

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программное обеспечение мобильной
поддержки учета финансовых факторов при
формировании бюджета**

Студент _____ Лукошкин Павел Андреевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Князев В.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой _____ Макарычев П.П.
(подпись) (фамилия, инициалы)

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

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

Пенза, 2022

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

Факультет
Вычислительной техники

Кафедра
МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
П.П. Макарычев
«___» _____ 202 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Лукошкин Павел Андреевич гр. 18ВП1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Программное обеспечение мобильной поддержки учета финансовых факторов при формировании бюджета

Тема утверждена приказом ПГУ № 327/О от “ 18 ” апреля 2022

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н. _____

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

1) *Назначение: мобильное приложение предназначено для учета финансовых факторов при формировании бюджета.*

2) *Среда выполнения приложения: операционная система Android 6 и выше*

Среда выполнения сервера: операционная система Windows 10

Языковые средства разработки: C#, XAML, UML, Linq

Инструментальные средства разработки: Microsoft SQL Server 2019,

Microsoft Visual Studio 2022, Microsoft Project, Visual Paradigm

Технологии разработки: RUP, ООП, Entity Framework Core 6, ASP.NET Core, .NET MAUI

3) *Основные функции мобильного приложения:*

а) управление списком счетов;

б) внесение операций по счетам;

в) формирование бюджета по категориям операций;

г) формирование отчета по внесенным операциям;

д) управление списком категорий;

е) управление списком мест.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	<i>Анализ предметной области постановка задачи</i>	15%	13.04.2022– 20.04.2022	
2	<i>Проектирование</i>	25%	21.04.2022– 05.05.2022	
3	<i>Реализация</i>	40%	06.05.2022– 01.06.2022	
4	<i>Контроль качества программного обеспечения</i>	5%	02.06.2022– 04.06.2022	
5	<i>Оформление пояснительной записки</i>	15%	05.06.2022– 17.06.2022	

Дата выдачи задания " 22 " декабря 2021

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить

Декан факультета _____

(подпись, дата)

Реферат

Пояснительная записка содержит 140 листов, 82 рисунка, 23 таблицы, 25 использованных источников, 7 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, MS SQL SERVER, C#, XAML, RUP, UML, LINQ, ASP.NET CORE, .NET MAUI

Объектом разработки является мобильное приложение для поддержки учета финансовых факторов при формировании бюджета.

Цель работы – разработка мобильного приложения учета финансовых факторов при формировании бюджета, позволяющего вносить операции с указанием счета и категории, формировать бюджет и смотреть отчеты по внесенным операциям.

Разработка серверной части приложения производилась на языке программирования C#. В качестве среды разработки использовалась Microsoft Visual Studio 2022. В качестве СУБД была использована СУБД MS SQL Server 2019.

Разработка мобильного приложения производилась на языке программирования C#. В качестве среды разработки использовалась Microsoft Visual Studio 2022. В качестве платформы был использован .NET MAUI.

Разработка проведена с использованием операционной системы Windows 10. Приложение предназначено для использования под управлением ОС Android версии 6.0 и выше, а его серверная часть работает под управлением операционной системы Windows 10 и выше.

					ПГУ 09.03.04 – 08BP181.06 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Программное обеспечение мобильной поддержки учета финансовых факторов при формировании бюджета. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Лукошкин П.А.					4	140
Пров.		Князев В.Н.				Группа 18ВП1		
Реценз.								
Н. контр.		Такташкин Д.В.						
Утв.		Макарычев П.П.						

Содержание

Введение.....	7
1 Анализ предметной области и формулирование требований.....	9
1.1 Анализ предметной области и постановка задачи	9
1.2 Анализ программного обеспечения аналогичного назначения.....	10
1.2.1 Анализ мобильного приложения «Money Manager»	10
1.2.2 Анализ мобильного приложения «Дзен-мани».....	11
1.2.3 Анализ мобильного приложения «Monefy»	12
1.2.4 Сравнение аналогов.....	13
1.3 Анализ целевой аудитории.....	14
1.4 Анализ функциональных требований	17
1.5 Анализ нефункциональных требований	20
1.6 Концептуальное моделирование базы данных.....	21
1.7 Планирование разработки и оценка бюджета.....	23
2 Проектирование программного обеспечения.....	30
2.1 Выбор технологий и инструментальных средств разработки.....	30
2.2 Проектирование архитектуры программного обеспечения.....	31
2.3 Моделирование базы данных.....	31
2.3.1 Логическое моделирование базы данных	31
2.3.2 Физическое моделирование базы данных	32
2.4 Проектирование серверной части приложения.....	36
2.5 Проектирование мобильного приложения	38
2.6 Проектирование интерфейса пользователя	38
3 Реализация программного обеспечения	49
3.1 Реализация и развертывание базы данных	49
3.2 Реализация серверной части приложения	50

