

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	Кафедра
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.04 «Программная инженерия»
Направленность	Программная инженерия

БАКАЛАВРСКАЯ РАБОТА
на тему
Веб-приложение для ведения финансового
учета

Студент	_____	Иванов Вадим Алексеевич
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Макарычев П.П.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026. № 12)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

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

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2026

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Иванов Вадим Алексеевич гр. 22ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Веб-приложение для ведения финансового учета

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025 г.

3. Руководитель работы Макарычев П. П.

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

Назначение: ведение личного финансового учёта

Функции:

1. Управление счетами

2. Управление кредитами

3. Управление вкладами

4. Управление финансовыми целями

5. Просмотр аналитики

6. Просмотр истории транзакций

7. Редактирование личного кабинета

Технологии разработки: клиент-сервер, ООП

Языки программирования: Java, JavaScript

Среды разработки: Microsoft Visual Studio Code, JetBrains IntelliJ Idea

5. Перечень подлежащих разработке вопросов

1. Анализ аналогов и выбор прототипа

2. Анализ требований на разработку

3. Проектирование системы

4. Реализация системы

5. Тестирование системы

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 _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

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

Реферат

Пояснительная записка содержит 106 листов, 24 рисунка, 27 таблиц, 21 использованный источник, 1 листинг и 2 приложения.

Веб-приложение, ФИНАНСОВЫЙ УЧЁТ, JAVA, INTELLIJ IDEA, VISUAL STUDIO CODE, JAVASCRIPT, REACT, SPRING BOOT, POSTGRESQL.

Объектом исследования является веб-приложение для ведения финансового учёта.

Предмет исследования – технологии и методы разработки веб-приложений для автоматизации учёта и анализа личных финансов пользователей.

Цель работы – разработать веб-приложение для управления личными финансами, обеспечивающее учёт доходов и расходов, работу со счетами, вкладами, кредитами и накоплениями.

Технологии разработки – Java, Spring Boot, PostgreSQL, React, Git, Rest API.

Результат работы – разработано веб-приложение, позволяющее вести учёт финансовых операций, управлять счетами, вкладами, кредитами и получать аналитическую информацию о финансовом состоянии.

					ПГУ 09.03.04 – 08ВП221.10 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Веб-приложение для ведения финансового учета» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Иванов В.А.						
Провер.		Макарычев П.П.					4	106
Реценз.						Группа 22ВП1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ аналогов и выбор прототипа.....	8
1.1 Анализ предметной области.....	8
1.2 Анализ существующих аналогов.....	9
1.2.1 Веб-сервис ХаниМани.....	9
1.2.2 Веб-сервис DrebeDengi.....	10
1.2.3 Веб-сервис Toshl.....	11
1.2.4 Веб-сервис Дзен-мани.....	12
1.2.5 Веб-сервис YNAB.....	13
1.3 Выбор прототипа.....	14
1.4 Анализ требований.....	25
1.4.1 Анализ функциональных требований.....	25
1.4.2 Анализ нефункциональных требований.....	30
1.5 Планирование разработки веб-приложения и оценка бюджета.....	32
2 Проектирование.....	35
2.1 Проектирование базы данных веб-приложения.....	35
2.2 Проектирование архитектуры веб-приложения.....	36
2.3 Проектирование интерфейса веб-приложения.....	39
3 Реализация веб-приложения.....	42
3.1 Обоснование средств разработки веб-приложения.....	42
3.2 Реализация базы данных веб-приложения.....	42
3.3 Реализация архитектуры веб-приложения.....	44
3.4 Реализация интерфейса веб-приложения.....	45
4 Тестирование.....	51
4.1 Функциональная диаграмма и таблица решений.....	51
4.2 Метрики кода веб-приложения.....	55
Заключение.....	59
Список использованных источников.....	60

Приложение А (обязательное) Исходной код серверной части	62
Приложение Б (обязательное) Исходной код клиентской части.....	86

Введение

В условиях экономической неопределенности и роста безналичных расчётов управление личными финансами становится особенно актуальным. Люди ежедневно сталкиваются с необходимостью отслеживать доходы и расходы, планировать бюджет и контролировать финансовые цели. Рост числа банковских карт и электронных кошельков усложняет поддержание полной картины денежного потока без специализированных решений.

Эффективным средством повышения финансовой грамотности и контроля над личным бюджетом считаются веб-сервисы для ведения финансового учёта. Подобные веб-сервисы предназначены для регистрации транзакций, категоризации операций, создания бюджетов, визуализации доходов и расходов, а также для формирования аналитики. Благодаря использованию таких систем пользователь получает возможность анализировать свои траты, выявлять неэффективные статьи расходов, накапливать сбережения и достигать поставленных перед собой финансовых целей.

Целью данной выпускной квалификационной работы является создание веб-приложения для ведения личного финансового учёта, обеспечивающего регистрацию операций, категоризацию транзакций, планирование бюджета и аналитику. Для этого выполняются анализ аналогов, проектирование архитектуры веб-приложения, выбор технологий, реализация функционала и тестирование.

В процессе разработки рассматриваются вопросы организации пользовательской аутентификации, структурирования финансовых данных, построения интерактивных отчётов, обеспечения безопасности приложения и удобства и функциональности интерфейса для обеспечения возможности быстро выполнять целевые действия.

1 Анализ аналогов и выбор прототипа

1.1 Анализ предметной области

Предметная область связана с процессами ведения личного финансового учёта. В широком смысле личный финансовый учёт включает в себя: работу с наличными и безналичными средствами, распределение операций по категориям, управление сбережениями, долгами и инвестициями, помимо этого формирование различных отчётов для принятия взвешенных финансовых решений. В рамках данной работы приложение должно предоставлять пользователю возможность в ручном режиме фиксировать финансовые операции, просматривать статистику, создавать и отслеживать бюджеты и получать наглядную аналитику.

Актуальность выбранной темы обусловлена растущей необходимостью людей эффективно контролировать свои личные финансы. В условиях экономической нестабильности, высокой инфляции и активного развития безналичных платежей людям всё сложнее отслеживать все свои траты без специальных инструментов. Количество банковских карт, электронных кошельков и кредитных продуктов значительно увеличилось за последние десятилетия, следствием этого является потеря человеком полной картины своего финансового состояния. Существующие решения в виде банковских приложений или универсальных сервисов часто оказываются разрозненными, что весьма затрудняет ведение финансового учёта. При этом грамотное ведение личного бюджета напрямую влияет на финансовую стабильность домохозяйства, снижение долговой нагрузки (если таковая имеется) и достижение важных жизненных целей – от создания «подушки безопасности» до покупки недвижимости [1].

Для наилучшего понимания предметной области ниже приведены основные понятия и термины, которые будут использоваться в работе:

- 1) доход – поступление денежных средств на баланс счёта;

- 2) расход – затраты денежных средств на покупку товаров, оплату услуг, выполнение обязательств и другие нужды;
- 3) категория – способ группировки доходов и расходов по однородным признакам, который позволяет структурировать данные и проводить анализ структуры бюджета;
- 4) бюджет – финансовый план на определённый период (неделя, месяц, год), устанавливающий плановые значения доходов и лимиты расходов по категориям;
- 5) транзакция – запись о совершённой финансовой операции, содержащая дату, сумму, категорию и счёт.

На основании проведенного анализа можно сделать вывод, что разрабатываемое веб-приложение должно представлять собой удобный инструмент, позволяющий пользователю легко и, главное, быстро вести учёт доходов и расходов, создавать бюджеты, отслеживать процесс погашения кредитов, и выплаты по вкладам, получать понятную аналитику в виде графиков, диаграмм и таблиц. При этом приложение должно иметь интуитивно понятный интерфейс.

1.2 Анализ существующих аналогов

В данном разделе проводится анализ аналогов веб-приложения для ведения финансового учёта. Анализ включает в себя рассмотрение функциональных возможностей, преимуществ и недостатков каждого аналога.

Для исследования были отобраны пять популярных веб-приложений: ХаниМани, DrebeDengi, Toshi, Дзен-мани и YNAB. Данные сервисы представляют как российские, так и международные решения, что позволяет провести сравнение различных подходов к управлению личными финансами.

1.2.1 Веб-сервис ХаниМани

ХаниМани представляет собой веб-сервис для ведения личного финансового учёта [2]. Пользователь может фиксировать доходы и расходы,

формировать отчёты и контролировать задолженности. Также сервис поддерживает работу с несколькими валютами, что удобно, если пользователь использует разные валютные счета. На рисунке 1 представлена страница с отчётами веб-сервиса ХаниМани.

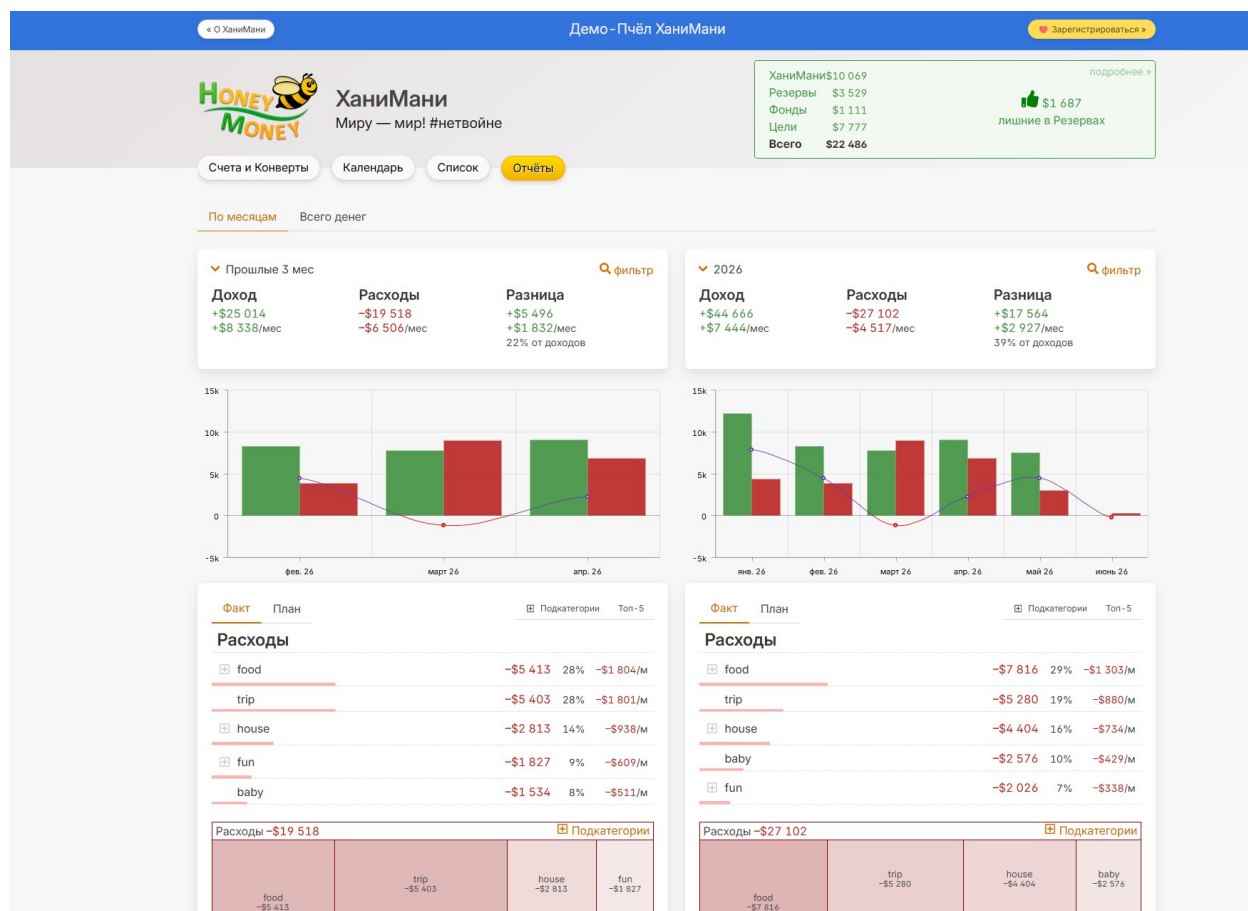


Рисунок 1 – Страница с отчётами веб-сервиса «ХаниМани»

К преимуществам ХаниМани можно отнести простой интерфейс и наличие инструментов для анализа операций. Пользователю доступны отчёты по доходам и расходам, а также средства контроля задолженностей.

Недостатком сервиса является ограниченная интеграция с банковскими приложениями, вследствие чего пользователю часто приходится вводить транзакции вручную.

1.2.2 Веб-сервис DrebeDengi

DrebeDengi – ещё одно российское веб-приложение для управления личными финансами [3]. Функциональные возможности во многом совпадают с

ХаниМани, однако пользователь имеет также возможность добавить кредит и поставить перед собой финансовую цель. На рисунке 2 представлена главная страница веб-сервиса DrebeDengi.

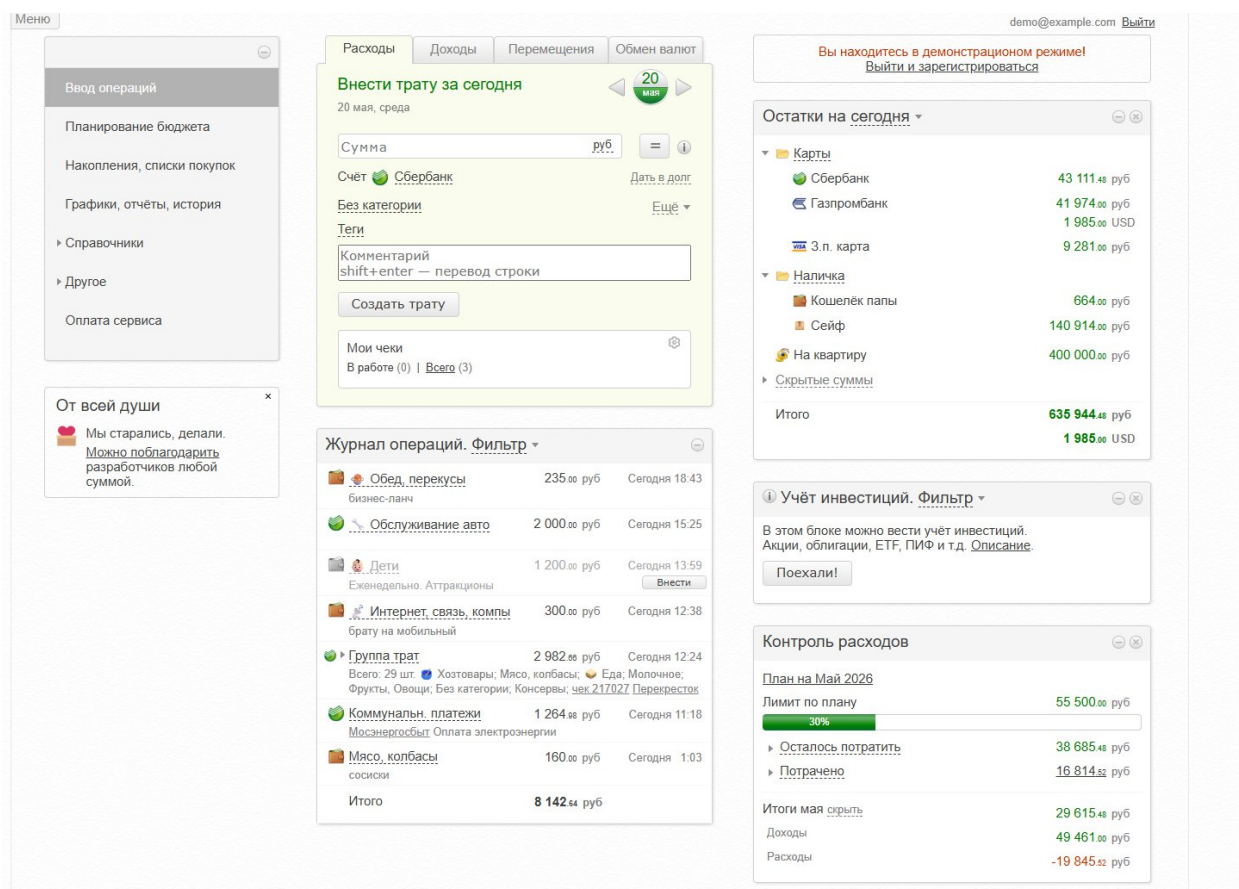


Рисунок 2 – Домашняя страница веб-сервиса «DrebeDengi»

К достоинствам сервиса можно отнести наличие широкой интеграции с банковскими приложениями, благодаря которой часть операций загружается автоматически без ручного ввода. Также пользователю доступны инструменты бюджетирования и финансового планирования.

Среди недостатков данного веб-сервиса можно выделить устаревший внешний вид интерфейса, а также ограниченное удобство работы с кредитами.

1.2.3 Веб-сервис Toshl

Toshl представляет собой веб-сервис для ведения личного финансового учёта [4]. Система позволяет регистрировать транзакции, распределять их по категориям (которые можно также создавать), планировать бюджет и

отслеживать погашение долгов. На рисунке 3 представлена вкладка планирования веб-сервиса Toshl.

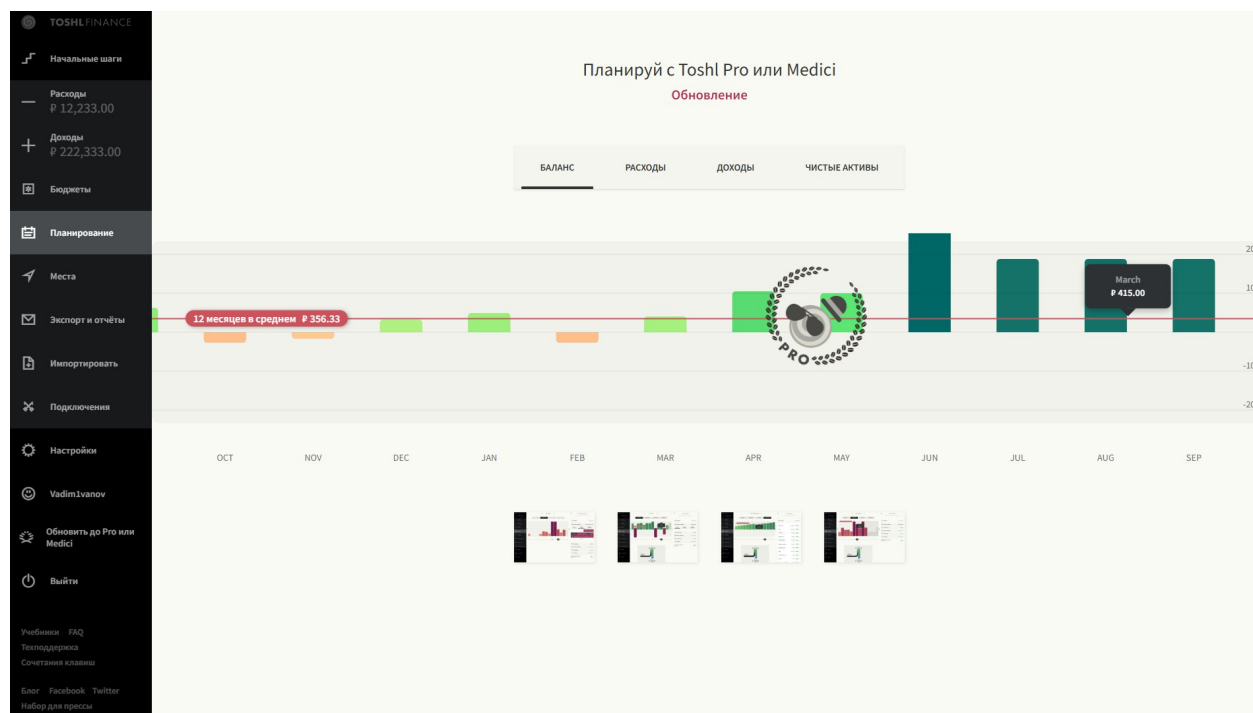


Рисунок 3 – Вкладка планирования веб-сервиса «Toshl»

К преимуществам сервиса относится поддержка нескольких валют, что делает его отличным выбором для людей, имеющих счета в различных валютах.

Недостатками является ограниченная интеграция с банковскими приложениями, из-за чего пользователям приходится вводить ручную транзакции, а также менее широкий набор возможностей по сравнению с конкурентами.

1.2.4 Веб-сервис Дзен-мани

Дзен-мани – одно из наиболее популярных российских веб-приложений для ведения личного финансового учёта [5]. Приложение имеет возможность автоматической синхронизации с банками, поддерживает бюджетирование, постановку финансовых целей, детальную аналитику и составление наглядных отчётов на основе накопленных данных. Использование визуальных инструментов упрощает представление финансовой информации. На рисунке 4 представлена главная страница сервиса Дзен-мани.

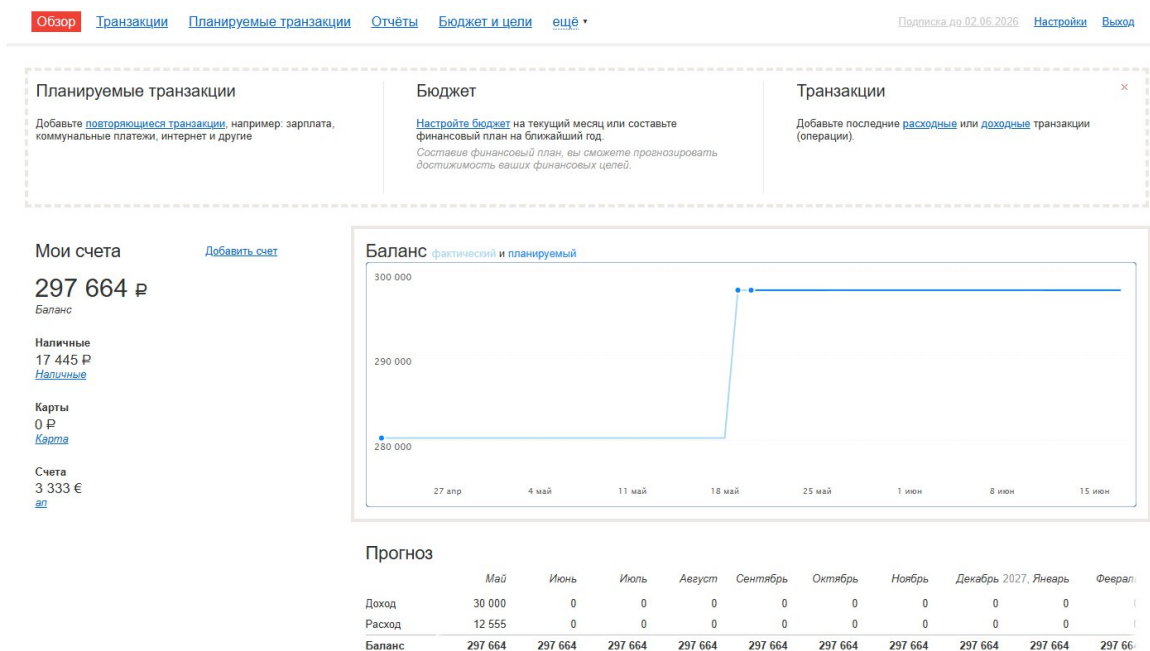


Рисунок 4 – Главная страница веб-сервиса «Дзен-мани»

Также к преимуществам сервиса можно отнести возможность сканирования чеков.

К недостаткам можно отнести интерфейс, который выглядит слишком устаревшим и уступает по дизайну конкурентам.

