработку	1 день	Пт 13.02.26
Анализ предметной области и выявление требований на разработку платформы для управления личными финансами	0 дней	Пт 13.02.26
▢ Проектирование платформы для управления личными финансами и планирования бюджета	17 дней	Пн 16.02.26
Проектирование архитектура приложения	6 дней	Пн 16.02.26
Проектирование структур данных	4 дней	Вт 24.02.26
Проектирование пользовательского интерфейса	3 дней	Пн 02.03.26
Проектирование классов программных средств	4 дней	Чт 05.03.26
Проектирование платформы для управления личными финансами и планирования бюджета завершено	0 дней	Вт 10.03.26
▢ Реализация платформы для управления личными финансами и планирования бюджета	27 дней	Ср 11.03.26
Реализация базы данных	2 дней	Ср 11.03.26
Реализация основного приложения	18 дней	Пт 13.03.26
Реализация интерфейса приложения	7 дней	Ср 08.04.26
Реализация платформы для управления личными финансами и планирования бюджета завершена	0 дней	Чт 16.04.26
▢ Контроль качества программного обеспечения	10 дней	Пт 17.04.26
Выбор режима тестирования и метрик кода	3 дней	Пт 17.04.26
Тестирование пользовательской части приложения для управления личными финансами и планирования бюджета	2 дней	Ср 22.04.26
Тестирование серверной части приложения для управления личными финансами и планирования бюджета	3 дней	Пт 24.04.26
Оценка качества программного кода	2 дней	Ср 29.04.26
Контроль качества программного обеспечения завершен	0 дней	Чт 30.04.26
▢ Оформление пояснительной записки	10 дней	Пт 01.05.26
Оформление отчета	10 дней	Пт 01.05.26
Оформление пояснительной записки завершено	0 дней	Чт 14.05.26
Конец проекта	0 дней	Чт 14.05.26

Рисунок 11 – Сведения по основным этапам проекта

Для наглядного представления сроков выполнения работ использован календарный план в виде диаграмма Ганта (рисунок 12).



Рисунок 12 – Диаграмма Ганта

Диаграмма отражает последовательность задач, их длительность и взаимозависимости, а также загрузку трудовых ресурсов по этапам разработки.

Для расчета бюджета проекта трудовым ресурсам были назначены тарифные ставки (рисунок 13).

Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных
Аналитик	Трудовой		А		100%	300,00 Р/ч	350,00 Р/ч
Разработчик ПО	Трудовой		Р		100%	500,00 Р/ч	600,00 Р/ч
Разработчик базы данных	Трудовой		РБД		100%	300,00 Р/ч	350,00 Р/ч
Дизайнер	Трудовой		Д		100%	250,00 Р/ч	300,00 Р/ч
Тестировщик ПО	Трудовой		Т		100%	350,00 Р/ч	400,00 Р/ч
Компьютер	Материальный		К			85 000,00 Р	
Электричество	Затраты		Э				
Интернет	Затраты		И				

Рисунок 13 – Ресурсы проекта

Также были сформированы отчеты об использовании ресурсов (рисунок В.1) и о бюджете проекта (рисунок В.2) и представлены в приложении В.

Таким образом, планирование показало, что разработка платформы для управления личными финансами и планирования бюджета требует 632 часа трудовых ресурсов при общей стоимости в 259030 рублей и календарной длительности 69 дней.

2 Проектирование платформы для управления личными финансами и планирования бюджета

2.1 Архитектура приложения

В качестве целевого архитектурного подхода была выбрана микросервисная архитектура. Микросервисная архитектура представляет собой способ построения приложения как набора небольших автономных сервисов, каждый из которых отвечает за отдельную бизнес-область, имеет четко определенные границы ответственности, собственную модель данных и независимый жизненный цикл развертывания [4]. Взаимодействие между сервисами выполняется по сетевым контрактам и через асинхронный обмен событиями.

В отличие от монолитного подхода, такой стиль проектирования позволяет развивать и масштабировать компоненты выборочно, не затрагивая всю систему целиком. Отдельные сервисы могут обновляться и сопровождаться независимо, что снижает связанность, повышает отказоустойчивость и упрощает расширение функциональности.

Для платформы управления личными финансами и планирования бюджета такой выбор обусловлен тем, что модули авторизации, транзакций, бюджетов, целей и аналитики имеют различную нагрузку при эксплуатации. Так, сервисы транзакций и кошельков наиболее нагружены с точки зрения интенсивности запросов, поскольку требуется оперативная обработка пользовательских операций и проверка бизнес-ограничений.

На рисунке 14 представлена диаграмма развертывания платформы. Данная диаграмма отражает архитектуру платформы с точки зрения эксплуатации: состав вычислительных узлов и контейнеров, размещение клиентской и серверной частей, а также маршруты взаимодействия между компонентами [5]. Также диаграмма развертывания позволяет наглядно продемонстрировать разделение ответственности между узлами и фактическую схему развертывания.

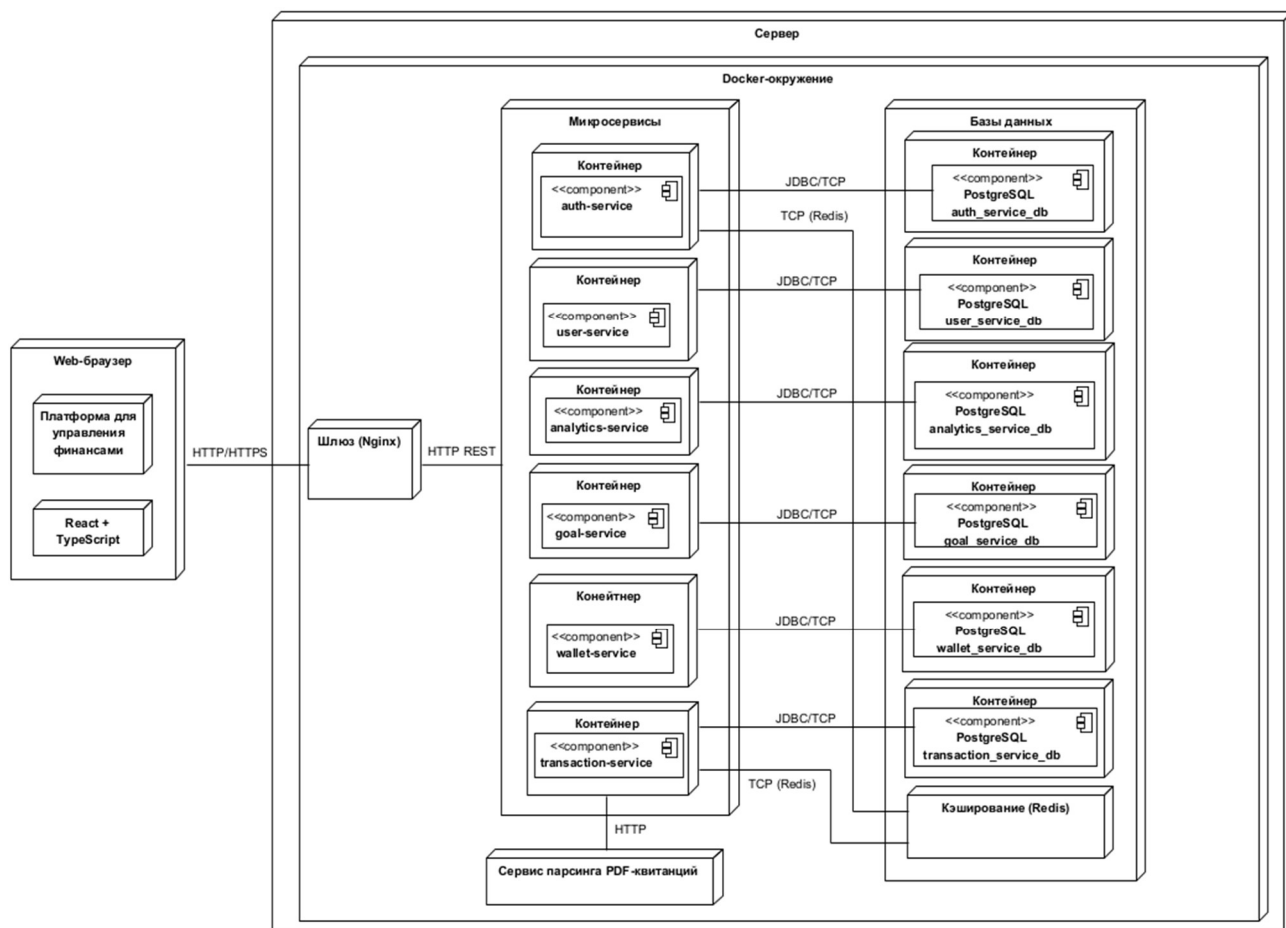


Рисунок 14 – Архитектура платформы для управления личными финансами и планирования бюджета

Развертывание архитектуры выполнено с использованием средств контейнеризации Docker, где каждый микросервис запускается в собственном Docker-контейнере и работает в общей внутренней сети докер. Такой подход позволяет обеспечить изоляцию окружения и единообразие конфигурации в разных средах. Для каждого доменного микросервиса используется отдельная база данных PostgreSQL в выделенном контейнере. Пользователь взаимодействует с системой через веб-интерфейс (React), а внешние API-запросы проходят через единый шлюз Nginx. На уровне шлюза реализуется маршрутизация запросов к доменным сервисам и централизованная проверка авторизации для защищенных маршрутов. Дополнительный сервис для парсинга PDF-квитанций используется модулем транзакций.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель данных используется для формализации структуры предметной области на раннем этапе проектирования. Такая модель отражает ключевые сущности и связи между ними [6].

Для платформы управления личными финансами и планирования бюджета в качестве базовой была выбрана концептуальная модель финансового ядра, которая включает в себя пользователей, кошельки, категории, транзакции, бюджеты и финансовые цели. Поскольку был выбран микросервисный подход к проектированию архитектуры, то согласно данному подходу каждая подсистема должна использовать собственную изолированную базу данных [7]. На этапе формулирования основных сущностей предметной области необходимо ответить, что в качестве идентификаторов сущностей используются имена соответствующих таблиц в границах конкретных сервисов, а связи между доменами носят логический (межсервисный характер). Табличное описание сущностей представлено в таблице 2.

Таблица 2 – Табличное описание основных сущностей для концептуальной модели базы данных

Название сущности	Идентификатор сущности	Назначение сущности
Пользователь	users	Хранит данные профиля пользователя системы
Учетные данные	credentials	Хранит данные аутентификации и авторизации пользователя
Кошелек	wallets	Содержит информацию о финансовых счетах пользователя (тип, валюта, баланс, статус)
Категория	categories	Используется для классификации доходов и расходов пользователя
Транзакция	transactions	Хранит факты финансовых операций: доход, расход, перевод между кошельками

Продолжение таблицы 2

Название сущности	Идентификатор сущности	Назначение сущности
История транзакций	transaction_history	Фиксирует изменения по операциям для последующего аудита
Бюджет	budgets	Содержит параметры бюджета пользователя на выбранный период
Лимит категории бюджета	budget_category_limits	Хранит ограничения расходов по категориям в рамках бюджета
Финансовая цель	goals	Содержит описание цели пользователя по накоплению или снижению расходов
Снимок прогресса цели	goal_progress_snapshots	Хранит историю изменения прогресса достижения цели

Графическое представление концептуальной модели представлено на рисунке 15.



Рисунок 15 – Диаграмма сущность-связь концептуальной модели

Концептуальная модель базы данных была построена в среде Visual Paradigm 17.0 в нотации IDEF1X.

2.2.2 Логическая модель базы данных

Логическая модель данных представляет собой развитие концептуальной модели и предназначена для формализации структуры реляционной базы данных на уровне сущностей, атрибутов, ключей, ограничений и связей [8]. На данном этапе для каждой сущности уточняются состав полей и типы данных. В отличие от концептуальной модели, которая описывает предметную область на уровне бизнес-сущностей и их взаимосвязей, логическая является более детализированной.

Физически таблицы распределены по изолированным базам данных сервисов, однако на логическом уровне образуют единую предметную модель. Для сущностей, принадлежащий разным сервисам, используются логические идентификаторы, которые обеспечивают согласованность связей между доменами.

Графическое представление логической модели приведено на рисунке 16.

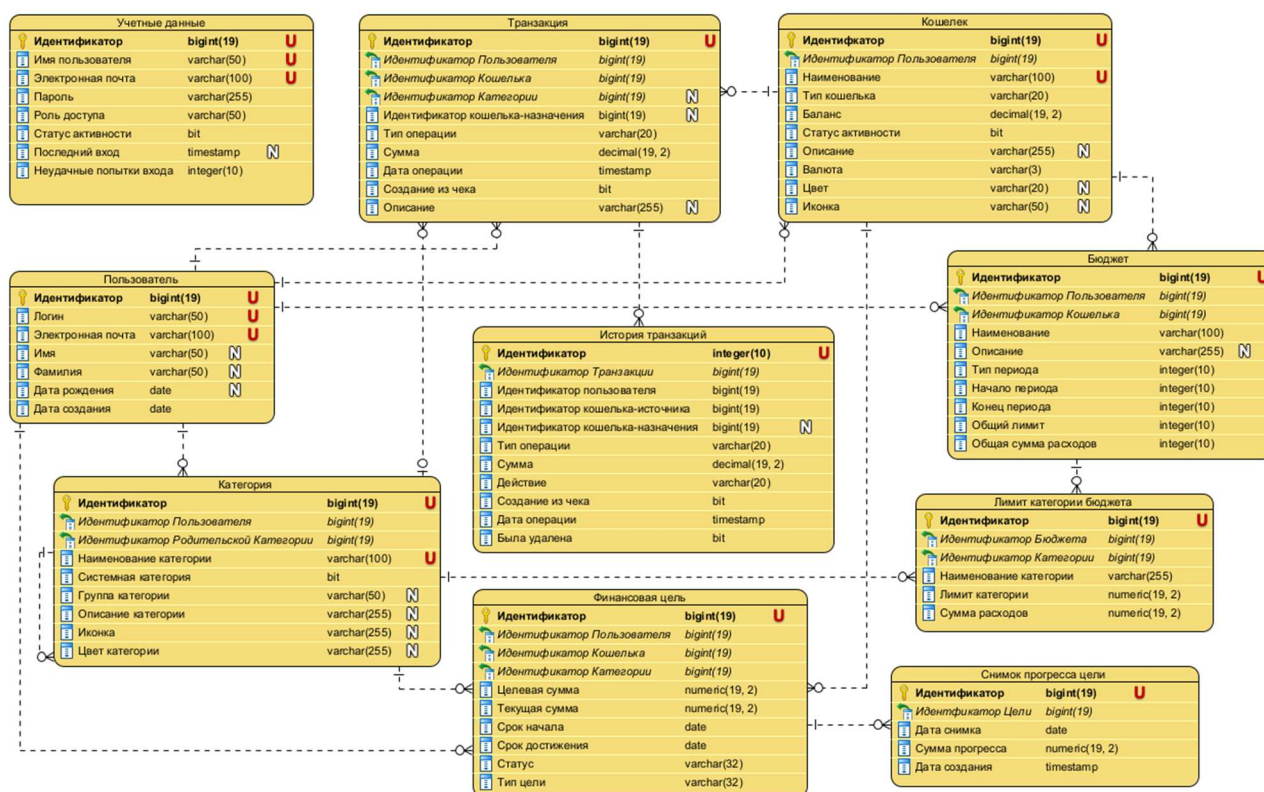


Рисунок 16 – Диаграмма сущность-связь логической модели

Таким образом, разработанная логическая модель обеспечивает наглядное представление структуры и взаимосвязей информационных сущностей системы.

2.3 Проектирование пользовательского интерфейса

При проектировании пользовательского интерфейса платформы для управления финансами необходимо учитывать функциональные требования системы и требования к удобству использования. Интерфейс должен обеспечивать быстрое и понятное выполнение ключевых пользовательских задач: учет операций, контроль бюджета, отслеживание целей и просмотр аналитики.

С учетом результатов сравнительного анализа существующих финансовых систем необходимо устранить типичные недостатки аналогов: перегруженность объектов на экране, избыточные сценарии, сложная навигация по платформе. Логика интерфейса должна строиться вокруг основных ежедневных действий пользователя, а второстепенные функции необходимо размещать без увеличения когнитивной нагрузки.

Доступ к разделам с персональными финансовыми данными доступен только для авторизованных пользователей с помощью навигационного меню по приложению в левой части экрана.

При проектировании интерфейса необходимо применить следующие решения по повышению удобства пользования:

- обеспечить визуальную иерархию экранов с выделением приоритетных данных (баланс, лимиты, прогресс целей, динамика расходов);
- сократить число шагов для базовых операций (добавление транзакции, создание бюджета, постановка цели);
- обеспечить единообразие форм, кнопок и сценариев подтверждения действий;
- реализовать предупреждение ошибок ввода;
- обеспечить обратную связь системы при успешных и ошибочных действиях;

- обеспечить адаптивность интерфейса под разные размеры экрана.

Технически пользовательский интерфейс будет проектирования как SPA-приложение на React с использованием TypeScript и компонентного подхода. SPA (Single Page Application) представляет собой архитектурный подход, при котором пользователь работает в пределах одной загруженной HTML-страницы, а смена разделов выполняется за счет динамического обновления представления без полного перезапроса документа [9]. Навигация по URL в таком приложении сохраняется, однако маршрутизация реализуется преимущественно на стороне клиента. Применение такого подхода позволит сократить количество полных перезагрузок интерфейса, а также повысить отзывчивость системы и обеспечить более целостный пользовательский сценарий при переходе между разделами «Транзакции», «Бюджеты», «Цели», «Кошельки» и «Категории».

Компонентный подход предполагает декомпозицию пользовательского интерфейса на автономные элементы (независимые модули): страницы, формы, таблицы, окна, элементы навигации и уведомления. Каждый компонент инкапсулирует собственную разметку, логику поведения и локальное состояние, а взаимодействие между компонентами выполняется через явно заданные параметры и обработчики событий. Такой подход повышает модульность интерфейса, упрощает повторное использование кода, а также локализует изменения при доработках.

2.4 Проектирование классов программных средств

В ходе проектирования серверной части приложения были разработаны диаграммы классов для каждого микросервиса в нотации UML. Диаграмма классов является структурной диаграммой, которая используется для формального представления статической модели системы: она показывает состав классов, а также связи между ними [10]. Такое представление позволяет заранее определить границы слоев приложения, проверить согласованность архитектурных решений и снизить риск ошибок на этапе реализации.

Диаграмма классов для микросервиса транзакций представлена на рисунке В.3 в приложении В. Сервис транзакций является одним из ключевых в системе, так как он наиболее полно отражает логику предметной области, а также через него проходит основной пользовательский сценарий финансового учета: создание и изменение операций, работа с категориями, переводы между кошельками, фиксация истории изменений, публикация событий.

Поскольку остальные микросервисы будут реализованы в едином архитектурном стиле (контроллеры, сервисный слой, репозитории, DTO-классы, мапперы и интеграционные компоненты), представленная диаграмма может рассматриваться как шаблон для проектирования классов других серверных модулей.

Краткое описание функционального назначения ключевых классов представлено в таблице 3.

Таблица 3 – Компоненты диаграммы классов сервиса транзакций

Компонент	Назначение
Transaction	Сущность транзакции, хранит данные о финансовой операции пользователя (доход, расход, перевод)
TransactionHistory	Сущность истории транзакций, фиксирует изменения операций (создание, редактирование, удаление) для аудита
Category	Сущность категории операций, описывает пользовательские и системные категории расходов/доходов.
CategoryRequestDto	DTO-класс входного запроса для операций создания и изменения категории, определяет допустимую структуру данных от клиента. Используется для валидации полей и изолирует API-контракт от внутренней сущности категорий.
CategoryResponseDto	DTO-класс выходного ответа с данными категории. Возвращает клиенту только необходимые атрибуты в удобном формате и исключает передачу внутренних служебных данных сущности
TransactionRequestDto	DTO-класс входного запроса для создания и изменения транзакции. Формализует состав передаваемых данных, поддерживает проверку корректности ввода и отделяет внешний контракт API от структуры сущности транзакции
TransactionResponseDto	DTO-класс выходного ответа по транзакции. Используется для передачи клиенту результата операции и данных транзакции в согласованном формате без прямой выдачи внутренней модели хранения
CategoryMapper	Класс преобразования данных между CategoryRequestDto, CategoryResponseDto и сущностью категории

Продолжение таблицы 3

Компонент	Назначение
TransactionMapper	Класс преобразования данных между TransactionRequestDto, TransactionResponseDto и сущностью транзакции
CategoryEventProducer	Компонент публикации событий категорий в Kafka для межсервисного обмена
TransactionEventProducer	Компонент публикации событий в Kafka для синхронизации с другими сервисами
TransactionRepository	Репозиторий доступа к данным транзакций, реализует базовые операции (просмотр, создание, изменение, удаление) и выборки по фильтрам
TransactionHistoryRepository	Репозиторий доступа к данным истории транзакций, обеспечивает хранение и получение журнала изменений
CategoryRepository	Репозиторий доступа к данным категорий, поддерживает операции поиска, проверки и управления иерархией категорий
CategoryEvent	Класс события категории, передаваемый через Kafka при изменениях в справочнике категорий
TransactionEvent	Класс события транзакции, передаваемый через Kafka при изменениях финансовых операций
ReceiptParseService	Сервис обработки PDF-чеков, извлекает данные операции и подготавливает их для создания транзакции
CategoryService	Сервисный интерфейс бизнес-логики управления категориями (создание, изменение, удаление, получение, статистика)
TransactionService	Сервисный интерфейс бизнес-логики управления транзакциями и их историей
CategoryController	REST-контроллер категорий, обрабатывает HTTP-запросы клиента и вызывает CategoryService
TransactionController	REST-контроллер транзакций, обрабатывает HTTP-запросы клиента, включая импорт данных из чеков и вызывает TransactionService

Спроектированная диаграмма классов наглядно отражает структуру микросервиса транзакций, состав его основных компонентов и характер их взаимодействия.

3 Реализация платформы для управления личными финансами и планирования бюджета

3.1 Реализация базы данных

Физическая ER-диаграмма представляет собой модель реляционной базы данных, которая приближена к реальной структуре хранения данных в выбранной СУБД. На данном уровне абстрактные сущности предметной области преобразуются в таблицы, их атрибуты получают конкретные системные наименования, а для каждого поля определяется подходящий тип данных [11].

Базы данных платформы для управления личными финансами и планирования бюджета были реализованы на основе PostgreSQL. Был также реализован принцип «Database per Service»: каждый микросервис работает со своей изолированной базой данных PostgreSQL [12]. Такой подход обеспечивает независимость схем данных и уменьшает риск каскадных изменений при развитии отдельных подсистем.

Также при таком подходе внешние связи между базами данных разных сервисов физически могут не оформляться с помощью внешних ключей, так как согласованность полей обеспечивается на уровне бизнес-логики. Связность между доменами поддерживается через согласованные интерфейсы и обмен событиями между сервисами.

Для каждой сущности были добавлены необязательные поля с датой создания и датой изменения в базе данных (`created_at`, `updated_at`). Данные поля предназначены для удобного отслеживания изменений в базе данных, что значительно упрощает фильтрацию и обнаружение возможных проблем при дальнейшей эксплуатации. Использование временных меток также повышает прозрачность аудита данных и облегчает анализ изменений записей.

С помощью среды построения диаграмм Visual Paradigm 18.0 была построена физическая модель базы данных в нотации IDEF1X, которая представлена на рисунке 17.