	6
3.3 Реализация мобильного приложения	52
4 Тестирование программного обеспечения	55
4.1 Тестирование серверной части приложения	55
4.1.1 Функциональное тестирование.....	55
4.1.2 Модульное тестирование.....	56
4.2 Тестирование мобильного приложения	57
4.3 Метрики кода.....	58
4.3.1 Метрика Мак-Кейба.....	59
4.3.2 Метрика Холстеда.....	60
Заключение	63
Список использованных источников	64
Приложение А. Глоссарий	67
Приложение Б. Исходный код миграции базы данных.....	69
Приложение В. Исходный код серверной части приложения.....	77
Приложение Г. Исходный код мобильного приложения.....	100
Приложение Д. Исходный код функций тестирования	120
Приложение Е. Результаты тестирования серверной части приложения	124
Приложение Ж. Результаты тестирования мобильного приложения.....	133

Введение

Современный мир вступил в новую фазу информационной эпохи, связанную с наступлением тотальной цифровизации всех сторон социальной жизни общества. Цифровая среда сегодня проникает во все сферы, как в область профессиональной деятельности, так и в, повседневные практики, меняя их формы, но и перестраивая сознание людей. Стремительное развитие цифровых технологий активизирует интерес ученых к проблемам трансформации социальной реальности, появления новой цифровой и финансовой культуры, изменения жизненного мира человека, преимуществ и рискам виртуализации социума.

Стоит отметить, что термин «цифровизация» стал впервые употребляться в 70-х гг. XX в., когда в ходе компьютерной революции произошел переход от аналоговых методов обработки информации к цифровому стандарту, использование которого привело к кардинальной трансформации всей технологической среды [1].

Данная тенденция обусловлена тем, что в современных условиях, внедрение новых информационных технологий позволяет повысить эффективность практически любого экономического, социального или бытового процесса. Осознавая данный факт, правительство Российской Федерации в 2017 году утвердило программу «Цифровая экономика Российской Федерации». Данный документ определяет стратегию развития «...цифровой экономики Российской Федерации, в которой данные в цифровой форме являются ключевым фактором производства во всех сферах социально-экономической деятельности, что повышает конкурентоспособность страны, качество жизни граждан, обеспечивает экономический рост» [2].

Еще одним документом, определяющим развитие информационных технологий в Российской Федерации, является указ Президента Российской Федерации «О стратегии развития информационного общества в Российской Федерации на 2017 – 2030 годы» от 9 мая 2017 года [3].

Одной из важнейших особенностей развития информационных технологий в нашей стране, да и во всем современном мире, является повсеместное распространение мобильных устройств с доступом к интернету. При этом мобильный интернет постепенно оттесняет на второй план традиционный доступ к сети с использованием стационарных компьютеров по причине своего существенно большего удобства. В частности, как отмечено в государственной программе «Цифровая экономика», в 2016 году в нашей стране «на 100 человек приходилось 159,95 мобильного телефона и из 100 человек 71,29 человека использовали мобильный доступ к сети Интернет» [3]. Со временем эта тенденция лишь будет усиливаться.

С развитием информационных технологий, цифровизации экономики и жизни общества, способствует изменению укладов общества и появления новой цифровой культуры. В этих условиях постепенно формируется финансовая культура, этому процессу активно способствует свободный доступ в интернет с мобильных устройств и компьютеров, неограниченный доступ к информации, а также просветительская деятельность Центрального Банка Российской Федерации работающего в этом направлении, формирующий культуру финансовой грамотности, среди студентов, молодежи и всех граждан, кто понял, что им это крайне необходимо в современном обществе.