1.2.5 Веб-сервис YNAB

YNAB (You Need A Budget) представляет собой веб-сервис для ведения личного финансового учёта [6], основной функционал которого ориентирован на планирование бюджета. Пользователь может распределять денежные средства по категориям расходов, контролировать выполнение бюджета и отслеживать финансовые цели. Также YNAB позволяет вести учёт транзакций. Благодаря удобному интерфейсу и развитым инструментам планирования сервис помогает пользователям более рационально распоряжаться своими денежными средствами и повышать уровень финансовой дисциплины. На рисунке 5 представлена главная страница веб-сервиса YNAB.

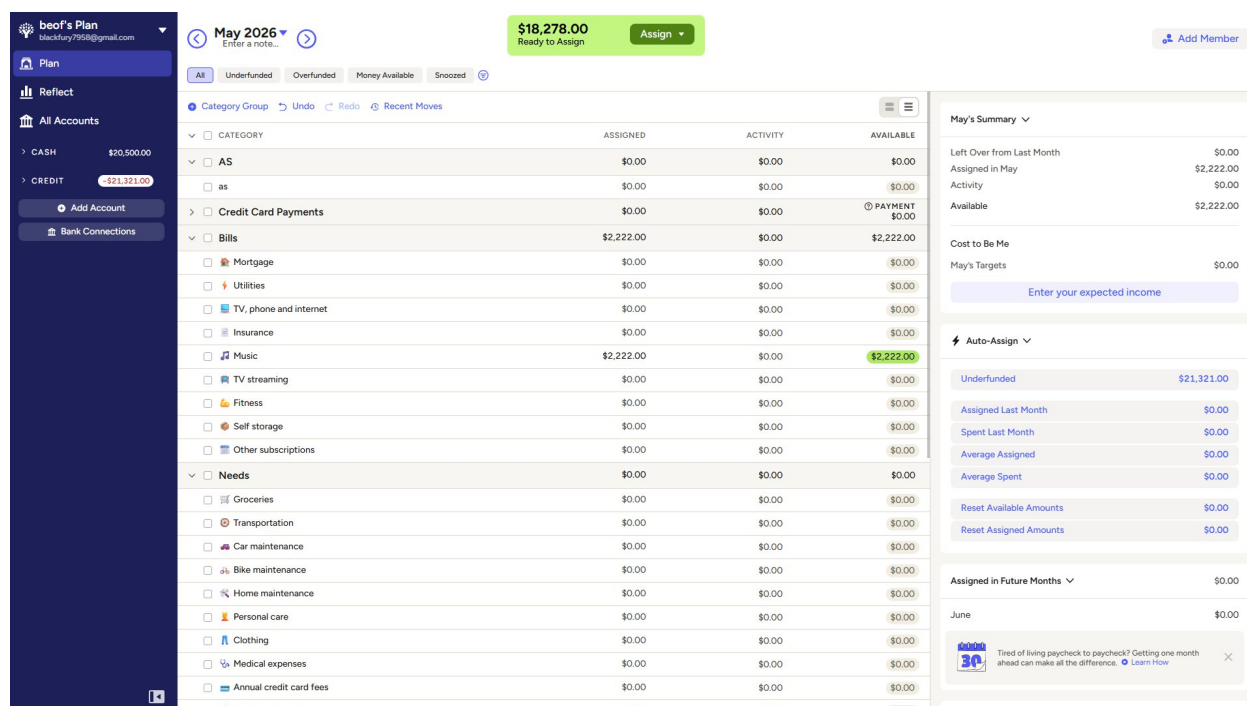


Рисунок 5 – Главная страница веб-сервиса «YNAB»

К преимуществам сервиса относится развитая система бюджетирования, которая помогает контролировать расходы и более эффективно распределять имеющиеся средства.

Недостатками являются необходимость ручного ввода транзакций из-за отсутствия интеграции с банками, а также отсутствие возможности добавлять кредиты и вклады.

1.3 Выбор прототипа

В данном разделе обосновывается выбор прототипа для разработки нового решения по ведению личного финансового учёта.

Для сравнения вариантов выбран метод анализа иерархий, поскольку он позволяет последовательно оценить альтернативы по нескольким значимым параметрам и определить наиболее предпочтительный вариант на основе совокупности критериев [7]. Также, были выделены критерии, отражающие ключевые требования к прототипу и обеспечивающие объективность дальнейшего сравнений:

Продолжение таблицы 1

	Финансо вая аналитик а	Удобство интерфейса и пользовательск ий опыт (UI/UX)	Интеграци я с банками	Поддержка нескольки х валют	Категоризаци я операций
Удобство интерфейса и пользовательс кий опыт (UI/UX)	1/3	1	1	3	5
Интеграция с банками	1/5	1	1	5	5
Поддержка нескольких валют	1/7	1/3	1/5	1	1/3
Категоризаци я операций	1/7	1/5	1/5	3	1

По критерию «Финансовая аналитика» веб-приложения можно расположить следующим образом:

1) DrebeDengi – предоставляет сведения по основным категориям расходов, динамике доходов и расходов, а также позволяет контролировать процесс накопления средств, однако сервис не поддерживает отслеживание погашения кредита и списка обязательных выплат;

2) Дзен-мани – позволяет анализировать доходы и расходы по категориям, контролировать бюджеты и финансовые цели, но при этом отсутствует возможность учёта кредитов;

3) ХаниМани – содержит инструменты анализа расходов по категориям и бюджету, а также отображает финансовые операции в календаре, при этом функционал работы с кредитами отсутствует;

4) Toshl – предоставляет лишь базовые средства анализа доходов, расходов и бюджетов, аналитические же возможности сервиса очень ограничены;

5) YNAB – ориентирован преимущественно на учёт доходов и расходов без развитых инструментов финансового анализа.

На основе проведенного сравнения была сформирована матрица попарных сравнений аналогов по критерию «Финансовая аналитика» (таблица 2).

Таблица 2 – Матрица попарных сравнений аналогов по критерию «Финансовая аналитика»

Финансовая аналитика	ХаниМани	DrebeDengi	Toshl	Дзен-мани	YNAB
ХаниМани	1	1/5	3	1/3	7
DrebeDengi	5	1	7	3	9
Toshl	1/3	1/7	1	1/5	3
Дзен-мани	3	1/3	5	1	7
YNAB	1/7	1/9	1/3	1/7	1

По критерию «Удобство интерфейса и пользовательский опыт (UI/IX)» рассматриваемые веб-приложения были расположены в следующем порядке:

1) Toshl – отличается современным оформлением, качественной визуализацией данных и удобной навигацией между разделами сервиса;

2) YNAB – обладает современным интерфейсом и хорошо продуманной структурой, что упрощает освоение основных функций сервиса;

3) ХаниМани – обеспечивает понятную навигацию и не вызывает сложностей при использовании, однако внешний вид веб-приложения выглядит менее современным по сравнению с первыми двумя сервисами;

4) DrebeDengi – основные функции доступны через боковое меню, что упрощает доступ к разделам приложения, в то же время оформление интерфейса и формы ввода данных выглядят устаревшими;

5) Дзен-мани – предоставляет простой доступ к основным функциям сервиса, но при этом дизайн веб-версии заметно уступает ранее рассмотренным аналогам.

После сравнения была сформирована матрица попарных сравнений аналогов по критерию «Удобство интерфейса и пользовательский опыт (UI/UX)» (таблица 3).

Таблица 3 – Матрица попарных сравнений аналогов по критерию «Удобство интерфейса и пользовательский опыт (UI/UX)»

Удобство интерфейса и пользовательский опыт (UI/UX)	ХаниМани	DrebeDengi	Toshl	Дзен-мани	YNAB
ХаниМани	1	3	1/5	5	1/3
DrebeDengi	1/3	1	1/7	3	1/3
Toshl	5	7	1	9	3
Дзен-мани	1/5	1/3	1/9	1	1/7
YNAB	3	3	1/3	7	1

По критерию «Интеграция с банками» аналоги были расположены в следующем порядке:

- 1) Дзен-мани – есть автоматическая синхронизация с большинством российских банков;
- 2) DrebeDengi – присутствует обработка СМС/Push от банков, импорт выписок, сканирование чеков, однако полноценная прямая API-интеграция с банками отсутствует;
- 3) Toshl – автоматическая синхронизация работает в Pro-версии в поддерживаемых странах, но для российского рынка охват крайне ограничен и практически неприменим;
- 4) ХаниМани – автоматическая синхронизация по API отсутствует;
- 5) YNAB – интеграция с банками РФ отсутствует.

Таким образом, получаем матрицу попарных сравнений аналогов по критерию «Интеграция с банками» (таблица 4).

Таблица 4 – Матрица попарных сравнений аналогов по критерию «Интеграция с банками»

Интеграция с банками	ХаниМани	DrebeDengi	Toshl	Дзен-мани	YNAB
ХаниМани	1	1/9	1/3	1/9	1
DrebeDengi	9	1	7	1/5	9
Toshl	3	1/7	1	1/7	3
Дзен-мани	9	5	7	1	9
YNAB	1	1/9	1/3	1/9	1

По критерию «Поддержка нескольких валют» аналоги можно расположить следующим образом:

- 1) Toshl – зарубежный сервис, отличающийся широкой поддержкой мультивалютного учёта, имеется более 300 валют, автоматически выполняет их конвертацию по актуальным курсам и предоставляет удобные инструменты для работы с отчетами/бюджетами в разных валютах;
- 2) DrebeDengi – поддерживается мультивалютный учет с конвертацией. Однако отсутствует суммирование баланса в разных валютах по каждому счёту;
- 3) Дзен-мани – предоставляет поддержку мультивалютного учёта и позволяет фиксировать доходы, расходы и переводы независимо от выбранной валюты, однако нет автоматической конвертации;
- 4) ХаниМани – сервис имеет хорошую реализацию мультивалютного учета с конвертацией в различные валюты. Однако отсутствует перевод со счёта с одной валютой на счёт с другой валютой;
- 5) YNAB – отсутствует поддержка нескольких валют при создании счёта.

Таким образом, получаем матрицу попарных сравнений аналогов по критерию «Поддержка нескольких валют» (таблица 5).

Таблица 5 – Матрица попарных сравнений аналогов по критерию «Поддержка нескольких валют»

Поддержка нескольких валют	ХаниМани	DrebeDengi	Toshl	Дзен-мани	YNAB
ХаниМани	1	1/3	1/5	1	9
DrebeDengi	3	1	1/3	3	9
Toshl	5	3	1	5	9
Дзен-мани	1	1/3	1/5	1	9
YNAB	1/9	1/9	1/9	1/9	1

По критерию «Категоризация операций» аналоги можно расположить следующим образом:

- 1) DrebeDengi – много категорий транзакций с возможностью создавать свои категории;
- 2) Toshl – основные категории транзакций с возможностью создавать свои категории;
- 3) Дзен-мани – есть основные категории с возможностью добавлять свои;
- 4) ХаниМани – основные категории транзакций, однако отсутствует перевод на русский язык;
- 5) YNAB – отсутствуют встроенные категории, только созданные пользователем.

Таким образом, получаем матрицу попарных сравнений аналогов по критерию «Категоризация операций» (таблица 6).

Таблица 6 – Матрица попарных сравнений аналогов по критерию «Категоризация операций»

Категоризация операций	ХаниМани	DrebeDengi	Toshl	Дзен-мани	YNAB
ХаниМани	1	1/7	1/5	1/5	7
DrebeDengi	7	1	3	3	9

Продолжение таблицы 6

Категоризация операций	ХаниМани	DrebeDengi	Toshl	Дзен-мани	YNAB
Toshl	5	1/3	1	1	7
Дзен-мани	5	1/3	1	1	7
YNAB	1/7	1/9	1/7	1/7	1

Далее необходимо вычислить весовые доли всех критериев. Сначала выполним нормировку матрицы. Для этого в первую очередь выполняется суммирование элементов каждого столбца матрицы. В результате всех вычислений получаем суммы столбцов и заносим в таблицу 7.

Таблица 7 – Суммы столбцов матрицы попарных сравнений критериев

a_{ij}	Финансов ая аналитика	Удобство интерфейса и пользовательс кий опыт (UI/UX)	Интеграц ия с банками	Поддержк а нескольки х валют	Категори зация операций
Финансовая аналитика	1	3	5	7	7
Удобство интерфейса и пользовательский опыт (UI/UX)	1/3	1	1	3	5
Интеграция с банками	1/5	1	1	5	5
Поддержка нескольких валют	1/7	1/3	1/5	1	1/3
Категоризация операций	1/7	1/5	1/5	3	1
Сумма	≈ 1,819	≈ 5,533	7,4	19	≈ 18,333

Далее делим все значения матрицы на сумму элементов соответствующего столбца и получаем нормированную матрицу (таблица 8).

Таблица 8 – Нормированная матрица сравнений критериев

A_{ij}	Финансов ая аналитик а	Удобство интерфейса и пользовательс кий опыт (UI/UX)	Интеграци я с банками	Поддержк а нескольки х валют	Категори зация операций
Финансовая аналитика	0,550	0,542	0,676	0,368	0,382
Удобство интерфейса и пользовательский опыт (UI/UX)	0,183	0,181	0,135	0,158	0,273
Интеграция с банками	0,110	0,181	0,135	0,263	0,273
Поддержка нескольких валют	0,078	0,060	0,027	0,052	0,018
Категоризация операций	0,078	0,036	0,027	0,158	0,055

Вычисляем среднее арифметическое каждой строки и получаем веса критериев (таблица 9).

Таблица 9 – Веса критериев

Критерий	Вес в долях	Вес в процентах
Финансовая аналитика	$\approx 0,504$	50,4 %
Удобство интерфейса и пользовательский опыт (UI/UX)	$\approx 0,186$	18,6 %
Интеграция с банками	$\approx 0,192$	19,2 %
Поддержка нескольких валют	$\approx 0,034$	3,4 %
Категоризация операций	$\approx 0,070$	7 %

В результате получаем наиболее весомый критерий «Финансовая аналитика», менее весомые критерии «Интеграция с банками» и «Удобство интерфейса и пользовательский опыт (UI/UX)» и наименее весомые критерии «Категоризация операций» и «Поддержка нескольких валют».

Далее аналогично вычисляем веса для аналогов по каждому критерию. Результаты вычислений для критерия «Финансовая аналитика» продемонстрированы в таблице 10.

Таблица 10 – Веса веб-сервисов по критерию «Финансовая аналитика»

Финансовая аналитика	Вес в долях	Вес в процентах
ХаниМани	$\approx 0,146$	14,6 %
DrebeDengi	$\approx 0,498$	49,8 %
Toshl	$\approx 0,066$	6,6 %
Дзен-мани	$\approx 0,257$	25,7 %
YNAB	$\approx 0,033$	3,3 %

Результаты вычислений для критерия «Удобство интерфейса и пользовательский опыт (UI/UX)» продемонстрированы в таблице 11.

Таблица 11 – Веса веб-сервисов по критерию «Удобство интерфейса и пользовательский опыт (UI/UX)»

Удобство интерфейса и пользовательский опыт (UI/UX)	Вес в долях	Вес в процентах
ХаниМани	$\approx 0,139$	13,9 %
DrebeDengi	$\approx 0,075$	7,5 %
Toshl	$\approx 0,511$	51,1 %
Дзен-мани	$\approx 0,035$	3,5 %
YNAB	$\approx 0,240$	24 %

Результаты вычислений для критерия «Интеграция с банками» продемонстрированы в таблице 12.

Таблица 12 – Веса веб-сервисов по критерию «Интеграция с банками»

Интеграция с банками	Вес в долях	Вес в процентах
ХаниМани	$\approx 0,039$	3,9 %
DrebeDengi	$\approx 0,303$	30,3 %
Toshl	$\approx 0,087$	8,7 %
Дзен-мани	$\approx 0,531$	53,1 %
YNAB	$\approx 0,039$	3,9 %

Результаты вычислений для критерия «Поддержка нескольких валют» продемонстрированы в таблице 13.

Таблица 13 – Веса веб-сервисов по критерию «Поддержка нескольких валют»

Поддержка нескольких валют	Вес в долях	Вес в процентах
ХаниМани	$\approx 0,124$	12,4 %
DrebeDengi	$\approx 0,245$	24,5 %
Toshl	$\approx 0,480$	48 %
Дзен-мани	$\approx 0,124$	12,4 %
YNAB	$\approx 0,026$	2,6 %

Результаты вычислений для критерия «Категоризация операций» продемонстрированы в таблице 14.

Таблица 14 – Веса веб-сервисов по критерию «Категоризация операций»

Категоризация операций	Вес в долях	Вес в процентах
ХаниМани	$\approx 0,086$	8,6 %
DrebeDengi	$\approx 0,464$	46,4 %
Toshl	$\approx 0,210$	21 %
Дзен-мани	$\approx 0,210$	21 %
YNAB	$\approx 0,030$	3 %

Далее собираем результаты всех вычислений для критериев в одну матрицу (таблица 15).

Общая матрица весов отражает значимость конкретного критерия для конкретного аналога.

Таблица 15 – Общая матрица весов

Веб-сервис	Финансовая аналитика	Удобство интерфейса и пользовательский опыт (UI/UX)	Интеграция с банками	Поддержка нескольких валют	Категоризация операций
ХаниМани	0,146	0,139	0,039	0,124	0,086
DrebeDengi	0,498	0,075	0,303	0,245	0,464
Toshl	0,066	0,511	0,087	0,480	0,210
Дзен-мани	0,257	0,035	0,531	0,124	0,210
YNAB	0,033	0,240	0,039	0,026	0,030

Для выбора наилучшего аналога умножаем получившуюся матрицу на матрицу весов критериев. Результат вычислений приведен в таблице 16.

Таблица 16 – Итоговые веса веб-сервисов

Веб-сервис	Вес в долях	Вес в процентах
ХаниМани	0,117	11,7 %
DrebeDengi	0,364	36,4 %
Toshl	0,176	17,6 %
Дзен-мани	0,257	25,7 %
YNAB	0,072	7,2 %

По итогам анализа было выявлено, что веб-сервис DrebeDengi является оптимальным выбором, так как он набрал весовую долю в 36,4 процента. Также веб-сервис DrebeDengi отличается интуитивно понятным интерфейсом и не требует специальных финансовых знаний, что делает его наиболее подходящим сервисом, который можно использовать в качестве прототипа.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

На основе проведённого анализа предметной области, а также изучения существующих аналогов (таких как личные финансовые менеджеры и

банковские приложения) был определён набор функциональных требований, которым должно соответствовать разрабатываемое веб-приложение. Эти требования охватывают ключевые операции: регистрацию доходов и расходов, управление категориями, создание бюджетов, создание вкладов и кредитов, отслеживание процесса накопления и формирование аналитических отчётов [8].

Для визуализации состава функций и взаимодействия пользователей с системой была построена диаграмма вариантов использования в нотации UML (Use Case Diagram). В рамках объектно-ориентированного проектирования данный вид диаграмм применяется для формального описания того, каким образом пользователь взаимодействует с программной системой и какие функции доступны ему в процессе работы. Она позволяет выделить границы системы, определить состав внешних актёров и зафиксировать набор сценариев, отражающих основные способы использования приложения. В отличие от текстового описания требований, диаграмма вариантов использования даёт более наглядное представление о логике работы системы и облегчает понимание её структуры на ранних этапах проектирования. Кроме того, она служит удобной основой для последующей детализации функциональности и согласования требований между этапами анализа и разработки.

Кроме того, использование UML-диаграмм позволяет структурировать требования и обеспечить их однозначную интерпретацию при дальнейшей реализации проекта. Построение диаграммы вариантов использования является важным этапом проектирования, поскольку оно помогает связать выявленные требования с конкретными действиями пользователя в системе. Такой подход упрощает последующую разработку функциональных модулей и делает структуру приложения более логичной и понятной. В результате проектирование становится более последовательным, а сам процесс реализации более удобным для дальнейшего сопровождения.

На рисунке 6 представлена диаграмма вариантов использования для веб-приложения ведения личного финансового учёта, отражающая весь выявленный функционал, связанный с управлением личными финансами.

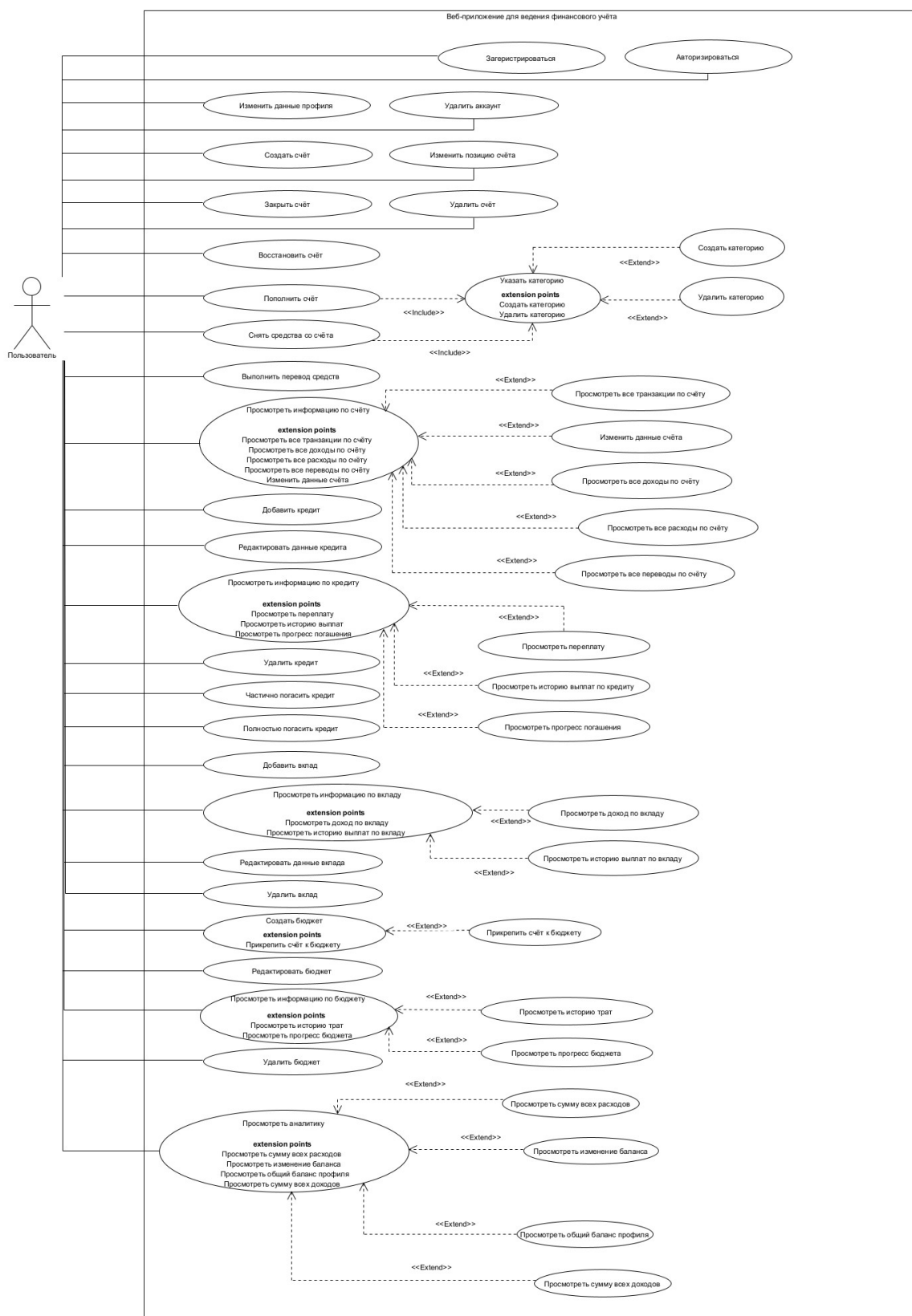


Рисунок 6 – Диаграмма вариантов использования веб-приложения для ведения
финансового учёта

Спецификация варианта использования «Пополнить счёт» представлена в таблице 17.