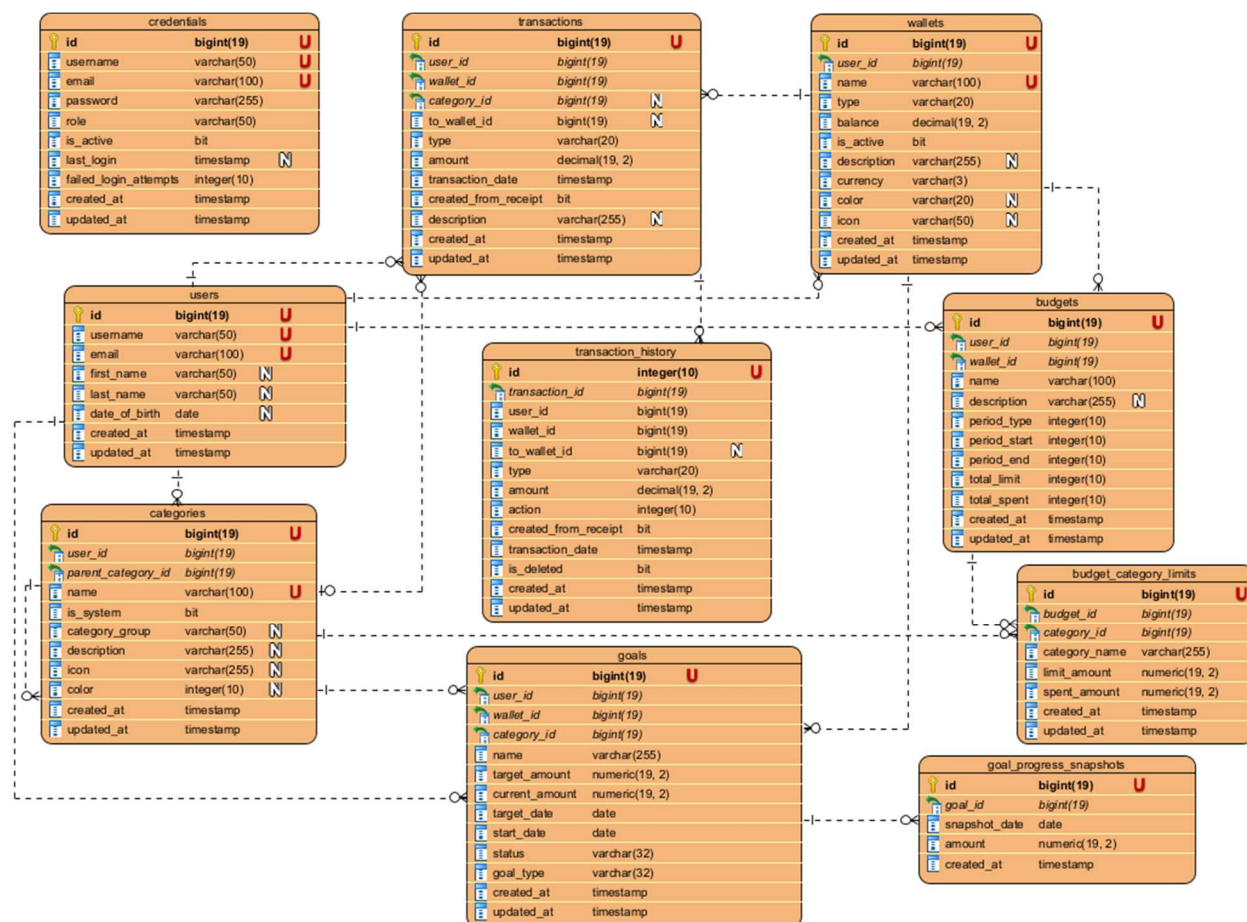


Рисунок 17 – Диаграмма сущность-связь физической модели базы данных

Описание атрибутов сущности «Учетные данные» приведено в таблице 4.

Таблица 4 – Сущность «Учетные данные»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор учетной записи	id	bigserial	PK	19	Да
Имя пользователя	username	varchar	-	50	Да
Электронная почта	email	varchar	-	100	Да
Хэш пароля	password	varchar	-	255	Да
Роль пользователя	role	varchar	-	50	Да
Признак активности учетной записи	is_active	boolean	-	-	Да
Количество неудачных попыток входа	failed_login_attempts	int	-	10	Нет

Продолжение таблицы 4

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Время последнего входа	last_login	timestamp	-	-	Нет
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Пользователи» приведено в таблице 5.

Таблица 5 – Сущность «Пользователи»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор пользователя	id	bigint	PK	19	Да
Логин пользователя	username	varchar	-	50	Да
Электронная почта	email	varchar	-	100	Да
Имя	first_name	varchar	-	50	Нет
Фамилия	last_name	varchar	-	50	Нет
Дата рождения	date_of_birth	date	-	-	Нет
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Категории» приведено в таблице 6.

Таблица 6 – Сущность «Категории»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор категории	id	bigserial	PK	19	Да
Идентификатор пользователя	user_id	bigint	FK	19	Да
Родительская категория	parent_category_id	bigint	FK	19	Нет
Наименование категории	name	varchar	-	100	Да
Описание категории	description	varchar	-	500	Нет

Продолжение таблицы 6

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Иконка категории	icon	varchar	-	50	Нет
Цвет категории	color	varchar	-	20	Нет
Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Группа категории	category_group	varchar	-	50	Нет
Признак системной категории	is_system	boolean	-	-	Да
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Кошельки» приведено в таблице 7.

Таблица 7 – Сущность «Кошельки»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор кошелька	id	bigserial	PK	19	Да
Идентификатор пользователя	user_id	bigint	FK	19	Да
Наименование кошелька	name	varchar	-	100	Да
Описание кошелька	description	varchar	-	500	Нет
Тип кошелька	type	varchar	-	20	Да
Валюта	currency	varchar	-	3	Да
Баланс	balance	decimal	-	19,2	Да
Признак активности	is_active	boolean	-	-	Да
Цвет	color	varchar	-	20	Нет
Иконка	icon	varchar	-	50	Нет
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Транзакции» приведено в таблице 8.

Таблица 8 – Сущность «Транзакции»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор транзакции	id	bigserial	PK	19	Да
Идентификатор пользователя	user_id	bigint	FK	19	Да
Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор кошелька-источника	wallet_id	bigint	FK	19	Да
Идентификатор кошелька-получателя	to_wallet_id	bigint	FK	19	Нет
Идентификатор категории	category_id	bigint	FK	19	Нет
Тип категории	type	varchar	-	20	Да
Сумма операции	amount	decimal	-	19,2	Да
Дата операции	transaction_date	timestamp	-	-	Да
Признак создания из PDF	created_from_receipt	boolean	-	-	Да
Описание операции	description	varchar	-	200	Нет
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «История транзакций» приведено в таблице 9.

Таблица 9 – Сущность «История транзакций»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор записи истории	id	bigserial	PK	19	Да
Идентификатор исходной транзакции	transaction_id	bigint	FK	19	Да
Идентификатор пользователя	user_id	bigint	FK	19	Да
Идентификатор кошелька-источника	wallet_id	bigint	FK	19	Да

Продолжение таблицы 9

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор кошелька-получателя	to_wallet_id	bigint	FK	19	Нет
Тип категории	type	varchar	-	20	Да
Сумма операции	amount	decimal	-	19,2	Да
Дата операции	transaction_date	timestamp	-	-	Да
Тип действия	action	varchar	-	20	Да
Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Признак удаления	is_deleted	boolean	-	-	Да
Признак создания из PDF	created_from_receipt	boolean	-	-	Да
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Бюджеты» приведено в таблице 10.

Таблица 10 – Сущность «Бюджеты»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор бюджета	id	bigserial	PK	19	Да
Идентификатор пользователя	user_id	bigint	FK	19	Да
Идентификатор кошелька	wallet_id	bigint	FK	19	Да
Наименование бюджета	name	varchar	-	255	Да
Описание бюджета	description	text	-	-	Нет
Тип периода	period_type	varchar	-	32	Да
Дата начала периода	period_start	date	-	-	Да
Дата окончания периода	period_end	date	-	-	Да
Лимит бюджета	total_limit	numeric	-	19,2	Да
Фактические расходы	total_spent	numeric	-	19,2	Да

Продолжение таблицы 10

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Лимиты категории бюджетов» приведено в таблице 11.

Таблица 11 – Сущность «Лимиты категории бюджетов»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор лимита	id	bigserial	PK	19	Да
Идентификатор бюджета	budget_id	bigint	FK	19	Да
Идентификатор категории	category_id	bigint	FK	19	Нет
Наименование категории	category_name	varchar	-	255	Да
Лимит по категории	limit_amount	numeric	-	19,2	Да
Фактические расходы по категории	spent_amount	numeric	-	19,2	Да
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Цели» приведено в таблице 12.

Таблица 12 – Сущность «Цели»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор цели	id	bigserial	PK	19	Да
Идентификатор пользователя	user_id	bigint	FK	19	Да
Идентификатор кошелька	wallet_id	bigint	FK	19	Нет
Идентификатор категории	category_id	bigint	FK	19	Нет

Продолжение таблицы 12

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Наименование цели	name	varchar	-	255	Да
Целевая сумма	target_amount	numeric	-	19,2	Да
Текущая сумма	current_amount	numeric	-	19,2	Да
Дата начала	start_date	date	-	-	Да
Целевая дата	target_date	date	-	-	Да
Статус цели	status	varchar	-	32	Да
Тип цели	goal_type	varchar	-	32	Да
Дата и время создания записи	created_at	timestamp	-	-	Да
Дата и время изменения записи	updated_at	timestamp	-	-	Да

Описание атрибутов сущности «Снимки прогресса цели» приведено в таблице 13.

Таблица 13 – Сущность «Снимки прогресса цели»

Содержание поля	Системное наименование	Тип	Ключ	Размер	Обязательность
Идентификатор снимка прогресса	id	bigserial	PK	19	Да
Идентификатор цели	goal_id	bigint	FK	19	Да
Дата снимка	snapshot_date	date	-	-	Да
Значение накопленной суммы	amount	numeric	-	19,2	Да
Дата и время создания	created_at	timestamp	-	-	Да

Для управления изменениями схемы и разворачивания базы данных использовался механизм миграций Flyway, позволяющий последовательно применять SQL-скрипты при запуске сервисов и поддерживать одинаковую структуру базы данных в разных средах развертывания.

3.2 Реализация программных компонентов

Серверная часть платформы реализована в виде набора независимых микросервисов на базе Java 17 и Spring Boot 3 с использованием интегрированной среды разработки JetBrains IntelliJ IDEA. Для сервиса

распознавания квитанций был использован Python и FastAPI. Архитектурно приложение построено с разделением на прикладные уровни: контроллеры, слой бизнес-логики, слой интеграции с внешними компонентами (Kafka, Redis, межсервисные HTTP-клиенты). Такой подход обеспечивает модульность, а также упрощает сопровождение, снижая связность между подсистемами.

Для асинхронного межсервисного взаимодействия используется Apache Kafka, что позволяет передавать доменные события и поддерживать актуальность аналитических представлений без жесткой синхронной связности [13]. Для кэширования и управления пользовательскими сессиями применяется Redis. Изменения структуры данных управляются через миграции Flyway, применяемые при запуске соответствующего сервиса.

Оркестрация контейнеров осуществляется с помощью Docker Compose, где определены зависимости между сервисами, параметры запуска и проверки доступности [14]. Каждый микросервис разворачивается в собственном контейнере и взаимодействует с такими компонентами, как PostgreSQL, Kafka, Redis, Nginx Gateway через внутреннюю сеть.

Всего в состав серверной части вошли следующие микросервисы: auth-service, user-service, transaction-service, wallet-service, budget-service, goal-service, analytics-service. Также был разработан вспомогательный сервис pdf-parser-service для парсинга квитанций. Основные сервисы реализованы в едином архитектурном стиле с разделением на слои API, бизнес-логики и доступа к данным, что обеспечивает согласованность архитектурных подходов и упрощает сопровождение.

Функционально auth-service отвечает за аутентификацию и авторизацию (JWT, refresh/logout), user-service за управление профилем пользователя, transaction-service за счет транзакций и категорий, включая интеграцию с импортом квитанций в PDF-формате. Микросервис wallet-service отвечает за управление кошельками и балансами, budget-service за бюджеты и лимиты категорий, а goal-service отвечает за финансовые цели и контроль прогресса. Микросервис analytics-service отвечает за агрегированную аналитику по

пользовательским транзакциям, дневные отчеты, динамику доходов и расходов, прогноз на будущий период на основе усреднения имеющихся данных.

Для каждого микросервиса разработана отдельная диаграмма компонентов. Так, на рисунке 18 представлена диаграмма компонентов микросервиса auth-service. Диаграммы компонентов остальных сервисов приведены на рисунках В.4 – В.9 в приложении В.

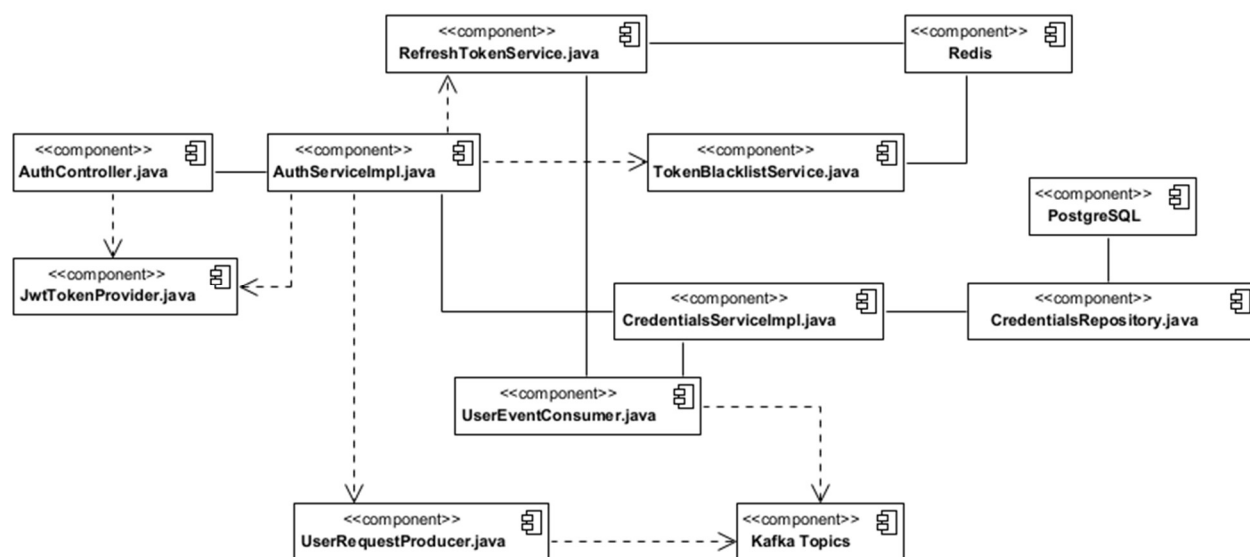


Рисунок 18 – Диаграмма компонентов микросервиса авторизации

Основная обработка запросов выполняется в AuthController, который передает операции в AuthServiceImpl, где реализована ключевая логика регистрации, входа, обновления профиля, проверки и отзыва токенов. Работа с учетными данными пользователя организована через CredentialsServiceImpl и CredentialsRepository с хранением в PostgreSQL. Для управления сессиями и отозванными токенами используется механизм кеширования Redis, что ускоряет проверку токенов и повышает управляемость контура с авторизацией [15]. Межсервисное взаимодействие реализовано в двух формах: синхронно через вызовы API сервиса для работы пользователями асинхронно через Kafka топики с использованием UserRequestProducer и UserEventConsumer. Такой набор компонентов обеспечивает целостную работу подсистемы авторизации в условиях микросервисной архитектуры и снижает связанность с другими сервисами платформы.

Аналогичный подход применен и в остальных микросервисах платформы. Сервис парсинга PDF-квитанций реализован с помощью Python/FastAPI, при этом логика сосредоточена в едином файле `app.py`. Сервис предоставляет две основные функции: разбор PDF-квитанции и проверку доступности. При обработке запроса файл читается в память, затем при помощи библиотеки `pdfplumber` извлекается текст, после чего применяются правила разбора для шаблонов квитанций популярных банков (Сбербанк, Тинькофф) и универсальный сценарий для остальных форматов. На финальном этапе выполняется нормализация суммы, даты и типа операции, и сервис возвращает результат в унифицированном JSON-формате для последующей обработки в сервисе транзакций [16].

Фрагмент серверной части платформы для управления финансами и планирования бюджета приведен в приложении Б. Полный код расположен на сервисе `git` [17].

3.3 Реализация пользовательского интерфейса

Пользовательский интерфейс платформы для управления личными финансами и планирования бюджета реализован в формате одностраничного веб-приложения на базе `React` и `TypeScript`.

Функционирование пользовательского интерфейса реализовано по последовательной схеме обработки пользовательских действий: переход к требуемому разделу, ввод и параметризация данных, передача запросов в серверные API, получение ответа и обновление представления на экране.

В результате были реализованы следующие основные страницы:

- страница входа (регистрация, авторизация);
- главная страница с аналитическими сводками;
- страница транзакций;
- страница категорий;
- страница кошельков;
- страница бюджетов;

- страница финансовых целей;
- страница профиля пользователя.

В ходе реализации были применены стандартные инструменты экосистемы React для организации маршрутизации, управления состоянием, взаимодействия с серверными API, визуализация аналитических данных и построения адаптивного интерфейса.

Работа пользователя начинается с регистрации (рисунок 19) и входа в систему, после чего становится доступен весь основной функционал приложения.

Скриншот экрана регистрации пользователя. Вверху находится иконка кошелька и заголовок «Создать аккаунт» с подзаголовком «Начните управлять своими финансами грамотно». Форма содержит следующие поля: «Имя» (подсказка 1), «Фамилия» (подсказка 2), «Имя пользователя» (подсказка 3), «Email» (подсказка 4), «Пароль» (подсказка 5) и «Подтвердите пароль» (подсказка 6). Внизу находится кнопка «Зарегистрироваться →» (подсказка 7) и ссылка «Уже есть аккаунт? Войти» (подсказка 8).

Рисунок 19 – Окно регистрации пользователя

Основные элементы окна регистрации пользователя:

- 1 – поле ввода имени
- 2 – поле ввода фамилии;
- 3 – поле ввода имени пользователя, отображаемого на платформе
- 4 – поле ввода электронной почты;

- 5 – поле ввода пароля;
- 6 – поле подтверждения пароля;
- 7 – кнопка «Зарегистрироваться»;
- 8 – ссылка на страницу входа в систему.

После успешной регистрации в системе пользователь имеет возможность перейти в профиль (рисунок 20), где доступен просмотр и корректировка данных (при необходимости), а также некоторые дополнительные настройки приложения (переключение темной/светлой темы, переключение тестового режима). С помощью навигационной панели в левой части экрана возможно осуществить переход на любую из следующих страниц: «Главная», «Транзакции», «Категории», «Кошельки», «Бюджеты», «Цели». Также пользователю всегда доступен выход из системы с помощью соответствующей кнопки «Выйти». Синим акцентирующим цветом на боковой панели навигации отмечена выбранная страница, на которой находится пользователь.

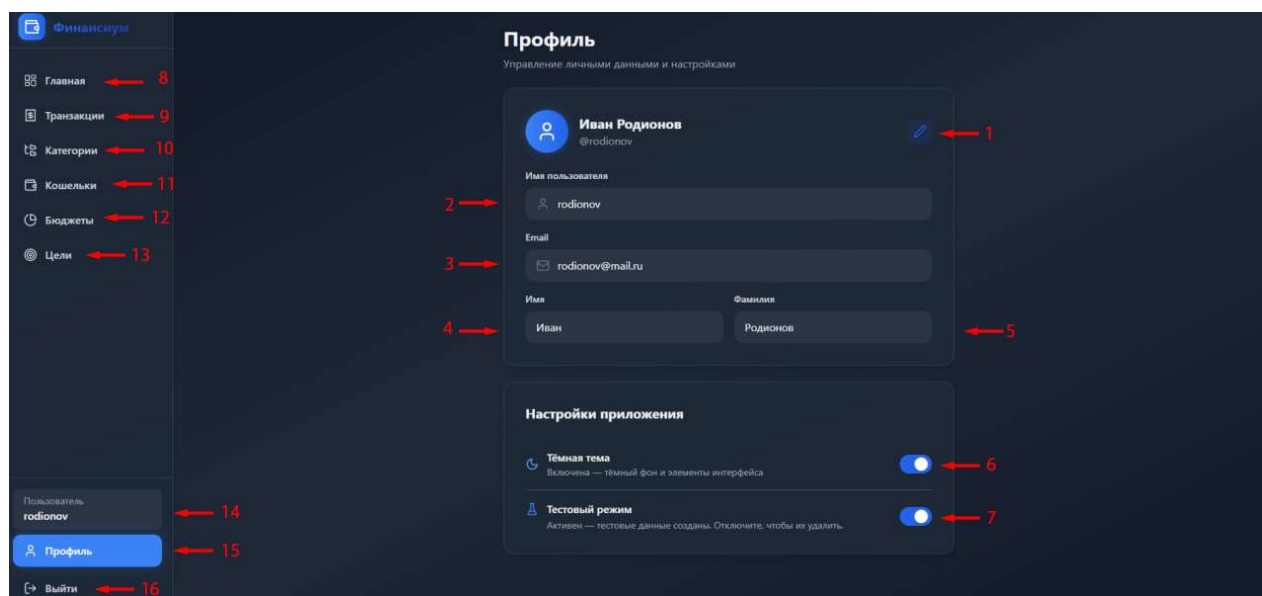


Рисунок 20 – Страница профиля пользователя

Основные элементы страницы профиля пользователя:

- 1 – кнопка включения режима редактирования для полей профиля;
- 2 – поле с уникальным именем пользователя внутри системы;
- 3 – поле с электронной почтой;
- 4 – поле с именем;

- 5 – поле с фамилией;
- 6 – переключатель оформления темной/светлой темы системы;
- 7 – переключатель тестового режима с искусственными данными;
- 8 – вкладка для перехода на страницу «Главная»;
- 9 – вкладка для перехода на страницу «Транзакции»;
- 10 – вкладка для перехода на страницу «Категории»;
- 11 – вкладка для перехода на страницу «Кошельки»;
- 12 – вкладка для перехода на страницу «Бюджеты»;
- 13 – вкладка для перехода на страницу «Цели»;
- 14 – поле с отображаемым именем пользователя в системе;
- 15 – вкладка для перехода на страницу «Профиль»;
- 16 – вкладка для выхода из системы.

При переходе на главную страницу пользователю доступна фильтрация финансов как по фиксированным периодам, так и по произвольному периоду, который пользователь может выбрать сам с помощью соответствующих кнопок. Данные изменяются автоматически. Пользователю также доступен не только просмотр доходов и расходов за период с итоговой суммой, но и проверка общего баланс всех счетов или выбранного с помощью выпадающего списка (рисунок 21).

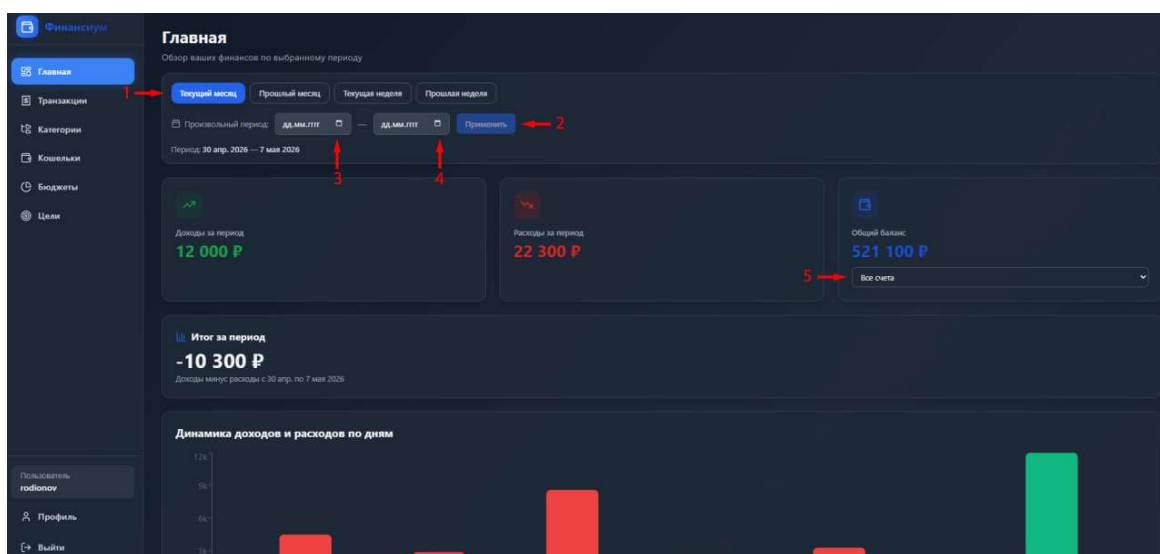


Рисунок 21 – Страница «Главная» с фильтрами по датам, общим балансом, итогами за период

Основные элементы страницы «Главная» с фильтрацией и просмотром данных по периодам:

- 1 – блок кнопок выбора периода фильтрации;
- 2 – кнопка «Применить» для применения фильтрации по произвольному периоду;
- 3 – кнопка с всплывающим меню календаря для выбора начала периода;
- 4 – кнопка с всплывающим меню календаря для выбора конца периода;
- 5 – кнопка с выпадающим списком всех счетов пользователя.