В этих условиях считаю, что финансовая составляющая, является неотрывной чертой в жизни современного общества и умение грамотно распоряжаться своими финансами, умение их отслеживать и приумножать основная задача, каждого человека. Приложения данного направления пользуется большим спросом, для учета своего бюджета.

Исходя из всего вышеперечисленного следует, что проект разработки мобильного приложения поддержки учета финансовых факторов при формировании бюджета является актуальным и перспективным.

1 Анализ предметной области и формулирование требований

1.1 Анализ предметной области и постановка задачи

Разрабатываемое мобильное приложение должно обеспечить пользователям простой способ управления личным бюджетом, легко получать информацию о остатках по счетам, а также показывать выполняемость поставленного плана. Благодаря приложению пользователю не потребуются осуществлять расчеты вручную.

Приложение должно предоставлять всем зарегистрированным и авторизованным пользователям доступ к информации о внесенных операциях, информация о которых загружена в систему. У каждого пользователя должна быть возможность ввода информации о уже совершенных операциях. Должна быть возможность просмотра отчетов по операциям за указанных месяц в табличной или графической форме.

Таким образом, приложение должно предоставлять следующие функции:

- регистрация, авторизация и аутентификация пользователя;
- управление списком счетов (добавление, изменение, удаление);
- внесение операций с указанием категории и счета;
- просмотр списка внесенных операций;
- управление списком категорий;
- управление списком мест;
- формирование бюджета по категориям операций;
- формирование отчетов по операциям.

Операции разделяются по 3 типам: доход (поступление средств от плательщика на личный счет), расход (платеж с личного счета в стороннее место), перевод (перевод между своими счетами).

Операции дохода разделяются по категориям: заработная плата, дивиденды, проценты по вкладу, инвестиции. Операции расхода разделяются по категориям: продукты, бытовая химия, транспорт, оплата ЖКХ.

Пользователь имеет возможность добавления собственных категорий, плательщиков и мест платежа.

1.2 Анализ программного обеспечения аналогичного назначения

В настоящее время существует целый ряд программных продуктов, направленных на управление личными финансами. Рассмотрим наиболее известные приложения такого рода и произведем для них сравнительный анализ.

1.2.1 Анализ мобильного приложения «Money Manager»

При входе в приложение пользователя встречает страница операций. В нижней части экрана располагается меню с четырьмя вкладками.

Интерфейс приложения представлен на рисунке 1.

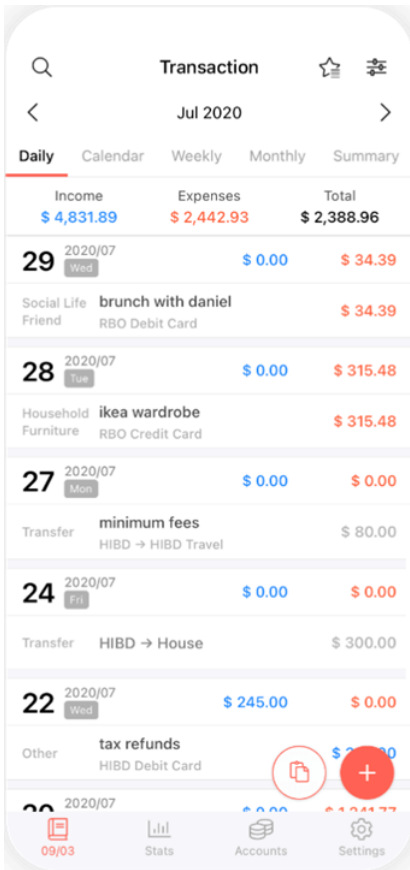


Рисунок 1 – Интерфейс «Money Manager»

На первой вкладке расположена страница операций, на ней можно добавить новые операции, посмотреть внесенные, а также посмотреть операции на календаре. Также есть страница итогов, на которой можно задать бюджет по категориям.

На второй вкладке выводится график доходов и расходов за выбранный период. На третьей вкладке отображаются текущие активы пользователя. Четвертая вкладка «Другое» содержит разделы настроек, помощи и предложение платного отключения рекламы [4].

Достоинствами данного приложения являются возможность быстрого внесения операций и просмотра информации по ним, возможность просмотра внесенных операций в календаре и возможность формирования бюджета по категориям.

Недостатком мобильного приложения является его условная бесплатность, отсутствие синхронизации между устройствами. У приложения перегруженный интерфейс, что приводит к усложнению его использования.