Таблица 17 – Спецификация варианта использования «Пополнить счёт»

Прецедент: пополнить счёт
ID: 1
Краткое содержание: пользователь выбирает счёт, категорию пополнения и сумму пополнения
Актеры: пользователь
Предусловия: <ol style="list-style-type: none"> 1. пользователь авторизовался в системе; 2. пользователь создал счёт.
Основной поток: <ol style="list-style-type: none"> 1. пользователь заходит на страницу управления счетами; 2. нажимает на кнопку «Пополнить» в карточке счёта; 3. вводит сумму пополнения, выбирает категорию; 4. нажимает на кнопку «Пополнить».
Постусловие: <ol style="list-style-type: none"> 1. Счёт пополнен

Спецификация варианта использования «Пополнить счёт» представлена в таблице 18.

Таблица 18 – Спецификация варианта использования «Создать счёт»

Прецедент: создать счёт
ID: 2
Краткое содержание: пользователь создаёт новый счёт
Актеры: пользователь
Предусловия: <ol style="list-style-type: none"> 1. пользователь авторизован в системе.

Продолжение таблицы 18

Основной поток:
1. пользователь заходит во вкладку управления счетами;
2. пользователь нажимает на кнопку «Создать счёт»;
3. пользователь вводит данные счёта: название, валюту, начальный баланс;
4. нажимает кнопку «Создать».
Постусловие:
1. счёт создан;
2. счёт отображается во вкладке управления счетами.

Спецификация варианта использования «Изменить данные профиля» представлена в таблице 19.

Таблица 19 – Спецификация варианта использования «Изменить данные профиля»

Прецедент: изменить данные профиля
ID: 3
Краткое содержание: пользователь изменяет личные данные аккаунта
Актеры: пользователь
Предусловия:
1. пользователь авторизован в системе.
Основной поток:
1. пользователь заходит во вкладку «Личный кабинет»;
2. пользователь вводит данные, которые хочет изменить;
3. нажимает на кнопку «Сохранить изменения».
Постусловие:
1. данные профиля обновлены.

В данном разделе был произведен анализ функциональных требований, были уточнены ключевые функции разрабатываемой системы и описаны основные способы взаимодействия пользователя с системой, а также создана соответствующая диаграмма вариантов использования.

Результаты анализа служат основой для последующего этапа проектирования веб-приложения.

1.4.2 Анализ нефункциональных требований

Помимо функциональных возможностей, разрабатываемое веб-приложение для ведения личного финансового учёта должно соответствовать ряду нефункциональных требований. Данные требования определяют характеристики качества системы, её производительность, удобство использования, безопасность и надёжность. Соблюдение нефункциональных требований является важным условием создания удобного, стабильного и безопасного программного продукта [9].

Одним из ключевых требований является удобство пользовательского интерфейса. Приложение должно обладать интуитивно понятной структурой и простой навигацией, чтобы пользователь мог быстро выполнять основные операции без необходимости длительного изучения системы. Интерфейс должен обеспечивать удобное добавление транзакций, просмотр отчётов, управление счетами, бюджетами, кредитами и вкладами. Особое внимание должно уделяться визуальному представлению информации, поскольку финансовые данные легче воспринимаются в виде графиков, диаграмм и прогресс-баров.

Важным нефункциональным требованием также является производительность системы. Веб-приложение должно обеспечивать быстрое выполнение основных операций, включая загрузку страниц, обработку пользовательских действий, формирование отчётов и отображение аналитической информации. Интерфейс системы должен работать плавно и без заметных задержек при стандартной пользовательской нагрузке. Для обеспечения высокой скорости работы предполагается использование современной серверной архитектуры [10].

Отдельное внимание уделяется надёжности хранения данных. Система должна обеспечивать корректное сохранение информации о транзакциях, счетах, кредитах, вкладах и финансовых целях. Для этого необходимо использование надёжной системы управления базами данных (СУБД) PostgreSQL.

Одним из наиболее значимых требований является безопасность приложения. Поскольку система хранит персональные и финансовые данные пользователей, необходимо обеспечить защиту информации от несанкционированного доступа. Для этого должна использоваться система аутентификации и авторизации пользователей. Пароли должны храниться в зашифрованном виде с применением современных алгоритмов хеширования. Также необходимо обеспечить защиту пользовательских сессий и ограничить доступ к данным только владельцу аккаунта.

Немаловажным требованием является поддержка программного продукта. Код приложения должен иметь модульную структуру, обеспечивающую удобство дальнейшей доработки и тестирования. Использование фреймворка Spring Framework позволяет разделить бизнес-логику, уровень доступа к данным и пользовательский интерфейс, что упрощает поддержку проекта и внесение изменений.

Дополнительно к указанным требованиям следует добавить необходимость обеспечения совместимости веб-приложения с различными браузерами. Система должна корректно функционировать и отображаться в таких браузерах как Google Chrome, Mozilla Firefox, Microsoft Edge, Yandex Browser. Также важно обеспечить адаптивность визуального интерфейса для корректного отображения на устройствах с различным разрешением экрана, включая настольные компьютеры, ноутбуки и мобильные устройства. Это позволит расширить доступность приложения для пользователей и повысить удобство его использования в различных условиях эксплуатации.

Таким образом, выполнение сформулированных нефункциональных требований позволит создать стабильное, безопасное и удобное веб-приложение для ведения личного финансового учёта, способное обеспечивать эффективную работу пользователей и дальнейшее развитие системы. В результате разрабатываемое приложение сможет эффективно решать задачи управления личными финансами и обеспечивать высокий уровень пользовательского опыта, способствуя повышению финансовой грамотности пользователей.

1.5 Планирование разработки веб-приложения и оценка бюджета

Перед началом проектирования веб-приложения необходимо выполнить планирование работ и предварительную оценку бюджета проекта. Данный этап позволяет определить последовательность выполнения задач, распределить трудозатраты по стадиям разработки, а также оценить требуемые временные и ресурсные затраты. Для структурирования процесса планирования будет использоваться программа ProjectLibre, которая обеспечивает возможность построения календарного графика выполнения проекта, распределения задач между этапами и контроля сроков реализации этих задач. В рамках планирования выделяются основные этапы разработки, включающие анализ аналогов и выбор прототипа, проектирование веб-приложения, реализацию и тестирование. Каждому этапу назначаются определённые временные рамки с учётом сложности выполняемых задач и их взаимосвязи. Задачи проекта, выявленные в ходе планирования, изображены на рисунке 7.

		Название	Продолжи...	Начало	Окончание	Предшествующие
1		Начало проекта	0 дней	05.05.2026, 08:00	05.05.2026, 08:00	
2		☒ Анализ аналогов и выбор прототи	6 дней?	05.05.2026, 08:00	12.05.2026, 17:00	
3		Анализ предметной области	1 день?	05.05.2026, 08:00	05.05.2026, 17:00	
4		Анализ существующих аналогов	1 день?	06.05.2026, 08:00	06.05.2026, 17:00	3
5		Выбор прототипа	2 дней?	07.05.2026, 08:00	08.05.2026, 17:00	4
6		Анализ требований	2 дней?	09.05.2026, 08:00	12.05.2026, 17:00	5
7		☒ Проектирование	8 дней?	13.05.2026, 08:00	22.05.2026, 17:00	
8		Проектирование базы данных	2 дней?	13.05.2026, 08:00	14.05.2026, 17:00	6
9		Проектирование архитектуры	3 дней?	15.05.2026, 08:00	19.05.2026, 17:00	8
10		Проектирование интерфейса	3 дней?	20.05.2026, 08:00	22.05.2026, 17:00	9
11		☒ Реализация веб-приложения	10,5 дней?	25.05.2026, 08:00	08.06.2026, 13:00	
12		Обоснование средств разработки	0,5 дней?	25.05.2026, 08:00	25.05.2026, 13:00	10
13		Реализация базы данных	4 дней?	25.05.2026, 13:00	29.05.2026, 13:00	12
14		Реализация архитектуры	3 дней?	29.05.2026, 13:00	03.06.2026, 13:00	13
15		Реализация интерфейса	3 дней?	03.06.2026, 13:00	08.06.2026, 13:00	14
16		☒ Тестирование	2 дней?	08.06.2026, 13:00	10.06.2026, 13:00	
17		Функциональная диаграмма и таблицы	1 день?	08.06.2026, 13:00	09.06.2026, 13:00	15
18		Метрики кода	1 день?	09.06.2026, 13:00	10.06.2026, 13:00	17
19		Конец проекта	0 дней?	10.06.2026, 13:00	10.06.2026, 13:00	18

Рисунок 7 – Ключевые задачи проекта

В ходе планирования были выявлены также ресурсы, необходимые для реализации проекта (рисунок 8).

	И	Название	Тип	Ед. изм. м...	Инициалы	Группировать	Максимальные единицы	Стандартная ставка	Ставка сверхурочных	Затраты на одно использование	Начислять	Основной календарь
1		Back-end программист	Работа		B-d		100 %	600/час	0 %	0,00 Р	Пропорционально	Пятидневка
2		Front-end программист	Работа		F-d		100 %	475/час	0 %	0,00 Р	Пропорционально	Пятидневка
3		Аналитик	Работа		A		100 %	500/час	0 %	0,00 Р	Пропорционально	Пятидневка
4		Тестировщик	Работа		T		100 %	450/час	0 %	0,00 Р	Пропорционально	Пятидневка

Рисунок 8 – Ресурсы проекта

Далее ресурсы были распределены под каждую задачу проекта. Список задач с ресурсами продемонстрирован на рисунке 9.

	И	Название	Продолжит...	Начало	Окончание	Предшествующие	Название ресурса
1		Начало проекта	0 дней	05.05.2026, 08:00	05.05.2026, 08:00		
2		Анализ аналогов и выбор прототи...	6 дней?	05.05.2026, 08:00	12.05.2026, 17:00		
3		Анализ предметной области	1 день?	05.05.2026, 08:00	05.05.2026, 17:00		Аналитик
4		Анализ существующих аналогов	1 день?	06.05.2026, 08:00	06.05.2026, 17:00	3	Аналитик
5		Выбор прототипа	2 дней?	07.05.2026, 08:00	08.05.2026, 17:00	4	Аналитик
6		Анализ требований	2 дней?	09.05.2026, 08:00	12.05.2026, 17:00	5	Аналитик
7		Проектирование	8 дней?	13.05.2026, 08:00	22.05.2026, 17:00		
8		Проектирование базы данных	2 дней?	13.05.2026, 08:00	14.05.2026, 17:00	6	Back-end программист
9		Проектирование архитектуры	3 дней?	15.05.2026, 08:00	19.05.2026, 17:00	8	Back-end программист
10		Проектирование интерфейса	3 дней?	20.05.2026, 08:00	22.05.2026, 17:00	9	Front-end программист
11		Реализация веб-приложения	10,5 дней?	25.05.2026, 08:00	08.06.2026, 13:00		
12		Обоснование средств разработки	0,5 дней?	25.05.2026, 08:00	25.05.2026, 13:00	10	Back-end программист[50 %];Front-end программист[50 %]
13		Реализация базы данных	4 дней?	25.05.2026, 13:00	29.05.2026, 13:00	12	Back-end программист
14		Реализация архитектуры	3 дней?	29.05.2026, 13:00	03.06.2026, 13:00	13	Back-end программист
15		Реализация интерфейса	3 дней?	03.06.2026, 13:00	08.06.2026, 13:00	14	Front-end программист
16		Тестирование	2 дней?	08.06.2026, 13:00	10.06.2026, 13:00		
17		Функциональная диаграмма и таблиц...	1 день?	08.06.2026, 13:00	09.06.2026, 13:00	15	Тестировщик
18		Метрики кода	1 день?	09.06.2026, 13:00	10.06.2026, 13:00	17	Тестировщик
19		Конец проекта	0 дней?	10.06.2026, 13:00	10.06.2026, 13:00	18	

Рисунок 9 – Список задач проекта с распределенными ресурсами

Для визуального представления сроков выполнения работ по разработке веб-приложения была сформирована диаграмма Ганта, позволяющая отразить последовательность и продолжительность всех этапов разработки проекта [11]. На рисунке 10 продемонстрирована получившаяся диаграмма Ганта.

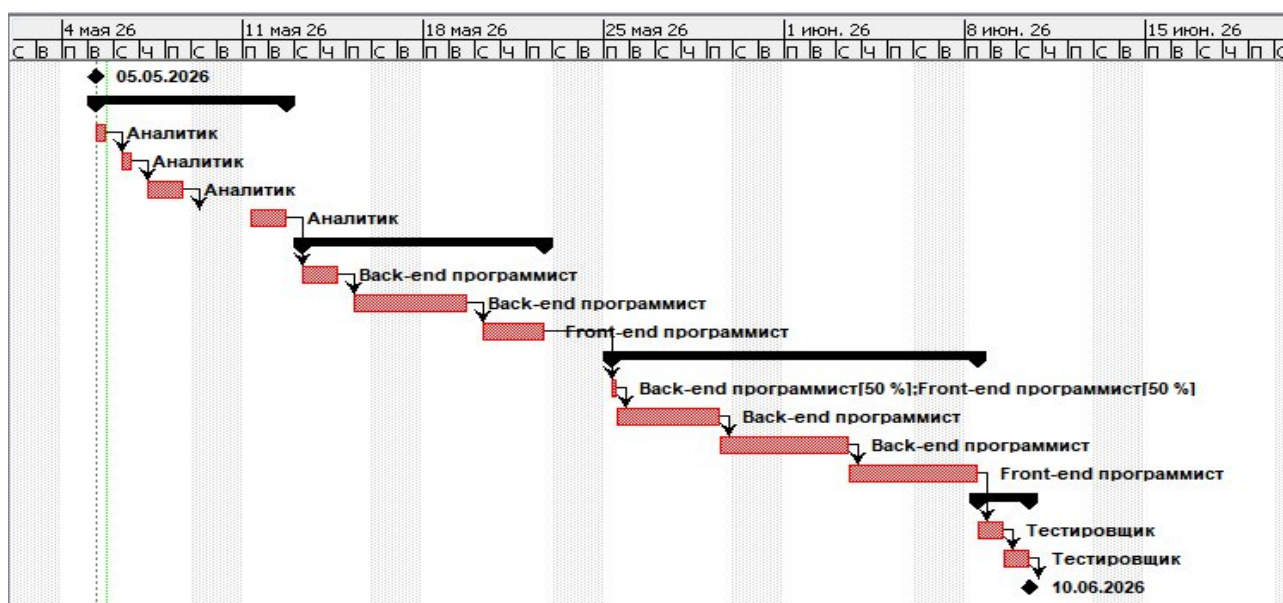


Рисунок 10 – Диаграмма Ганта проекта

При помощи программного средства ProjectLibre была сформирована статистическая информация по проекту, отражающая основные параметры планирования и выполнения работ. Результат статистики продемонстрирован на рисунке 11.

fintime

Dates			
Start	05.05.2026, 08:00	Finish	10.06.2026, 13:00
Baseline Start		Baseline Finish	
Actual Start		Actual Finish	

Duration			
Scheduled	26,5 дней	Remaining	26,5 дней
Baseline	0 дней	Actual	0 дней
		Percent Complete	0 %

Work			
Scheduled	212 часов	Remaining	212 часов
Baseline	0 часов	Actual	0 часов

Costs			
Scheduled	113750,00 Р	Remaining	113750,00 Р
Baseline	0,00 Р	Actual	0,00 Р
		Variance	0,00 Р

Рисунок 11 – Статистика проекта

В результате планирования установлено, что итоговая стоимость проекта равна 113750,00 рублей, а продолжительность реализации всего проекта равна 212 часов, что эквивалентно 26,5 рабочим дням. Полученные показатели отражают трудоёмкость разработки и позволяют оценить необходимые временные, ресурсные и финансовые затраты для реализации веб-приложения.

2 Проектирование

2.1 Проектирование базы данных веб-приложения

В ходе проектирования базы данных (БД) была создана логическая модель базы данных (рисунок 12).

Логическая модель данных является одним из видов моделей представления данных и является промежуточной моделью между концептуальной и физической. В рамках проектирования используется реляционная ER-модель (Entity-Relationship model, модель «сущность – связь») базы данных. Она включает в себя сущности, связи между сущностями, атрибуты сущностей, а также ключи [12]. Также она помогает наглядно и без привязки к конкретной СУБД формализовать структуру хранимых данных.

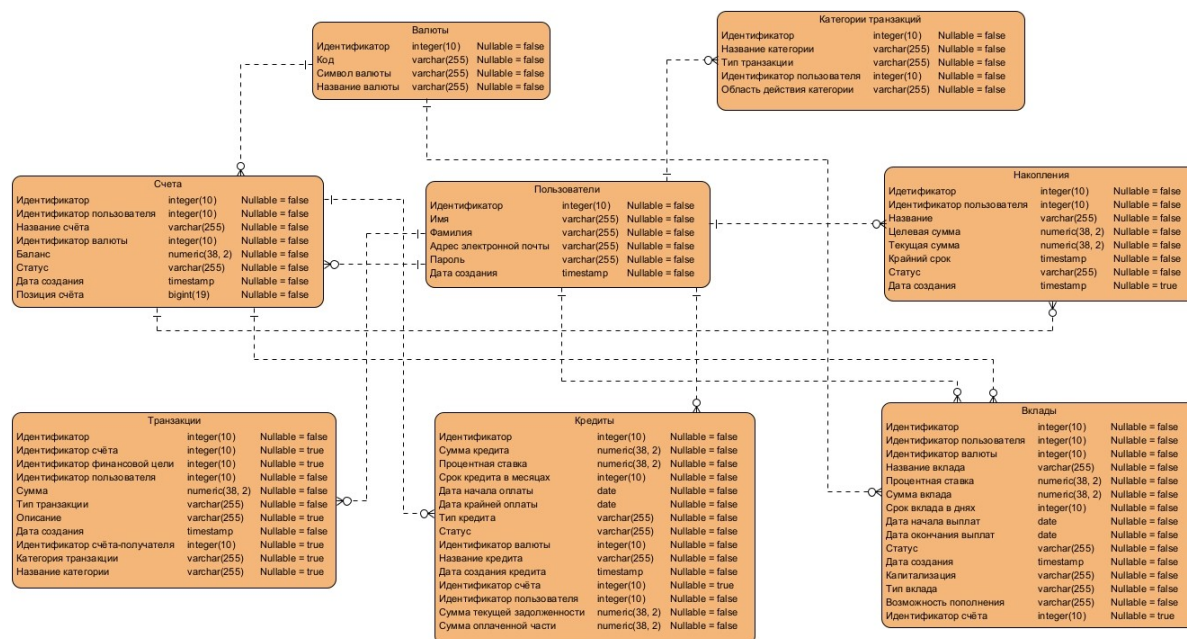


Рисунок 12 – Логическая модель базы данных

Описание основных сущностей, используемых в разрабатываемой системе, с указанием их назначения, логической модели базы данных представлено в таблице 20. Описание сущностей служит основой для последующей реализации физической модели базы данных с учётом поставленных требований.

Таблица 20 – Описание сущностей логической модели базы данных

Название сущности	Описание сущности
Пользователи	Хранение пользовательских данных (имя, фамилия, адрес эл. почты, пароль)
Счета	Хранение данных о счетах пользователя
Валюты	Хранение данных о валютах
Категории транзакций	Хранение данных о категориях транзакций
Кредиты	Хранение данных о кредитах пользователя
Вклады	Хранение данных о вкладах пользователя
Накопления	Хранение данных о целевых накоплениях пользователя
Транзакции	Хранение данных о транзакциях пользователя

Представленная модель БД позволит без дублирования данных хранить информацию о счетах, кредитах, накоплениях, вкладах и действиях пользователя. Также данная структура является масштабируемой и при добавлении новых сущностей не требует кардинального исправления схемы БД.

2.2 Проектирование архитектуры веб-приложения

В ходе проектирования архитектуры веб-приложения была выбрана диаграмма классов.

Диаграмма классов – это распространенный инструмент, позволяющий визуализировать процесс проектирования архитектуры. Она помогает разработчикам программного обеспечения понять структуру проекта и организовать код таким образом, чтобы исключить циклические зависимости,

слишком сильное зацепление или недостаточную связность. Также она является точкой коммуникации между разработчиками, менеджерами и заказчиками проекта [13].

Для серверной части веб-приложения, связанного с управлением счетами, была разработана диаграмма классов, представленная на рисунке 13.

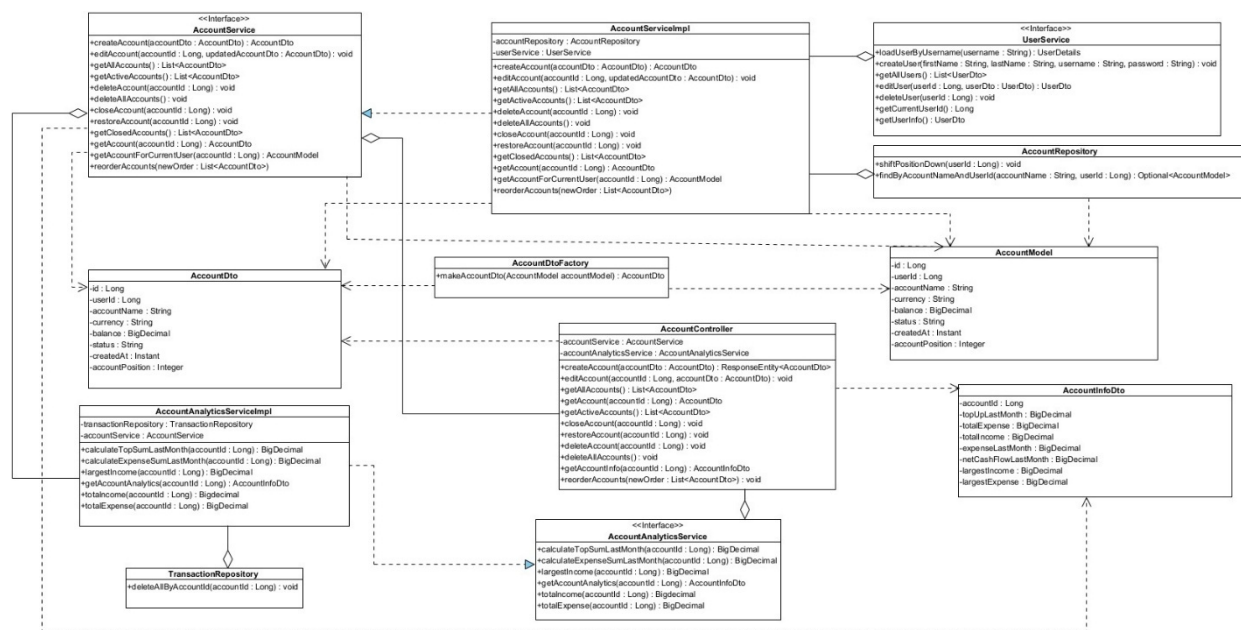


Рисунок 13 – Диаграмма классов

Краткое описание классов представлено в таблице 21.