Ниже на странице «Главная» представлена графическая аналитика доходов и расходов пользователя по дням и по категориям. Также для удобного просмотра финансов за выбранный период был вынесен блок с транзакциями за текущий период (рисунок 22).

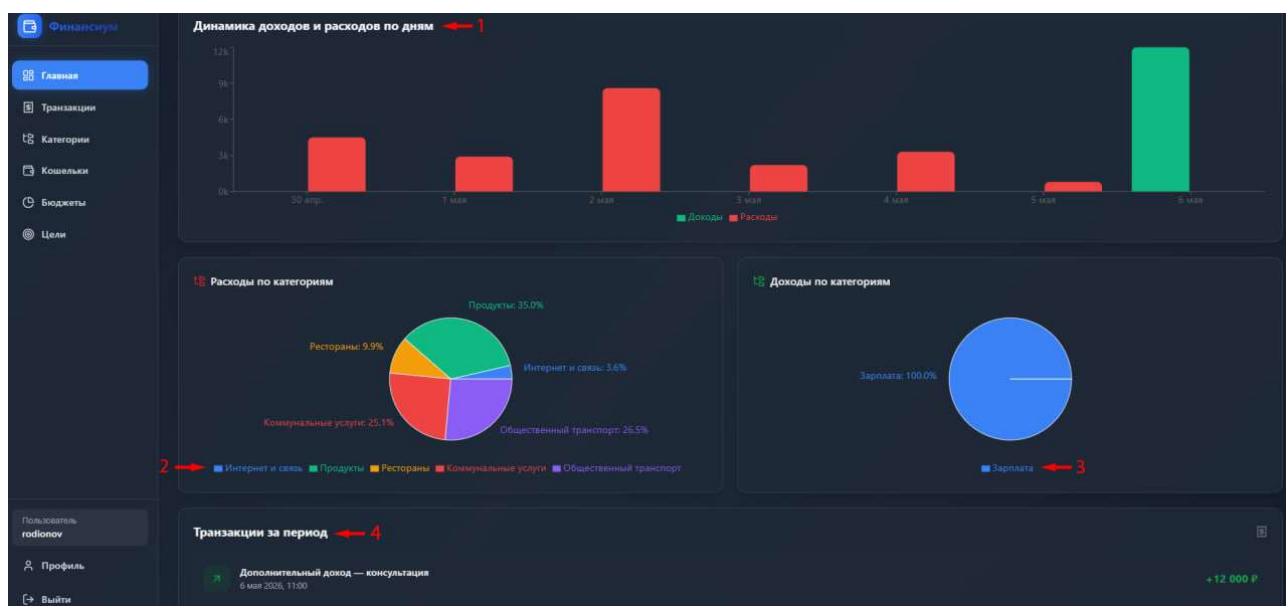


Рисунок 22 – Страница «Главная» с графической аналитикой и транзакциями за период

Основные элементы страницы «Главная» с динамикой доходов и расходов по дням и категориям:

- 1 – блок «Динамика доходов и расходов по дням» с графиком;
- 2 – наименование категорий текущих расходов;
- 3 – наименование категорий текущих доходов;
- 4 – блок с транзакциями за выбранный период.

Страница «Транзакции» содержит все активные транзакции со всех счетов пользователя (рисунок 23). Есть возможность создания транзакции с помощью кнопки «Добавить», редактирования и удаления транзакции. С помощью кнопки «Импорт квитанций в PDF» пользователь может загрузить чеки об операциях для ведения единой истории. Во вкладке «История» хранятся все транзакции, в том числе удаленные. Такие транзакции имеют специальную пометку «Удалена».

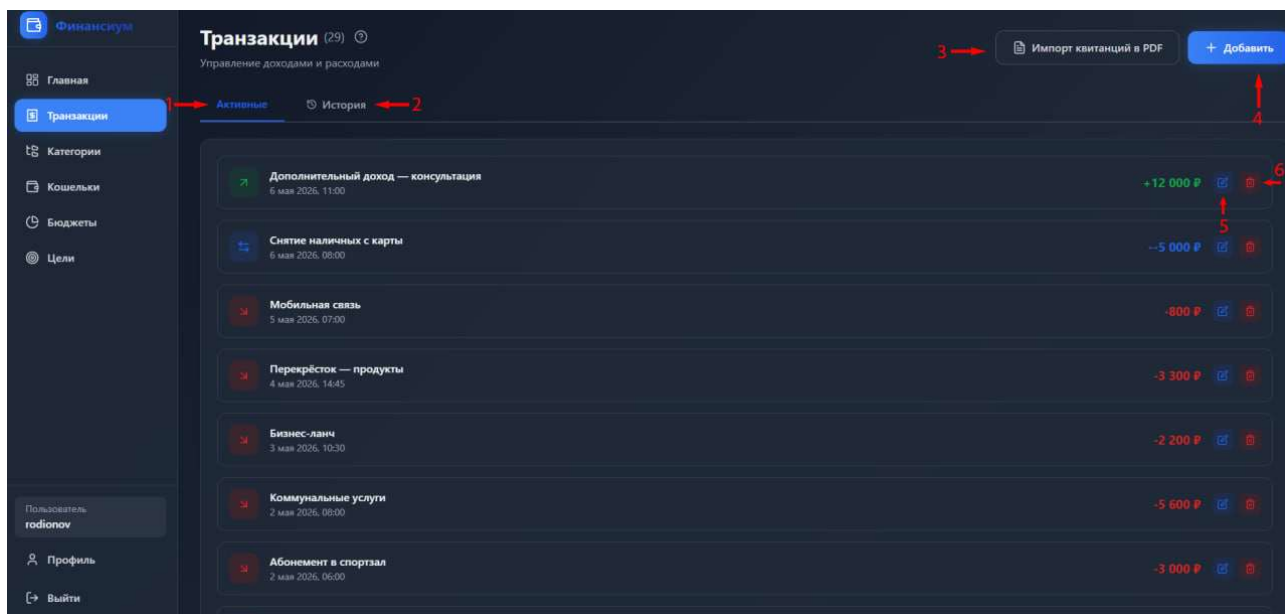


Рисунок 23 – Страница «Транзакции»

Основные элементы страницы «Транзакции»:

- 1 – вкладка с активными транзакциями;
- 2 – история со всеми транзакциями, в том числе удаленными;
- 3 – кнопка «Импорт квитанций в PDF»;
- 4 – кнопка «Добавить» для создания транзакции;
- 5 – кнопка изменения транзакции;
- 6 – кнопка удаления транзакции.

Окно импортирования квитанций из PDF визуально и по функционалу совпадает с окном созданием транзакции за исключением информационного сообщения о том, что данные были загружены из квитанции и требуют проверки перед сохранением (рисунок 24).

Новая транзакция

1 → **Информационное сообщение:** Данные загружены из квитанции. Файл: Документ-2026-02-25-22-34-22.pdf. Рекомендуем проверить автоматически подставленные поля и при необходимости скорректировать их перед сохранением.

2 → **Описание:** Стипендия ФГБОУ ВО "ПГУ", ПЕНЗЕНСКИЙ ГОСУД

3 → **Категория *:** Прочее

4 → **Тип:** Доход

5 → **Счет *:** Основная карта (243 500 ₽)

6 → **Сумма:** 19000

7 → **Дата:** 25.02.2026 10:15

8 → **Сохранить** (кнопка)

9 → **Отмена** (кнопка)

Рисунок 24 – Окно создания транзакции

Основные элементы окна импортирования квитанций из PDF:

- 1 – информационное сообщение о загруженных данных;
- 2 – поле описания транзакции;
- 3 – выпадающий список с категориями транзакции;
- 4 – выпадающий список с типом транзакции;
- 5 – выпадающий список с выбором счета для привязки к транзакции;
- 6 – поле с суммой;
- 7 – поле с датой транзакции;
- 8 – кнопка «Сохранить»;
- 9 – кнопка «Отмена».

На странице «Категории» пользователю доступны различные операции с категориями: применить фильтры по группам категорий, создать пользовательскую категорию, просмотреть статистику расходов (рисунок 25).

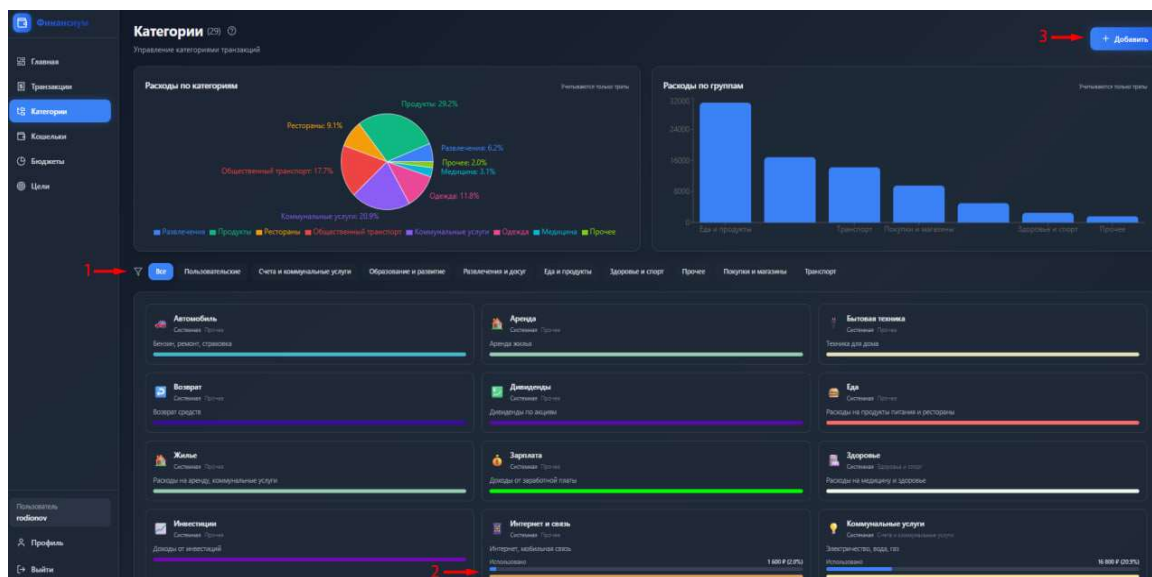


Рисунок 25 – Страница «Категории»

Основные элементы страницы «Категории»:

- 1 – блок фильтров по группам категорий;
- 2 – шкала прогресса использованных средств (доля от всех расходов);
- 3 – кнопка «Добавить».

При нажатии на кнопку «Добавить» открывается окно добавления новой категории. Пользователю доступна возможность задать название новой категории, описание, иконку и цвет для быстрого поиска, а также группу, к которой будет относиться категория. Интерфейс данного окна представлен на рисунке 26.

Новая категория

Название: Подписки

Описание:

Иконка: [Иконка]

Цвет: [Цвет]

Группа: Финансы и услуги

Сохранить Отмена

Рисунок 26 – Окно добавления новой категории

Основные элементы окна добавления категории:

- 1 – поле с названием категории;
- 2 – поле с описанием категории (необязательно);
- 3 – иконка для обозначения и быстрого поиска;
- 4 – выпадающий список с группами категории;
- 5 – кнопка «Сохранить»;
- 6 – поле с выпадающим меню для выбора цвета;
- 7 – кнопка «Отмена».

На странице «Кошельки» доступно управление кошельками пользователя (просмотр, создание, редактирование, удаление). Интерфейс страницы представлен на рисунке 27. Страница используется основой для последующих операций с транзакциями, бюджетами и целями, поскольку именно кошельки выступают базовым источником данных о доступных средствах.

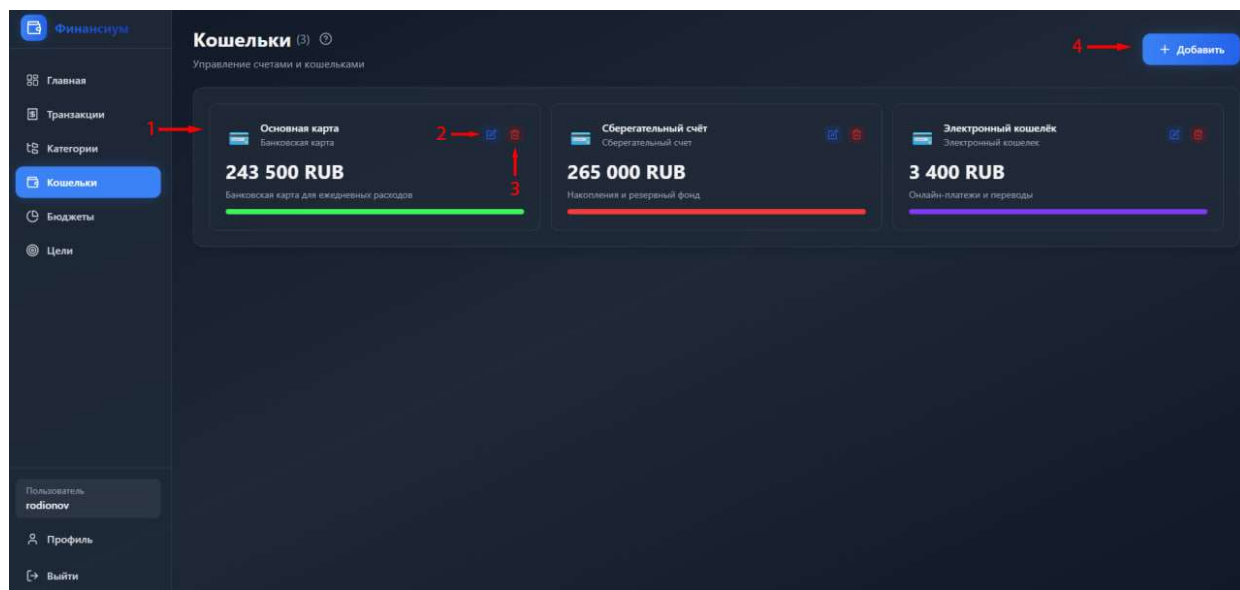


Рисунок 27 – Страница «Кошельки»

Основные элементы страницы «Кошельки»:

- 1 – карточка кошелька с названием, типом, суммой, описанием, иконкой и цветом;
- 2 – кнопка редактирования кошелька;
- 3 – кнопка удаления кошелька;
- 4 – кнопка «Добавить».

Страница «Цели» предназначена для контроля персональных финансовых целей пользователя. Интерфейс раздела реализует полный цикл работы с целями: создание, просмотр прогресса, анализ связанных операций, редактирование и удаление (рисунок 28). Каждая карточка с целью содержит название, текущую и целевую суммы, тип цели, категорию (если задана), статус выполнения, индикатор прогресса и рекомендуемый ежемесячный взнос. Также в карточке доступны действия редактирования и удаления. Выбор карточки активирует детальный просмотр выбранной цели.

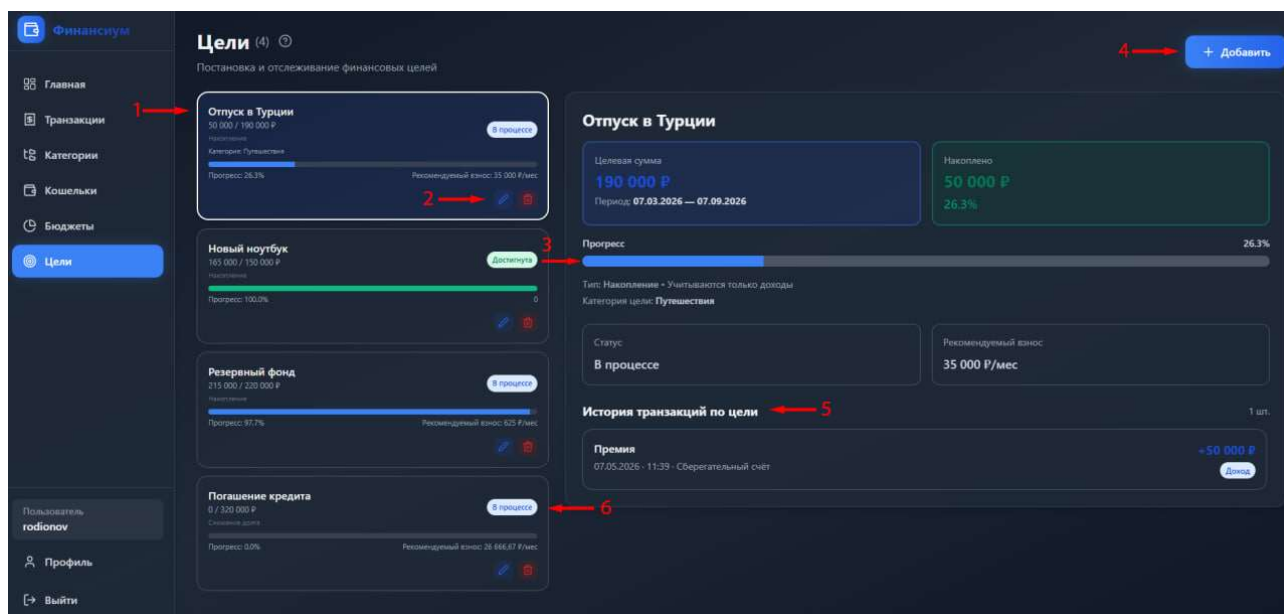


Рисунок 28 – Страница «Цели»

Основные элементы страницы «Цели»:

- 1 – карточка с целью;
- 2 – блок кнопок редактирования и удаления цели;
- 3 – шкала прогресса выполнения конкретной цели;
- 4 – кнопка «Добавить»;
- 5 – блок истории транзакций по конкретной цели;
- 6 – статус цели.

Страница «Бюджеты» предназначена для планирования и контроля расходов по выбранному периоду, счету и категориям. В разделе доступны все основные операции с бюджетами: создание, просмотр фактического исполнения,

редактирование параметров и удаление. Ключевой функциональный блок страницы связан с выбором периода анализа. Пользователь может использовать фильтры по времени с возможностью задавать их вручную. После выбора интерфейс загружает соответствующие бюджеты и отображает итоговые показатели в привязке к выбранному временному промежутку (рисунок 29).

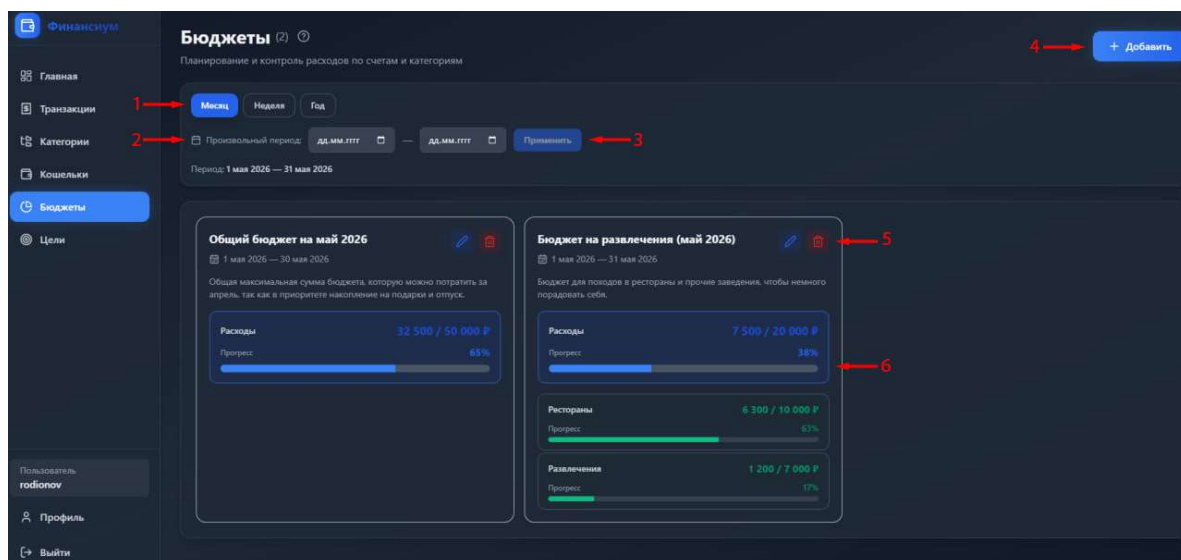


Рисунок 29 – Страница «Бюджеты»

Основные элементы страницы «Бюджеты»:

- 1 – блок фильтрации по определенным периодам;
- 2 – блок выбора произвольного периода фильтрации;
- 3 – кнопка «Применить»;
- 4 – кнопка «Добавить»;
- 5 – блок кнопок редактирования и удаления;
- 6 – шкала контролирования общих расходов и по категориям.

При нажатии на кнопку «Добавить» во вкладке с бюджетами пользователю открывается окно с добавлением нового бюджета (рисунок 30). При нажатии на кнопку редактирования бюджета в виде соответствующей иконки в блоке кнопок открывается окно редактирования бюджета, которое практически не имеет никаких различий с окном создания, так как есть возможно редактировать все поля.

Новый бюджет

1 → Название: Бюджет на траты (май 2026)

2 → Описание: Бюджет для походов в рестораны и прочие заведения, чтобы немного порадовать себя.

3 → Счет *: Основная карта

4 → Общий лимит бюджета *: 20000
Максимальная сумма расходов, которую нельзя превышать

5 → Дата начала: 01.05.2026

7 → Дата окончания: 31.05.2026

6 → Лимиты по категориям (опционально):
Если категории не выбраны, бюджет будет применяться ко всем категориям.
Если выбраны конкретные категории, бюджет будет применяться только к ним.

8 → + Добавить

9 → Лимиты по категориям:
Рестораны: 10000
Развлечения: 7000

10 → + Создать бюджет

11 → Отмена

Рисунок 30 – Окно создания нового бюджета

Основные элементы окна создания нового бюджета:

- 1 – поле с названием бюджета;
- 2 – описание бюджета (опционально);
- 3 – выпадающий список с существующими счетами для привязки;
- 4 – поле с общим лимитом бюджета;
- 5 – дата начала действия бюджета;
- 6 – информационное сообщение о лимитах по категориям;
- 7 – дата окончания действия бюджета;
- 8 – кнопка «Добавить» для добавления лимита по категории;
- 9 – блок с выбором категории, суммы ограничения и иконкой удаления;
- 10 – кнопка «Создать бюджет»;
- 11 – кнопка «Отмена».

Применение React и TypeScript позволило обеспечить интерактивность интерфейса, своевременное обновление данных после выполнения пользовательских действий и удобное взаимодействие с серверной частью приложения. Наличие аналитических графиков, фильтров по периодам, окон взаимодействия и визуальных индикаторов прогресса повышает наглядность представления финансовой информации и упрощает контроль доходов, расходов, бюджетов и целей.

4 Контроль качества программного обеспечения

4.1 Выбор режима тестирования и метрик кода

Контроль качества программного обеспечения является обязательной частью разработки платформы для управления личными финансами и планирования бюджета, поскольку система работает с критически важными для пользователя данными: денежными операциями, балансами кошельков, бюджетами, аналитикой и финансовыми целями. Ошибки в таких сценариях могут привести не только к техническим сбоям, но и к искажению пользовательской информации относительно финансового состояния, поэтому проверка должна охватывать как корректность бизнес-логики, так и качество программной реализации.

Для разработанной платформы было проведено тестирование, организованное на нескольких уровнях. Тестирование программного обеспечения представляет собой проверку поведения системы с целью выявления дефектов, подтверждения соответствия требованиям и снижения риска появления ошибок на этапе эксплуатации пользователем.

Современная практика разработки программного обеспечения предполагает использование нескольких видов тестирования, которые дополняют друг друга. В настоящий момент распространены такие виды тестирования, как функциональное, модульное, интеграционное, системное, нагрузочное.

Функциональное тестирование ориентировано на проверку системы с точки зрения требований и пользовательских сценариев. Модульное тестирование – вид тестирования, которое направлено на изолированную проверку отдельных методов и компонентов. Системное тестирование – вид тестирования, которое позволяет подтвердить работоспособность продукта как единого целого, а нагрузочное тестирование позволяет оценить поведение системы под реальной или повышенной интенсивностью запросов.

В рамках микросервисной архитектуры функциональные проверки особенно важны для сервисного слоя, где сосредоточены основные правила предметной области. В качестве автоматизированного тестирования было выбрано модульное тестирование с использованием unit-тестов.

Unit-тестирование – это проверка минимально изолированной единицы кода без реального подключения к внешним системам. Тест проверяет один конкретный аспект поведения: либо результат является корректным, либо выводится корректная реакция на ошибку [18].