1.2.2 Анализ мобильного приложения «Дзен-мани»

Навигация в приложении устроена с помощью нижней панели, в которой размещены вкладки приложения. На первой вкладке располагается страница счетов с возможностью их редактирования и просмотра информации по счетам. На второй вкладке расположен список всех внесенных операций и кнопка добавления новой операции. Существует возможность объединения операций в одну.

На третьей вкладке расположена страница с различными вариантами представления информации по операциям: график по доходам, расходам за выбранный период, график изменения общего баланса счетов. На последней вкладке располагаются разделы настроек, связи с разработчиком и возможности входа в аккаунт [5].

Интерфейс программы представлена на рисунке 2.

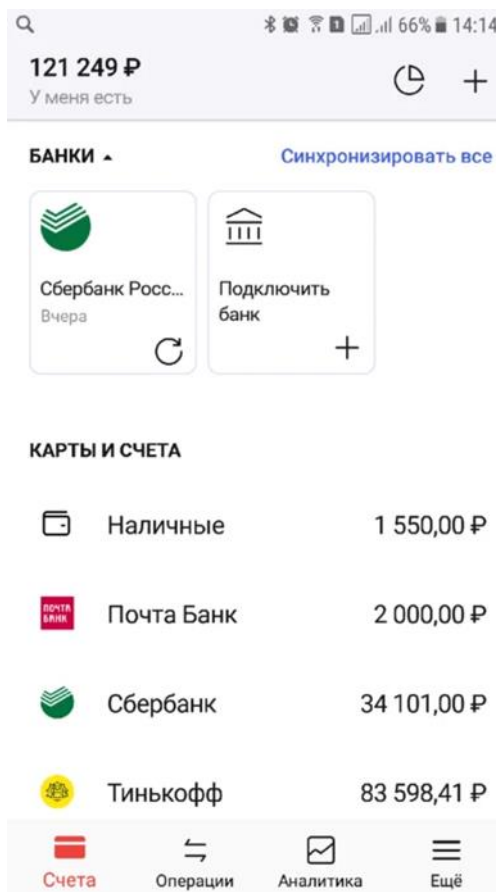


Рисунок 2 – Интерфейс «Дзен-мани»

Достоинствами данного приложения являются удобный интерфейс, возможность просмотра таблиц и графиков по внесенным операциям, а также синхронизация между устройствами.

Недостатком мобильного приложения является его условная бесплатность, отсутствие рекомендаций для помощи достижения поставленных целей.

1.2.3 Анализ мобильного приложения «Monefy»

Приложение имеет одностраничный вид. Все функции: добавления, изменения операций, а также добавления, изменения категорий выполняются на этой странице. Существует возможность просмотра списка операций на определенный период [6].

Интерфейс программы представлен на рисунке 3.



Рисунок 3 – Интерфейс «Monefy»

К достоинствам приложения относится возможность быстрого внесения операций, возможность синхронизации между устройствами.

К недостаткам мобильного приложения относится его условная бесплатность, отсутствие функции формирования отчета по операциям, отсутствие функции формирования бюджета. У приложения не интуитивно понятное расположение всех функций.

1.2.4 Сравнение аналогов

Был произведен сравнительный анализ рассмотренных выше мобильных приложений [7]. В результате анализа приложений сформирована таблица 1.

Таблица 1 – Сравнение аналогов

Мобильное приложение	Удобство интерфейса	Платный функционал	Функция формирования отчета по операциям	Формирование бюджета по категориям	Синхронизация между устройствами
Money Manager	Перегруженный	Есть, содержит рекламу в бесплатной версии	Есть	Есть	Нет
Дзен-мани	Удобный	Есть, одноразовая оплата	Есть	Нет	Есть
Monefy	Запутанный	Есть, подписка	Нет	Нет	Есть
Разрабатываемое приложение	Удобный	Нет, все функции бесплатны	Есть	Есть	Есть

Из проведенного сравнения можно сделать вывод что большинство приложений содержит платный функционал. Также в одном приложении нет функции формирования отчета по операциям. В двух нет функции формирования бюджета по категориям. И так же в одном нет синхронизации между устройствами.

Таким образом, в результате анализа программных продуктов аналогичного назначения было принято решение о разработке собственного приложения, которое будет лишено недостатков, выявленных у рассмотренных выше аналогов и предоставляющее весь необходимый функционал.