Таблица 21 – Описание классов

Класс	Описание класса
AccountService	Интерфейс сервиса управления счетами
AccountRepository	Репозиторий счетов, обеспечивающий хранение позиции счета в списке
AccountModel	Модель, хранящая данные счета
AccountDto	Объект передачи данных (DTO), используемый для передачи данных счета на клиентскую часть веб-приложения

Продолжение таблицы 21

Класс	Описание класса
AccountServiceImpl	Реализация сервиса управления счетами (создание, редактирование, удаление, список всех счетов, список активных счетов, удаление счета, удаление всех счетов, закрытие счета, восстановление, список закрытых счетов, изменение места счёта)
AccountDtoFactory	Фабрика, преобразовывающая модель счета в DTO
UserService	Интерфейс сервиса управления аккаунтом (создание, редактирование профиля, удаление, информация)
AccountController	Контроллер, обрабатывающий HTTP-запросы по управлению счетами
AccountAnalyticsService	Интерфейс сервиса аналитики
AccountAnalyticsServiceImpl	Реализация сервиса аналитики (самый крупный расход, самый крупный доход, сумма доходов за месяц, сумма расходов за месяц, всего доходов, всего расходов)
TransactionRepository	Репозиторий транзакций
AccountInfoDto	DTO, используемый для передачи данных аналитики на клиентскую часть веб-приложения

Таким образом, разработанная диаграмма классов отражает структуру серверной части веб-приложения, отвечающую за работу со счетами и сопутствующими финансовыми операциями.

Выделение классов и слоёв системы позволяет разграничить бизнес-логику веб-приложения, а также работу с данными и взаимодействие с внешними запросами, что в конечном итоге упрощает масштабирование и поддержку кода. Кроме того, такая архитектура также минимизирует риск возникновения циклических зависимостей [14].

2.3 Проектирование интерфейса веб-приложения

Эффективность работы пользователя с любым веб-приложением во многом определяется качеством пользовательского интерфейса (UI) и пользовательского опыта (UX). При проектировании интерфейса были сформулированы следующие ключевые требования [15]:

- интерфейс не должен перегружать пользователя лишней информацией;
- пользователь всегда должен получать подтверждение своих действий (как успешных, так и неуспешных);
- все однотипные элементы должны выглядеть и вести себя одинаково во всём веб-приложении;
- пользователь всегда должен знать, где он находится в структуре веб-приложения;
- веб-приложение должно обладать приятным внешним видом для повышения лояльности пользователей.

Исходя из вышеперечисленных требований, был разработан прототип интерфейса страницы управления счетами (рисунок 14). Слева расположена навигационная панель, которая предоставляет пользователю быстрый доступ ко всем ключевым страницам веб-приложения (личный кабинет, управление счетами, кредиты, вклады, планирование, аналитика, история операций). В правом верхнем углу расположен выпадающее меню, позволяющее перейти в личный кабинет, сменить аккаунт или выйти из аккаунта. В центре расположен список счетов пользователя в виде карточек с быстрой возможностью пополнить, снять, перевести, закрыть или удалить счёт. Пользователь может видеть какие счета являются закрытыми, а какие активными, благодаря статусу в карточке. Также при желании пользователь имеет возможность переместить как активный, так и закрытый счёт из одного места сетки счетов в другое. Для этого ему необходимо нажать левой кнопкой мыши счёт и перенести его в любое место сетки.

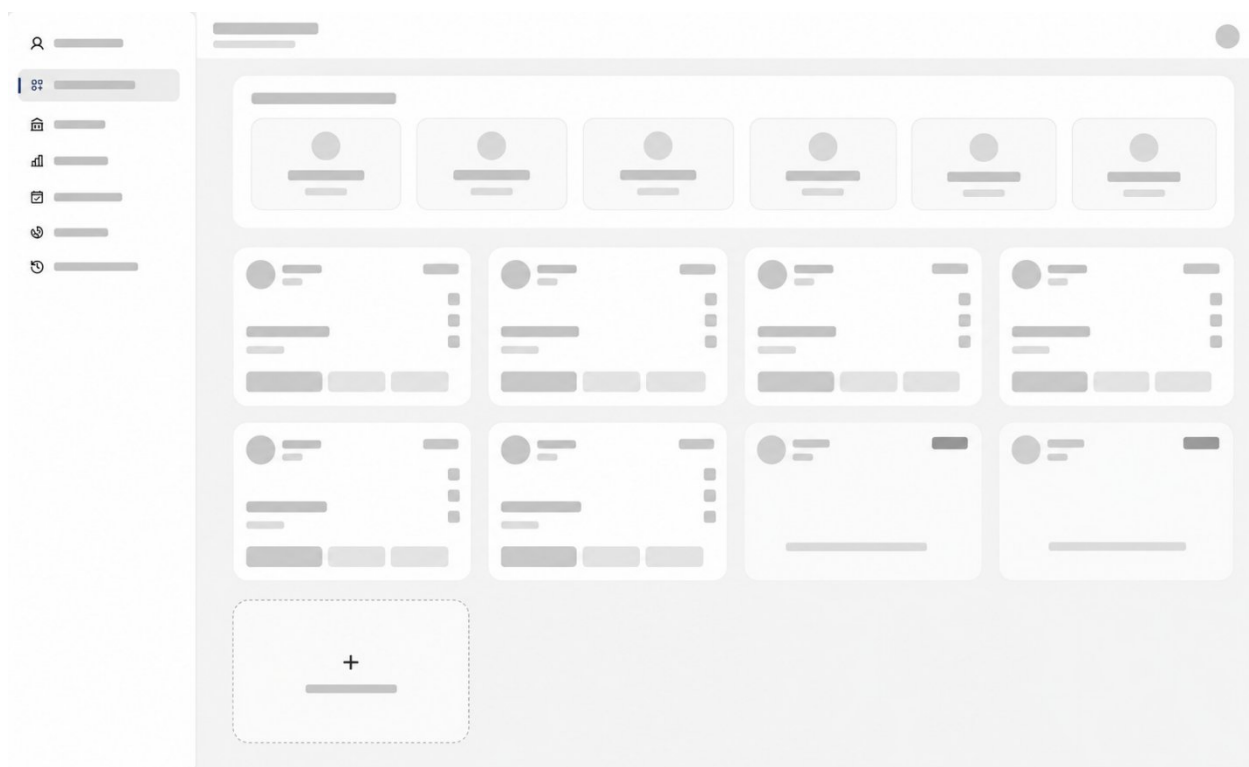


Рисунок 14 – Прототип страницы управления счетами

Для создания счёта пользователю необходимо нажать на поле «Создать счёт» и в открывшееся модальное окно (рисунок 15) ввести название счета, валюту, начальный баланс, описание (при желании) и нажать на кнопку «Создать счёт».

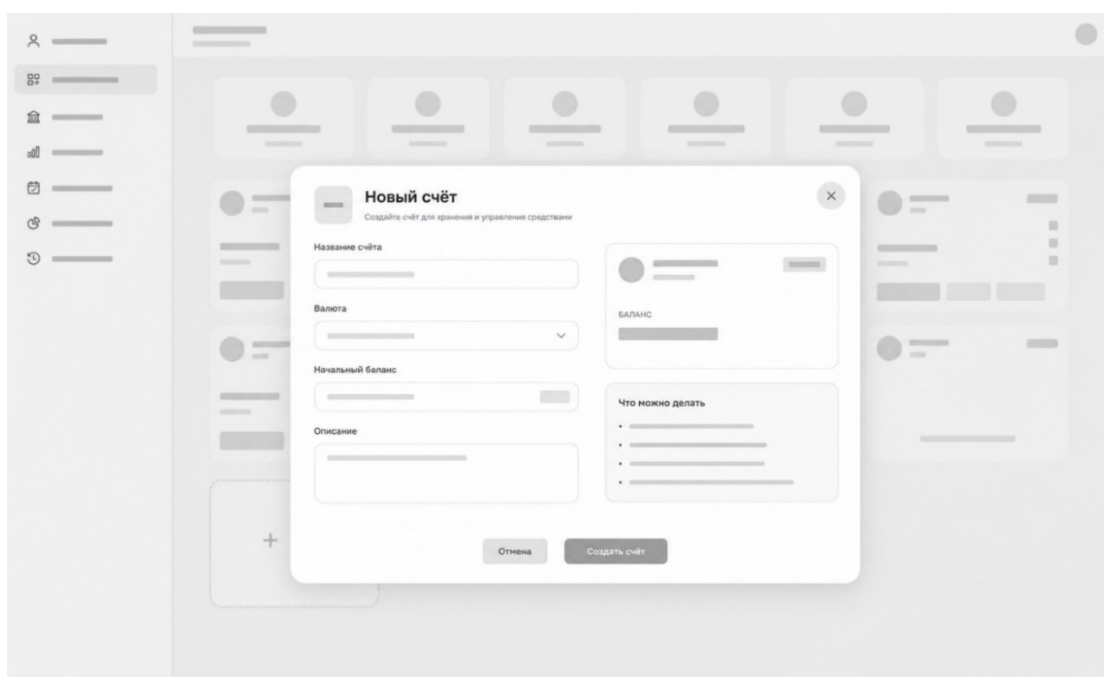


Рисунок 15 – Прототип страницы добавления нового счёта

Таким образом, пользователь сможет сразу увидеть созданный счёт в списке. При желании пользователь может поменять счета местами, зажав нужный левой кнопкой мыши и перетаскив в другое место.

При нажатии на счёт пользователь должен попадать на информационную страницу (рисунок 16), которая показывает последние транзакции, информацию счета и аналитику.

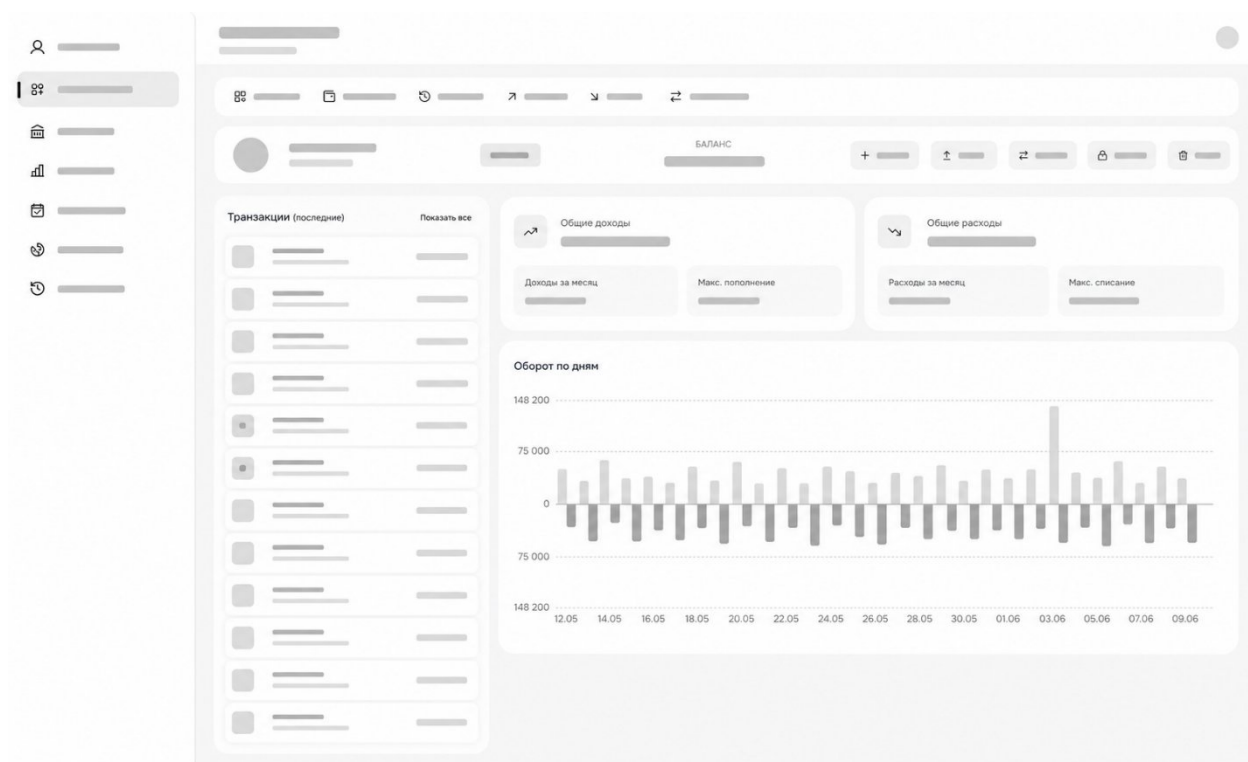


Рисунок 16 – Прототип информационной страницы счёта

В результате проектирования был сформирован функциональный и логически структурированный прототип интерфейса управления счетами. Разработанный прототип учитывает основные принципы UI/UX-дизайна, включая минимизацию когнитивной нагрузки и обеспечивает предсказуемость поведения интерфейса. Использование единого стиля оформления и логичного расположения элементов управления способствует повышению удобства использования системы, а также обеспечивает целостность пользовательского опыта при работе с различными разделами приложения. Благодаря этому пользователь имеет возможность быстро ориентироваться в системе и выполнять необходимые финансовые операции без долгого обучения.

3 Реализация веб-приложения

3.1 Обоснование средств разработки веб-приложения

При разработке серверной части веб-приложения был выбран язык программирования Java, так как он является надежным, кроссплатформенным и обладает развитой экосистемой, и фреймворк Spring (Spring Boot), который обеспечивает быстрый старт проекта и упрощенную конфигурацию [16]. Spring Boot также предоставляет встроенные механизмы для настройки REST-контроллеров, управления зависимостями, безопасности и интеграцию с базой данных PostgreSQL через Spring Data JPA для удобной работы с сущностями и репозиториями.

В качестве среды разработки для серверной части была использована IntelliJ IDEA, так как она предоставляет широкий набор инструментов для эффективной разработки на языке Java с использованием Spring. Среда имеет интеграцию с системой сборки кода Maven, системой контроля версий Git и базой данных PostgreSQL.

Для клиентской части веб-приложения была использована библиотека React, так как она использует удобный компонентный подход к построению пользовательского интерфейса и упрощает повторное использование элементов.

Данный набор технологий является оптимальным выбором, так как обеспечивает удобство и скорость разработки, за счёт интеграции, а также надежность и масштабируемость разрабатываемого веб-приложения.

3.2 Реализация базы данных веб-приложения

С использованием логической модели базы данных была разработана физическая модель базы данных «fintime», при помощи инструментария СУБД PostgreSQL. Результат продемонстрирован на рисунке 17. Физическая модель БД представляет собой конкретную реализацию логической модели и учитывает особенности конкретной СУБД (в данном случае PostgreSQL) [17].

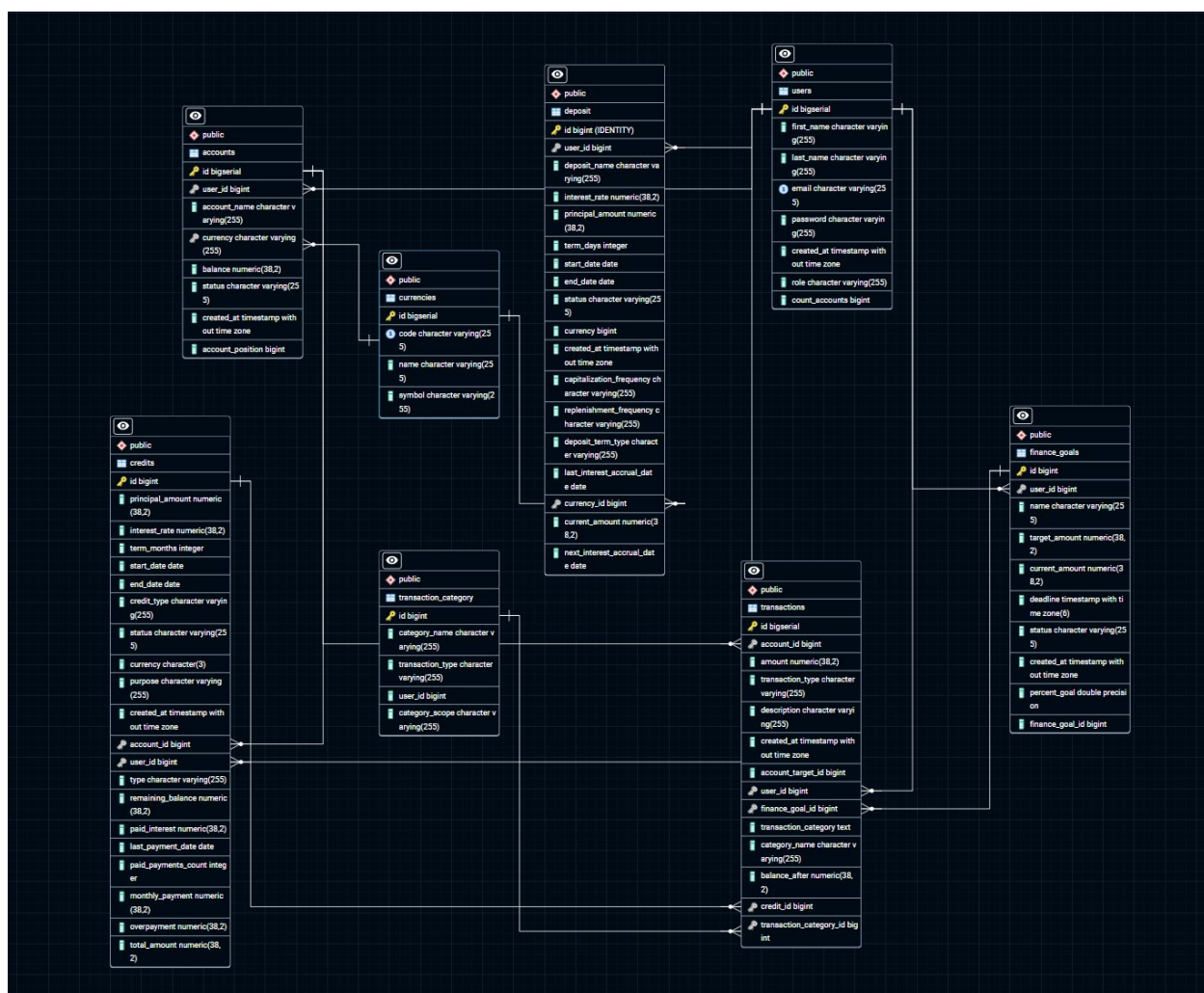


Рисунок 17 – Физическая модель базы данных

Все типы связей между сущностями в физической модели описаны в таблице 22.

Таблица 22 – Описание связей между сущностями физической модели базы данных

Сущность 1	Сущность 2	Связь
users	accounts	Один-ко-многим
users	deposit	Один-ко-многим
users	credits	Один-ко-многим
users	finance_goals	Один-ко-многим
users	transactions	Один-ко-многим
currencies	accounts	Один-ко-многим

Продолжение таблицы 22

Сущность 1	Сущность 2	Связь
currencies	deposit	Один-ко-многим
currencies	credits	Один-ко-многим
currencies	finance_goals	Один-ко-многим
currencies	transactions	Один-ко-многим
accounts	credits	Один-ко-многим
accounts	deposit	Один-ко-многим
accounts	finance_goals	Один-ко-многим
accounts	transactions	Один-ко-многим
finance_goals	transactions	Один-ко-многим
credits	transactions	Один-ко-многим
deposit	transactions	Один-ко-многим

Таким образом, была реализована физическая модель базы данных «fintime», включающая структуру таблиц, их атрибуты, типы данных и связи между сущностями, что обеспечивает эффективное хранение и обработку информацию.

3.3 Реализация архитектуры веб-приложения

Диаграмма развёртывания предназначена для представления топологии разрабатываемой системы и визуализации компонентов веб-приложения. Диаграмма развёртывания также разрабатывается для выявления узких мест системы, которые потенциально могут быть связаны с ограничением производительности. В архитектуре разрабатываемой системы выделяются три ключевых компонента: сервер, разработанный на Java с использованием фреймворка Spring, клиентское устройство и сервер базы данных PostgreSQL.

Взаимодействие с системой осуществляется при помощи веб-браузера. Пользователь отправляет HTTP-запросы к серверу, сервер выполняет обработку запросов, обращается к серверу базы данных PostgreSQL для получения или изменения данных, после чего передаёт результаты обратно пользователю.

Диаграмма развёртывания представлена на рисунке 18.

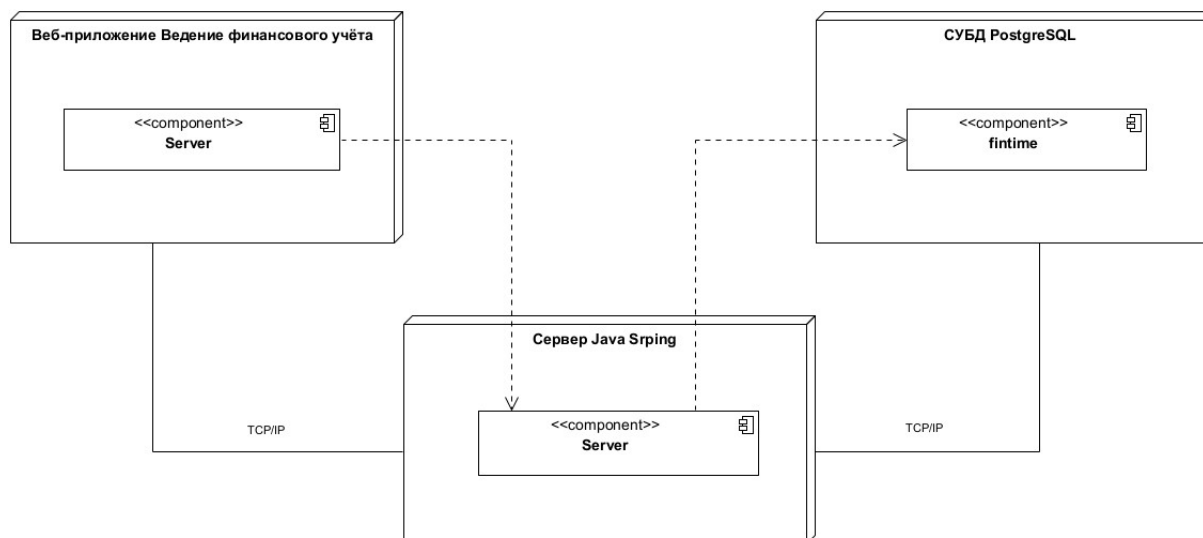


Рисунок 18 – Диаграмма развёртывания

Данная диаграмма развёртывания наглядно демонстрирует связи между компонентами системы, что помогает получить целостное представление об архитектуре веб-приложения. Чёткое разделение на клиентский и серверный компоненты, а также выделение отдельного сервера базы данных, позволяет эффективно распределять нагрузку и минимизировать потенциальные узкие места, что критично для обеспечения высокой производительности системы при росте числа пользователей.

3.4 Реализация интерфейса веб-приложения

При реализации интерфейса веб-приложения был сделан акцент на максимальную простоту, удобство и интуитивную понятность. Основная цель заключалась в создании такого интерфейса, при котором пользователь мог бы максимально быстро освоиться в веб-приложении и сосредоточиться непосредственно на управлении своими счетами, кредитами, вкладами и накоплениями.