Чтобы наиболее точно отразить архитектурную и структурную сложность программного модуля, необходимо произвести количественную оценку исходного кода. Метрики кода – это формализованные показатели, которые позволяют измерить свойства программной реализации: объем, алгоритмическую сложность, связность, уровень абстракции [19]. Использование метрик дает возможность перейти от субъективной оценки к объективным числовым значениям, а также позволяет сравнить компоненты между собой и выявлять участки, требующие рефакторинга или дополнительного тестового контроля.

4.2 Тестирование платформы для управления личными финансами и планирования бюджета

4.2.1 Тестирование пользовательской части приложения для управления личными финансами и планирования бюджета

Для проведения функционального тестирования пользовательской части был выбран метод тестирования черным ящиком. В качестве основного пользовательского сценария необходимо отметить создание транзакции, поскольку именно через этот сценарий формируются дальнейшие записи о доходах, расходах и переводах, а также выполняется пересчет финансовых показателей всей платформы. Ключевым параметром в данном сценарии является поле «amount»: оно обязательно для заполнения, передается как десятичное число и должно быть не меньше 0,01. Дополнительно на уровне базы

данных применяются ограничения точности в 19 символов из-за типа данных DECIMAL, а также проверка на пустое и неотрицательное значения.

Рассмотрим описание метода классов эквивалентности для данного поля. В таблице 14 представлены технические требования к полю «amount».

Таблица 14 – Технические требования к полю «amount»

№	Требование	Содержание
1	Обязательность	Поле является обязательным: не может быть null и не может отсутствовать в запросе
2	Тип данных	Значение должно быть числом в десятичном формате
3	Логическое ограничение	Значение должно быть строго больше нуля
4	Минимальное значение	Минимальное значение составляет: 0,01
5	Ограничение дробной части	Не более двух знаков после точки
6	Ограничение базы данных	Тип DECIMAL(19, 2) позволяет хранить не более 10 цифр, из них до 17 цифр в целой части
7	Реакция системы	При нарушении ограничений возникает ошибка валидации, а при корректных значениях появляется сигнал об успешном создании

На основе требований было выполнено разбиение входных данных на правильные и неправильные классы эквивалентности, которые представлены в таблице 15. Правильные классы эквивалентности представлены сокращениями от их названий КЭ-П1, где КЭ означает класс эквивалентности, а сокращения П1 и Н1 – правильный и неправильный КЭ с порядковым номером соответственно.

Таблица 15 – Классы эквивалентности

Критерий	Правильные КЭ	Неправильные КЭ
Длина значения (разрядность)	Не более 19 цифр, целая часть от 1 до 17 цифр, дробная от 0 до 2 цифр (КЭ-П1)	Общая длина больше 19 цифр (переполнение) (КЭ-Н1), целая часть больше 17 цифр (КЭ-Н2)
Структура	Число в десятичной записи, например: 123; 10,5; 999,99 (КЭ-П2)	Нечисловой формат (строка, объект, массив) (КЭ-Н3), больше 2 знаков после запятой (КЭ-Н4)
Содержание	Положительное значение больше или равно 0,01 (КЭ-П3)	Отсутствует поле (КЭ-Н5), нулевое значение (КЭ-Н6), отрицательное значение (КЭ-Н7)

Проверка классов эквивалентности и граничных значений для поля «amount» позволяет подтвердить корректность валидации вводимых значений и надежность работы пользовательского функционала при создании транзакции.

По выделенным классам эквивалентности был сформирован набор из 10 тест-кейсов, включающий как корректные, так и некорректные значения каждого класса эквивалентности. Набор тест-кейсов представлен в таблице 16. Результаты представлены в приложении Г.

Таблица 16 – Тест-кейсы по классам эквивалентности поля «amount»

№	КЭ	Входные данные	Выходные данные
1	КЭ-П1, КЭ-П2, КЭ-П3	150	Транзакция создана (рисунок Г.1)
2	КЭ-П1, КЭ-П2, КЭ-П3	150,5	Транзакция создана (рисунок Г.2)
3	КЭ-П1, КЭ-П2, КЭ-П3	100000000000000000	Транзакция создана (рисунок Г.3)
4	КЭ-П1, КЭ-П2, КЭ-П3	0,01	Транзакция создана (рисунок Г.4)
5	КЭ-Н6	0	Ошибка: сумма должна быть больше 0.01 (рисунок Г.5). Транзакция не создана
6	КЭ-Н7	-1	Ошибка: сумма должна быть больше 0.01 (рисунок Г.6). Транзакция не создана
7	КЭ-Н4	10,111	Ошибка: переполнение знаков после запятой (рисунок Г.7). Транзакция не создана
8	КЭ-Н5	поле отсутствует	Ошибка валидации обязательного поля (рисунок Г.8). Транзакция не создана
9	КЭ-Н3	«abc»	Ошибка несоответствия типа (рисунок Г.9). Транзакция не создана
10	КЭ-Н3	{«value»:100}	Ошибка несоответствия типа (рисунок Г.10). Транзакция не создана

Тест-кейсы под номерами 8 - 10 проверялись напрямую через API с помощью инструмента для тестирования Postman, так как форма ввода транзакции через пользовательский интерфейс содержит обязательное поле суммы и ограничение ввода отрицательных и нулевых значений на клиентской стороне [20].

Также к классам эквивалентности были составлены граничные условия для вводимых данных. Граничные условия к полю с суммой представлены в таблице 17.

Таблица 17 – Граничные условия

Границы	Содержание
Физические	Поле суммы должно существовать во входном сообщении (Ф1)
Логические	Сумма должна быть строго больше нуля (Л1) Для транзакции типа EXPENSE (списание) и TRANSFER (перевод) баланс исходного кошелька не должен быть с недостающим количеством средств (Л2)

Продолжение таблицы 17

Границы	Содержание
Технологические	Ограничение базы данных задается типом DECIMAL(19,2): максимум 19 цифр, из них 17 в целой части (T1) Формат числа: не более двух знаков после запятой (T2)
Произвольные	Минимально допустимое значение составляет 0.01

Для выделенных граничных условий были составлены тест-кейсы, представленные в таблице 18.

Таблица 18 – Тест-кейсы для выделенных граничных условий

№	Границы	Входные данные	Выходные данные
1	Ф1	Поле отсутствует или равно null	Ошибка: пропущено обязательное поле
2	Л1	0	Ошибка: значение должно быть больше или равно 0,01
3	T1	100000000000000000	Успешно
4	T1	100000000000000000	Ошибка: переполнение типа
5	T2	10,99	Успешно
6	T2	10,999	Ошибка: недопустимое количество знаков после запятой
7	Л2	Баланс кошелька: 1000,00 Сумма списания: 999,99	Успешно
8	Л2	Баланс кошелька: 1000,00 Сумма списания: 1001,00	Ошибка: недостаточно средств на балансе
9	П1	0,01	Успешно
10	T1, П1	0,0091232	Ошибка: недопустимое количество знаков после запятой и значение ниже минимума

Полученные результаты соответствуют ожидаемым, что позволяет сделать вывод о том, что пользовательская часть приложения работает корректно.

4.2.2 Тестирование серверной части приложения для управления личными финансами и планирования бюджета

Для обеспечения корректной работы разработанного программного продукта и подтверждения ожидаемого поведения бизнес-логики серверной части приложения для управления личными финансами и планирования бюджета было проведено модульное тестирование. В первую очередь была осуществлена проверка сервисных компонентов, которые реализуют ключевые

сценарии платформы: создание и обработка транзакций, управление кошельками, бюджетами, целями, пользователями и аналитикой.

Тестирование серверных компонентов выполнялось с использованием таких инструментов, как Junit 5, Mockito, AssertJ, что позволило изолированно проверять отдельные методы, моделировать поведение внешних зависимостей, а также валидировать как успешные, так и ошибочные ветви выполнения. Такой подход позволяет снизить риск регрессий и получить объективное подтверждение корректности реализации бизнес-правил на уровне сервисного слоя. Фрагмент кода юнит-тестов представлен в приложении Д. Полный код юнит-тестов представлен на сервисе git [17]. Тестирование было проведено для каждого сервиса приложения. Рассмотрим спецификации тестов на примере сервиса авторизации. Тесты представлены в таблице 19.

Таблица 19 – Спецификация тестов для сервиса авторизации

№	Название теста	Сценарий	Ожидаемый результат
1	login_success	Успешный вход: возвращаются access/refresh токены, обновляется lastLogin	Тест пройден
2	login_wrongPassword_throws	Неверный пароль: AuthenticationException, увеличивается счетчик неудачных попыток	Тест пройден
3	login_deactivatedAccount_throws	Вход в деактивированный аккаунт: AuthenticationException	Тест пройден
4	login_existingSession_isRevoked	При активной сессии выполняется ее отзыв перед новым входом	Тест пройден
5	validateToken_valid_returnsData	Валидный токен: успешный ValidationResponse с userId/username/role	Тест пройден
6	validateToken_blacklisted_returnsInvalid	Токен в blacklist: возвращается невалидный ответ	Тест пройден
7	validateToken_jwtException_returnsInvalid	Ошибка валидации JWT (TokenException): возвращается невалидный ответ без проброса исключения	Тест пройден
8	refreshToken_success	Валидный refresh token: выдается новый access token	Тест пройден
9	refreshToken_invalid_throws	Невалидный refresh token: TokenException	Тест пройден
10	logout_success	Успешный logout: refresh-токены отзываются, access-токен добавляется в blacklist	Тест пройден

Продолжение таблицы 19

№	Название теста	Сценарий	Ожидаемый результат
11	logout_expiredAccessToken_notBlacklisted	Истекший access-токен не добавляется в blacklist.	Тест пройден
12	changeUserRole_delegates	Смена роли делегируется в credentialsService	Тест пройден

Рассмотрим спецификацию тестов на примере сервиса управления транзакциями. Спецификация тестов представлена в таблице 20.

Таблица 20 – Спецификация тестов для сервиса управления транзакциями

№	Название теста	Сценарий	Ожидаемый результат
1	createTransaction_income_success	Успешное создание INCOME: сохранение, история, публикация события	Тест пройден
2	createTransaction_expense_validatesBalance	Для EXPENSE вызывается проверка баланса	Тест пройден
3	createTransaction_expense_insufficientBalance_throws	Недостаточно средств: InsufficientBalanceException, сохранение не выполняется	Тест пройден
4	createTransaction_transfer_missingToWallet_throws	TRANSFER без toWalletId: IllegalArgumentException	Тест пройден
5	createTransaction_transfer_sameWallets_throws	TRANSFER с одинаковыми кошельками: IllegalArgumentException	Тест пройден
6	createTransaction_transfer_success	Успешный TRANSFER между разными кошельками	Тест пройден
7	deleteTransaction_success	Успешное удаление транзакции с записью в историю и публикацией события удаления	Тест пройден
8	deleteTransaction_notFound_throws	Удаляемая транзакция не найдена: ResourceNotFoundException	Тест пройден
9	deleteTransaction_wrongUser_throws	Попытка удалить чужую транзакцию: IllegalArgumentException	Тест пройден
10	getTransactionById_success	Получение транзакции по ID возвращает корректный DTO	Тест пройден
11	getUserTransactions_returnPage	Возвращается страница транзакций пользователя	Тест пройден
12	clearTransactionHistory_deletesAll	Очистка истории пользователя вызывает удаление всех записей истории	Тест пройден

Рассмотрим модульные тесты на примере сервиса работы с кошельками. Спецификация тестов представлена в таблице 21.

Таблица 21 – Спецификация тестов для сервиса управления кошельками

№	Название теста	Сценарий	Ожидаемый результат
1	createWallet_success	Успешное создание нового кошелька и публикация события	Тест пройден

Продолжение таблицы 21

№	Название теста	Сценарий	Ожидаемый результат
2	createWallet_duplicateName_throws	Дублирующееся имя кошелька: <code>IllegalArgumentException</code>	Тест пройден
3	updateWallet_success	Успешное обновление данных кошелька.	Тест пройден
4	updateWallet_notFound_throws	Обновляемый кошелек не найден: <code>ResourceNotFoundException</code>	Тест пройден
5	updateWallet_wrongUser_throws	Обновление чужого кошелька: <code>IllegalArgumentException</code>	Тест пройден
6	deleteWallet_success	Успешный soft-delete: кошелек деактивируется	Тест пройден
7	deleteWallet_notFound_throws	Удаляемый кошелек не найден: <code>ResourceNotFoundException</code>	Тест пройден
8	deleteWallet_wrongUser_throws	Удаление чужого кошелька: <code>IllegalArgumentException</code>	Тест пройден
9	getWalletById_success	Успешное получение кошелька по идентификатору	Тест пройден
10	getWalletById_notFound_throws	Кошелек не найден: <code>ResourceNotFoundException</code>	Тест пройден
11	getUserWallets_returnsList	Получение списка всех кошельков пользователя	Тест пройден
12	getActiveUserWallets_returnsOnlyActive	Получение только активных кошельков пользователя	Тест пройден
13	getUserWallets_empty	При отсутствии кошельков возвращается пустой список	Тест пройден

Было также проведено модульное тестирования для сервиса целей. Спецификация тестов представлена в таблице 22.

Таблица 22 – Спецификация тестов для сервиса целей

№	Название теста	Сценарий	Ожидаемый результат
1	createGoal_success	Успешное создание цели и стартового snapshot	Тест пройден
2	createGoal_initialSnapshotHasZeroAmount	Стартовый snapshot создается с суммой 0	Тест пройден
3	updateGoal_categoryChanged_recalculatesAmount	При смене категории выполняется перерасчет текущей стоимости через <code>transactionServiceClient</code>	Тест пройден
4	updateGoal_categoryUnchanged_noRecalculation	Если категория не изменилась, перерасчет не выполняется	Тест пройден
5	updateGoal_notFound_throws	Обновляемая цель не найдена: <code>IllegalArgumentException</code>	Тест пройден
6	updateGoal_wrongUser_throws	Обновление чужой цели: <code>IllegalArgumentException</code>	Тест пройден
7	deleteGoal_success	Успешное удаление цели.	Тест пройден
8	deleteGoal_notFound_throws	Удаляемая цель не найдена: <code>IllegalArgumentException</code>	Тест пройден

Продолжение таблицы 22

№	Название теста	Сценарий	Ожидаемый результат
9	deleteGoal_wrongUser_throws	Удаление чужой цели: <code>IllegalArgumentException</code>	Тест пройден
10	getUserGoals_returnsList	Возврат списка целей пользователя в виде DTO	Тест пройден
11	getGoalProgress_addsStartPoint	В прогресс добавляется стартовая точка, если ее нет	Тест пройден
12	getGoalProgress_existingStartSnapshot_noDuplicate	При наличии стартового snapshot дублирование не происходит	Тест пройден
13	getGoalProgress_wrongUser_throws	Доступ к чужой цели: <code>IllegalArgumentException</code>	Тест пройден

Рассмотрим спецификацию тестов на примере сервиса аналитики. Модульные тесты представлены в таблице 23.

Таблица 23 – Спецификация тестов для сервиса аналитики

№	Название теста	Сценарий	Ожидаемый результат
1	getDashboardSummary_returnsCorrectValues	Корректный расчет доходов, расходов, баланса и данных прошлого периода	Тест пройден
2	getDashboardSummary_noTransactions_allZero	При отсутствии транзакций все агрегаты и проценты изменений равны нулю	Тест пройден
3	getDailyReport_returnsListFromRows	Преобразование строк репозитория в <code>DailyReportDto</code>	Тест пройден
4	getDailyReport_noData_emptyList	При отсутствии данных возвращается пустой дневной отчет	Тест пройден
5	getIncomeExpenseTrend_returnsTrend	Формирование тренда доходов/расходов по дневным данным	Тест пройден
6	getCategoryShare_calculatesPercent	Корректный расчет процентной доли категорий (например, 60/40)	Тест пройден
7	getCategoryShare_empty	При отсутствии транзакций по категориям возвращается пустой список	Тест пройден
8	getForecast_monthly_returnsAverages	Прогноз на месяц считается как среднее по последним периодам	Тест пройден
9	getForecast_weekly_returnsAverages	Прогноз на неделю рассчитывается корректно по недельным данным	Тест пройден
10	getForecast_noData_zeroForecast	При отсутствии данных прогнозные значения равны нулю	Тест пройден

Рассмотрим также спецификацию модульных тестов для сервиса управления пользователями. Спецификация тестов представлена в таблице 24.

Таблица 24 – Спецификация тестов для сервиса управления пользователями

№	Название теста	Сценарий	Ожидаемый результат
1	createUser_success	Успешное создание пользователя с публикацией события.	Тест пройден

Продолжение таблицы 24

№	Название теста	Сценарий	Ожидаемый результат
2	createUser_duplicate_throws	Дубликат username/email: выбрасывается исключение, сохранение не выполняется	Тест пройден
3	createUserInternal_success	Внутреннее создание пользователя без вызова полной валидации	Тест пройден
4	createUserInternal_duplicate_throws	Внутреннее создание при дубликате: исключение	Тест пройден
5	getUserById_success	Успешное получение пользователя по ID	Тест пройден
6	getUserById_notFound_throws	Пользователь по ID не найден: исключение	Тест пройден
7	getUserByUsername_success	Успешное получение пользователя по username	Тест пройден
8	getUserByUsername_notFound_throws	Пользователь по username не найден: исключение	Тест пройден
9	getAllUsers_returnsList	Возвращается список всех пользователей	Тест пройден
10	getAllUsers_empty	При отсутствии пользователей возвращается пустой список	Тест пройден
11	updateUser_success	Успешное обновление профиля и публикация события обновления	Тест пройден
12	updateUser_notFound_throws	Обновляемый пользователь не найден: исключение	Тест пройден
13	deleteUser_success	Успешное удаление пользователя и публикация события удаления	Тест пройден
14	deleteUser_notFound_throws	Удаляемый пользователь не найден: исключение	Тест пройден
15	userExists_delegates	Проверка существования пользователя делегируется репозиторию	Тест пройден
16	existsByUsername_false	Проверка занятости username возвращает false для свободного имени	Тест пройден
17	existsByEmail_true	Проверка занятости email возвращает true для занятого адреса	Тест пройден

Рассмотрим модульные тесты для сервиса управления бюджетами. Спецификация тестов представлена в таблице 25.

Таблица 25 – Спецификация тестов для сервиса управления бюджетами

№	Название теста	Сценарий	Ожидаемый результат
1	getBudgetsForUserAndPeriod_returnsList	Возвращается список бюджетов пользователя за указанный период.	Тест пройден
2	getBudgetsForUserAndPeriod_empty	Если бюджетов нет, возвращается пустой список.	Тест пройден
3	createBudget_noCategoryLimits_sumsExpenses	Создание бюджета без лимитов категорий с суммированием расходов.	Тест пройден
4	createBudget_noAuthToken_totalSpentIsZero	Без auth-токена расходы не запрашиваются, totalSpent = 0.	Тест пройден
5	createBudget_otherWalletTransactions_notCounted	Расходы из других кошельков не учитываются в бюджете.	Тест пройден

Продолжение таблицы 25

№	Название теста	Сценарий	Ожидаемый результат
6	createBudget_withCategoryLimits_accountsPerCategory	При лимитах категорий расходы учитываются по соответствующим категориям.	Тест пройден
7	updateBudget_success	Успешное обновление бюджета.	Тест пройден
8	updateBudget_notFound_throws	Бюджет не найден при обновлении: исключение.	Тест пройден
9	updateBudget_wrongUser_throws	Попытка обновления чужого бюджета: исключение.	Тест пройден
10	deleteBudget_success	Успешное удаление бюджета.	Тест пройден
11	deleteBudget_notFound_throws	Бюджет не найден при удалении: исключение.	Тест пройден
12	deleteBudget_wrongUser_throws	Попытка удаления чужого бюджета: исключение.	Тест пройден

Результаты модульного тестирования для каждого из сервисов представлены на рисунках Е.1 – Е.7 в приложении Е.

Полученные результаты модульного тестирования соответствуют ожидаемым, что подтверждает корректность ключевых сценариев работы сервисного слоя.

4.3 Оценка качества программного кода

4.3.1 Метрика SLOC

Метрика SLOC используется для количественной оценки размера программного модуля [21]. Рост значения SLOC обычно коррелирует с повышением трудоемкости понимания кода и его модификации. Были проанализированы такие категории строк, как code (строки с исполняемым кодом), comment (строки комментариев), blank (пустые строки), total (общее количество строк файла или модуля).

Для расчета метрики использовалась утилита cloc, которая является кроссплатформенным инструментом статического анализа. Утилита помогает распознать язык файла, посчитать строки кода по категориям, а также сформировать табличный отчет в консоли. Расчет выполнялся для Java-кода сервиса управления транзакциями, чтобы результаты анализа было удобно

оценивать относительно бизнес-логики приложения. Анализ кода по каждому модулю сервиса транзакций представлен в таблице 26.

Таблица 26 – Анализ кода сервиса для управления транзакциями по модулям

Модуль (package)	Количество файлов	Blank	Comment	Code	Total
client	2	13	21	64	98
config	6	20	21	114	155
controller	3	34	79	242	355
dto	7	177	21	439	637
entity	2	125	16	334	475
exception	7	31	15	128	174
filter	1	13	5	53	71
kafka	3	79	31	371	481
mapper	1	12	24	123	159
repo	4	29	70	89	188
service	2	134	67	542	743
util	1	8	10	47	65
TransactionServiceApplication.java	1	3	4	9	16
Итого	39	678	384	2555	3617

Полученные значения по модулю отражают агрегированный объем всего Java-слоя сервиса, а не одного класса. Превышение порога в данном случае не является признаком ошибки в проектировании, так как модуль включает в себя контроллеры, сущности и их обертки для взаимодействия с пользователем, слой для работы с Kafka, репозитории, сервисы и другие компоненты, которые суммарно дают большой объем строк. Также был получен результат анализа кода в виде консольного вывода утилиты cloc (рисунок 31).

```
PS C:\Users\akeru\Documents\Github\finance-manager> cloc transaction-service/src/main/java --include-lang=Java
39 text files.
39 unique files.
0 files ignored.

github.com/AlDanial/cloc v 2.08 T=0.12 s (325.4 files/s, 30180.9 lines/s)
-----
Language             files      blank     comment      code
-----
Java                  39         678       384         2555
SUM:                  39         678       384         2555
-----
```

Рисунок 31 – Графическое представление анализа кода сервиса для управления транзакциями

Основная бизнес-логика сервиса содержится в модуле под названием «service» и содержит 5 файлов. Результат анализа модуля service представлен на рисунке 32.

```
PS C:\Users\akeru\Documents\Github\finance-manager> cloc transaction-service/src/main/java/com/akerumort/transactionservice/service
```

5 text files.
5 unique files.
0 files ignored.

github.com/AlDanial/cloc v 2.08 T=0.03 s (164.0 files/s, 24376.0 lines/s)

Language	files	blank	comment	code
Java	5	134	67	542
SUM:	5	134	67	542

Рисунок 32 – Графическое представление анализа кода для модуля service внутри сервиса для управления транзакциями

По результатам анализа можно сделать вывод о том, что модуль содержит 743 строки кода, из них 542 строки являются логическими, 67 представляют собой комментарии, а пустых строк содержится в количестве 134 единиц. Результаты демонстрируют, что ключевая бизнес-функция транзакции имеет контролируемый размер, а общий размер сервиса для управления транзакциями находится в приемлемых границах.

4.3.2 Метрика Холстеда

Метрика Холстеда применяется для оценки алгоритмической сложности кода на основе размера программы. В отличие от метрики SLOC, метрика Холстеда оценивает количество операторов и операнд у метода, что позволяет сделать вывод о трудоемкости понимания кода и его тестирования. Практическая ценность данной метрики заключается в том, что она помогает сравнивать методы одинакового размера, но с разной логической сложностью.

Анализ метрики производился на основе метода создания транзакции. Данный метод обрабатывает несколько сценариев (доход, расход, перевод), содержит множество проверок и исключений, а также внутри метода происходит валидация баланса, сохранение, запись истории, публикация события в Kafka. Функционально метод принимает следующие параметры:

- userId: идентификатор пользователя, который совершает операцию;
- request: запрос-обертка с типом транзакции, суммой, кошельком и дополнительными полями;

– authToken: токен авторизации для вызова проверки баланса во внешнем сервисе.

Код функции создания транзакции представлен в листинге 1.