1.3 Анализ целевой аудитории

Целевой аудиторией проекта являются пользователи мобильных устройств, интересующиеся управлением личными финансами. В настоящее время спрос на мобильные приложения в категории финансовых технологий растет. На рисунке 4 приведен график, на котором показана динамика роста доли приложений этой категории [8].

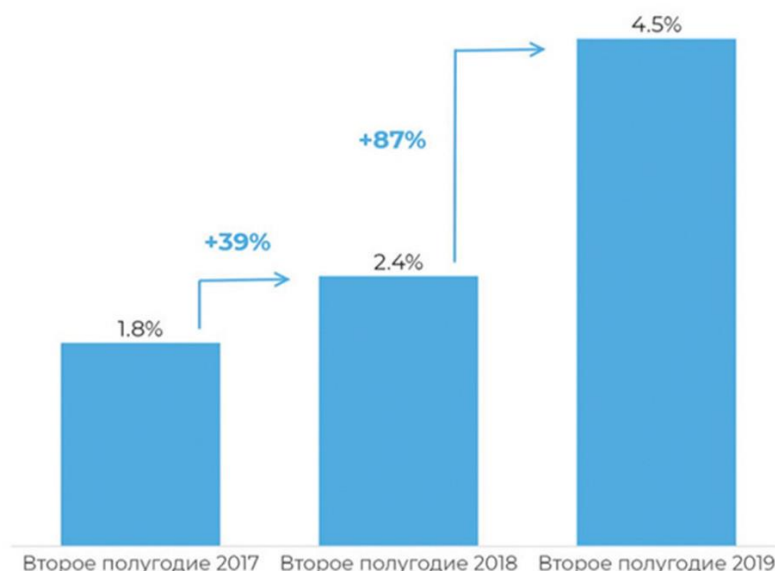


Рисунок 4 – Доля приложений финансовых технологий

Исходя из графика, наблюдается устойчивая тенденция роста доли приложений финансовых технологий с каждым годом. Среди всех подкатегорий по доле установок лидирует подкатегория финансовые услуги, это видно на графике, представленном на рисунке 5.

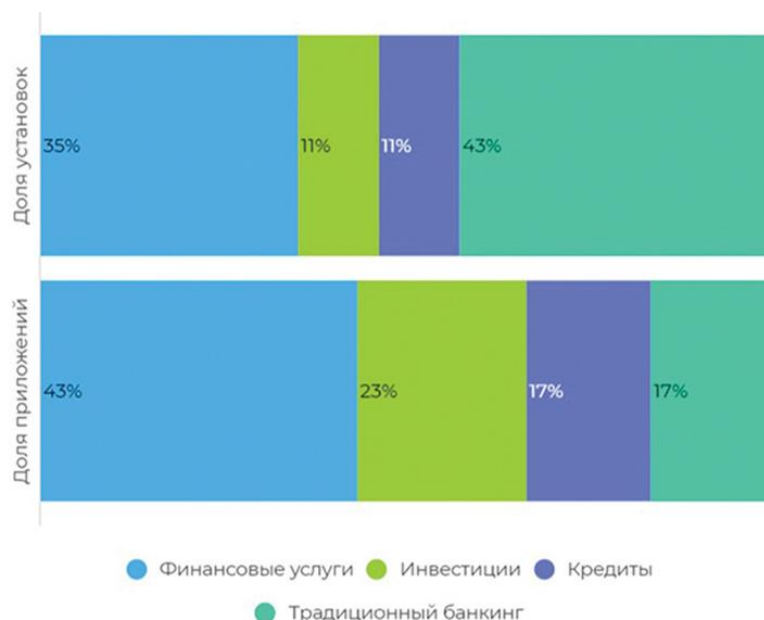


Рисунок 5 – Доля финансовых приложений по подкатегориям

В подкатеорию финансовых услуг входят приложения управления личными финансами [8]. На рисунке 6 представлены доли основных производителей смартфонов на мировом рынке [9].