Страница управления счетами предоставляет пользователю возможность создать счёт, пополнить счёт, снять со счёта, перевести средства на другой счёт, закрыть, восстановить или удалить счёт. Интерфейс представлен на рисунке 19.

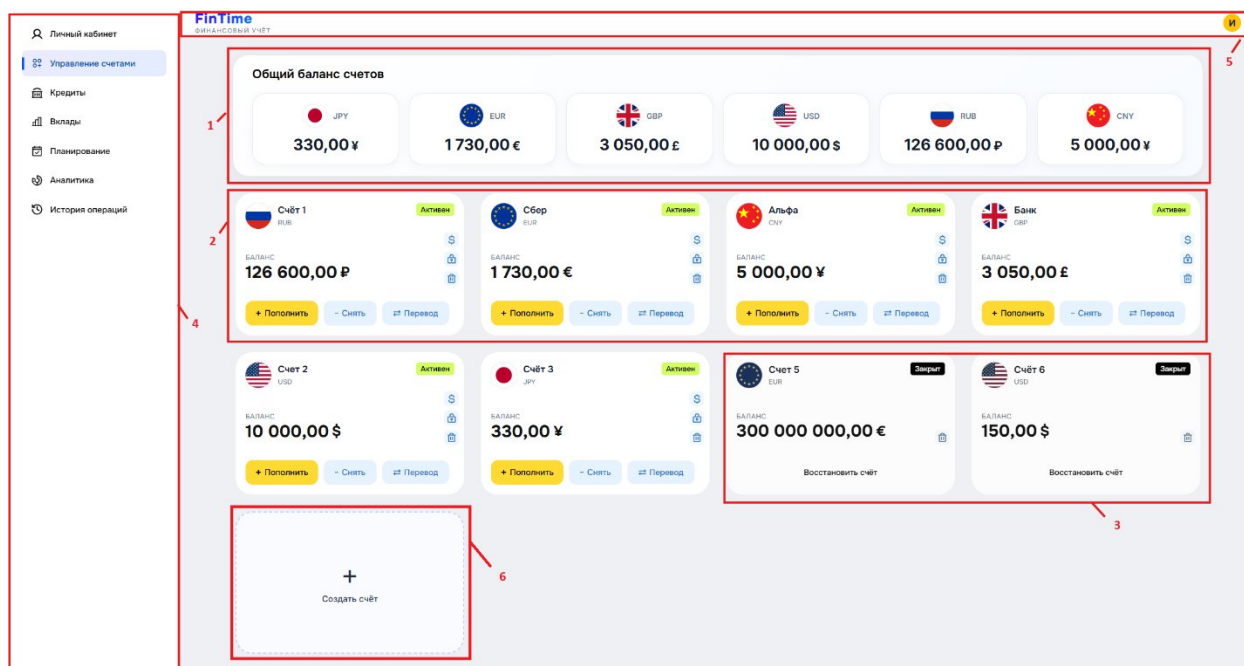


Рисунок 19 – Интерфейс страницы управления счетами

На данном рисунке изображены:

- 1) аналитическая панель с общими балансами в различных валютах;
- 2) список активных счетов пользователя в виде карточек с кнопками быстрого доступа к основным действиям над счетами;
- 3) список закрытых счетов пользователя с возможностью восстановить или удалить счёт;
- 4) навигационная панель быстрого доступа к основным страницам веб-приложения;
- 5) шапка веб-приложения, в которой изображено название сервиса в левой стороне и кнопка выпадающего меню в правой стороне;
- 6) кнопка создания счёта.

При нажатии на «Создать счёт» открывается модальное окно (рисунок 20), в котором пользователь может ввести данные счёта: название счёта, валюта, начальный баланс, описание (при желании).

Справа пользователь сразу может увидеть, как будет выглядеть будущая карточка счёта. Под карточкой находится информационный блок, который показывает пользователю возможные варианты использования счёта.

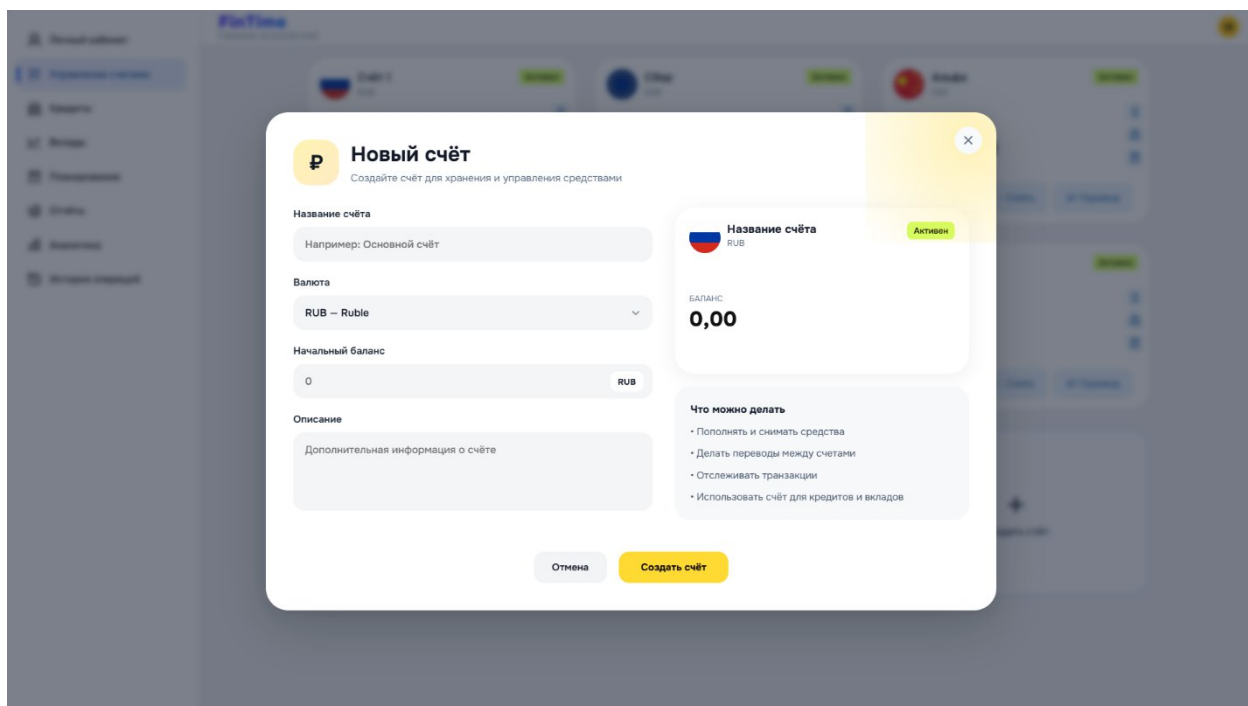


Рисунок 20 – Интерфейс страницы добавления нового счёта

Пользователь также имеет возможность просмотреть данные конкретного счёта (рисунок 21), нажав на соответствующую карточку в общем списке счетов.

На данной странице пользователь может увидеть как основные данные счёта (название, валюта, баланс, статус), так и список последних транзакций, аналитические данные: общие доходы по данному счёту и общие расходы, доходы и расходы за последний месяц, максимальная сумма пополнения или списания за всё время. На представленной гистограмме пользователь имеет возможность быстро просмотреть сколько он потратил или заработал за конкретный день в течение последнего месяца. Для этого он должен навести на конкретную колонку гистограммы, и система выведет модальное информационное окно с доходом и расходом в этот день.

Также пользователь может перейти к другим страницам (все транзакции, все расходы, все доходы, все переводы) с информацией по счёту выбрав нужный пункт в горизонтальной навигационной панели.

На страницу «Все транзакции» пользователь может также перейти через альтернативу в виде кнопки «Показать все» в блоке последний транзакций.

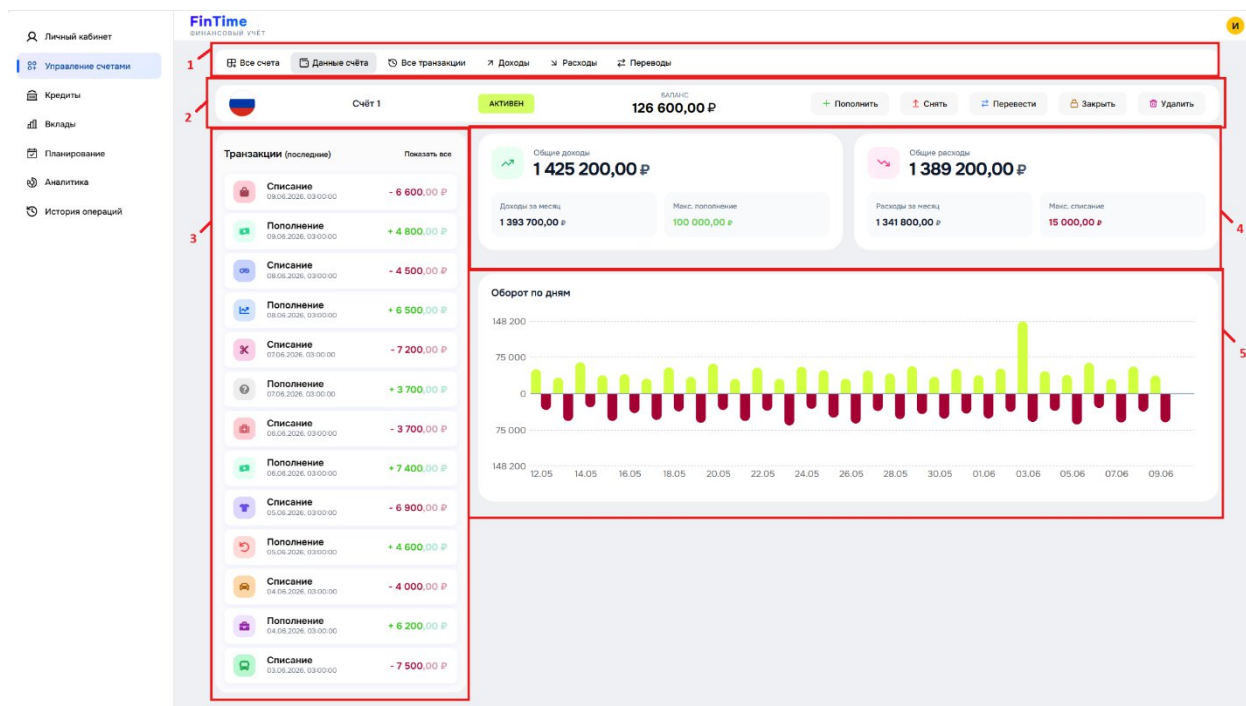


Рисунок 21 – Интерфейс информационной страницы счёта

На рисунке 14 представлено:

- 1) навигационная панель для перехода к другим информационным страницам.
- 2) основная информация счёта с элементами управления;
- 3) список последних транзакций по счёту;
- 4) аналитические данные по счёту: общий доход и расход, доход и расход за месяц, а также самые крупные суммы пополнения и списания;
- 5) гистограмму, демонстрирующую оборот денежных средств за последний месяц.

Категории транзакций пользователь может выбрать в открывшемся модальном окне (рисунок 22) при нажатии на кнопки «Пополнить» или «Снять». В зависимости от выбранной кнопки у пользователя будет разный выбор категорий (категории для пополнения и категории для снятия).

Для добавления личной категории пользователь должен ввести название категории и нажать на кнопку «Добавить». Добавленные пользователем категории будут отображаться сверху списка категорий.

Также в форме находятся уже готовые суммы для быстрого ввода.

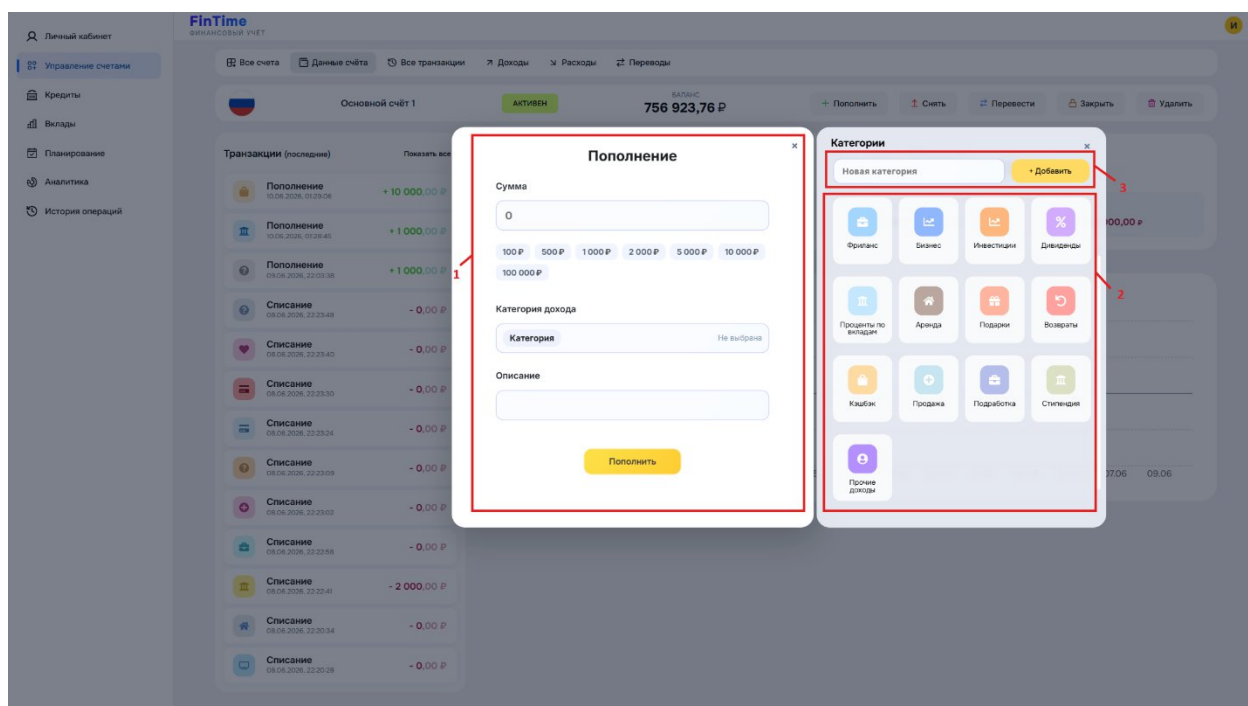


Рисунок 22 – Интерфейс страницы пополнения счёта

На рисунке 15 представлено:

- 1) форма добавления новой транзакции;
- 2) список категорий транзакции;
- 3) форма добавления личной категории транзакции.

На странице «Все транзакции» (рисунок 23) пользователь может просмотреть полный список операций по счёту. Каждая транзакция содержит информацию: категорию транзакции, тип транзакции (пополнение, списание, перевод), дату совершения транзакции и сумму.

Список транзакций может быть отсортирован по дате с помощью выбора начальной и конечной даты, что позволяет пользователю формировать выборку операций за определенный период. Также пользователь может посмотреть общее количество совершенных транзакций по счёту с учётом сортировки.

У пользователя имеется возможность изменить информацию или удалить конкретную транзакцию, нажав на кнопку в колонке «Действия».

Если количество транзакций становится слишком большим, то они начинают разделяться на страницы.

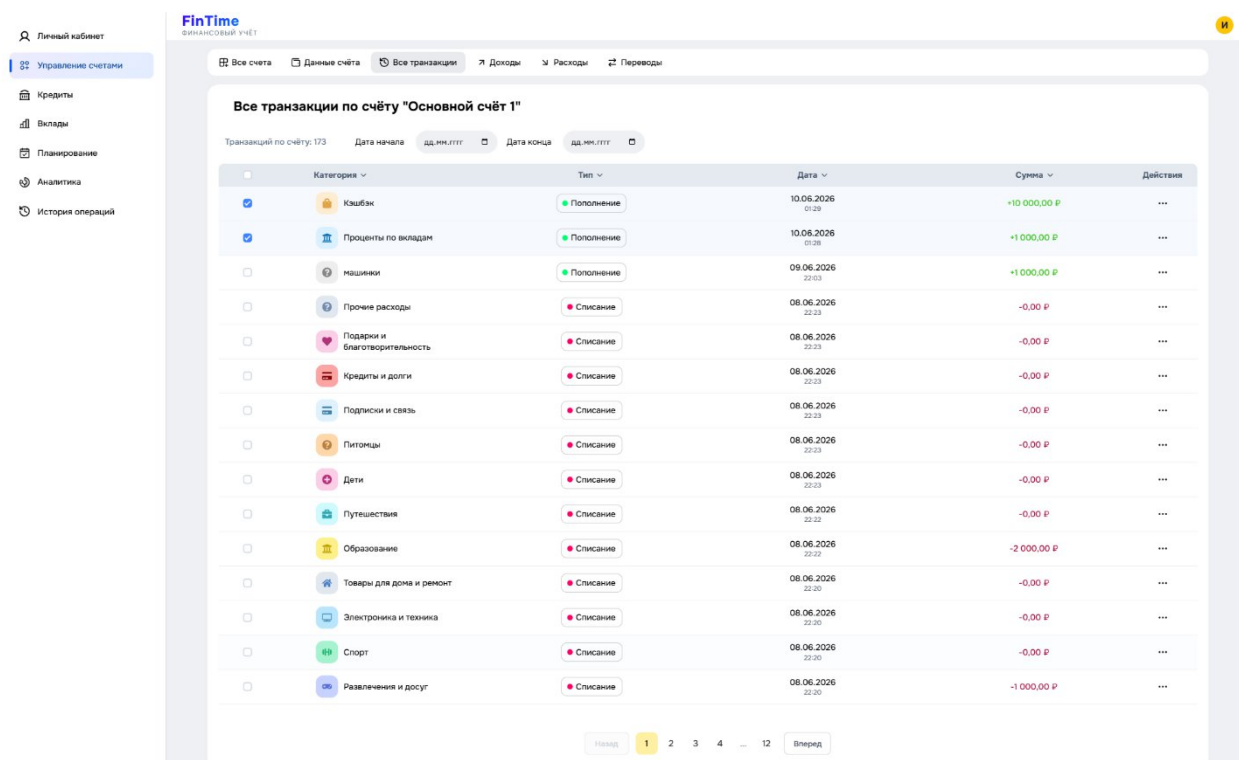


Рисунок 23 – Интерфейс страницы всех транзакций

Списки только доходов, только расходов или только переводов можно увидеть, выбрав соответствующую вкладку на навигационной панели.

В ходе разработки был реализован интерфейс веб-приложения, ориентированный на обеспечение высокой скорости и удобства выполнения основных пользовательских операций.

Для повышения удобства использования особое внимание было уделено навигации и расположению функциональных элементов. Основные операции доступны пользователю из соответствующих разделов системы, что позволяет сократить количество действий, необходимых для выполнения типовых задач.

Все элементы интерфейса приведены к единому стилю, что обеспечивает визуальную целостность системы и снижает вероятность ошибок при взаимодействии с функциональными компонентами у новых пользователей.

В результате реализованный интерфейс представляет собой завершенное решение, ориентированное на повышение эффективности работы пользователя, упрощение взаимодействия с финансовыми данными и обеспечение удобного доступа ко всем основным компонентам веб-приложения.

4 Тестирование

4.1 Функциональная диаграмма и таблица решений

Пользователь должен иметь возможность добавить вклад. Для этого он должен ввести название уникальное вклада и процентную ставку вклада. Название вклада может содержать от 2 до 64 символов латинского и русского алфавита, цифр, пробелов, тире и нижнего подчеркивания. Процентная ставка может содержать от 1 до 3 цифр. Пользователь не может добавить вклад с уже добавленным названием. Реакция программы на пользовательский ввод:

- программа должна сообщать о неверной длине процентной ставки или названия вклада, если их размер не соответствует требуемому;
- программа должна сообщать об использовании некорректных символов в составе процентной ставки или названия вклада, если таковые имеются;
- программа должна сообщать о том, что вклад уже добавлен, если такое название уже используется;
- программа должна сообщить об успешном добавлении вклада, если все введенные пользователем данные оказались допустимыми.

Был произведен анализ нескольких полей и поведения системы на основе технического задания. Выделены причины и следствия событий. Результат продемонстрирован в таблице 23.

Таблица 23 – Причины и следствия событий

Причины	
П1	Корректная длина названия вклада
П2	Корректная длина процентной ставки
П3	Название вклада состоит из корректных символов
П4	Процентная ставка состоит из корректных символов
П5	Название вклада свободно
Следствия	
С1	Сообщение: «Некорректная названия вклада»

Продолжение таблицы 24

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
C2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
C3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
C4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
C5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
C6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Исходя из получившейся таблицы был построен набор тест-кейсов по столбцам таблицы решений. Результаты продемонстрированы в таблице 25.

Таблица 25 – Результаты тест-кейсов

№	Входные данные	Ожидаемый результат
1	Название вклада: «\$» Процентная ставка: «44#4»	Некорректная длина названия вклада, Некорректная длина процентной ставки, Некорректные символы в составе названия вклада, Некорректные символы в составе процентной ставки
2	Название вклада: «Вклад*» Процентная ставка: «441»	Некорректная длина процентной ставки, Некорректные символы в составе названия вклада, Некорректные символы в составе процентной ставки
3	Название вклада: «%» Процентная ставка: «;12»	Некорректная длина названия вклада, Некорректные символы в составе названия вклада, Некорректные символы в составе процентной ставки
4	Название вклада: «Выгода\$» Процентная ставка: «123#»	Некорректные символы в составе названия вклада, Некорректные символы в составе процентной ставки
5	Название вклада: «G» Процентная ставка: «13^23»	Некорректная длина названия вклада, Некорректная длина процентной ставки, Некорректные символы в составе процентной ставки

Продолжение таблицы 25

№	Входные данные	Ожидаемый результат
6	Название вклада: «Новый-2» Процентная ставка: «12355!»	Некорректная длина процентной ставки, Некорректные символы в составе процентной ставки
7	Название вклада: «Н» Процентная ставка: «@13»	Некорректная длина названия вклада, Некорректные символы в составе процентной ставки
8	Название вклада: «Mine» Процентная ставка: «1%3»	Некорректные символы в составе процентной ставки
9	Название вклада: «{» Процентная ставка: «1100»	Некорректная длина названия вклада, Некорректная длина процентной ставки, Некорректные символы в составе названия вклада
10	Название вклада: «{Вклад 6» Процентная ставка: «1231243»	Некорректная длина процентной ставки, Некорректные символы в составе названия вклада
11	Название вклада: «Vklad nomer 33112345123455 67890» Процентная ставка: «14№»	Некорректная длина названия вклада, Некорректные символы в составе названия вклада
12	Название вклада: «%вклад» Процентная ставка: «12»	Некорректные символы в составе названия вклада
13	Название вклада: «Р» Процентная ставка: «1199»	Некорректная длина названия вклада, Некорректная длина процентной ставки
14	Название вклада: «Зеленый» Процентная ставка: «9991»	Некорректная длина процентной ставки
15	Название вклада: «Х» Процентная ставка: «25»	Некорректная длина названия вклада
16	Название вклада: (уже добавленное название) Процентная ставка: «18»	Вклад уже добавлен
17	Название вклада:(название свободно) Процентная ставка: «14»	Вклад успешно добавлен

В результате тестирования были успешно пройдены все тест-кейсы.