Листинг 1 – Код функции создания транзакции

```
@Override
    public TransactionResponseDto createTransaction(Long userId,
        TransactionRequestDto request, String authToken) {
        logger.info("Creating transaction for user: {}", userId);
        if (request.getType() == TransactionType.TRANSFER) {
            if (request.getToWalletId() == null) {
                throw new IllegalArgumentException("To wallet is
required for transfer transactions");
            }
            if
(request.getWalletId().equals(request.getToWalletId())) {
                throw new IllegalArgumentException("Source and
destination wallets must be different");
            }
        }
        if (request.getType() == TransactionType.EXPENSE ||
request.getType() == TransactionType.TRANSFER) {
            try {

walletServiceClient.validateBalance(request.getWalletId(),
request.getAmount(), authToken);
            } catch (InsufficientBalanceException e) {
                logger.warn("Insufficient balance for transaction:
userId={}, walletId={}, amount={}",
                        userId, request.getWalletId(),
request.getAmount());
                throw e;
            }
        }
        Transaction transaction =
transactionMapper.toEntity(request, userId);
        transaction = transactionRepository.save(transaction);
        // сохраняем в историю
        saveTransactionHistory(transaction,
TransactionHistory.TransactionAction.CREATED, false);
        // публикуем событие о создании транзакции
        transactionEventProducer.publishTransactionCreated(transaction);

        logger.info("Transaction created with id: {}",
transaction.getId());
        return transactionMapper.toResponseDto(transaction);
    }
```

Рассмотрим операторы данной функции с числом их вхождений, которые представлены в таблице 27.

Таблица 27 – Операторы функции создания транзакции

Номер	Оператор	Число вхождений
1	«(»	29
2	«)»	29
3	«.»	25
4	«.»	12
5	«;»	12
6	«{»	7
7	«}»	7
8	«if»	4
9	«==»	4
10	«getType»	3
11	«throw»	3
12	«getWalletId»	3
13	«info»	2
14	«getToWalletId»	2
15	«new»	2
16	«IllegalArgumentException»	2
17	«getAmount»	2
18	«=»	2
19	«createTransaction»	1
20	«equals»	1
21	« »	1
22	«try»	1
23	«validateBalance»	1
24	«catch»	1
25	«warn»	1
26	«toEntity»	1
27	«save»	1
28	«saveTransactionHistory»	1
29	«publishTransactionCreated»	1
30	«getId»	1
31	«return»	1
32	«toResponseDto»	1

Операнды функции создания транзакции с числом вхождений представлены в таблице 28.

Таблица 28 – Операнды функции создания транзакции

Номер	Операнд	Число вхождений
1	«request»	12
2	«transaction»	7
3	«userId»	4
4	«logger»	3
5	«transactionType»	3
6	«authToken»	2
7	«TRANSFER»	2
8	«e»	2
9	«transactionMapper»	2
10	«public»	1
11	«TransactionResponseDto»	1
12	«String»	1
13	«Creating transaction for user: {}»	1
14	«null»	1
15	«To wallet is required for transfer transactions»	1
16	«Source and destination wallets must be different»	1
17	«EXPENSE»	1

Продолжение таблицы 28

Номер	Операнд	Число вхождений
18	«walletServiceClient»	1
19	«InsufficientBalanceException»	1
20	«Insufficient balance for transaction: userId={}, walletId={}, amount={}»	1
21	«Transaction»	1
22	«transactionRepository»	1
23	«TransactionHistory»	1
24	«TransactionAction»	1
25	«CREATED»	1
26	«false»	1
27	«transactionEventProducer»	1
28	«Transaction created with id: {}»	1
29	«TransactionRequestDto»	1
30	«Long»	1

На основе данных из таблиц был произведен расчет характеристик программы по формулам метрики Холстеда.

Словарь программы рассчитан по формуле:

$$n = n_1 + n_2 = 32 + 30 = 62,$$

где n – словарь программы;

n_1 – число уникальных операторов программы;

n_2 – число уникальных операндов программы.

Длина кода программы рассчитана по формуле:

$$N = N_1 + N_2 = 164 + 58 = 222,$$

где N – длина программы;

N_1 – общее число операторов в программе;

N_2 – общее число операндов в программе.

Теоретическая длина программы рассчитана по формуле:

$$N' = n_1 \times \log_2(n_1) + n_2 \times \log_2(n_2) = 32 \times 5 + 30 \times 4,91 = 307,21,$$

где N' – теоретическая длина программы;

n_1 – число уникальных операторов программы;

n_2 – число уникальных операндов программы.

Объем программы рассчитан по формуле:

$$V = N \times \log_2(n) = 222 \times 5,95 = 1321,83,$$

где V – объем программы;

N – длина программы;

n – словарь программы.

Теоретический объем программы рассчитан по формуле:

$$V' = N' \times \log_2(n) = 307.21 \times 5,95 = 1829,17,$$

где V' – теоретический объем программы;

N' – теоретическая длина программы;

n – словарь программы.

Уровень качества программирования рассчитан по формуле:

$$L = V' \div V = 1829,17 \div 1321,83 = 1,38,$$

где L – уровень качества программирования;

V' – теоретический объем программы;

V – объем программы.

Относительный уровень качества рассчитан по формуле:

$$L' = (2 \times n_2) \div (n_1 \times N_2) = (2 \times 30) \div (32 \times 58) = 0,032,$$

где L' – относительный уровень качества программирования;

n_2 – число уникальных операндов программы;

n_1 – число уникальных операторов программы;

N_2 – общее число операндов в программе.

Показатель Maintainability Index (MI) представляет собой интегральную метрику сопровождаемости, которая объединяет несколько характеристик кода в единое числовое значение. Суть данного показателя состоит в том, чтобы оценить, насколько сложно поддерживать модуль при его дальнейшем развитии: чем выше показатель, тем легче вносить изменения.

Практическая интерпретация выполняется по следующим диапазонам:

- $MI > 85$: высокая сопровождаемость;
- $65 \leq MI \leq 85$: средняя/хорошая сопровождаемость;
- $MI < 65$: повышенные риски сопровождения, требуется упрощение кода.

Показатель вычислительной сложности кода равен объему программы по определению Холстеда и составляет 1321.83, а показатель цикломатической сложности кода равен 5 по числу выделенных решений.

Для комплексной оценки сопровождаемости была использована следующая формула:

$$\begin{aligned} MI &= 171 - 5.2 \times \ln(HV) - 0.23 \times CC - 16.2 \times \ln(LOC) = \\ &= 171 - 5.2 \times \ln(1321.83) - 0.23 \times 5 - 16.2 \times \ln(26) = \\ &= 171 - 37.40 - 1.15 - 52.78 = 79.67, \end{aligned}$$

где HV – вычислительная сложность кода (Halstead Volume);

CC – цикломатическая сложность кода;

LOC – количество строк кода.

Таким образом, показатель индекса сопровождаемости равен 79.67, что соответствует хорошему уровню сопровождаемости метода, код является поддерживаемым. Полученные показатели находятся в пределах нормы, что позволяет сделать вывод о том, что код удовлетворяет требованиям метрик.

Заключение

В рамках выпускной квалификационной работы спроектирована и разработана платформа для управления личными финансами и планирования бюджета.

На этапе анализа предметной области исследованы процессы управления личными финансами, определены ключевые пользовательские сценарии и сущности системы. Выполнен сравнительный анализ существующих программных продуктов-аналогов, выявлены их функциональные и архитектурные ограничения. По результатам анализа сформированы функциональные и нефункциональные требования к разрабатываемой платформе.

В части проектирования обоснован выбор микросервисного подхода и разработана архитектура приложения с разделением ответственности между доменными сервисами. Спроектированы концептуальная, логическая и физическая модели данных, а также пользовательский интерфейс и структурные модели программных компонентов. Рассчитан бюджет и сроки выполнения проекта.

На стадии реализации разработана серверная часть платформы на языке Java с использованием фреймворка Spring, базы данных PostgreSQL и механизма управления миграциями Flyway. Также были использованы такие технологии, как Kafka, Redis, Docker, Nginx. Клиентская часть разработана в формате SPA-приложения с помощью React и TypeScript. Вспомогательный сервис распознавания квитанций в формате PDF реализован с помощью языка Python и фреймворка FastAPI. Реализованы ключевые функциональные модули: авторизация и управление пользователями, транзакции и категории, кошельки, бюджеты и лимиты, финансовые цели, аналитика и прогнозирование, импорт данных из PDF-квитанций.

В рамках контроля качества проведено функциональное тестирование пользовательской части, модульное тестирование серверной части, а также

количественная оценка качества кода с использованием метрик кода. Полученные результаты тестирования подтвердили корректность реализации основных бизнес-сценариев.

Таким образом, в результате выполнения выпускной квалификационной работы разработана платформа управления личными финансами и планирования бюджета, которая объединяет инструменты повседневного учета операций, контроля бюджетных ограничений, мониторинга финансовых целей и аналитической поддержки пользователя.

Список использованных источников

1. Родионов, И.В. Сравнительный анализ современных систем управления личными финансами: архитектурные и функциональные решения / И.В. Родионов // Информационные технологии в науке и образовании. Проблемы и перспективы: материалы XIII Всероссийской науч.-практ. Конф. – Пенза : Изд-во Пензенского государственного университета. 2026.
2. Родионов, И.В. Техники обеспечения удобства использования интерфейсов финансовых систем / И.В. Родионов // Информационные технологии в науке и образовании. Проблемы и перспективы: материалы XIII Всероссийской науч.-практ. Конф. – Пенза : Изд-во Пензенского государственного университета. 2026.
3. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/articles/566218/> (дата обращения: 03.02.2026).
4. Ньюмен, С. Создание микросервисов / С. Ньюмен – 2-е изд. – Санкт-Петербург ; Москва ; Минск : Питер, 2023. – 624 с.
5. UML – полное руководство // Блог Visual Paradigm. [Электронный ресурс]. URL: <https://blog.visual-paradigm.com/ru/uml-a-comprehensive-guide/> (дата обращения: 07.02.2026).
6. Моделирование данных: обзор // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/articles/556790/> (дата обращения: 10.02.2026)
7. Ричардс, М. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / М.Ричардс, Н.Форд – Санкт-Петербург : Питер, 2023 – 448 с.
8. Концептуальная, логическая и физическая модели данных // Официальная документация Visual Paradigm. [Электронный ресурс]. URL: <https://circle.visual-paradigm.com/docs/database-design-engineering/database-designers-guide/conceptual-logical-and-physical-data-model/> (дата обращения: 21.02.2026)

9. Single Page Application в облачном хранилище // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/selectel/articles/246019/> (дата обращения: 01.03.2026).
10. Использование диаграммы классов UML при проектировании и документировании программного обеспечения // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/articles/572234/> (дата обращения: 03.03.2026)
11. Что такое диаграмма «сущность–связь» (ERD) // Baeldung on Computer Science. [Электронный ресурс]. URL: <https://www.baeldung.com/cs/erd> (дата обращения: 11.03.2026).
12. Паттерн «Database per Service» для микросервисов // GeeksForGeeks. [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/system-design/database-per-service-pattern-for-microservices/> (дата обращения: 12.03.2026).
13. Шапира, Г. Apache Kafka. Поточковая обработка и анализ данных / Г. Шапира, Т. Палино. – 2-е изд. – Санкт-Петербург : Питер, 2023. – 512 с.
14. Docker Compose в DevOps // GeeksforGeeks. [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/devops/docker-compose/> (дата обращения: 14.03.2026).
15. Введение в Redis Server // GeeksforGeeks. [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/system-design/introduction-to-redis-server/> (дата обращения: 20.03.2026).
16. Что такое JSON // Oracle. [Электронный ресурс]. URL: <https://www.oracle.com/database/what-is-json/> (дата обращения: 02.04.2026).
17. Родионов, И.В. Платформа для управления личными финансами и планирования бюджета на основе микросервисной архитектуры // GitHub [Электронный ресурс]. URL: <https://github.com/akerugen/finance-manager> (дата обращения: 15.04.2026).
18. Лучшие практики юнит тестирования // IBM. [Электронный ресурс]. URL: <https://www.ibm.com/think/insights/unit-testing-best-practices> (дата обращения: 18.04.2026).

19. Метрики кода: значения метрик кода // Microsoft Learn. [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=visualstudio> (дата обращения: 25.04.2026).
20. Postman: основы тестирования API и первые шаги с инструментом // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/companies/vk/articles/750096/> (дата обращения: 26.05.2026).
21. Программный код и его метрики // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 04.05.2026).

Приложение А
(справочное)
Глоссарий

Микросервисная архитектура – подход к построению системы как набора независимых сервисов, каждый из которых отвечает за отдельную бизнес-область. Имеет четко определенные границы ответственности. Взаимодействие между сервисами выполняется через API и/или обмен событиями, что позволяет развертывать и масштабировать отдельные компоненты независимо друг от друга.

СУБД (система управления базами данных) – программный комплекс для создания, хранения, изменения и управления данными в базе данных.

API (Application Programming Interface) – программный интерфейс, через который компоненты системы обмениваются данными и вызывают функции.

REST API – архитектурный стиль взаимодействия клиент-серверных приложений по HTTP с использованием стандартных методов (GET, POST, PUT, DELETE).

CRUD-операции – базовые операции над данными: создание (create), чтение (read), изменение (update), удаление (delete).

PostgreSQL – реляционная система управления базами данных, применяемая для хранения данных сервисов платформы.

Apache Kafka – платформа потоковой передачи сообщений (брокер), используемая для асинхронного обмена событиями между микросервисами.

DTO (Data Transfer Object) – объект передачи данных между слоями и сервисами без включения внутренней бизнес-логики сущностей.

JWT (JSON Web Token) – формат для передачи данных аутентификации и авторизации между клиентом и сервером.

Redis – высокопроизводительное хранилище данных типа «ключ-значение», применяемое для кэширования и быстрого доступа к используемым данным. Хранит данные в оперативной памяти.

Nginx – веб-сервер и обратный прокси-сервер, используемый как API-шлюз для маршрутизации внешних запросов к микросервисам.

Spring (Spring Boot) – фреймворк Java для быстрой разработки серверных приложений и микросервисов.

React – библиотека JavaScript для разработки пользовательских интерфейсов на основе компонентов.

Postman – инструмент для ручного и автоматизированного тестирования API-запросов и ответов.

Валидация – процесс проверки входных данных на соответствие заданным правилам, ограничениям формата и бизнес-логике перед их обработкой или сохранением.

JSON (JavaScript Object Notation) – текстовый формат обмена данными в виде пар «ключ-значение», широко используемый при взаимодействии клиентской и серверной частей приложения.

Docker Compose – инструмент для декларативного запуска и управления набором взаимосвязанных контейнеров Docker на основе единого файла конфигурации. Позволяет описать в одном файле все сервисы приложения, их зависимости, переменные окружения, порты, тома и правила доступности.

Приложение Б
(обязательное)
Листинг приложения

Листинг Б.1 – Файл AnalyticsController.java

```
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDate;
import java.util.List;

@RestController
@RequestMapping("/api/analytics")
@Tag(name = "Analytics", description = "Аналитика и отчеты")
public class AnalyticsController {

    private final AnalyticsService analyticsService;

    public AnalyticsController(AnalyticsService analyticsService)
    {
        this.analyticsService = analyticsService;
    }

    @GetMapping("/dashboard/summary")
    @Operation(summary = "Сводка для дашборда", description =
"Агрегированная статистика за период с сравнением с предыдущим
периодом")
    public ResponseEntity<DashboardSummaryDto>
getDashboardSummary(
        @RequestAttribute("userId") Long userId,
        @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate from,
        @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate to) {
        DashboardSummaryDto dto =
analyticsService.getDashboardSummary(userId, from, to);
        return ResponseEntity.ok(dto);
    }

    @GetMapping("/reports/daily")
    @Operation(summary = "Дневной отчет", description = "Доходы и
расходы по дням за период")
    public ResponseEntity<List<DailyReportDto>> getDailyReport(
        @RequestAttribute("userId") Long userId,
        @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate from,
        @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate to) {
        List<DailyReportDto> list =
analyticsService.getDailyReport(userId, from, to);
        return ResponseEntity.ok(list);
    }

    @GetMapping("/charts/income-expense-trend")
    @Operation(summary = "Динамика доходов и расходов",
description = "Данные для линейного графика по дням")
```

```

        public ResponseEntity<List<IncomeExpenseTrendDto>>
getIncomeExpenseTrend(
            @RequestAttribute("userId") Long userId,
            @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate from,
            @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate to) {
            List<IncomeExpenseTrendDto> list =
analyticsService.getIncomeExpenseTrend(userId, from, to);
            return ResponseEntity.ok(list);
        }
        @GetMapping("/charts/by-category")
        @Operation(summary = "Распределение по категориям",
description = "Данные для круговой диаграммы (INCOME или
EXPENSE)")
        public ResponseEntity<List<CategoryShareDto>>
getCategoryShare(
            @RequestAttribute("userId") Long userId,
            @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate from,
            @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate to,
            @RequestParam(defaultValue = "EXPENSE") String type) {
            List<CategoryShareDto> list =
analyticsService.getCategoryShare(userId, type, from, to);
            return ResponseEntity.ok(list);
        }
        @GetMapping("/forecast")
        @Operation(summary = "Прогноз на следующий период",
description = "Простой прогноз (среднее за последние периоды).
period: MONTH или WEEK")
        public ResponseEntity<ForecastDto> getForecast(
            @RequestAttribute("userId") Long userId,
            @RequestParam(defaultValue = "MONTH") String period) {
            ForecastDto dto = analyticsService.getForecast(userId,
period);
            return ResponseEntity.ok(dto);
        }
    }
}

```

Листинг Б.2 – Файл BudgetController.java

```

import com.akerumort.budgetservice.dto.BudgetDto;
import com.akerumort.budgetservice.dto.BudgetRequest;
import com.akerumort.budgetservice.service.BudgetService;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import jakarta.validation.Valid;

import java.time.LocalDate;
import java.util.List;

```

```

@RestController
@RequestMapping("/api/budgets")
public class BudgetController {
    private final BudgetService budgetService;

    public BudgetController(BudgetService budgetService) {
        this.budgetService = budgetService;
    }

    @GetMapping
    public ResponseEntity<List<BudgetDto>> getBudgets(
        @RequestAttribute("userId") Long userId,
        @RequestParam("from") @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate from,
        @RequestParam("to") @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate to
    ) {
        List<BudgetDto> budgets =
budgetService.getBudgetsForUserAndPeriod(userId, from, to);
        return ResponseEntity.ok(budgets);
    }

    @PostMapping
    public ResponseEntity<BudgetDto> createBudget(
        @RequestAttribute("userId") Long userId,
        @RequestHeader("Authorization") String authHeader,
        @Valid @RequestBody BudgetRequest request
    ) {
        BudgetDto result = budgetService.createBudget(userId,
request, authHeader);
        return ResponseEntity.ok(result);
    }

    @PutMapping("/{id}")
    public ResponseEntity<BudgetDto> updateBudget(
        @RequestAttribute("userId") Long userId,
        @RequestHeader("Authorization") String authHeader,
        @PathVariable("id") Long budgetId,
        @Valid @RequestBody BudgetRequest request
    ) {
        BudgetDto result = budgetService.updateBudget(userId,
budgetId, request, authHeader);
        return ResponseEntity.ok(result);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteBudget(
        @RequestAttribute("userId") Long userId,
        @PathVariable("id") Long budgetId
    ) {
        budgetService.deleteBudget(userId, budgetId);
        return ResponseEntity.noContent().build();
    }
}

```

Приложение В
(рекомендуемое)
Графическая часть

Использование ресурсов от СР 27.05.26
Платформа для управления личными финансами

	Итого
Аналитик	120 ч
Анализ условий применения объектов предметной области	8 ч
Анализ приложений-аналогов для управления финансами	8 ч
Анализ функциональных требований	8 ч
Анализ нефункциональных требований	8 ч
Планирование процесса разработки приложения и оценка затрат на разработку	8 ч
Оформление отчета	80 ч
Разработчик ПО	332 ч
Проектирование архитектура приложения	48 ч
Проектирование классов программных средств	32 ч
Реализация основного приложения	144 ч
Реализация интерфейса приложения	28 ч
Оформление отчета	80 ч
Разработчик базы данных	52 ч
Проектирование пользовательского интерфейса	24 ч
Реализация интерфейса приложения	28 ч
Дизайнер	48 ч
Проектирование структур данных	32 ч
Реализация базы данных	16 ч
Тестировщик ПО	80 ч
Выбор режима тестирования и метрик кода	24 ч
Тестирование пользовательской части приложения для управления личными финансами и планирования бюджета	16 ч
Тестирование серверной части приложения для управления личными финансами и планирования бюджета	24 ч
Оценка качества программного кода	16 ч
Компьютер	
Электричество	
Анализ условий применения объектов предметной области	
Анализ приложений-аналогов для управления финансами	
Анализ функциональных требований	
Анализ нефункциональных требований	
Планирование процесса разработки приложения и оценка затрат на разработку	
Проектирование архитектура приложения	
Проектирование структур данных	
Проектирование пользовательского интерфейса	
Проектирование классов программных средств	
Реализация базы данных	
Реализация основного приложения	
Реализация интерфейса приложения	
Выбор режима тестирования и метрик кода	
Тестирование пользовательской части приложения для управления личными финансами и планирования бюджета	
Тестирование серверной части приложения для управления личными финансами и планирования бюджета	
Оценка качества программного кода	
Оформление отчета	
Интернет	
Анализ условий применения объектов предметной области	
Анализ приложений-аналогов для управления финансами	
Анализ функциональных требований	
Анализ нефункциональных требований	
Проектирование архитектура приложения	
Проектирование пользовательского интерфейса	
Реализация базы данных	
Реализация основного приложения	
Реализация интерфейса приложения	
Выбор режима тестирования и метрик кода	
Тестирование пользовательской части приложения для управления личными финансами и планирования бюджета	
Тестирование серверной части приложения для управления личными финансами и планирования бюджета	
Итого	632 ч

Рисунок В.1 – Отчет об использовании ресурсов

Отчет о бюджете от Ср 27.05.26
Платформа для управления личными финансами

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
17	Реализация основного приложения	0,00 Р	Пропорциональное	72 240,00 Р	0,00 Р	72 240,00 Р
27	Оформление отчета	0,00 Р	Пропорциональное	64 200,00 Р	0,00 Р	64 200,00 Р
10	Проектирование архитектура прило	0,00 Р	Пропорциональное	24 120,00 Р	0,00 Р	24 120,00 Р
18	Реализация интерфейса приложени	0,00 Р	Пропорциональное	22 600,00 Р	0,00 Р	22 600,00 Р
13	Проектирование классов программн	0,00 Р	Пропорциональное	16 050,00 Р	0,00 Р	16 050,00 Р
23	Тестирование серверной части приг	0,00 Р	Пропорциональное	8 480,00 Р	0,00 Р	8 480,00 Р
21	Выбор режима тестирования и метр	0,00 Р	Пропорциональное	8 440,00 Р	0,00 Р	8 440,00 Р
11	Проектирование структур данных	0,00 Р	Пропорциональное	8 050,00 Р	0,00 Р	8 050,00 Р
12	Проектирование пользовательского	0,00 Р	Пропорциональное	7 280,00 Р	0,00 Р	7 280,00 Р
22	Тестирование пользовательской час	0,00 Р	Пропорциональное	5 680,00 Р	0,00 Р	5 680,00 Р
24	Оценка качества программного кода	0,00 Р	Пропорциональное	5 625,00 Р	0,00 Р	5 625,00 Р
16	Реализация базы данных	0,00 Р	Пропорциональное	4 080,00 Р	0,00 Р	4 080,00 Р
3	Анализ условий применения объект	0,00 Р	Пропорциональное	2 440,00 Р	0,00 Р	2 440,00 Р
4	Анализ приложений-аналогов для уг	0,00 Р	Пропорциональное	2 440,00 Р	0,00 Р	2 440,00 Р
5	Анализ функциональных требовани	0,00 Р	Пропорциональное	2 440,00 Р	0,00 Р	2 440,00 Р
6	Анализ неф функциональных требова	0,00 Р	Пропорциональное	2 440,00 Р	0,00 Р	2 440,00 Р
7	Планирование процесса разработки	0,00 Р	Пропорциональное	2 425,00 Р	0,00 Р	2 425,00 Р
1	Начало проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
8	Анализ предметной области и выявл	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
14	Проектирование платформы для уп	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
19	Реализация платформ ы для управл	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
25	Контроль качества программного об	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
28	Оформление пояснительной записк	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
29	Конец проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
		0,00 Р		259 030,00 Р	0,00 Р	259 030,00 Р