4.2 Метрики кода веб-приложения

Для оценки сложности программного кода обработки платежей по кредиту была использована метрика Холстеда. В основе расчёта метрики лежат четыре ключевых параметра: число уникальных операторов (включая разделители, имена функций и знаки операций), число уникальных операндов, суммарное количество операторов и суммарное количество операндов [19].

При анализе была выбрана функция, которая выполняет обработку платежа по кредитной задолженности пользователя. Сам программный код представлен в листинге 1.

Листинг 1 – Функция обработки платежа по кредитной задолженности

```
@Override
    public void payCreditDebt(Long creditId, TransactionDto
transactionDto) {
        CreditModel credit = getCreditForCurrentUser(creditId);
        AccountModel account =
accountService.getAccountForCurrentUser(credit.getAccountId());
        if (transactionDto.getAmount() == null ||
transactionDto.getAmount().compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("Payment amount
must be positive");
        }
        if
(transactionDto.getAmount().compareTo(account.getBalance()) > 0) {
            throw new IllegalArgumentException("Недостаточно
средств на счёте");
        }
        BigDecimal amountRounded =
transactionDto.getAmount().setScale(2, RoundingMode.HALF_UP);
        BigDecimal balanceRounded =
credit.getRemainingBalance().setScale(2, RoundingMode.HALF_UP);
        if (amountRounded.compareTo(balanceRounded) > 0) {
            throw new IllegalArgumentException("Сумма превышает
остаток по кредиту");
        }
        transactionDto.setCreditId(creditId);
        transactionDto.setAccountId(credit.getAccountId());
        transactionDto.setUserId(credit.getUserId());
        transactionDto.setCreatedAt(Instant.now());

transactionDto.setTransactionType(TransactionType.EXPENSE);
        transactionDto.setCategoryName("Кредиты и долги");
        transactionService.createExpenseTransaction(credit.getAccountId(),
transactionDto);
    }
```

```

        BigDecimal newBalance =
credit.getRemainingBalance().subtract(amountRounded).max(BigDecimal.ZERO);
        credit.setRemainingBalance(newBalance);
        credit.setLastPaymentDate(LocalDate.now());
        credit.setPaidPaymentsCount(credit.getPaidPaymentsCount()
+ 1);

        int newPaidCount = credit.getPaidPaymentsCount();
        int remainingMonths = Math.max(0, credit.getTermMonths() -
newPaidCount);

        if (remainingMonths == 0 ||
newBalance.compareTo(BigDecimal.ZERO) <= 0) {
            credit.setStatus(CreditStatus.PAID_OFF);
            credit.setMonthlyPayment(BigDecimal.ZERO);
        }
        creditRepository.saveAndFlush(credit);

```

Этот код представляет функцию `payCreditDebt`, которая используется для проведения платежа по кредиту пользователя и обновления состояния кредитной задолженности. Функция принимает идентификатор кредита (`creditId`) и объект транзакции (`TransactionDto`).

Для выполнения операции функция сначала проверяет принадлежность кредита пользователю. Далее выполняется проверка корректности входных данных: сумма платежа должна быть не null и больше нуля.

После проверок сумма платежа и остаток кредита округляются до двух знаков после запятой (`HALF_UP`), формируется и создаётся расходная транзакция, затем уменьшается остаток задолженности, обновляется дата и счётчик платежей.

Для использования метрики Холстеда требуется подсчитать количество операторов и операндов. Данные об операторах представлены в таблице 26.

Таблица 26 – Операторы

Операторы	Число вхождений
if	4
throw	3
=	10
.	20

Продолжение таблицы 26

Операторы	Число вхождений
new	1
+	1
-	1
>	2
<=	2
compareTo	6
setScale	2
max	1
setCreditId	1
setAccountId	1
setUserId	1
setCreatedAt	1
setTransactionType	1
setCategoryName	1
saveAndFlush	1
n1 = 19	N1 = 60

Данные об операндах продемонстрированы в таблице 27.

Таблица 27 – Операнды

Операнды	Число вхождений
credit	9
transactionDto	12
account	4
creditId	3
amountRounded	3
balanceRounded	2
account.getBalance	1
BigDecimal.ZERO	4
RoundingMode.HALF_UP	2
Instant.now	1

Продолжение таблицы 27

Операнды	Число вхождений
TransactionType.EXPENSE	1
newBalance	2
remainingMonths	2
credit.getRemainingBalance	3
credit.getPaidPaymentsCount	2
credit.getTermMonths	2
n2 = 16	N2 = 53

Исходя из этих данных, можно рассчитать промежуточные значения, используя которые можно получить искомую сложность функции [20].

Длина программы определяется следующим образом:

$$N = N1 + N2 = 60 + 53 = 113.$$

Словарь программы вычисляется по формуле:

$$n = n1 + n2 = 19 + 16 = 35.$$

Объем программы находится по выражению:

$$HV = N * \log_2 n,$$

$$HV = 113 * 5,129 = 579,6.$$

Потенциальный объем программы рассчитывается по формуле:

$$V^* = (2 + n) * \log_2 (2 + n) = 192,73.$$

Теоретический уровень программы определяется как:

$$L' = \frac{V^*}{HV},$$

$$L' = \frac{192,73}{579,60} = 0,3325.$$

Сложность программы вычисляется по формуле:

$$HD = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right),$$

$$HD = \left(\frac{19}{2}\right) * \left(\frac{53}{16}\right) = 31,47.$$

Исходя из расчётов сложность функции payCreditDebt равна 31,47, что соответствует среднему уровню сложности.

Заключение

В результате выполнения выпускной квалификационной работы было разработано веб-приложение для ведения личного финансового учёта, предназначенное для контроля доходов и расходов, управления счетами, вкладами, кредитами и финансовыми целями пользователей. В рамках работы были выполнены анализ предметной области, исследование существующих аналогов, формирование требований, проектирование архитектуры системы, реализация программного продукта и его тестирование.

На этапе анализа предметной области были рассмотрены особенности ведения личного финансового учёта и определены основные функции, необходимые пользователям для эффективного управления своими финансами. Также был проведён сравнительный анализ существующих веб-сервисов, среди которых ХаниМани, DrebeDengi, Tosh1, Дзен-мани и YNAB. Для объективной оценки аналогов был использован метод анализа иерархий, позволивший определить наиболее подходящий прототип для дальнейшей разработки. По результатам анализа в качестве прототипа был выбран веб-сервис DrebeDengi.

На основе требований спроектированы база данных, архитектура и интерфейс системы, а также основные UML-диаграммы. Реализованы back-end на Spring Boot, PostgreSQL и front-end на React для работы с финансовыми операциями. Тестирование подтвердило корректность системы, а метрика Холстеда показала среднюю сложность функции.

Подводя итоги, можно констатировать, что поставленная цель выпускной квалификационной работы была достигнута [21]. Разработанное веб-приложение предоставляет пользователям удобный инструмент для ведения личного финансового учёта, соответствует предъявленным функциональным и нефункциональным требованиям, обладает возможностью дальнейшего расширения и может использоваться в качестве основы для развития полнофункциональной системы управления личными финансами.

Список использованных источников

1. Хикматов У.С Основы финансовой грамотности: учеб. пособие / У.С. Хикматов, М.Т. Койчуева. – Бишкек: КРСУ, 2015. – 163 с.
2. Домашняя бухгалтерия ХаниМани // ХаниМани [Электронный ресурс]. URL: <https://hmbee.ru/> (дата обращения 05.05.2026).
3. DrebeDengi // DrebeDengi [Электронный ресурс]. URL: <https://www.drebedengi.ru/> (дата обращения 05.05.2026).
4. Tosh1 // Tosh1 [Электронный ресурс]. URL: <https://tosh1.com/> (дата обращения 06.05.2026).
5. Дзен-мани // Дзен-мани [Электронный ресурс]. URL: <https://zenmoney.ru/> (дата обращения 06.05.2026).
6. YNAB // YNAB [Электронный ресурс]. URL: <https://www.ynab.com/> (дата обращения 07.05.2026).
7. Саати Т. Принятие решений. Метод анализа иерархий / Т. Саати. – Москва: Радио и связь, 1993. – 278 стр.
8. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/566218/> (дата обращения: 16.05.2026).
9. Павлова А. И., Кусков Н.С. Разработка структуры web-приложения для учета личных финансов // CyberLeninka [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/razrabotka-struktury-web-prilozheniya-dlya-ucheta-lichnyh-finansov/viewer> (дата обращения: 19.05.2026).
10. Функциональные и нефункциональные требования: Полное руководство // Scand [Электронный ресурс]. URL: <https://scand.com/ru/company/blog/functional-vs-non-functional-requirements/> (дата обращения: 19.05.2026).
11. Как диаграммы Ганта упрощают работу с проектами // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/hygger/articles/415271/> (дата обращения 20.05.2026).

12. Стружкин Н. П. Базы данных: проектирование: учебник для вузов / Н. П. Стружкин, В. В. Годин. – Москва: Юрайт, 2024. – 477 с.
13. Мартин Р. Чистая архитектура / Р. Мартин. – Санкт-Петербург: Питер, 2019. – 352 с.
14. Гамма Э. Приёмы объектно-ориентированного проектирования / Р. Хелм, Р. Джонсон, Дж. Влиссидес. – Санкт-Петербург: Питер, 2020. – 395 с.
15. Основы проектирования интерфейсов для системных аналитиков: ключевые принципы и подходы // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/872168/> (дата обращения 21.05.2026).
16. Шилдт Г. Java. Полное руководство: учеб. пособие / Г. Шилд. – Москва: Вильямс, 2019. – 1104 с.
17. PostgreSQL Documentation // PostgreSQL Global Development Group [Электронный ресурс]. URL: <https://www.postgresql.org/docs> (дата обращения: 10.06.2026).
18. Decision Table – что это и как применять // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/546432/> (дата обращения 10.06.2026).
19. Halstead's Software Metrics - Software Engineering // GeeksforGeeks [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-engineering/software-engineering-halsteads-software-metrics/> (дата обращения 11.06.2026).
20. Оценка характеристик программ системой метрик Холстеда // Справочник [Электронный ресурс]. URL: https://spravochnick.ru/informatika/ocenka_harakteristik_programm_sistemoy_metrik_holsteda/ (дата обращения 11.06.2026).
21. FinTime // GitHub [Электронный ресурс]. URL: <https://github.com/vadim1vanov/FinTime> (дата обращения 16.06.2026).

Приложение А
(обязательное)
Исходной код серверной части

Листинг А.1 – Модуль управления счетами

```
package com.fintime.fintime.Services.Impl;

import com.fintime.fintime.DTO.AccountDto;
import com.fintime.fintime.DTO.AccountInfoDto;
import com.fintime.fintime.Exceptions.BadRequestException;
import com.fintime.fintime.Exceptions.NotFoundException;
import com.fintime.fintime.Factories.AccountDtoFactory;
import com.fintime.fintime.Models.AccountModel;
import com.fintime.fintime.Repository.AccountRepository;
import com.fintime.fintime.Repository.UserRepository;
import com.fintime.fintime.Services.AccountService;
import com.fintime.fintime.Services.TransactionService;
import com.fintime.fintime.Services.UserService;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.time.Instant;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class AccountServiceImpl implements AccountService {
    private final AccountRepository accountRepository;
    private final UserService userService;

    @Override
    @Transactional
    public AccountDto createAccount(AccountDto accountDto) {
        Long currentUserId = userService.getCurrentUserId();

        if
(accountRepository.findByAccountNameAndUserId(accountDto.getAccountName(), currentUserId).isPresent()) {
            throw new BadRequestException("Счёт с данным названием уже создан!");
        }
        List<AccountModel> accountModels =
accountRepository.findAll().stream()
            .filter(accountModel ->
accountModel.getUserId().equals(currentUserId))
            .toList();

        accountRepository.shiftPositionsDown(currentUserId);
    }
}
```

```

        AccountModel account = accountRepository.saveAndFlush(
            AccountModel.builder()
                .accountName(accountDto.getAccountName())
                .currency(accountDto.getCurrency())
                .balance(accountDto.getBalance())
                .status("active")
                .createdAt(Instant.now())
                .userId(currentUserId)
                .accountPosition(accountModels.size() + 1)
                .build()
        );
        return AccountDtoFactory.makeAccountDto(account);
    }

    @Override
    @Transactional
    public void editAccount(Long accountId, AccountDto
updatedAccountDto) {
        AccountModel account =
getAccountForCurrentUser(accountId);
        if(updatedAccountDto.getAccountName() != null
&& !updatedAccountDto.getAccountName().isBlank()){

account.setAccountName(updatedAccountDto.getAccountName());
        }
        if(updatedAccountDto.getStatus() != null
&& !updatedAccountDto.getStatus().isBlank()) {
            account.setStatus(updatedAccountDto.getStatus());
        }
        if(updatedAccountDto.getBalance() != null) {
            account.setBalance(updatedAccountDto.getBalance());
        }
        if(updatedAccountDto.getCurrency() != null
&& !updatedAccountDto.getCurrency().isBlank()) {
            account.setCurrency(updatedAccountDto.getCurrency());
        }
        AccountModel updatedAccount =
accountRepository.saveAndFlush(account);
        AccountDtoFactory.makeAccountDto(updatedAccount);
    }

    @Override
    public List<AccountDto> getAllAccounts(){
        Long currentUserId = userService.getCurrentUserId();
        return accountRepository.findAll().stream()
            .map(AccountDtoFactory::makeAccountDto)
            .filter(accountDto ->
Objects.equals(accountDto.getUserId(), currentUserId))
            .collect(Collectors.toList());
    }

```

```

@Override
public AccountDto getAccount(Long accountId) {
    AccountModel account =
getAccountForCurrentUser(accountId);
    return AccountDtoFactory.makeAccountDto(account);
}

@Override
public List<AccountDto> getActiveAccounts() {
    Long currentUserId = userService.getCurrentUserId();
    return accountRepository.findAll().stream()
        .map(AccountDtoFactory::makeAccountDto)
        .filter(accountDto ->
Objects.equals(accountDto.getStatus(), "active") &&
                Objects.equals(accountDto.getUserId(),
currentUserId))
        .collect(Collectors.toList());
}

@Override
public List<AccountDto> getClosedAccounts() {
    return accountRepository.findAll().stream()
        .map(AccountDtoFactory::makeAccountDto)
        .filter(accountDto ->
Objects.equals(accountDto.getStatus(), "closed"))
        .collect(Collectors.toList());
}

@Override
public void deleteAccount(Long accountId) {
    AccountModel account =
getAccountForCurrentUser(accountId);
    accountRepository.delete(account);
}

@Override
public void closeAccount(Long accountId) {
    AccountModel account =
getAccountForCurrentUser(accountId);
    account.setStatus("closed");
    AccountModel updatedAccount =
accountRepository.saveAndFlush(account);
    AccountDtoFactory.makeAccountDto(updatedAccount);
}

@Override
public void restoreAccount(Long accountId) {
    AccountModel account =
getAccountForCurrentUser(accountId);
    account.setStatus("active");
    AccountModel updatedAccount =
accountRepository.saveAndFlush(account);
}

```

```

        AccountDtoFactory.makeAccountDto(updatedAccount);
    }

    @Override
    public void deleteAllAccounts() {
        Long currentUserId = userService.getCurrentUserId();
        List<AccountModel> accounts =
accountRepository.findAll().stream()
        .filter(account ->
account.getUserId().equals(currentUserId))
        .toList();
        accountRepository.deleteAll(accounts);
    }

    @Override
    public AccountModel getAccountForCurrentUser(Long accountId) {
        Long currentUser = userService.getCurrentUserId();
        AccountModel account =
accountRepository.findById(accountId).orElseThrow(
        () -> new NotFoundException("Account not
found!"));
        if(!account.getUserId().equals(currentUser)) {
            throw new AccessDeniedException("You don't have
access!");
        }
        return account;
    }

    @Override
    public void increaseAccountBalance(Long accountId, BigDecimal
amount) {
        AccountModel accountModel =
getAccountForCurrentUser(accountId);

accountModel.setBalance(accountModel.getBalance().add(amount));
        accountRepository.saveAndFlush(accountModel);
    }

    @Override
    public void decreaseAccountBalance(Long accountId, BigDecimal
amount) {
        AccountModel accountModel =
getAccountForCurrentUser(accountId);

accountModel.setBalance(accountModel.getBalance().subtract(amount)
);
        accountRepository.saveAndFlush(accountModel);
    }

    @Override
    @Transactional
    public void reorderAccounts(List<AccountDto> newOrder) {

```

```

        Long currentUserId = userService.getCurrentUserId();
        List<AccountModel> userAccounts =
accountRepository.findAll().stream()
                .filter(accountModel ->
accountModel.getUserId().equals(currentUserId))
                .toList();

        Map<Long, AccountModel> accountMap = userAccounts.stream()
                .collect(Collectors.toMap(AccountModel::getId, a
-> a));

        for (int i = 0; i < newOrder.size(); i++) {
            AccountDto dto = newOrder.get(i);
            AccountModel account = accountMap.get(dto.getId());
            if (account != null) {
                account.setAccountPosition(i);
            }
        }

        accountRepository.saveAll(userAccounts);
    }
}

```

Листинг А.2 – Модуль управления кредитами

```

package com.fintime.fintime.Services.Impl;

import com.fintime.fintime.DTO.CreditDataDto;

import com.fintime.fintime.DTO.CreditDto;
import com.fintime.fintime.DTO.CreditPaymentDto;
import com.fintime.fintime.DTO.TransactionDto;
import com.fintime.fintime.Enums.CreditStatus;
import com.fintime.fintime.Enums.CreditType;
import com.fintime.fintime.Enums.TransactionType;
import com.fintime.fintime.Exceptions.BadRequestException;
import com.fintime.fintime.Exceptions.NotFoundException;
import com.fintime.fintime.Factories.CreditDataDtoFactory;
import com.fintime.fintime.Models.AccountModel;
import com.fintime.fintime.Models.CreditModel;
import com.fintime.fintime.Repository.AccountRepository;
import com.fintime.fintime.Repository.CreditRepository;
import com.fintime.fintime.Services.AccountService;
import com.fintime.fintime.Services.CreditService;
import com.fintime.fintime.Services.TransactionService;
import com.fintime.fintime.Services.UserService;
import lombok.AllArgsConstructor;
import lombok.RequiredArgsConstructor;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;

```

```

import java.math.RoundingMode;
import java.time.Instant;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class CreditServiceImpl implements CreditService {
    private final UserService userService;
    private final AccountService accountService;
    private final CreditRepository creditRepository;
    private final AccountRepository accountRepository;
    private final TransactionService transactionService;

    @Override
    public BigDecimal calculateMonthlyPayment(BigDecimal
principalAmount,
                                                BigDecimal
interestRate,
                                                Integer termMonths){
        BigDecimal monthlyRate =
interestRate.divide(BigDecimal.valueOf(12), 15,
RoundingMode.HALF_UP)
                .divide(BigDecimal.valueOf(100), 10,
RoundingMode.HALF_UP);
        if(monthlyRate.compareTo(BigDecimal.ZERO) == 0){
            return
principalAmount.divide(BigDecimal.valueOf(termMonths), 2,
RoundingMode.HALF_UP);
        }
        BigDecimal one = BigDecimal.ONE;
        BigDecimal calculateAnnuityFactor =
monthlyRate.add(one).pow(termMonths);

        BigDecimal divideAnnuity =
calculateAnnuityFactor.multiply(monthlyRate);
        BigDecimal divisorAnnuity =
calculateAnnuityFactor.subtract(BigDecimal.valueOf(1));

        return
principalAmount.multiply( divideAnnuity.divide(divisorAnnuity,
15, RoundingMode.HALF_UP) )
                .setScale(2, RoundingMode.HALF_UP);
    }

    @Override
    public BigDecimal calculateTotalAmount(BigDecimal
principalAmount,

```

```

                                BigDecimal
interestRate,
                                Integer termMonths){
    return calculateMonthlyPayment(principalAmount,
interestRate, termMonths)
        .multiply(BigDecimal.valueOf(termMonths)).setScale
(2, RoundingMode.HALF_UP);
    }

    @Override
    public BigDecimal calculateOverpayment(BigDecimal
principalAmount,
                                BigDecimal
interestRate,
                                Integer termMonths ){
        return calculateTotalAmount(principalAmount, interestRate,
termMonths).subtract(principalAmount);
    }

    @Override
    public BigDecimal calculateRemainingBalance(BigDecimal
principalAmount,
                                BigDecimal
interestRate,
                                Integer
termMonths,
                                Integer
numberPayment) {
        BigDecimal monthlyRate = interestRate
            .divide(BigDecimal.valueOf(100), 15,
RoundingMode.HALF_UP)
            .divide(BigDecimal.valueOf(12), 15,
RoundingMode.HALF_UP);

        if (monthlyRate.compareTo(BigDecimal.ZERO) == 0) {
            BigDecimal paid =
calculateMonthlyPayment(principalAmount, interestRate, termMonths)
                .multiply(BigDecimal.valueOf(numberPayment));
            return principalAmount.subtract(paid).setScale(2,
RoundingMode.HALF_UP);
        }

        BigDecimal monthlyPayment =
calculateMonthlyPayment(principalAmount, interestRate,
termMonths);
        BigDecimal onePlusRatePowN =
BigDecimal.ONE.add(monthlyRate).pow(numberPayment);

        BigDecimal firstPart =
principalAmount.multiply(onePlusRatePowN);
        BigDecimal secondPart = monthlyPayment.multiply(

```

```

onePlusRatePowN.subtract(BigDecimal.ONE).divide(monthlyRate, 15,
RoundingMode.HALF_UP)
    );

    return firstPart.subtract(secondPart).setScale(2,
RoundingMode.HALF_UP);
}

@Override
public CreditDataDto createCredit(Long accountId,
CreditDataDto creditDto) {
    AccountModel account =
accountService.getAccountForCurrentUser(accountId);

    BigDecimal principal = creditDto.getPrincipalAmount();
    BigDecimal rate = creditDto.getInterestRate();
    int term = creditDto.getTermMonths();

    BigDecimal monthlyPayment =
calculateMonthlyPayment(principal, rate, term);
    BigDecimal totalAmount =
monthlyPayment.multiply(BigDecimal.valueOf(term));
    BigDecimal overpayment = totalAmount.subtract(principal);
    CreditModel creditModel = CreditModel.builder()
        .accountId(accountId)
        .userId(account.getUserId())
        .principalAmount(principal)
        .interestRate(rate)
        .termMonths(term)
        .startDate(creditDto.getStartDate())
        .endDate(creditDto.getStartDate().plusMonths(term)
)

        .type(CreditType.CREDIT_LINE)
        .status(CreditStatus.ACTIVE)
        .purpose(creditDto.getPurpose())
        .createdAt(Instant.now())
        .totalAmount(totalAmount)
        .remainingBalance(totalAmount)
        .monthlyPayment(monthlyPayment)
        .paidPaymentsCount(0)
        .overpayment(overpayment)
        .paidInterest(BigDecimal.ZERO)
        .lastPaymentDate(null)
        .build();

    creditRepository.saveAndFlush(creditModel);
    return
CreditDataDtoFactory.makeCreditDataDto(creditModel);
}

@Override