Рисунок В.2 – Отчет о бюджете проекта

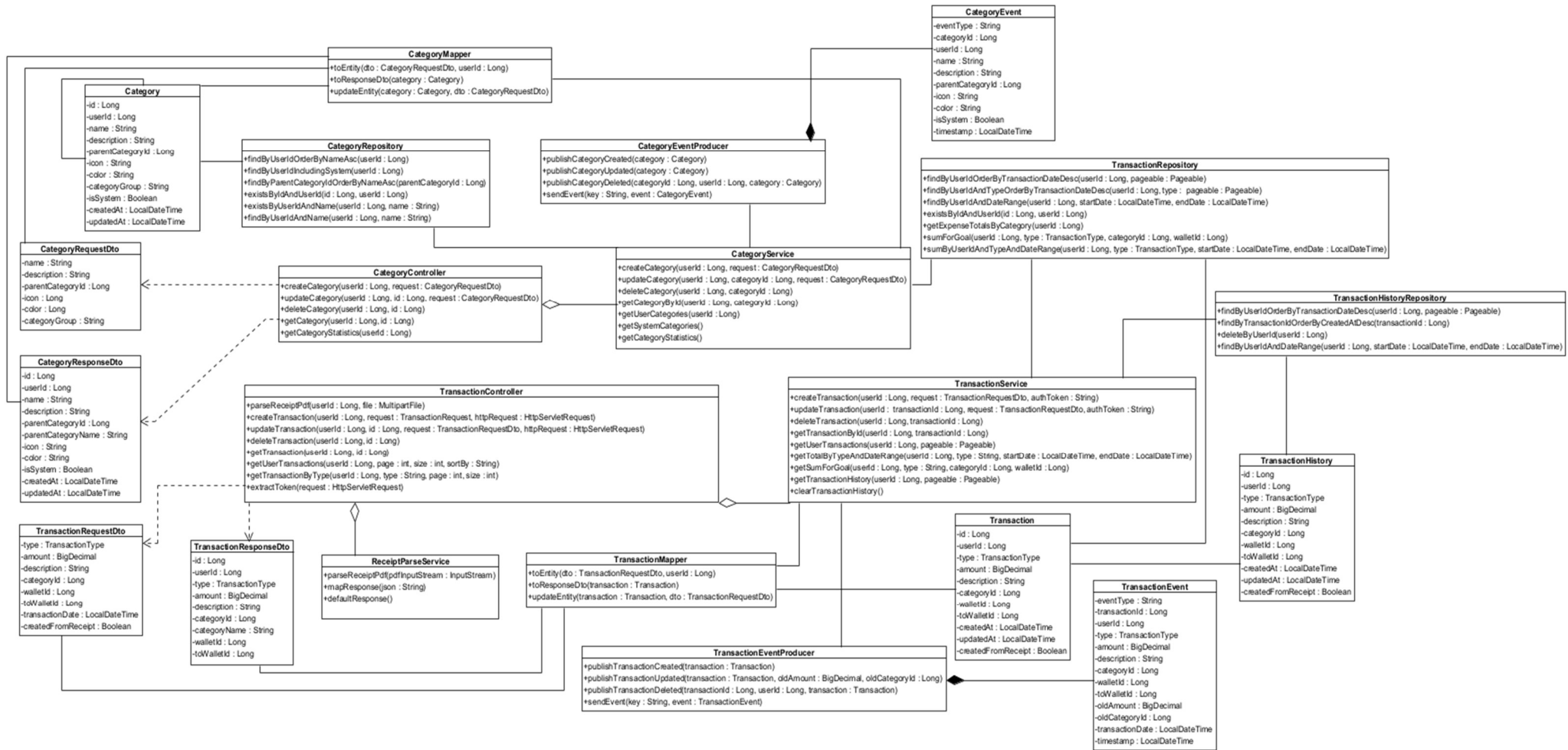


Рисунок В.3 – Диаграмма классов для сервиса транзакций

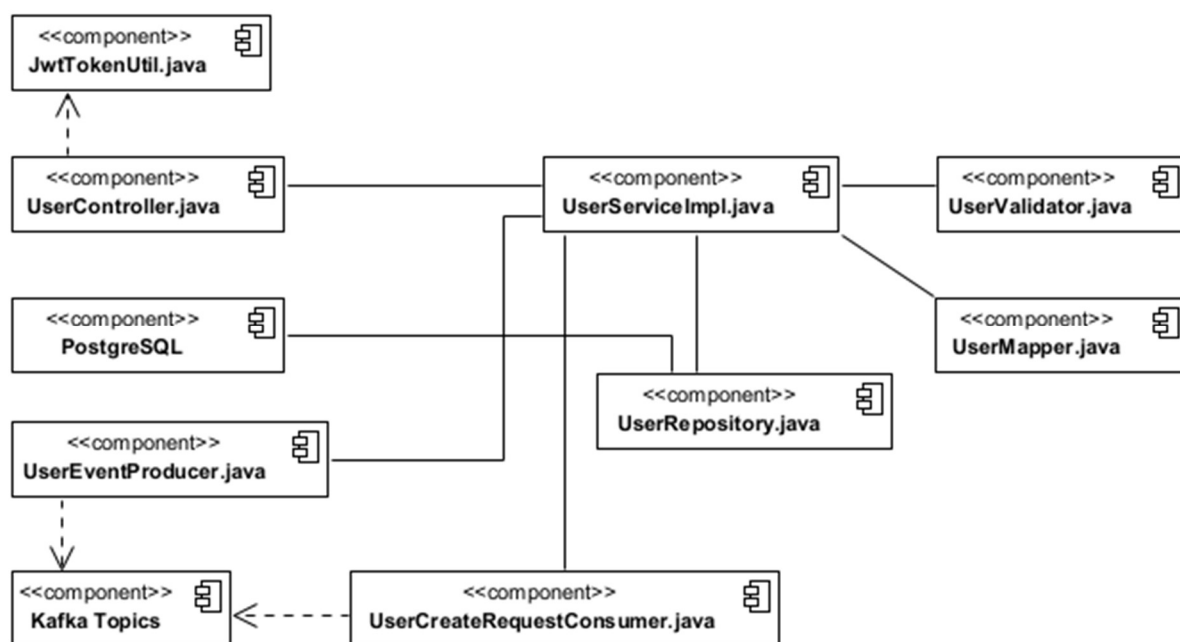


Рисунок В.4 – Диаграмма компонентов сервиса для работы с пользователями

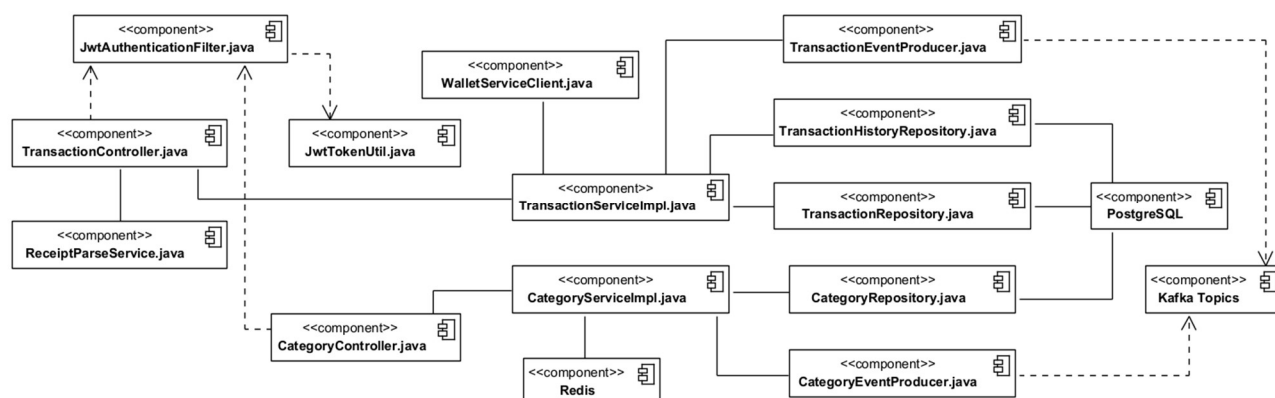


Рисунок В.5 – Диаграмма компонентов сервиса для работы с транзакциями

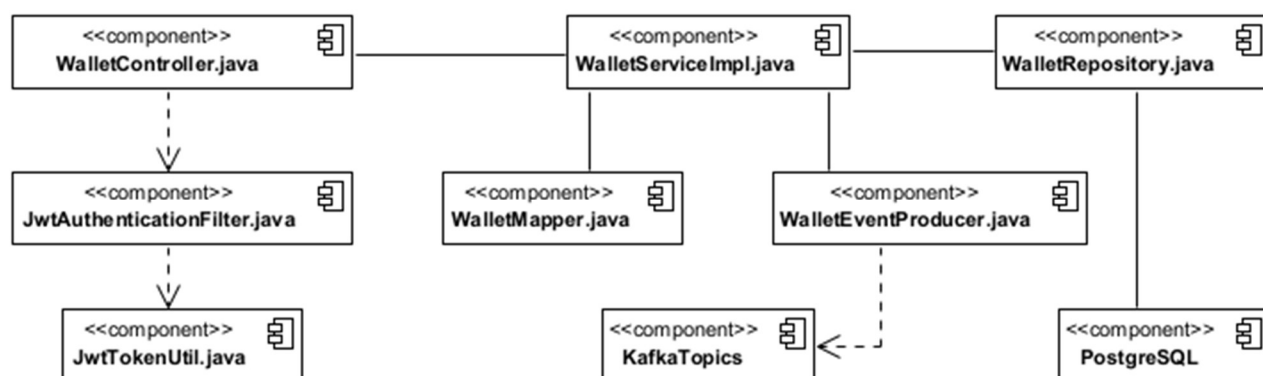


Рисунок В.6 – Диаграмма компонентов сервиса для работы с кошельками

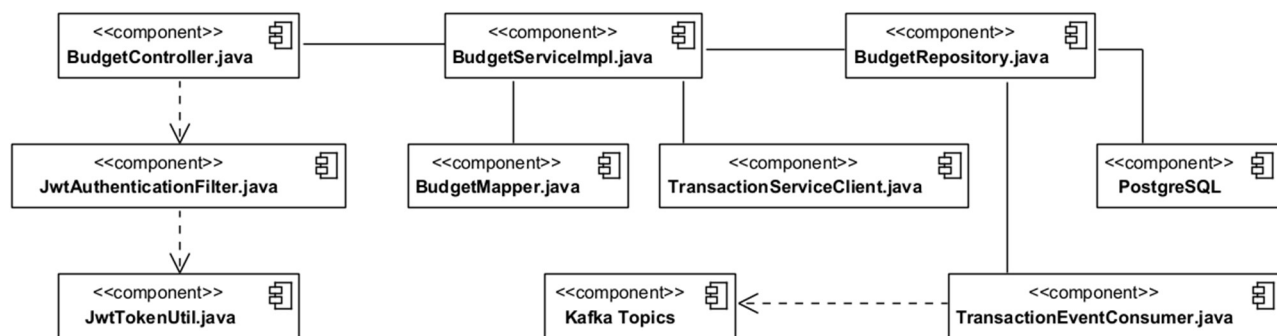


Рисунок В.7 – Диаграмма компонентов сервиса для работы бюджетами

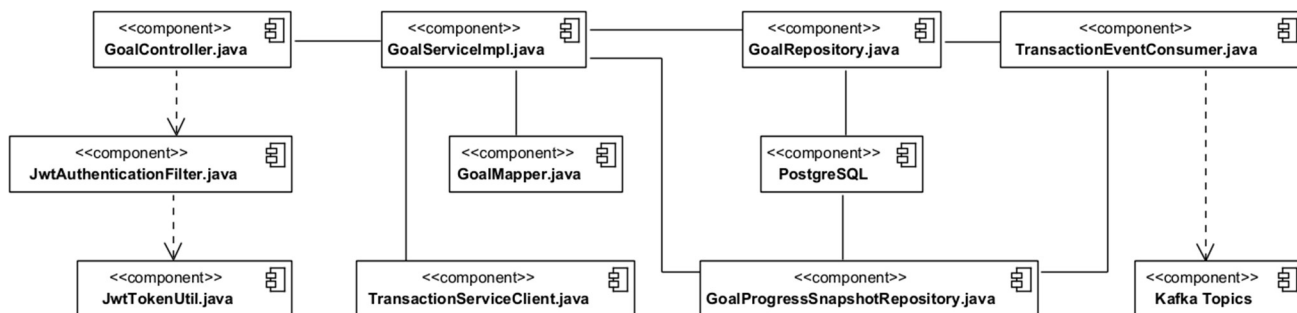


Рисунок В.8 – Диаграмма компонентов сервиса для работы с целями

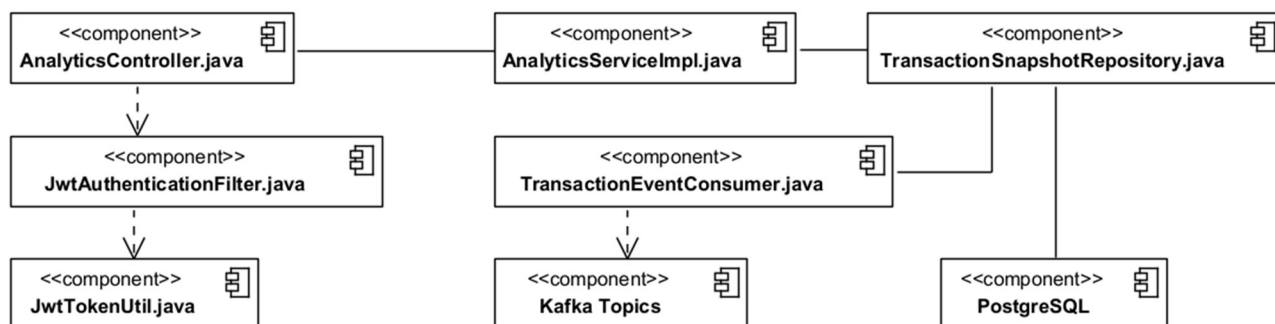


Рисунок В.9 – Диаграмма компонентов сервиса аналитики

Приложение Г
(рекомендуемое)
Результаты функционального тестирования

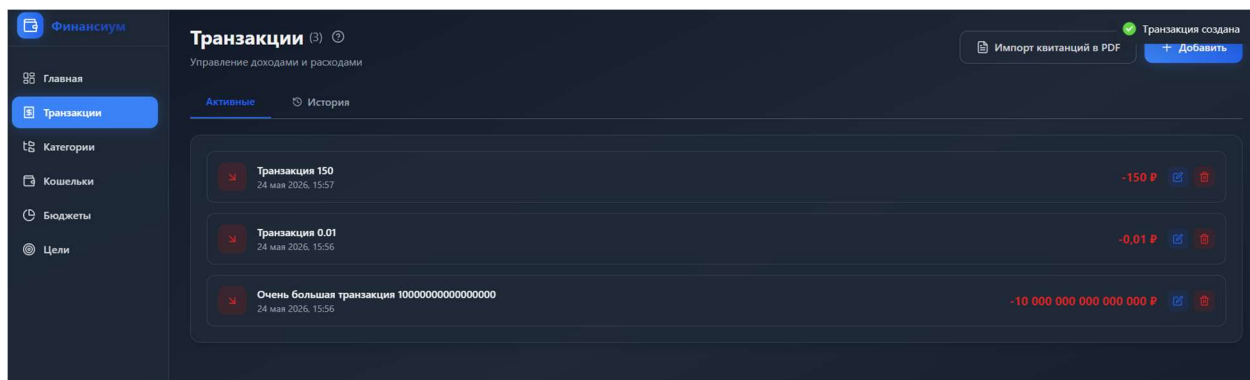


Рисунок Г.1 – Результат создания транзакции со значением «150»

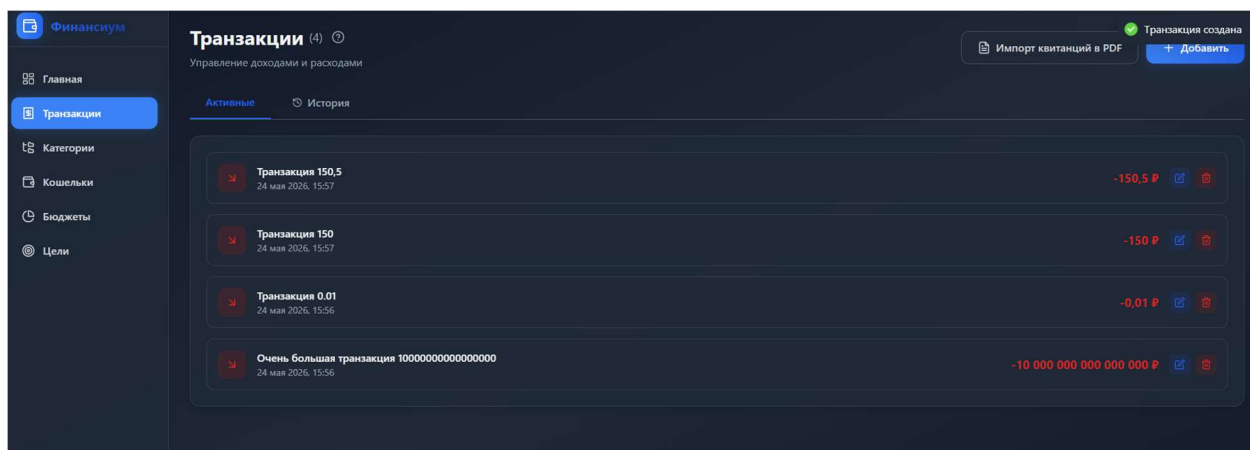


Рисунок Г.2 – Результат создания транзакции со значением «150,5»

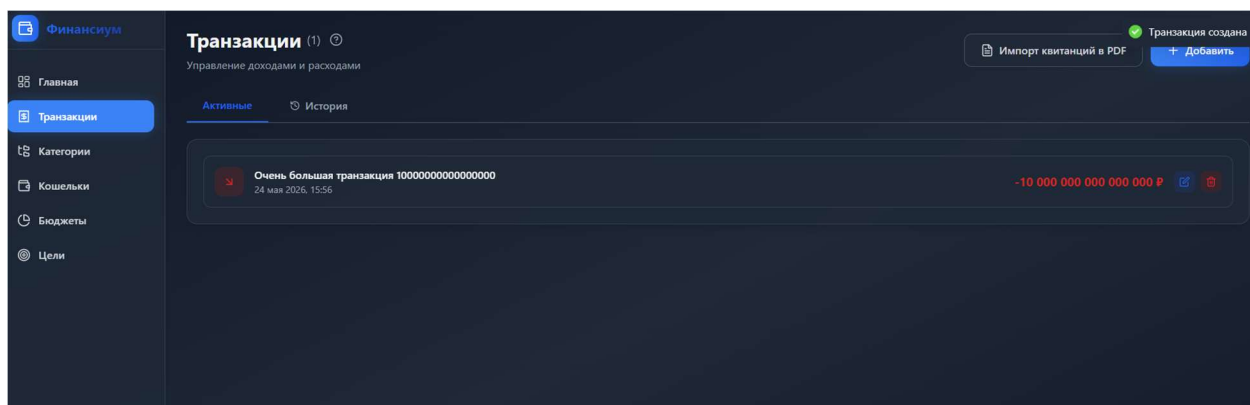


Рисунок Г.3 – Результат создания транзакции со значением
«10000000000000000»

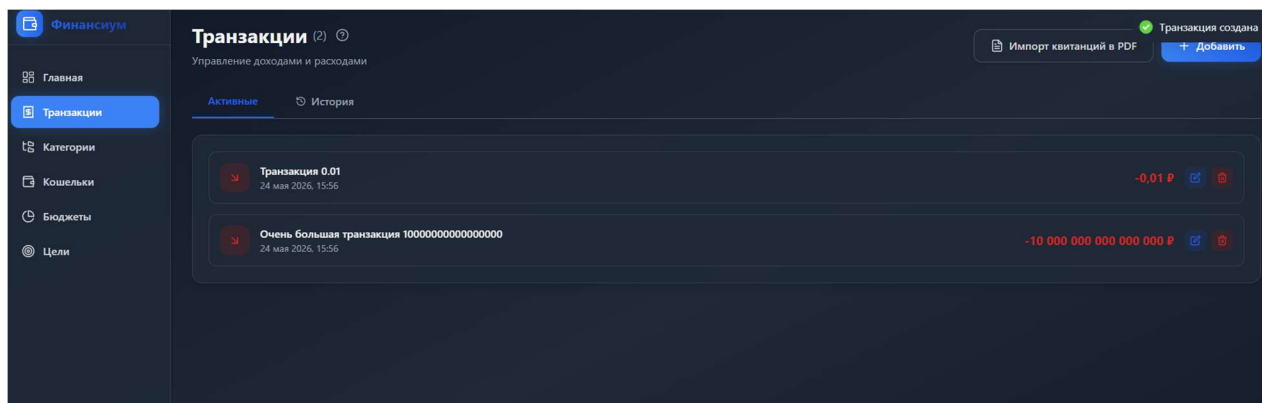


Рисунок Г.4 – Результат создания транзакции со значением «0,01»

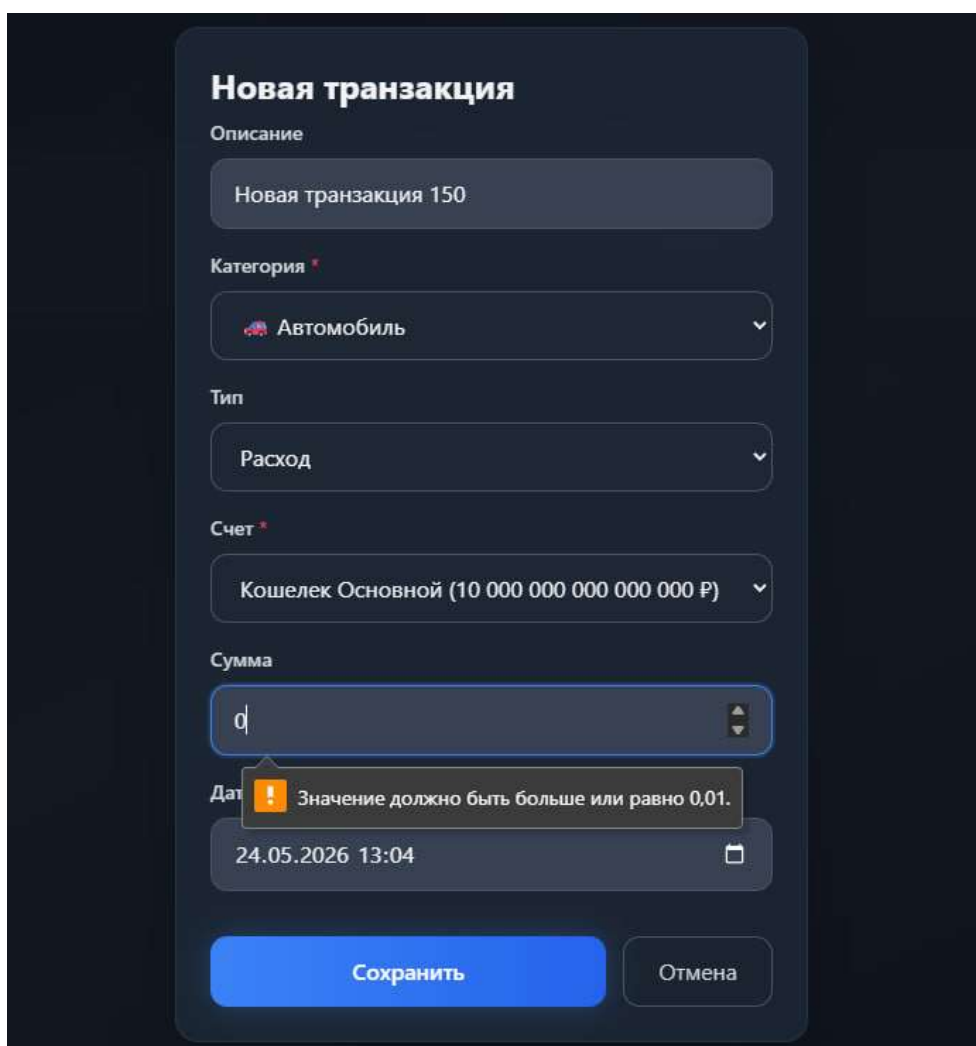


Рисунок Г.5 – Результат создания транзакции со значением «0»

Новая транзакция

Описание
Отрицательная транзакция

Категория *
Автомобиль

Тип
Расход

Счет *
Кошелек Основной (10 000 000 000 000 000 Р)

Сумма
-1

Дата
24.05.2026 13:08

Сохранить Отмена

Значение должно быть больше или равно 0,01.