```

```

    public List<CreditDataDto> getAllCredits() {
        Long currentUserId = userService.getCurrentUserId();
        return creditRepository.findAll().stream()
            .map(credit -> {
                CreditDataDto creditDataDto =
                    CreditDataDtoFactory.makeCreditDataDto(credit);

                creditDataDto.setProgress(calculateCreditProgress(credit.getId()));
            })
            .filter(creditDto ->
                Objects.equals(creditDto.getUserId(), currentUserId))
            .collect(Collectors.toList());
    }

    @Override
    public CreditModel getCreditForCurrentUser(Long creditId) {
        Long currentUser = userService.getCurrentUserId();
        CreditModel credit =
            creditRepository.findById(creditId).orElseThrow(
                () -> new NotFoundException("Account not
                found!"));
        if (!credit.getUserId().equals(currentUser)) {
            throw new AccessDeniedException("You don't have
            access!");
        }
        return credit;
    }

    @Override
    public CreditDto creditInfo(Long creditId) {
        CreditModel credit = getCreditForCurrentUser(creditId);
        BigDecimal progress = calculateCreditProgress(creditId);
        return CreditDto.builder()
            .id(creditId)
            .nextPaymentDate(LocalDate.now())
            .type(CreditType.CREDIT_LINE)
            .paidInterest(BigDecimal.valueOf(228))
            .startDate(credit.getStartDate())
            .endDate(credit.getEndDate())
            .userId(credit.getUserId())
            .accountId(credit.getAccountId())
            .createdAt(credit.getCreatedAt())
            .purpose(credit.getPurpose())
            .status(credit.getStatus())
            .termMonths(credit.getTermMonths())
            .principalAmount(credit.getPrincipalAmount())
            .interestRate(credit.getInterestRate())
    }

```

```

        .monthlyPayment(credit.getMonthlyPayment() !=
null ?
credit.getMonthlyPayment() : calculateMonthlyPayment(
        credit.getPrincipalAmount(),
        credit.getInterestRate(),
        credit.getTermMonths()))
        .remainingBalance(credit.getRemainingBalance())
        .progress(progress)
        .build();
    }

    @Override
    public void closeCredit(Long creditId){
        CreditModel credit = getCreditForCurrentUser(creditId);
        credit.setStatus(CreditStatus.PAID_OFF);
        CreditModel updatedCredit =
creditRepository.saveAndFlush(credit);
    }

    @Override
    public void payCreditDebt(Long creditId, TransactionDto
transactionDto) {
        CreditModel credit = getCreditForCurrentUser(creditId);
        AccountModel account =
accountService.getAccountForCurrentUser(credit.getAccountId());

        if (transactionDto.getAmount() == null ||
transactionDto.getAmount().compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("Payment amount
must be positive");
        }

        if
(transactionDto.getAmount().compareTo(account.getBalance()) > 0) {
            throw new IllegalArgumentException("Недостаточно
средств на счёте");
        }

        BigDecimal amountRounded =
transactionDto.getAmount().setScale(2, RoundingMode.HALF_UP);
        BigDecimal balanceRounded =
credit.getRemainingBalance().setScale(2, RoundingMode.HALF_UP);

        if (amountRounded.compareTo(balanceRounded) > 0) {
            throw new IllegalArgumentException("Сумма превышает
остаток по кредиту");
        }

        transactionDto.setCreditId(creditId);
        transactionDto.setAccountId(credit.getAccountId());
        transactionDto.setUserId(credit.getUserId());
    }

```

```

        transactionDto.setCreatedAt(Instant.now());

transactionDto.setTransactionType(TransactionType.EXPENSE);
        transactionDto.setCategoryName("Кредиты и долги");

transactionService.createExpenseTransaction(credit.getAccountId(),
transactionDto);

        BigDecimal newBalance =
credit.getRemainingBalance().subtract(amountRounded).max(BigDecimal.ZERO);

        credit.setRemainingBalance(newBalance);
        credit.setLastPaymentDate(LocalDate.now());
        credit.setPaidPaymentsCount(credit.getPaidPaymentsCount()
+ 1);

        int newPaidCount = credit.getPaidPaymentsCount();
        int remainingMonths = Math.max(0, credit.getTermMonths() -
newPaidCount);

        if (remainingMonths == 0 ||
newBalance.compareTo(BigDecimal.ZERO) <= 0) {
            credit.setStatus(CreditStatus.PAID_OFF);
            credit.setMonthlyPayment(BigDecimal.ZERO);
        }

        creditRepository.saveAndFlush(credit);
    }

    @Override
    public void payOffCredit(Long creditId, TransactionDto
transactionDto) {
        CreditModel credit = getCreditForCurrentUser(creditId);
        AccountModel account =
accountService.getAccountForCurrentUser(credit.getAccountId());

        BigDecimal amount = credit.getRemainingBalance();

        if (amount.compareTo(BigDecimal.ZERO) <= 0) {
            throw new BadRequestException("Credit already paid
off");
        }

        if (amount.compareTo(account.getBalance()) > 0) {
            throw new IllegalArgumentException("Account balance is
insufficient to pay off the credit");
        }

        transactionDto.setAmount(amount);
    }

```

```

        transactionDto.setAccountId(credit.getAccountId());
        transactionDto.setUserId(credit.getUserId());
        transactionDto.setCreatedAt(Instant.now());

transactionDto.setTransactionType(TransactionType.EXPENSE);

        payCreditDebt(creditId, transactionDto);

        credit.setStatus(CreditStatus.PAID_OFF);
        creditRepository.saveAndFlush(credit);
    }

    @Override
    public BigDecimal calculateCreditProgress(Long creditId) {
        CreditModel credit = getCreditForCurrentUser(creditId);
        BigDecimal originalTotal = calculateTotalAmount(
            credit.getPrincipalAmount(),
            credit.getInterestRate(),
            credit.getTermMonths()
        );

        if (originalTotal.compareTo(BigDecimal.ZERO) == 0) return
BigDecimal.ZERO;

        BigDecimal paid =
originalTotal.subtract(credit.getRemainingBalance());
        return paid.divide(originalTotal, 10,
RoundingMode.HALF_UP)
            .multiply(BigDecimal.valueOf(100))
            .setScale(2, RoundingMode.HALF_UP);
    }

    @Override
    public void deleteCredit(Long creditId){
        CreditModel credit = getCreditForCurrentUser(creditId);
        creditRepository.delete(credit);
    }

    @Override
    public void setPaidPaymentsCount(Long creditId, Integer count)
{
        CreditModel credit = getCreditForCurrentUser(creditId);

        if (count < 0 || count > credit.getTermMonths()) {
            throw new BadRequestException("Invalid payments
count");
        }
    }

```

```

        credit.setPaidPaymentsCount(count);

        BigDecimal remaining = calculateRemainingBalance(
            credit.getPrincipalAmount(),
            credit.getInterestRate(),
            credit.getTermMonths(),
            count
        );
        credit.setRemainingBalance(remaining);

        if (count >= credit.getTermMonths()) {
            credit.setStatus(CreditStatus.PAID_OFF);
        } else {
            credit.setStatus(CreditStatus.ACTIVE);
        }

        creditRepository.saveAndFlush(credit);
    }

    @Override
    public List<TransactionDto> getCreditTransactions(Long
creditId) {

        return
transactionService.getTransactionsByCredit(creditId);
    }

    @Override
    public List<CreditPaymentDto> getFuturePaymentSchedule(Long
creditId) {
        CreditModel credit = getCreditForCurrentUser(creditId);

        if
(credit.getRemainingBalance().compareTo(BigDecimal.ZERO) <= 0) {
            return List.of();
        }

        int remainingMonths = Math.max(0, credit.getTermMonths() -
credit.getPaidPaymentsCount());
        if (remainingMonths <= 0) return List.of();

        return buildFutureSchedule(
            credit.getRemainingBalance(),
            credit.getMonthlyPayment(),
            credit.getInterestRate(),
            remainingMonths,
            credit.getPaidPaymentsCount(),
            credit.getStartDate()
        );
    }
}

```

```

private List<CreditPaymentDto> buildFutureSchedule(
    BigDecimal remainingBalance,
    BigDecimal monthlyPayment,
    BigDecimal rate,
    int remainingMonths,
    int paidCount,
    LocalDate startDate) {

    List<CreditPaymentDto> schedule = new ArrayList<>();
    if (remainingMonths <= 0 ||
remainingBalance.compareTo(BigDecimal.ZERO) <= 0) {
        return schedule;
    }

    BigDecimal balance = remainingBalance;
    LocalDate currentDate = startDate != null ? startDate :
LocalDate.now();

    BigDecimal fixedPayment = monthlyPayment;
    if (fixedPayment == null ||
fixedPayment.compareTo(BigDecimal.ZERO) <= 0) {
        fixedPayment =
calculateMonthlyPayment(remainingBalance, rate, remainingMonths);
    }

    BigDecimal monthlyRate = rate
        .divide(BigDecimal.valueOf(100), 12,
RoundingMode.HALF_UP)
        .divide(BigDecimal.valueOf(12), 12,
RoundingMode.HALF_UP);

    for (int i = 1; i <= remainingMonths; i++) {
        BigDecimal interest =
balance.multiply(monthlyRate).setScale(2, RoundingMode.HALF_UP);

        BigDecimal payment = fixedPayment;
        BigDecimal principal = payment.subtract(interest);

        if (principal.compareTo(BigDecimal.ZERO) < 0) {
            principal = BigDecimal.ZERO;
        }

        balance =
balance.subtract(principal).max(BigDecimal.ZERO);

        if (i == remainingMonths) {
            principal = principal.add(balance);
            balance = BigDecimal.ZERO;
        }
    }
}

```

```

        LocalDate paymentDate =
currentDate.plusMonths(paidCount + i);

        schedule.add(CreditPaymentDto.builder()
            .num(paidCount + i)
            .date(paymentDate)
            .payment(payment.setScale(2,
RoundingMode.HALF_UP)) // всегда 33890
            .interest(interest)
            .principal(principal.setScale(2,
RoundingMode.HALF_UP))
            .balance(balance.setScale(2,
RoundingMode.HALF_UP))
            .build());
    }

    return schedule;
}
}

```

Листинг А.3 – Модуль управления вкладами

```

package com.fintime.fintime.Services.Impl;

import com.fintime.fintime.DTO.DepositDto;
import com.fintime.fintime.Enums.CapitalizationFrequency;
import com.fintime.fintime.Enums.DepositStatus;
import com.fintime.fintime.Enums.ReplenishmentFrequency;
import com.fintime.fintime.Exceptions.BadRequestException;
import com.fintime.fintime.Exceptions.NotFoundException;
import com.fintime.fintime.Factories.DepositDtoFactory;
import com.fintime.fintime.Models.DepositModel;
import com.fintime.fintime.Repository.DepositRepository;
import com.fintime.fintime.Services.DepositService;
import com.fintime.fintime.Services.UserService;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class DepositServiceImpl implements DepositService {

```

```

private final UserService userService;
private final DepositRepository depositRepository;

@Override
public DepositDto createDeposit(DepositDto depositDto) {
    Long currentUserId = userService.getCurrentUserId();

    DepositModel deposit = depositRepository.saveAndFlush(
        DepositModel.builder()
            .userId(currentUserId)
            .depositName(depositDto.getDepositName())
            .principalAmount(depositDto.getPrincipalAmount())
            .interestRate(depositDto.getInterestRate())
            .termDays(depositDto.getTermDays())
            .startDate(depositDto.getStartDate())
            .endDate(depositDto.getEndDate())
            .status(DepositStatus.ACTIVE)
            .createdAt(depositDto.getCreatedAt())
            .replenishmentFrequency(depositDto.getReplenishmentFrequency())
            .capitalizationFrequency(depositDto.getCapitalizationFrequency())
            .termType(depositDto.getTermType())
            .lastInterestAccrualDate(depositDto.getLastInterestAccrualDate())
            .currentAmount(depositDto.getPrincipalAmount())
            .build()
    );

    return DepositDtoFactory.makeDepositDto(deposit);
}

@Override
public DepositModel getDepositById(Long depositId) {
    Long currentUserId = userService.getCurrentUserId();
    DepositModel deposit =
        depositRepository.findById(depositId)
            .orElseThrow(() -> new NotFoundException("Deposit not found!"));

    if (!deposit.getUserId().equals(currentUserId)) {
        throw new AccessDeniedException("You don't have access!");
    }

    return deposit;
}

```

```

    }

    @Override
    public void deleteDeposit(Long depositId) {
        DepositModel deposit = getDepositById(depositId);
        depositRepository.delete(deposit);
    }

    @Override
    public List<DepositDto> getAllDeposits() {
        Long currentUserId = userService.getCurrentUserId();
        return depositRepository.findAll().stream()
            .map(DepositDtoFactory::makeDepositDto)
            .filter(dto -> Objects.equals(dto.getUserId(),
currentUserId))
            .collect(Collectors.toList());
    }

    @Override
    public BigDecimal calculateSimpleInterest(DepositDto
depositDto) {
        LocalDate startDate = depositDto.getStartDate();
        LocalDate endDate = depositDto.getEndDate();

        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Deposit dates are
missing");
        }
        if (depositDto.getInterestRate() == null ||
depositDto.getInterestRate().compareTo(BigDecimal.ZERO) < 0) {
            throw new IllegalArgumentException("Invalid interest
rate");
        }
        if (depositDto.getPrincipalAmount() == null ||
depositDto.getPrincipalAmount().compareTo(BigDecimal.ZERO) < 0) {
            throw new IllegalArgumentException("Invalid principal
amount");
        }

        long days = ChronoUnit.DAYS.between(startDate, endDate);
        BigDecimal timeInYears =
BigDecimal.valueOf(days).divide(BigDecimal.valueOf(365), 15,
RoundingMode.HALF_UP);

        return depositDto.getPrincipalAmount()
            .multiply(depositDto.getInterestRate())
            .divide(BigDecimal.valueOf(100), 15,
RoundingMode.HALF_UP)
    }

```

```

        .multiply(timeInYears)
        .setScale(2, RoundingMode.HALF_UP);
    }

    @Override
    public BigDecimal calculateCompoundInterest(
        BigDecimal principal,
        BigDecimal annualRate,
        Integer termDays,
        CapitalizationFrequency capitalizationFrequency
    ) {
        if (annualRate == null ||
            annualRate.compareTo(BigDecimal.ZERO) < 0) {
            throw new IllegalArgumentException("Invalid interest
            rate");
        }
        if (termDays == null || termDays <= 0) {
            throw new IllegalArgumentException("Term days must be
            positive");
        }

        BigDecimal rateDecimal =
            annualRate.divide(BigDecimal.valueOf(100), 15,
            RoundingMode.HALF_UP);
        int n;

        switch (capitalizationFrequency) {
            case DAILY -> n = 365;
            case WEEKLY -> n = 52;
            case MONTHLY -> n = 12;
            case HALF_YEARLY -> n = 2;
            case YEARLY -> n = 1;
            case NONE -> n = 1;
            default -> throw new BadRequestException("Unsupported
            capitalization frequency: " + capitalizationFrequency);
        }

        BigDecimal timeInYears =
            BigDecimal.valueOf(termDays).divide(BigDecimal.valueOf(365), 15,
            RoundingMode.HALF_UP);
        BigDecimal nVal = BigDecimal.valueOf(n);
        BigDecimal rt = rateDecimal.divide(nVal, 15,
            RoundingMode.HALF_UP);

        BigDecimal base = BigDecimal.ONE.add(rt);
        BigDecimal power =
            timeInYears.multiply(BigDecimal.valueOf(n));

        BigDecimal amount =
            principal.multiply(base.pow(power.intValue()));
        return amount.setScale(2, RoundingMode.HALF_UP);
    }

```

```

    }

    @Override
    public BigDecimal calculateTotalAmount(
        BigDecimal principal,
        BigDecimal annualRate,
        Integer termDays,
        CapitalizationFrequency capitalizationFrequency
    ) {
        if
(capitalizationFrequency.equals(CapitalizationFrequency.NONE)) {
            return principal.add(
                calculateSimpleInterest(
                    DepositDto.builder()
                        .principalAmount(principal)
                        .interestRate(annualRate)
                        .startDate(LocalDate.now())
                        .endDate(LocalDate.now().plusD
ays(termDays))
                        .build()
                )
            );
        }
        return calculateCompoundInterest(principal, annualRate,
termDays, capitalizationFrequency);
    }

    @Override
    public BigDecimal calculateProfit(
        BigDecimal principal,
        BigDecimal annualRate,
        Integer termDays,
        CapitalizationFrequency capitalizationFrequency
    ) {
        return calculateTotalAmount(principal, annualRate,
termDays, capitalizationFrequency)
            .subtract(principal)
            .setScale(2, RoundingMode.HALF_UP);
    }

    @Override
    public BigDecimal calculateDepositProgress(Long depositId) {
        DepositModel deposit = getDepositById(depositId);
        LocalDate startDate = deposit.getStartDate();
        LocalDate endDate = deposit.getEndDate();
        LocalDate today = LocalDate.now();

        if (startDate == null || endDate == null) {

```

```

        throw new IllegalArgumentException("Deposit dates are
missing");
    }

    if (today.isBefore(startDate)) {
        return BigDecimal.ZERO;
    }
    if (today.isAfter(endDate)) {
        return BigDecimal.valueOf(100);
    }

    long totalDays = ChronoUnit.DAYS.between(startDate,
endDate);
    if (totalDays <= 0) {
        return BigDecimal.ZERO;
    }

    long elapsedDays = ChronoUnit.DAYS.between(startDate,
today);
    return BigDecimal.valueOf(elapsedDays)
        .divide(BigDecimal.valueOf(totalDays), 15,
RoundingMode.HALF_UP)
        .multiply(BigDecimal.valueOf(100))
        .setScale(2, RoundingMode.HALF_UP);
}

@Override
public void closeDeposit(Long depositId) {
    DepositModel deposit = getDepositById(depositId);
    deposit.setStatus(DepositStatus.CLOSED);
    deposit.setEndDate(LocalDate.now());
    depositRepository.saveAndFlush(deposit);
}
private long getPassedPeriods(
    LocalDate lastDate,
    LocalDate currentDate,
    CapitalizationFrequency frequency
) {
    return switch (frequency) {
        case DAILY ->
            ChronoUnit.DAYS.between(
                lastDate,
                currentDate
            );
        case WEEKLY ->
            ChronoUnit.WEEKS.between(
                lastDate,
                currentDate
            );
    }
}

```

```

        );

    case MONTHLY ->
        ChronoUnit.MONTHS.between(
            lastDate,
            currentDate
        );

    case HALF_YEARLY ->
        ChronoUnit.MONTHS.between(
            lastDate,
            currentDate
        ) / 6;

    case YEARLY ->
        ChronoUnit.YEARS.between(
            lastDate,
            currentDate
        );

    default -> 0;
};
}
private BigDecimal calculatePeriodInterest(
    BigDecimal amount,
    BigDecimal annualRate,
    CapitalizationFrequency frequency
) {

    BigDecimal periodsPerYear;

    switch (frequency) {

        case DAILY ->
            periodsPerYear =
                BigDecimal.valueOf(365);

        case WEEKLY ->
            periodsPerYear =
                BigDecimal.valueOf(52);

        case MONTHLY ->
            periodsPerYear =
                BigDecimal.valueOf(12);

        case HALF_YEARLY ->
            periodsPerYear =
                BigDecimal.valueOf(2);

        case YEARLY ->
            periodsPerYear =
                BigDecimal.ONE;
    }
}

```

```

        default ->
            periodsPerYear =
                BigDecimal.ONE;
    }

    return amount
        .multiply(
            annualRate.divide(
                BigDecimal.valueOf(100),
                10,
                RoundingMode.HALF_UP
            )
        )
        .divide(
            periodsPerYear,
            10,
            RoundingMode.HALF_UP
        );
    }

    @Transactional
    public void accrueInterest(DepositModel deposit) {
        LocalDate today = LocalDate.now();

        if (deposit.getNextInterestAccrualDate() == null) {
            deposit.setNextInterestAccrualDate(
                calculateNextDate(deposit.getStartDate(),
deposit.getCapitalizationFrequency())
            );
        }

        while
            (!today.isBefore(deposit.getNextInterestAccrualDate())
                && deposit.getStatus() == DepositStatus.ACTIVE) {

            BigDecimal interest = calculatePeriodInterest(
                deposit.getCurrentAmount(),
                deposit.getInterestRate(),
                deposit.getCapitalizationFrequency()
            );

            deposit.setCurrentAmount(

deposit.getCurrentAmount().add(interest).setScale(2,
RoundingMode.HALF_UP)
            );

            deposit.setNextInterestAccrualDate(
                calculateNextDate(
                    deposit.getNextInterestAccrualDate(),
                    deposit.getCapitalizationFrequency()
                )
            );
        }
    }

```

```
        );  
    }  
  
    depositRepository.save(deposit);  
}  
  
    private LocalDate calculateNextDate(LocalDate baseDate,  
CapitalizationFrequency frequency) {  
        return switch (frequency) {  
            case DAILY -> baseDate.plusDays(1);  
            case WEEKLY -> baseDate.plusWeeks(1);  
            case MONTHLY -> baseDate.plusMonths(1);  
            case HALF_YEARLY -> baseDate.plusMonths(6);  
            case YEARLY -> baseDate.plusYears(1);  
            case NONE -> baseDate;  
        };  
    }  
}
```

Приложение Б
(обязательное)
Исходной код клиентской части

Листинг Б.1 – Модуль управления счетами

```

import { useNavigate, useParams } from "react-router-dom";
import { useEffect, useState, useCallback } from "react";
import {
  DndContext,
  closestCenter,
  KeyboardSensor,
  PointerSensor,
  useSensor,
  useSensors,
  DragOverlay,
  MeasuringStrategy,
} from "@dnd-kit/core";
import {
  arrayMove,
  SortableContext,
  sortableKeyboardCoordinates,
  rectSortingStrategy,
  useSortable,
  defaultAnimateLayoutChanges,
} from "@dnd-kit/sortable";
import { CSS } from "@dnd-kit/utilities";
import { NavLink } from "react-router-dom";
import Card from "../Card/Card";
import ConfirmModal from "../Modal/ConfirmModal";
import TransactionModal from "../Modal/TransactionModal";
import TransferModal from "../Modal/TransferModal";
import {
  getAccounts,
  deleteAccount,
  closeAccount,
  restoreAccount,
  income,
  expense,
  reorderAccounts,
  transfer
} from "../api/api";
import CreateAccountModal from "../Modal/CreateAccountModal";
import { createAccount, getTotalAccountsAmount } from
"../api/api";

import {
  FaArrowLeft
} from "react-icons/fa"
import styles from "../Accounts.module.css";

function customAnimateLayoutChanges(args) {
  if (args.isSorting || args.wasDragging) {
    return defaultAnimateLayoutChanges(args);
  }
  return true;
}

```

```

function SortableCard({
  id,
  account,
  onDelete,
  onClose,
  onRestore,
  onIncome,
  onExpense,
  onTransfer,
  navigateToAccount
}) {
  const { attributes, listeners, setNodeRef, transform,
transition, isDragging } = useSortable({
    id,
    animateLayoutChanges: customAnimateLayoutChanges,
  });

  const [dragging, setDragging] = useState(false);
  const [hovered, setHovered] = useState(false);

  const style = {
    transform: CSS.Transform.toString(transform),
    transition: transition || "transform 0.3s cubic-bezier(0.34,
1.56, 0.64, 1)",
    opacity: isDragging ? 0 : 1,
    visibility: isDragging ? "hidden" : "visible",
    zIndex: isDragging ? 1000 : hovered ? 50 : 1,
    boxShadow: isDragging ? "0 20px 40px rgba(0,0,0,0.25)" :
"none",
    borderRadius: "22px",
    height: "auto",
    width: "100%",
    pointerEvents: isDragging ? "none" : "auto",
    margin: "0",
    overflow: "visible",
    cursor: "pointer",
  };

  return (
    <div
      ref={setNodeRef}
      style={style}
      {...attributes}
      {...listeners}
      onMouseEnter={() => !isDragging && setHovered(true)}
      onMouseLeave={() => !isDragging && setHovered(false)}
      onMouseDown={() => setDragging(false)}
      onDragStart={() => setDragging(true)}
      onClick={() => {
        if (!dragging) navigateToAccount();
      }}
    >

```

```

    }}
  >
  <Card
    account={account}
    onDelete={onDelete}
    onClose={onClose}
    onRestore={onRestore}
    onIncome={onIncome}
    onExpense={onExpense}
    onTransfer={onTransfer}
    isDragging={isDragging} // <-- вот здесь
  />
</div>
);
}

function DraggableOverlay({ account }) {
  if (!account) return null;

  return (
    <div
      style={{
        width: "300px",
        height: "230px",
        pointerEvents: "none",
        opacity: 1,
        borderRadius: "30px",
        background: "#ffffff",
      }}
    >
      <Card account={account} />
    </div>
  );
}

export default function Accounts() {
  const [accounts, setAccounts] = useState([]);
  const [modal, setModal] = useState(null);
  const [activeId, setActiveId] = useState(null);
  const navigate = useNavigate();
  const [loading, setLoading] = useState(true);
  const handleCreateAccount = () => {
    setModal({ type: "create" });
  };

  const [activeTab, setActiveTab] = useState("Все счета");
  const sensors = useSensors(
    useSensor(PointerSensor, { activationConstraint: { distance:
5 } }),
    useSensor(KeyboardSensor, { coordinateGetter:
sortableKeyboardCoordinates })

```

```

    );

const loadAccounts = useCallback(async () => {
  try {
    const data = await getAccounts();
    const sorted = [...data].sort(
      (a, b) => (a.account_position ?? 0) - (b.account_position ??
0)
    );
    setAccounts(sorted);
  } catch (err) {
    console.error("Ошибка загрузки:", err);
  } finally {
    setLoading(false);
  }
}, []);

const [totalAccountsAmount, setTotalAccountsAmount] =
useState({});
const loadAnalytics = async () => {
  try {
    const amount = await getTotalAccountsAmount();
    setTotalAccountsAmount(amount);
  } catch (err) {
    console.error(err);
  }
};

useEffect(() => {
  loadAccounts();
  loadAnalytics();
}, [loadAccounts]);

const handleDragStart = (event) => setActiveId(event.active.id);

const handleDragEnd = (event) => {
  setActiveId(null);
  const { active, over } = event;

  if (over && active.id !== over.id) {
    setAccounts((items) => {
      const oldIndex = items.findIndex((i) => i.id ===
active.id);
      const newIndex = items.findIndex((i) => i.id === over.id);
      const newOrder = arrayMove(items, oldIndex, newIndex);
      reorderAccounts(newOrder);
      return newOrder;
    });
  }
};

const activeAccount = accounts.find((acc) => acc.id ===
activeId);

```

```

const getCurrencySymbol = (currency) => {
  switch (currency) {
    case "RUB": return "₽";
    case "USD": return "$";
    case "EUR": return "€";
    case "JPY": return "¥";
    case "CNY": return "¥";
    case "GBP": return "£";
    default: return currency;
  }
};

const getCurrencyFlag = (currency) => {
  switch (currency) {
    case "RUB": return "/flags/ru.svg";
    case "USD": return "/flags/us.svg";
    case "EUR": return "/flags/eu.svg";
    case "JPY": return "/flags/jp.svg";
    case "CNY": return "/flags/cn.svg";
    case "GBP": return "/flags/gb.svg";
    default: return "/flags/default.svg";
  }
};

const currencyBalances = Object.entries(totalAccountsAmount)
  .filter(([_, amount]) => Number(amount) !== 0);
return (
  <div className={styles.container}>
    <div className={styles.header}>
    </div>
    <div className={styles.summaryCard}>
      <div className={styles.summaryLabel}>
        Общий баланс счетов
      </div>
    </div>

    {currencyBalances.length > 0 && (
      <div className={styles.currencyGrid}>
        {currencyBalances.map(([currency, amount]) => (
          <div
            key={currency}
            className={styles.currencyCard}
          >
            <div className={styles.currencyCardHeader}>
              <div className={styles.currencyFlagWrapper}>
                <img
                  src={getCurrencyFlag(currency)}
                  alt={currency}
                  className={
                    currency === "USD"
                      ? styles.flagUs
                      : currency === "CNY"
                      ? styles.flagChina

```

```

        : styles.flag
      }
    />
  </div>

  <span className={styles.currencyCode}>
    {currency}
  </span>
</div>

<div className={styles.currencyCardAmount}>
  {Number(amount).toLocaleString("ru-RU", {
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  })}
  <span className={styles.currencySign}>
    {getCurrencySymbol(currency)}
  </span>
</div>
</div>
  )})
</div>
})

</div>
  <DndContext
    sensors={sensors}
    collisionDetection={closestCenter}
    onDragStart={handleDragStart}
    onDragEnd={handleDragEnd}
    measuring={{ droppable: { strategy:
MeasuringStrategy.Always } }}
  >
    <SortableContext items={accounts.map((acc) => acc.id)}
strategy={rectSortingStrategy}>
      <div className={styles.grid}>
        {accounts.map((acc) => (
<SortableCard
  key={acc.id}
  id={acc.id}
  account={acc}
  onDelete={() => setModal({ type: "delete", acc })}
  onClose={() => setModal({ type: "close", acc })}
  onRestore={() => setModal({ type: "restore", acc })}
  onIncome={() => setModal({ type: "income", acc })}
  onExpense={() => setModal({ type: "expense", acc })}
  onTransfer={() => setModal({ type: "transfer", acc })}
  navigateToAccount={() => navigate(`${acc.id}`)}
/>
      )})
    )})
  )})

```

```

{!loading && (
  <div
    className={styles.addCard}
    onClick={handleCreateAccount}
  >
    <div className={styles.addInner}>
      <div className={styles.plus}>+</div>
      <div className={styles.addText}>Создать счёт</div>
    </div>
  </div>)}

  </div>
</SortableContext>

  <DragOverlay dropAnimation={{ duration: 250, easing:
"cubic-bezier(0.34, 1.56, 0.64, 1)" }}>
    {activeAccount && <DraggableOverlay
account={activeAccount} />}
  </DragOverlay>
</DndContext>

{/* Модалки */}
{modal?.type === "delete" && (
  <ConfirmModal
    title="Удалить счёт"
    text={`Удалить "${modal.acc.account_name}"?`}
    onCancel={() => setModal(null)}
    onConfirm={() => {
      deleteAccount(modal.acc.id).then(loadAccounts);
      setModal(null);
    }}
  />
)}
{modal?.type === "close" && (
  <ConfirmModal
    title="Закрыть счёт"
    text={`Закрыть "${modal.acc.account_name}"?`}
    onCancel={() => setModal(null)}
    onConfirm={() => {
      closeAccount(modal.acc.id).then(loadAccounts);
      setModal(null);
    }}
  />
)}
{modal?.type === "restore" && (
  <ConfirmModal
    title="Восстановить счёт"
    text={`Восстановить "${modal.acc.account_name}"?`}
    onCancel={() => setModal(null)}
    onConfirm={() => {
      restoreAccount(modal.acc.id).then(loadAccounts);
      setModal(null);
    }}
  />
)}

```

```

    }}
  />
)}
{modal?.type === "income" && (
  <TransactionModal
    title="Пополнение счёта"
    onCancel={() => setModal(null)}
    currency={modal.acc.currency}
    onSubmit={(data) => {
      income(modal.acc.id, data).then(loadAccounts);
      setModal(null);
    }}
  />
)}
{modal?.type === "expense" && (
  <TransactionModal
    title="Снятие со счёта"
    currency={modal.acc.currency}
    onCancel={() => setModal(null)}
    onSubmit={(data) => {
      expense(modal.acc.id, data).then(loadAccounts);
      setModal(null);
    }}
  />
)}
{modal?.type === "transfer" && (
  <TransferModal
    title="Перевод на счёт"
    fromAccount={modal.acc}
    currency={modal.acc.currency}
    accounts={accounts.filter(a => a.id !== modal.acc.id)}
    onCancel={() => setModal(null)}
    onSubmit={(data) => {
      transfer(
        modal.acc.id,          // from
        data.toAccountId,      // to
        { amount: data.amount }
      ).then(loadAccounts);

      setModal(null);
    }}
  />
)}
{modal?.type === "create" && (
  <CreateAccountModal
    onCancel={() => setModal(null)}
    onSubmit={(data) => {
      createAccount(data).then(loadAccounts);
      setModal(null);
    }}
  />
)}

```

```

    </div>
  );
}

```

Листинг Б.2 – Модуль управления кредитами

```

import { useEffect, useState } from "react";
import {
  getCredits,
  getAccounts,
  deleteCredit,
  getTotalCreditAmount,
  getTotalCreditOverpayment
} from "../../api/api";
import styles from "../Credits.module.css";
import DeleteCreditModal from "../DeleteCreditModal";
import CreditDetailsModal from "../CreditDetailsModal";
import CreateCreditModal from "../CreateCreditModal";
import PayCreditModal from "../PayCreditModal";
import PayoffCreditModal from "../PayOffCreditModal";

export default function Credits() {
  const [credits, setCredits] = useState([]);
  const [accountsMap, setAccountsMap] = useState({});

  const [loading, setLoading] = useState(true);

  const [payModal, setPayModal] = useState(null);
  const [payoffModal, setPayoffModal] = useState(null);
  const [totalDebt, setTotalDebt] = useState(0);
  const [detailsModal, setDetailsModal] = useState(null);
  const [modalOpen, setModalOpen] = useState(false);
  const [totalOverpayment, setTotalOverpayment] = useState(0);
  const [deleteModal, setDeleteModal] = useState(null);
  const handleDeleteClick = (credit) => {
    setDeleteModal(credit);
  };

  useEffect(() => {
    load();
  }, []);

  const load = async () => {
    setLoading(true);
    try {
      const [
        creditsData,
        accountsData,
        totalDebtData,
        totalOverpaymentData
      ] = await Promise.all([

```

```

        getCredits(),
        getAccounts(),
        getTotalCreditAmount(),
        getTotalCreditOverpayment()
    ]);

    setTotalDebt(totalDebtData || 0);
    setTotalOverpayment(totalOverpaymentData || 0);

    setCredits(Array.isArray(creditsData) ? creditsData : []);

    const map = {};
    accountsData.forEach((acc) => {
        map[acc.id] = acc;
    });
    setAccountsMap(map);

    } finally {
        setLoading(false);
    }
};

const getStatusClass = (status) => {
    switch (status?.toLowerCase()) {
        case "active":
            return styles.active;
        case "paid_off":
            return styles.paidOff;
        case "overdue":
            return styles.overdue;
        default:
            return styles.default;
    }
};

const handleDelete = async (creditId) => {
    const confirmed = window.confirm("Удалить кредит?");
    if (!confirmed) return;

    try {
        await deleteCredit(creditId);
        setCredits((prev) => prev.filter((c) => c.id !== creditId));
    } catch (e) {
        console.error(e);
        alert("Ошибка удаления");
    }
};

const calculateMonthlyPayment = (principal, rate, months) => {
    const monthlyRate = rate / 100 / 12;
    if (!monthlyRate) return principal / months;

```

```

    const annuity = (monthlyRate * Math.pow(1 + monthlyRate,
months)) / (Math.pow(1 + monthlyRate, months) - 1);
    return principal * annuity;
  };

  if (loading) {
    return <div className={styles.loading}>Загрузка...</div>;
  }

  return (
    <div className={styles.container}>
      <div className={styles.header}>
        <div>
          <h2 className={styles.title}>Кредиты</h2>
          <p className={styles.subtitle}>Управление кредитами и
платежами</p>
        </div>

        <button className={styles.addBtn} onClick={() =>
setModalOpen(true)}>
          + Новый кредит
        </button>
      </div>

      <div className={styles.analyticsCard}>
        <div className={styles.analyticsLabel}>Кредитная
аналитика</div>
        <div className={styles.currencyGrid}>
          <div className={styles.currencyCard}>
            <div className={styles.currencyCardHeader}>
              <div className={styles.currencyFlagWrapper}>
                
              </div>
              <span className={styles.currencyCode}>Общая
задолженность</span>
            </div>
            <div className={styles.currencyCardAmount}>
              {Number(totalDebt).toLocaleString("ru-RU", {
                minimumFractionDigits: 2,
                maximumFractionDigits: 2
              })}
              <span className={styles.currencySign}>₽</span>
            </div>
          </div>

          <div className={styles.currencyCard}>
            <div className={styles.currencyCardHeader}>
              <div className={styles.currencyFlagWrapper}>
                
              </div>

```

```

        <span className={styles.currencyCode}>Общая
переплата</span>
    </div>
    <div className={styles.currencyCardOverAmount}>
        {Number(totalOverpayment).toLocaleString("ru-RU", {
            minimumFractionDigits: 2,
            maximumFractionDigits: 2
        })}
        <span className={styles.currencySign}>₽</span>
    </div>
</div>
</div>
</div>

{credits.length === 0 ? (
    <div className={styles.empty}>Кредитов нет</div>
) : (
    <div className={styles.grid}>
        {credits.map((c) => {
            const progress = Number(c.progress || 0);
            const monthlyPayment = Number(c.monthly_payment || 0);
            const overpayment = Number(c.overpayment || 0);

            let progressColor = "#0077ff";
            if (progress >= 100) progressColor = "#22c55e";
            else if (progress >= 75) progressColor = "#d2ff6a";
            else if (progress < 25) progressColor = "#ef4444";
            else if (progress >= 25) progressColor = "#eeb830";

            return (
                <div key={c.id} className={styles.card}>

                    <div className={styles.top}>
                        <div className={styles.topLeft}>
                            <div className={styles.creditTitle}>
                                {c.purpose || "Кредит"}
                            </div>
                        <div className={styles.accountName}>
                            {accountsMap[c.account_id]?.account_name ||
"Счёт"}
                        </div>
                    </div>

                    <div className={` ${styles.status}
${getStatusClass(c.status)} `}>
                        {c.status === "ACTIVE" ? "В процессе" :
c.status === "PAID_OFF" ? "Уплачен" :
c.status}
                    </div>
                </div>
            )
        })}
    </div>

```

```

<div className={styles.mainSection}>
  <div className={styles.balanceSide}>
    <span className={styles.balanceLabel}>Остаток
долга</span>
    <div className={styles.balanceValue}>
      {Number(c.remaining_balance ||
0).toLocaleString("ru-RU")} ₸
    </div>
    <div className={styles.totalAmount}>
      из{" "}
      {Number(c.total_amount ||
0).toLocaleString("ru-RU")} ₸
    </div>
  </div>

  <div className={styles.circleWrap}>
    <div
      className={styles.progressCircle}
      style={{
        background: `conic-
gradient(${progressColor} ${progress}%, #edf2f7 ${progress}%)`
      }}
    >
      <div className={styles.circleInner}>
        <div
className={styles.circlePercent}>{progress}%</div>
        <div
className={styles.circleText}>погашено</div>
      </div>
    </div>
  </div>

  <div className={styles.stats}>
    <div className={styles.statCard}>
      <span>Платёж / мес</span>
      <b>{monthlyPayment.toLocaleString("ru-RU")}
₸</b>
    </div>

    <div className={styles.statCard}>
      <span>Ставка</span>
      <b>{c.interest_rate}%</b>
    </div>

    <div className={styles.statCard}>
      <span>Переплата</span>
      <b>
        {Number(c.overpayment ||
0).toLocaleString("ru-RU")} ₸
      </b>
    </div>
  </div>

```

```

        </div>

        <div className={styles.statCard}>
            <span>Срок</span>
            <b>{c.term_months} мес</b>
        </div>
    </div>

    <div className={styles.actions}>

        <button
            className={styles.deleteBtn}
            onClick={() => handleDeleteClick(c)}
        >
            Удалить
        </button>

        <button
            className={styles.darkBtn}
            onClick={() => setDetailsModal(c)}
        >
            Платежи
        </button>
        <button
            className={styles.primaryBtn}
            onClick={() => setPayoffModal(c)}
        >
            Погасить
        </button>
    </div>
</div>
    );
    })}
</div>
)}

{modalOpen && (
    <CreateCreditModal
        onClose={() => setModalOpen(false)}
        onSubmit={() => {
            setModalOpen(false);
            load();
        }}
    />
)}

    {deleteModal && (
    <DeleteCreditModal
        credit={deleteModal}
        onClose={() => setDeleteModal(null)}
    >

```

```

      onSuccess={() => {
        setDeleteModal(null);
        load();
      }}
    />
  )}

  {detailsModal && (
    <CreditDetailsModal
      credit={detailsModal}
      onClose={() => setDetailsModal(null)}
      onSuccess={() => {
        setDetailsModal(null);
        load();
      }}
    />
  )}

  {payModal && (
    <PayCreditModal
      credit={payModal}
      onClose={() => setPayModal(null)}
      onSuccess={load}
    />
  )}

  {payoffModal && (
    <PayoffCreditModal
      credit={payoffModal}
      onClose={() => setPayoffModal(null)}
      onSuccess={load}
    />
  )}
</div>
);
}

```

Листинг Б.3 – Модуль управления вкладами

```

import { useEffect, useState } from "react";
import { getDeposits, deleteDeposit, getTotalDepositAmount } from
"../../api/api";
import styles from "../Deposits.module.css";
import CreateDepositModal from "../CreateDepositModal";
import DepositDetailsModal from "../DepositDetailsModal";

export default function Deposits() {
  const [deposits, setDeposits] = useState([]);
  const [loading, setLoading] = useState(true);
  const [modalOpen, setModalOpen] = useState(false);
  const [detailsModal, setDetailsModal] = useState(null);
  const [totalAmount, setTotalAmount] = useState(0);

```

```

useEffect(() => {
  load();
}, []);

const load = async () => {
  setLoading(true);
  try {
    const [depositsData, totalAmountData] = await Promise.all([
      getDeposits(),
      getTotalDepositAmount(),
    ]);
    setDeposits(Array.isArray(depositsData) ? depositsData :
[]);
    setTotalAmount(totalAmountData || 0);
  } finally {
    setLoading(false);
  }
};

const handleDelete = async (id) => {
  if (!window.confirm("Удалить вклад?")) return;
  try {
    await deleteDeposit(id);
    setDeposits((prev) => prev.filter((d) => d.id !== id));
  } catch (e) {
    console.error(e);
    alert("Ошибка удаления");
  }
};

const getDaysLeft = (endDate) => {
  const diff = Math.ceil((new Date(endDate) - new Date()) /
(1000 * 60 * 60 * 24));
  return diff < 0 ? 0 : diff;
};

const calculateProfit = (principal, rate, days) =>
  principal * (rate / 100) * (days / 365);

const calculateTotal = (principal, rate, days) =>
  principal + calculateProfit(principal, rate, days);

if (loading) return <div
className={styles.loading}>Загрузка...</div>;

return (
  <div className={styles.container}>

    <div className={styles.header}>
      <div>
        <h2 className={styles.title}>Вклады</h2>

```

```

        <p className={styles.subtitle}>Доходность и
накопления</p>
    </div>
    <button className={styles.addBtn} onClick={() =>
setModalOpen(true)}>
        + НОВЫЙ ВКЛАД
    </button>
</div>

<div className={styles.analyticsCard}>
    <div className={styles.analyticsLeft}>
        <span className={styles.analyticsLabel}>Общая сумма всех
вкладов</span>
        <div className={styles.analyticsAmount}>
            {Number(totalAmount).toLocaleString("ru-RU")} ₽
        </div>
    </div>
</div>

{deposits.length === 0 ? (
    <div className={styles.empty}>Вкладов нет</div>
) : (
    <div className={styles.grid}>
        {deposits.map((d) => {
            const progress = (() => {
                const start = new Date(d.start_date);
                const end = new Date(d.end_date);
                const now = new Date();
                if (now <= start) return 0;
                if (now >= end) return 100;
                return Math.round(((now - start) / (end - start)) *
100);
            })();

            let progressColor = "#22c55e";
            if (progress >= 100) progressColor = "#ffd42a";

            return (
                <div key={d.id} className={styles.card}>

                    <div className={styles.top}>
                        <div>
                            <div
className={styles.depositTitle}>{d.deposit_name}</div>
                            <div className={styles.depositType}>
                                {{ FIXED: "Срочный", INDEFINITE:
"Бессрочный" }}[d.deposit_term_type] || d.deposit_term_type
                            </div>
                        </div>
                        <div className={styles.status}>
                            {d.status === "ACTIVE" ? "Активен" : "Закрыт"}

```

```

    </div>
  </div>

  <div className={styles.mainSection}>
    <div className={styles.balanceSide}>
      <span className={styles.balanceLabel}>Сумма
вклада</span>
      <div className={styles.balanceValue}>
{Number(d.principal_amount).toLocaleString("ru-RU")} ₺
      </div>
      <div className={styles.totalAmount}>
        Доход: +{calculateProfit(
          Number(d.principal_amount),
          Number(d.interest_rate),
          Number(d.term_days)
        ).toLocaleString("ru-RU",
{ maximumFractionDigits: 0 })} ₺
      </div>
    </div>

    <div className={styles.circleWrap}>
      <div
        className={styles.progressCircle}
        style={{
          background: `conic-
gradient(${progressColor} ${progress}%, #edf2f7 ${progress}%)`,
        }}
      >
        <div className={styles.circleInner}>
          <div
className={styles.circlePercent}>{progress}%</div>

          <div
className={styles.circleText}>с пока</div>
        </div>
      </div>
    </div>
  </div>

  <div className={styles.stats}>
    <div className={styles.statCard}>
      <span>Ставка</span>
      <b>{d.interest_rate}%</b>
    </div>

    <div className={styles.statCard}>
      <span>Срок</span>
      <b>{d.term_days} дн.</b>
    </div>
  </div>

```

```

</div>
<div className={styles.statCard}>
  <span>К получению</span>
  <b>
    {calculateTotal(
      Number(d.principal_amount),
      Number(d.interest_rate),
      Number(d.term_days)
    ).toLocaleString("ru-RU",
{ maximumFractionDigits: 0 })} ₺
  </b>
</div>

<div className={styles.statCard}>
  <span>Осталось</span>
  <b>{getDaysLeft(d.end_date)} дн.</b>
</div>
</div>

<div className={styles.actions}>
  <button
    className={styles.secondaryBtn}
    onClick={() => setDetailsModal(d)}
  >
    Детали
  </button>

  <button
    className={styles.deleteBtn}
    onClick={() => handleDelete(d.id)}
  >
    Удалить
  </button>
  <button className={styles.primaryBtn}>
    Пополнить
  </button>
</div>

</div>
);
}}
</div>
)}

{modalOpen && (
  <CreateDepositModal
    onClose={() => setModalOpen(false)}
    onSubmit={() => {
      setModalOpen(false);
      load();
    }}
  >

```

```
        />
    )}

    {detailsModal && (
        <DepositDetailsModal
            deposit={detailsModal}
            onClose={() => setDetailsModal(null)}
        />
    )}

</div>
);}
```