Рисунок Г.6 – Результат попытки создания транзакции со значением «-1»

Новая транзакция

Описание
123123

Категория *
Автомобиль

Тип
Расход

Счет *
Кошелек Основной (10 000 000 000 000 000 Р)

Сумма
10,111

Дата
24.05.2026 13:27

Сохранить Отмена

Введите допустимое значение. Ближайшие допустимые значения: 10,11 и 10,12.

Рисунок Г.7 – Результат попытки создания транзакции со значением «10,111»

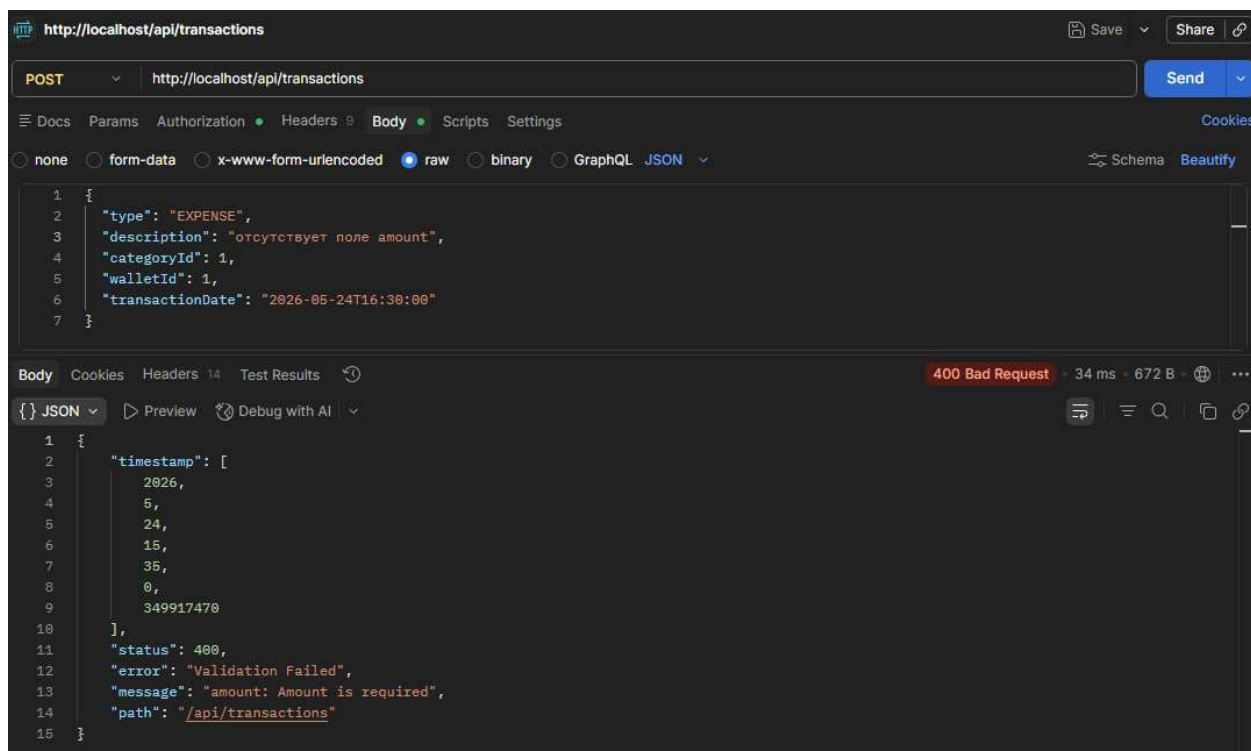


Рисунок Г.8 – Результат попытки создания транзакции с отсутствующим полем «amount»

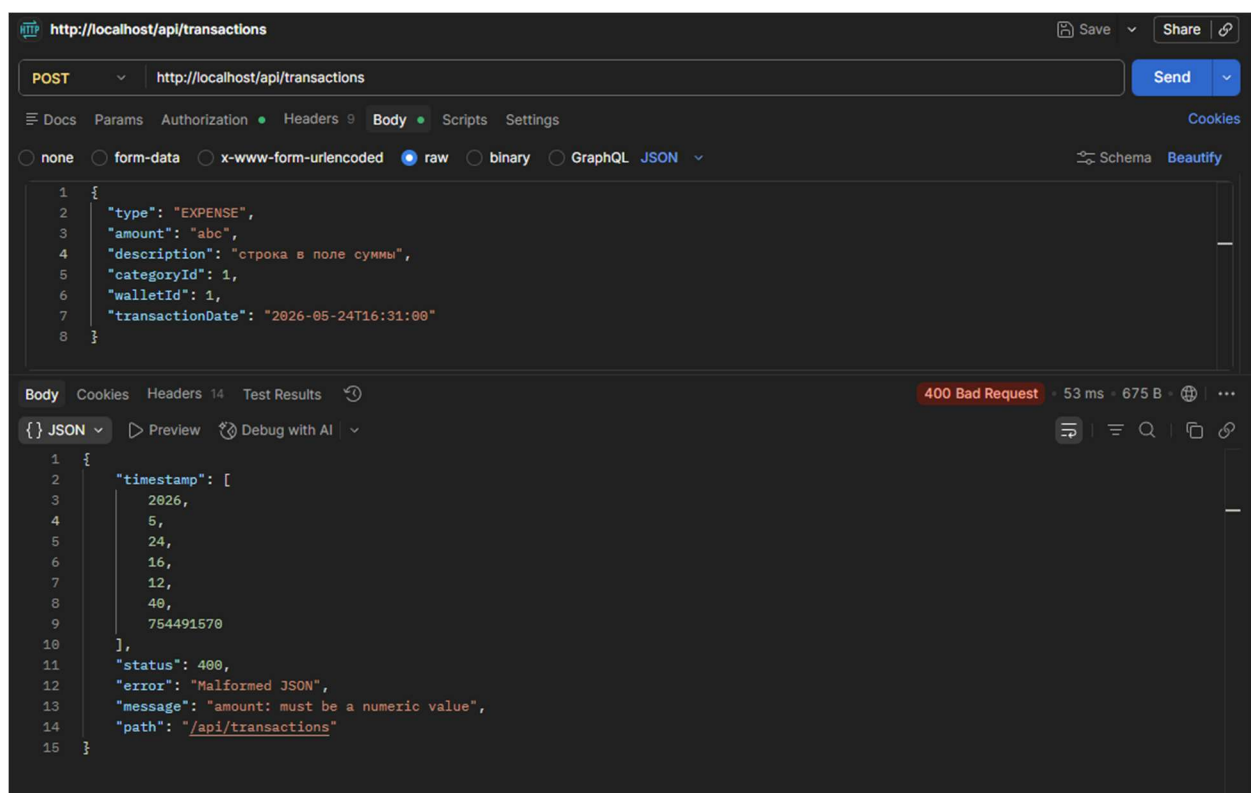


Рисунок Г.9 – Результат попытки создания транзакции со строковым значением в поле «amount»

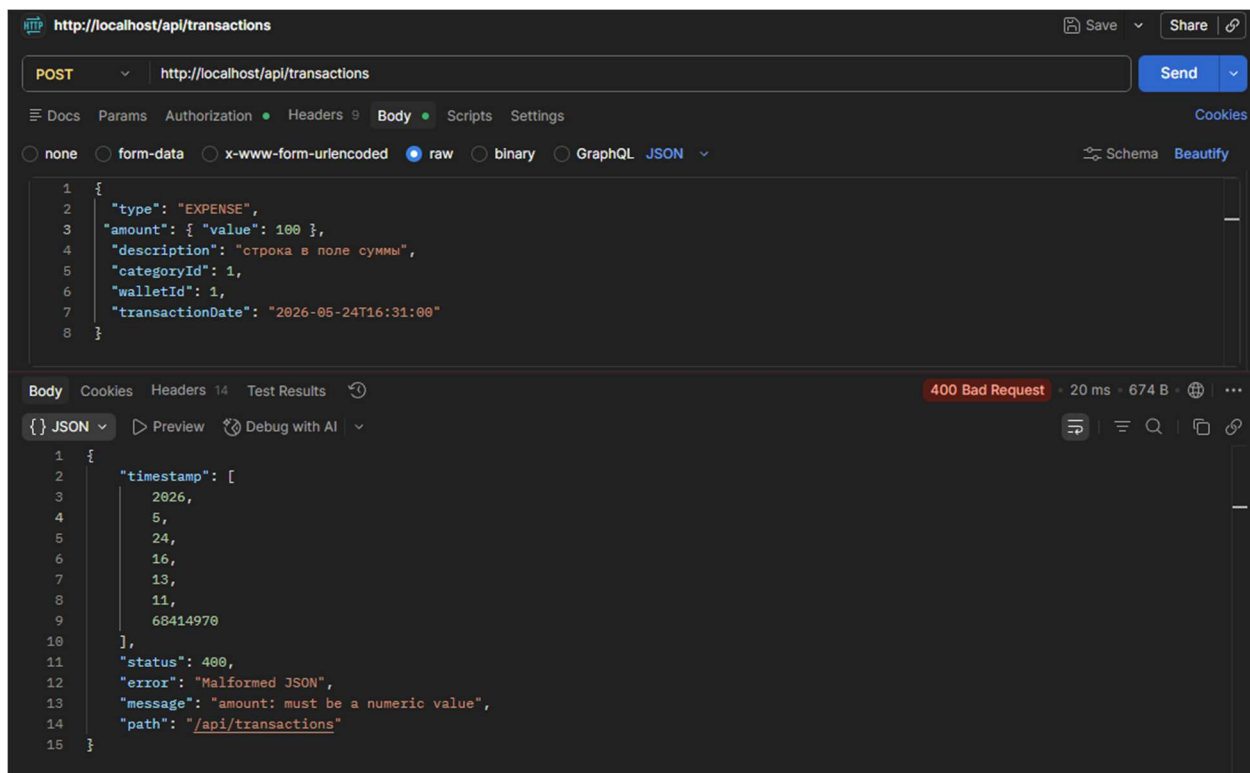


Рисунок Г.10 – Результат попытки создания транзакции с недопустимым значением в поле «amount»

Приложение Д
(рекомендуемое)
Листинг модульных тестов

Листинг Д.1 – Файл AuthServiceImpl.java

```

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.test.util.ReflectionTestUtils;
import org.springframework.web.client.RestTemplate;

import java.time.LocalDateTime;

import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.assertThatThrownBy;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyLong;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.ArgumentMatchers.eq;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("AuthServiceImpl – unit tests")
class AuthServiceImplTest {

    @Mock private CredentialsService credentialsService;
    @Mock private JwtTokenProvider jwtTokenProvider;
    @Mock private RefreshTokenService refreshTokenService;
    @Mock private CredentialsValidator credentialsValidator;
    @Mock private PasswordEncoder passwordEncoder;
    @Mock private RestTemplate restTemplate;
    @Mock private UserRequestProducer userRequestProducer;
    @Mock private TokenBlacklistService tokenBlacklistService;

    @InjectMocks
    private AuthServiceImpl authService;

    @BeforeEach
    void injectFieldMocks() {
        // tokenBlacklistService внедряется через @Autowired (не
        конструктор), поэтому
        // @InjectMocks его пропускает, устанавливаем вручную
        через ReflectionTestUtils.
        ReflectionTestUtils.setField(authService,
        "tokenBlacklistService", tokenBlacklistService);
    }

    private static final Long CRED_ID = 42L;
    private static final String USERNAME = "testuser";

```

```

        private static final String ACCESS_TOKEN =
"access.token.value";
        private static final String REFRESH_TOKEN =
"refresh.token.value";

        private Credentials buildCredentials() {
            Credentials c = new Credentials(USERNAME, "test@mail.com",
"encodedPass",
                LocalDateTime.now());
            c.setId(CRED_ID);
            c.setIsActive(true);
            return c;
        }

@Test
@DisplayName("login: успешный вход возвращает токены")
void login_success() {
    Credentials cred = buildCredentials();
    LoginRequest req = new LoginRequest();
    req.setUsernameOrEmail(USERNAME);
    req.setPassword("password123");

    when(credentialsService.findByUsernameOrEmail(USERNAME,
USERNAME)).thenReturn(cred);

when(refreshTokenService.hasActiveSession(USERNAME)).thenReturn(false);

    when(passwordEncoder.matches("password123",
"encodedPass")).thenReturn(true);
    when(jwtTokenProvider.generateAccessToken(CRED_ID,
USERNAME, Role.USER.getAuthority()))
        .thenReturn(ACCESS_TOKEN);
    when(jwtTokenProvider.generateRefreshToken(CRED_ID,
USERNAME, Role.USER.getAuthority()))
        .thenReturn(REFRESH_TOKEN);

    AuthResponse result = authService.login(req);

    assertThat(result.getAccessToken()).isEqualTo(ACCESS_TOKEN);

    assertThat(result.getRefreshToken()).isEqualTo(REFRESH_TOKEN);
    assertThat(result.getUsername()).isEqualTo(USERNAME);

    verify(credentialsService).resetFailedLoginAttempts(USERNAME);
    verify(credentialsService).updateLastLogin(USERNAME);

    verify(refreshTokenService).storeRefreshToken(eq(REFRESH_TOKEN),
eq(USERNAME), anyLong());
}

@Test

```

```

        @DisplayName("login: неверный пароль ->
AuthenticationException + счетчик неудачных попыток")
        void login_wrongPassword_throws() {
            Credentials cred = buildCredentials();
            LoginRequest req = new LoginRequest();
            req.setUsernameOrEmail(USERNAME);
            req.setPassword("wrong");

            when(credentialsService.findByUsernameOrEmail(USERNAME,
USERNAME)).thenReturn(cred);

            when(refreshTokenService.hasActiveSession(USERNAME)).thenReturn(false);

            when(passwordEncoder.matches("wrong",
"encodingPass")).thenReturn(false);

            assertThatThrownBy(() -> authService.login(req))
                .assertInstanceOf(AuthenticationException.class)
                .hasMessageContaining("Invalid credentials");

            verify(credentialsService).incrementFailedLoginAttempts(USERNAME);
        }

        @Test
        @DisplayName("login: деактивированный аккаунт ->
AuthenticationException")
        void login_deactivatedAccount_throws() {
            Credentials cred = buildCredentials();
            cred.setIsActive(false);
            LoginRequest req = new LoginRequest();
            req.setUsernameOrEmail(USERNAME);
            req.setPassword("password123");

            when(credentialsService.findByUsernameOrEmail(USERNAME,
USERNAME)).thenReturn(cred);

            when(refreshTokenService.hasActiveSession(USERNAME)).thenReturn(false);

            assertThatThrownBy(() -> authService.login(req))
                .assertInstanceOf(AuthenticationException.class)
                .hasMessageContaining("deactivated");
        }

        @Test
        @DisplayName("login: при наличии активной сессии она
отзывается перед новым входом")
        void login_existingSession_isRevoked() {
            Credentials cred = buildCredentials();
            LoginRequest req = new LoginRequest();
            req.setUsernameOrEmail(USERNAME);
            req.setPassword("password123");

```

```

        when(credentialsService.findByUsernameOrEmail(USERNAME,
        USERNAME)).thenReturn(cred);

when(refreshTokenService.hasActiveSession(USERNAME)).thenReturn(true);
        when(passwordEncoder.matches("password123",
        "encodedPass")).thenReturn(true);
        when(jwtTokenProvider.generateAccessToken(anyLong(),
        anyString(), anyString()))
            .thenReturn(ACCESS_TOKEN);
        when(jwtTokenProvider.generateRefreshToken(anyLong(),
        anyString(), anyString()))
            .thenReturn(REFRESH_TOKEN);

        authService.login(req);

verify(refreshTokenService).revokeAllUserSessions(USERNAME);
    }

    @Test
    @DisplayName("validateToken: валидный токен возвращает
    успешный ответ с данными пользователя")
    void validateToken_valid_returnsData() {

when(tokenBlacklistService.isTokenBlacklisted(ACCESS_TOKEN)).thenReturn(false);

when(jwtTokenProvider.getUserIdFromToken(ACCESS_TOKEN)).thenReturn(CRED_ID);

when(jwtTokenProvider.getUsernameFromToken(ACCESS_TOKEN)).thenReturn(USERNAME);

when(jwtTokenProvider.getRoleFromToken(ACCESS_TOKEN)).thenReturn(Role.USER.getAuthority());

        ValidationResponse result =
        authService.validateToken(ACCESS_TOKEN);

        assertThat(result.isValid()).isTrue();
        assertThat(result.getUserId()).isEqualTo(CRED_ID);
        assertThat(result.getUsername()).isEqualTo(USERNAME);
    }

    @Test
    @DisplayName("validateToken: токен в черном списке ->
    невалидный ответ")
    void validateToken_blacklisted_returnsInvalid() {

when(tokenBlacklistService.isTokenBlacklisted(ACCESS_TOKEN)).thenReturn(true);

```

```

        ValidationResponse result =
authService.validateToken (ACCESS_TOKEN);

        assertThat(result.getValid()).isFalse();
        assertThat(result.getUserId()).isNull();
        verify(jwtTokenProvider, never()).validateToken(any());
    }

    @Test
    @DisplayName("validateToken: TokenException при валидации ->
невалидный ответ (не прокидывается)")
    void validateToken_jwtException_returnsInvalid() {

when(tokenBlacklistService.isTokenBlacklisted(ACCESS_TOKEN)).thenReturn(false);
        doThrow(new
TokenException("expired")).when(jwtTokenProvider).validateToken(ACCESS_TOKEN);

        ValidationResponse result =
authService.validateToken (ACCESS_TOKEN);

        assertThat(result.getValid()).isFalse();
    }

    @Test
    @DisplayName("refreshToken: действующий refresh-token -> новый
access-token")
    void refreshToken_success() {
        RefreshTokenRequest req = new RefreshTokenRequest();
        req.setRefreshToken(REFRESH_TOKEN);

when(jwtTokenProvider.getUserIdFromToken(REFRESH_TOKEN)).thenReturn(CRED_ID);

when(jwtTokenProvider.getUsernameFromToken(REFRESH_TOKEN)).thenReturn(USERNAME);

when(jwtTokenProvider.getRoleFromToken(REFRESH_TOKEN)).thenReturn(
Role.USER.getAuthority());
        when(jwtTokenProvider.generateAccessToken(CRED_ID,
USERNAME, Role.USER.getAuthority()))
            .thenReturn(ACCESS_TOKEN);

        AuthResponse result = authService.refreshToken(req);

assertThat(result.getAccessToken()).isEqualTo(ACCESS_TOKEN);
assertThat(result.getRefreshToken()).isEqualTo(REFRESH_TOKEN);
    }

```

```

@Test
@DisplayName("refreshToken: невалидный refresh-token ->
TokenException")
void refreshToken_invalid_throws() {
    RefreshTokenRequest req = new RefreshTokenRequest();
    req.setRefreshToken("bad.token");

    doThrow(new
TokenException("invalid")).when(jwtTokenProvider).validateToken("b
ad.token");

    assertThatThrownBy(() -> authService.refreshToken(req))
        .assertInstanceOf(TokenException.class);
}

@Test
@DisplayName("logout: токены отзываются и access token
попадает в blacklist")
void logout_success() {

when(refreshTokenService.getUsernameFromToken(REFRESH_TOKEN)).then
Return(USERNAME);

when(jwtTokenProvider.getExpirationTimeMs(ACCESS_TOKEN)).thenRetur
n(60_000L);

    authService.logout(REFRESH_TOKEN, ACCESS_TOKEN);

verify(refreshTokenService).revokeAllUserTokens(REFRESH_TOKEN);
    verify(tokenBlacklistService).blacklistToken(ACCESS_TOKEN,
60_000L);
}

@Test
@DisplayName("logout: истекший access-token в blacklist не
добавляется")
void logout_expiredAccessToken_notBlacklisted() {

when(refreshTokenService.getUsernameFromToken(REFRESH_TOKEN)).then
Return(USERNAME);

when(jwtTokenProvider.getExpirationTimeMs(ACCESS_TOKEN)).thenRetur
n(0L);

    authService.logout(REFRESH_TOKEN, ACCESS_TOKEN);

    verify(tokenBlacklistService,
never()).blacklistToken(any(), anyLong());
}

@Test

```

```

        @DisplayName("changeUserRole: делегирует вызов
credentialsService")
        void changeUserRole_delegates() {
            authService.changeUserRole(USERNAME, "ADMIN");

            verify(credentialsService).changeUserRole(USERNAME,
"ADMIN");
        }
    }
}

```

Листинг Д.2 – Файл AnalyticsServiceImplTest.java

```

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.eq;
import static org.mockito.Mockito.when;

@ExtendWith(MockitoExtension.class)
@DisplayName("AnalyticsServiceImpl – unit tests")
class AnalyticsServiceImplTest {

    @Mock
    private TransactionSnapshotRepository snapshotRepository;

    @InjectMocks
    private AnalyticsServiceImpl analyticsService;

    private static final Long USER_ID = 1L;

    @Test
    @DisplayName("getDashboardSummary: корректно считает баланс и
дельту к предыдущему периоду")
    void getDashboardSummary_returnsCorrectValues() {
        LocalDate from = LocalDate.of(2025, 2, 1);
        LocalDate to = LocalDate.of(2025, 2, 28);

        when(snapshotRepository.sumIncomeByUserAndPeriod(eq(USER_ID),
any(LocalDateTime.class), any(LocalDateTime.class)))

```

```

        .thenReturn(BigDecimal.valueOf(50_000),
BigDecimal.valueOf(45_000));

when(snapshotRepository.sumExpenseByUserAndPeriod(eq(USER_ID),
any(LocalDate.class), any(LocalDate.class)))
    .thenReturn(BigDecimal.valueOf(20_000),
BigDecimal.valueOf(18_000));

    DashboardSummaryDto result =
analyticsService.getDashboardSummary(USER_ID, from, to);

assertThat(result.getTotalIncome()).isEqualByComparingTo(BigDecimal.
valueOf(50_000));

assertThat(result.getTotalExpense()).isEqualByComparingTo(BigDecimal.
valueOf(20_000));

assertThat(result.getBalance()).isEqualByComparingTo(BigDecimal.va
lueOf(30_000));

assertThat(result.getPreviousPeriodIncome()).isEqualByComparingTo(
BigDecimal.valueOf(45_000));
    }

    @Test
    @DisplayName("getDashboardSummary: нет транзакций, значит все
суммы равны нулю")
    void getDashboardSummary_noTransactions_allZero() {
        LocalDate from = LocalDate.of(2025, 1, 1);
        LocalDate to = LocalDate.of(2025, 1, 31);

when(snapshotRepository.sumIncomeByUserAndPeriod(eq(USER_ID),
any(), any()))
    .thenReturn(BigDecimal.ZERO, BigDecimal.ZERO);

when(snapshotRepository.sumExpenseByUserAndPeriod(eq(USER_ID),
any(), any()))
    .thenReturn(BigDecimal.ZERO, BigDecimal.ZERO);

    DashboardSummaryDto result =
analyticsService.getDashboardSummary(USER_ID, from, to);

assertThat(result.getTotalIncome()).isEqualByComparingTo(BigDecimal.
ZERO);

assertThat(result.getTotalExpense()).isEqualByComparingTo(BigDecimal.
ZERO);

assertThat(result.getBalance()).isEqualByComparingTo(BigDecimal.ZE
RO);

```

```

assertThat(result.getIncomeChangePercent()).isEqualByComparingTo(BigDecimal.ZERO);

assertThat(result.getExpenseChangePercent()).isEqualByComparingTo(BigDecimal.ZERO);
    }

    @Test
    @DisplayName("getDailyReport: преобразует строки репозитория в DailyReportDto")
    void getDailyReport_returnsListFromRows() {
        LocalDate from = LocalDate.of(2025, 1, 1);
        LocalDate to = LocalDate.of(2025, 1, 3);

        java.sql.Date sqlDate1 = java.sql.Date.valueOf("2025-01-01");
        java.sql.Date sqlDate2 = java.sql.Date.valueOf("2025-01-02");

        Object[] row1 = {sqlDate1, BigDecimal.valueOf(10_000), BigDecimal.valueOf(5_000)};
        Object[] row2 = {sqlDate2, BigDecimal.valueOf(0), BigDecimal.valueOf(3_000)};
        List<Object[]> rowList = new ArrayList<>();
        rowList.add(row1);
        rowList.add(row2);

        when(snapshotRepository.dailyTotals(eq(USER_ID), any(LocalDateTime.class), any(LocalDateTime.class)))
            .thenReturn(rowList);

        List<DailyReportDto> result =
analyticsService.getDailyReport(USER_ID, from, to);

        assertThat(result).hasSize(2);

        assertThat(result.get(0).getIncome()).isEqualByComparingTo(BigDecimal.valueOf(10_000));

        assertThat(result.get(0).getExpense()).isEqualByComparingTo(BigDecimal.valueOf(5_000));

        assertThat(result.get(1).getExpense()).isEqualByComparingTo(BigDecimal.valueOf(3_000));
    }

    @Test
    @DisplayName("getDailyReport: нет данных = пустой список")
    void getDailyReport_noData_emptyList() {
        when(snapshotRepository.dailyTotals(eq(USER_ID), any(), any()))
            .thenReturn(List.of());
    }

```

```

        assertThat(analyticsService.getDailyReport(USER_ID,
            LocalDate.of(2025, 1, 1), LocalDate.of(2025, 1,
31))).isEmpty();
    }

    @Test
    @DisplayName("getIncomeExpenseTrend: преобразует DailyReport")
    void getIncomeExpenseTrend_returnsTrend() {
        LocalDate from = LocalDate.of(2025, 1, 1);
        LocalDate to = LocalDate.of(2025, 1, 1);

        java.sql.Date sqlDate = java.sql.Date.valueOf("2025-01-
01");
        Object[] row = {sqlDate, BigDecimal.valueOf(7_000),
BigDecimal.valueOf(2_000)};
        List<Object[]> singleRow = new ArrayList<>();
        singleRow.add(row);

        when(snapshotRepository.dailyTotals(eq(USER_ID), any(),
any())) .thenReturn(singleRow);

        List<IncomeExpenseTrendDto> result =
analyticsService.getIncomeExpenseTrend(USER_ID, from, to);

        assertThat(result).hasSize(1);

        assertThat(result.get(0).getIncome()).isEqualByComparingTo(BigDecimal.valueOf(7_000));

        assertThat(result.get(0).getExpense()).isEqualByComparingTo(BigDecimal.valueOf(2_000));
    }

    @Test
    @DisplayName("getCategoryShare: считает процентную долю каждой категории")
    void getCategoryShare_calculatesPercent() {
        LocalDate from = LocalDate.of(2025, 1, 1);
        LocalDate to = LocalDate.of(2025, 1, 31);

        Object[] row1 = {1L, BigDecimal.valueOf(6_000)};
        Object[] row2 = {2L, BigDecimal.valueOf(4_000)};
        List<Object[]> catRows = new ArrayList<>();
        catRows.add(row1);
        catRows.add(row2);

        when(snapshotRepository.totalsByCategory(eq(USER_ID),
eq("EXPENSE"), any(), any()))
            .thenReturn(catRows);
    }

```

```

        List<CategoryShareDto> result =
analyticsService.getCategoryShare(USER_ID, "EXPENSE", from, to);

        assertThat(result).hasSize(2);
        // 6000 / (6000+4000) * 100 = 60%

assertThat(result.get(0).getPercent()).isEqualByComparingTo(BigDecimal
imal.valueOf(60));
        // 4000 / 10000 * 100 = 40%

assertThat(result.get(1).getPercent()).isEqualByComparingTo(BigDecimal
imal.valueOf(40));
    }

    @Test
    @DisplayName("getCategoryShare: нет транзакций -> пустой
список")
    void getCategoryShare_empty() {
        when(snapshotRepository.totalsByCategory(eq(USER_ID),
any(), any(), any()))
            .thenReturn(List.of());

        assertThat(analyticsService.getCategoryShare(USER_ID,
"EXPENSE",
                LocalDate.of(2025, 1, 1), LocalDate.of(2025, 1,
31))).isEmpty();
    }

    @Test
    @DisplayName("getForecast (MONTH): прогноз рассчитывается как
среднее по последним 3 месяцам")
    void getForecast_monthly_returnsAverages() {

        // репозиторий вернет одинаковые значения на каждый запрос
в цикле

when(snapshotRepository.sumIncomeByUserAndPeriod(eq(USER_ID),
any(), any()))
            .thenReturn(BigDecimal.valueOf(60_000));

when(snapshotRepository.sumExpenseByUserAndPeriod(eq(USER_ID),
any(), any()))
            .thenReturn(BigDecimal.valueOf(30_000));

        ForecastDto result = analyticsService.getForecast(USER_ID,
"MONTH");

        assertThat(result.getPeriod()).isEqualTo("MONTH");

assertThat(result.getForecastedIncome()).isEqualByComparingTo(BigDecimal
imal.valueOf(60_000));

```

```

assertThat(result.getForecastedExpense()).isEqualByComparingTo(Big
Decimal.valueOf(30_000));
    assertThat(result.getDescription()).isNotBlank();
}

@Test
@DisplayName("getForecast (WEEK): прогноз по неделям")
void getForecast_weekly_returnsAverages() {

when(snapshotRepository.sumIncomeByUserAndPeriod(eq(USER_ID),
any(), any()))
    .thenReturn(BigDecimal.valueOf(15_000));

when(snapshotRepository.sumExpenseByUserAndPeriod(eq(USER_ID),
any(), any()))
    .thenReturn(BigDecimal.valueOf(8_000));

    ForecastDto result = analyticsService.getForecast(USER_ID,
"WEEK");

    assertThat(result.getPeriod()).isEqualTo("WEEK");

assertThat(result.getForecastedIncome()).isEqualByComparingTo(BigD
ecimal.valueOf(15_000));
}

@Test
@DisplayName("getForecast: нет данных -> прогноз равен нулю")
void getForecast_noData_zeroForecast() {

when(snapshotRepository.sumIncomeByUserAndPeriod(eq(USER_ID),
any(), any()))
    .thenReturn(BigDecimal.ZERO);

when(snapshotRepository.sumExpenseByUserAndPeriod(eq(USER_ID),
any(), any()))
    .thenReturn(BigDecimal.ZERO);

    ForecastDto result = analyticsService.getForecast(USER_ID,
"MONTH");

assertThat(result.getForecastedIncome()).isEqualByComparingTo(BigD
ecimal.ZERO);

assertThat(result.getForecastedExpense()).isEqualByComparingTo(Big
Decimal.ZERO);
}
}

```

Приложение Е
(рекомендуемое)
Результаты модульного тестирования

```
Run AuthServiceImplTest x
AuthServiceImpl — unit tests (com.akerugen.authservice.service) 1 sec 585 ms
Tests passed: 12 of 12 tests - 1 sec 585 ms
login: деактивированный аккаунт -> AuthenticationException 1 sec 500 ms
refreshToken: действующий refresh-token -> новый access-token 14 ms
login: неверный пароль -> AuthenticationException + счётчик неудачных попыток 14 ms
login: успешный вход возвращает токены 0 ms
login: при наличии активной сессии она отзывается перед новым входом 0 ms
validateToken: валидный token возвращает успешный ответ с данными пользователя 0 ms
changeUserRole: делегирует вызов credentialsService 0 ms
logout: истёкший access-token в blacklist не добавляется 0 ms
validateToken: TokenException при валидации -> невалидный ответ (не прокидывается) 0 ms
logout: токены отзываются и access token попадает в blacklist 0 ms
refreshToken: невалидный refresh-token -> TokenException 7 ms
validateToken: token в чёрном списке -> невалидный ответ 3 ms
19:53:43.485 [main] ERROR com.akerugen.authservice.service.impl.AuthServiceImpl - Login failed: User account is deactivated
19:53:43.453 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Starting token refresh
19:53:43.454 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Token refreshed for user: testuser with role: ROLE_USER
19:53:43.466 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Starting login for user: testuser
19:53:43.466 [main] WARN com.akerugen.authservice.service.impl.AuthServiceImpl - Invalid password for user: testuser
19:53:43.467 [main] ERROR com.akerugen.authservice.service.impl.AuthServiceImpl - Login failed: Invalid credentials
19:53:43.482 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Starting login for user: testuser
19:53:43.483 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Login successful for user: testuser with role: ROLE_USER
19:53:43.491 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Starting login for user: testuser
19:53:43.492 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Revoking previous session for user: testuser to allow re-
19:53:43.492 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Login successful for user: testuser with role: ROLE_USER
19:53:43.510 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Changing role for user: testuser to ADMIN
19:53:43.510 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Role changed successfully for user: testuser
19:53:43.516 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Processing logout
19:53:43.517 [main] WARN com.akerugen.authservice.service.impl.AuthServiceImpl - Access token already expired, skipping blacklist
19:53:43.517 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - User testuser logged out successfully
19:53:43.526 [main] ERROR com.akerugen.authservice.service.impl.AuthServiceImpl - Token validation failed: expired
19:53:43.531 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Processing logout
19:53:43.533 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Access token added to blacklist for user: testuser, TTL:
19:53:43.533 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - User testuser logged out successfully
19:53:43.541 [main] INFO com.akerugen.authservice.service.impl.AuthServiceImpl - Starting token refresh
19:53:43.541 [main] ERROR com.akerugen.authservice.service.impl.AuthServiceImpl - Token refresh failed: invalid
19:53:43.545 [main] WARN com.akerugen.authservice.service.impl.AuthServiceImpl - Token validation failed: token is blacklisted
Process finished with exit code 0
```

Рисунок Е.1 – Результат модульного тестирования сервиса авторизации

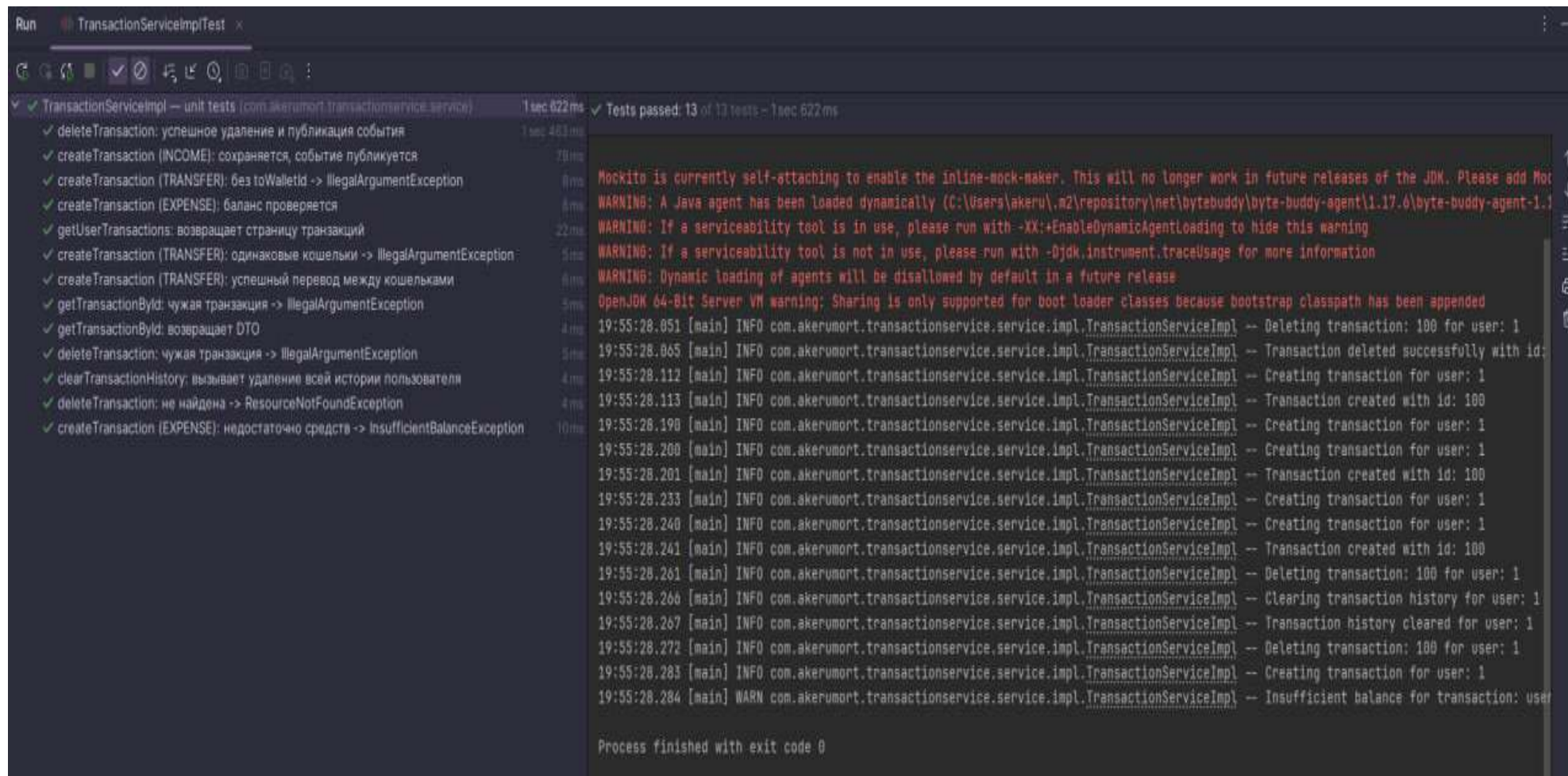


Рисунок Е.2 – Результат модульного тестирования сервиса управления транзакциями

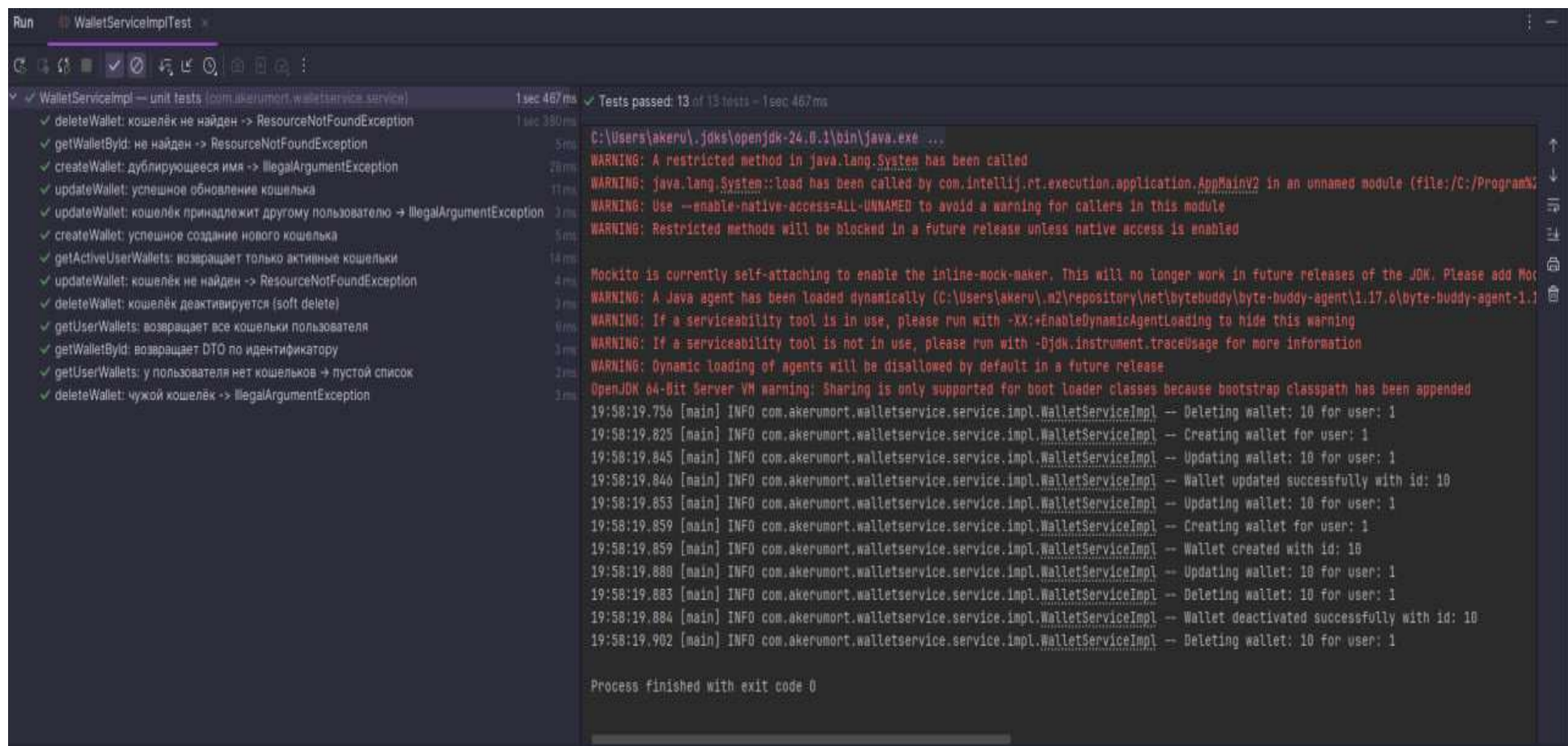


Рисунок Е.3 – Результат модульного тестирования сервиса управления кошельками

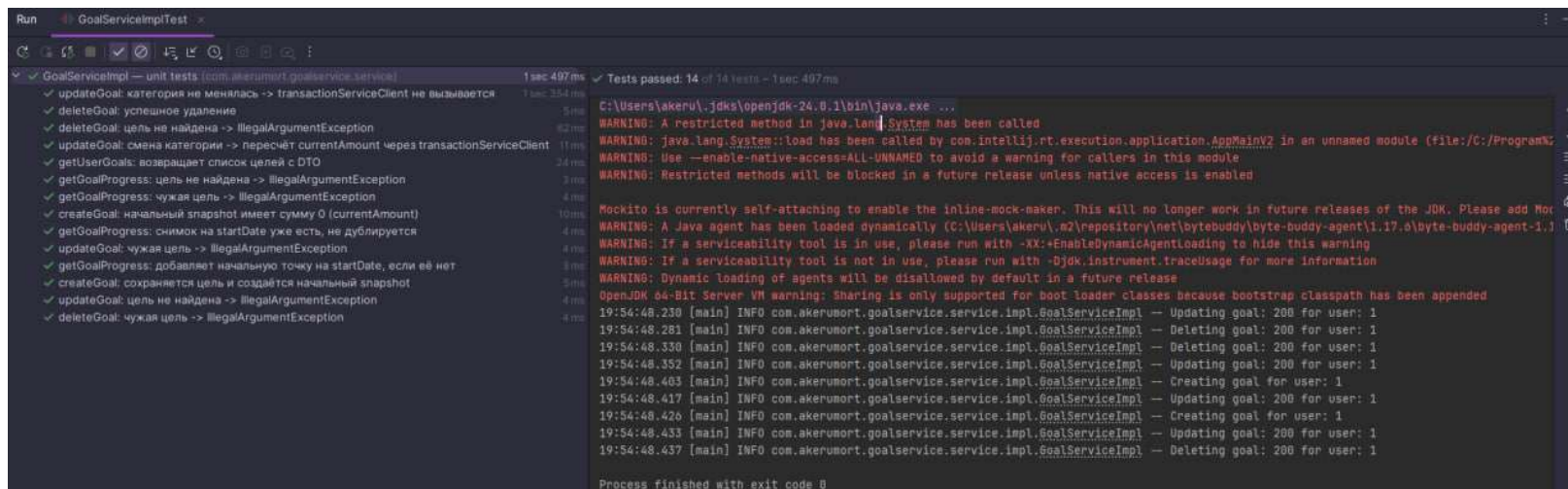


Рисунок Е.4 – Результат модульного тестирования сервиса целей

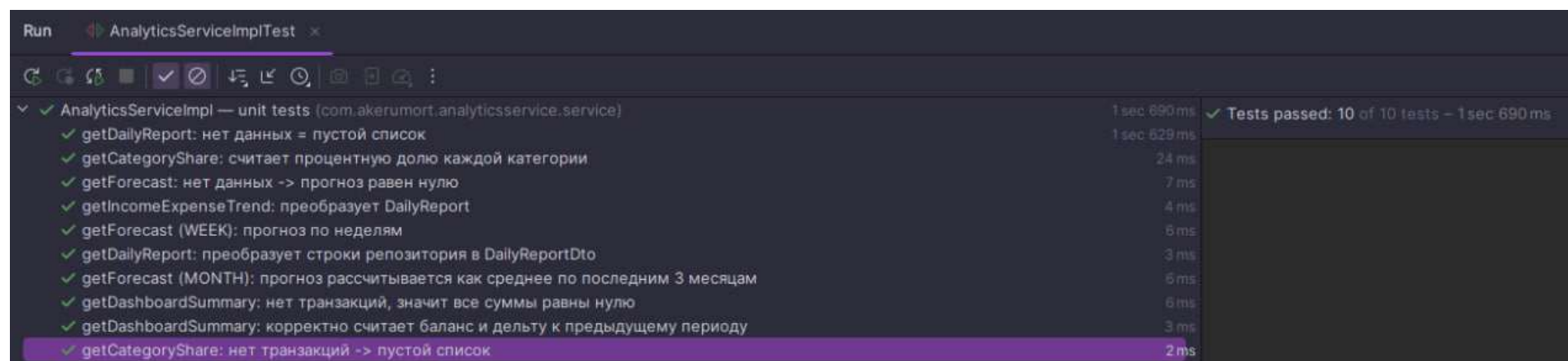


Рисунок Е.5 – Результат модульного тестирования сервиса аналитики

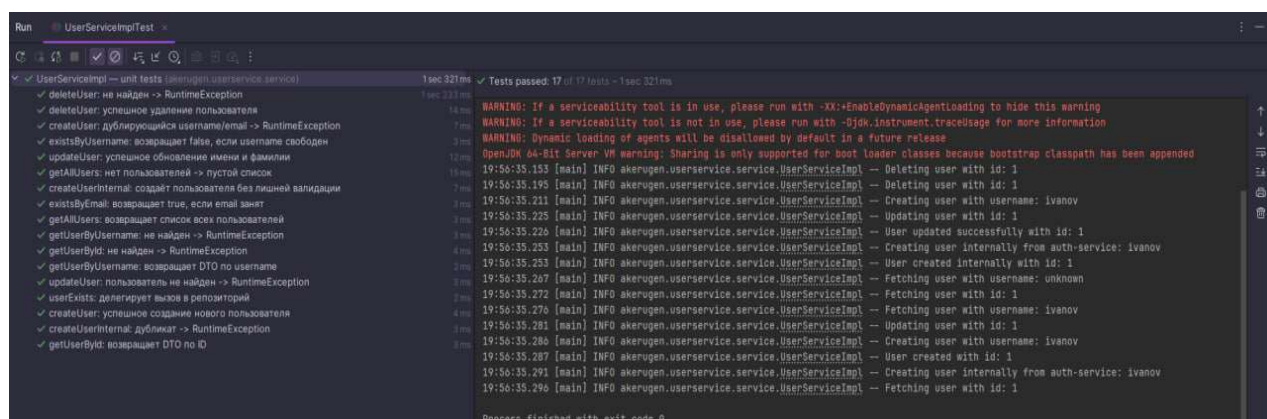


Рисунок Е.6 – Результат модульного тестирования сервиса управления пользователями

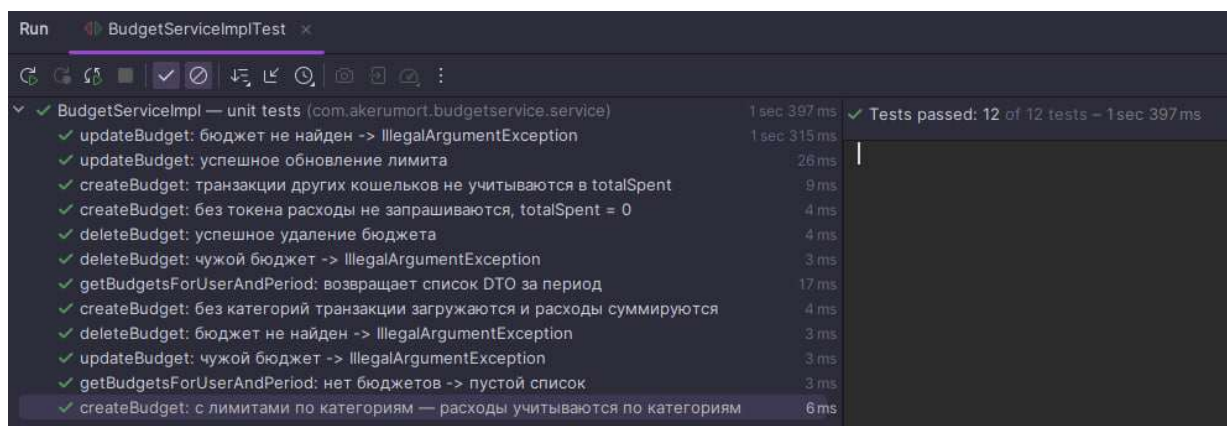


Рисунок Е.7 – Результат модульного тестирования сервиса управления бюджетами