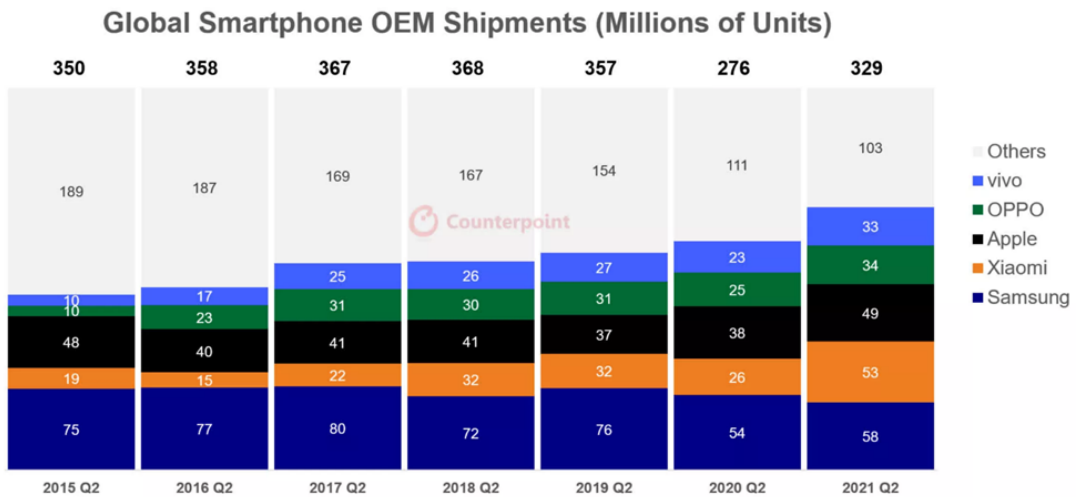


Рисунок 6 – Доли производителей смартфонов на мировом рынке

По графику видно, что доля компаний, производящих смартфоны на операционной системе Android более 85%, это значит, что Android является более приоритетной операционной системой для мобильных устройств, так как позволяет охватить большее число пользователей.

На рисунке 7 представлена статистика распространения разных версий системы Android [10].

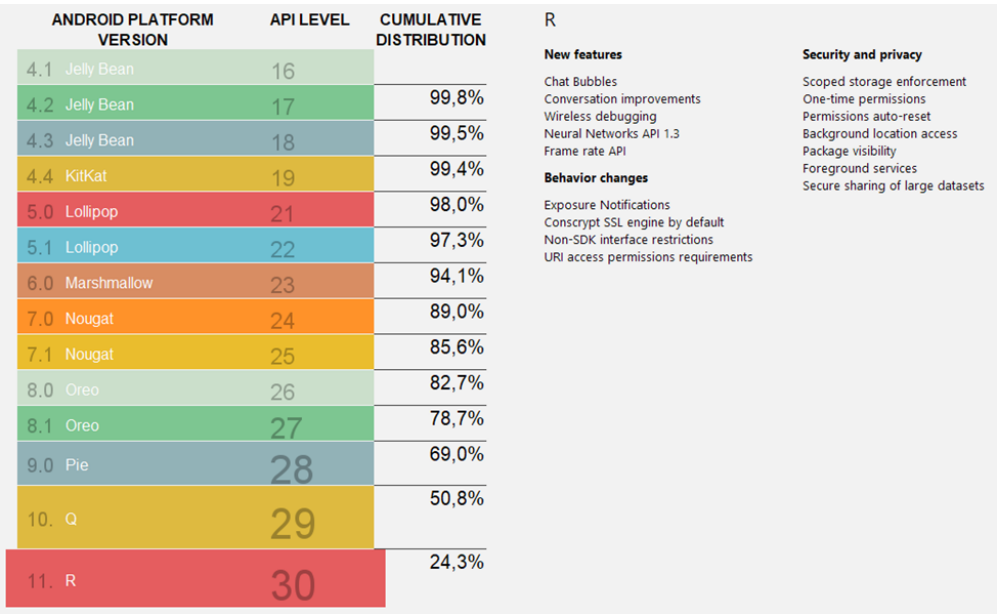


Рисунок 7 – Статистика распространения разных версий Android

По рисунку можно увидеть, что 94,1 % пользователей системы Android использует версии не младше 6.0. Поэтому приложение будет разрабатываться для Android 6.0 и более новых версий системы.

1.4 Анализ функциональных требований

На рисунке 8 отображена диаграмма вариантов использования, созданная в результате анализа требований задания [11, 12].

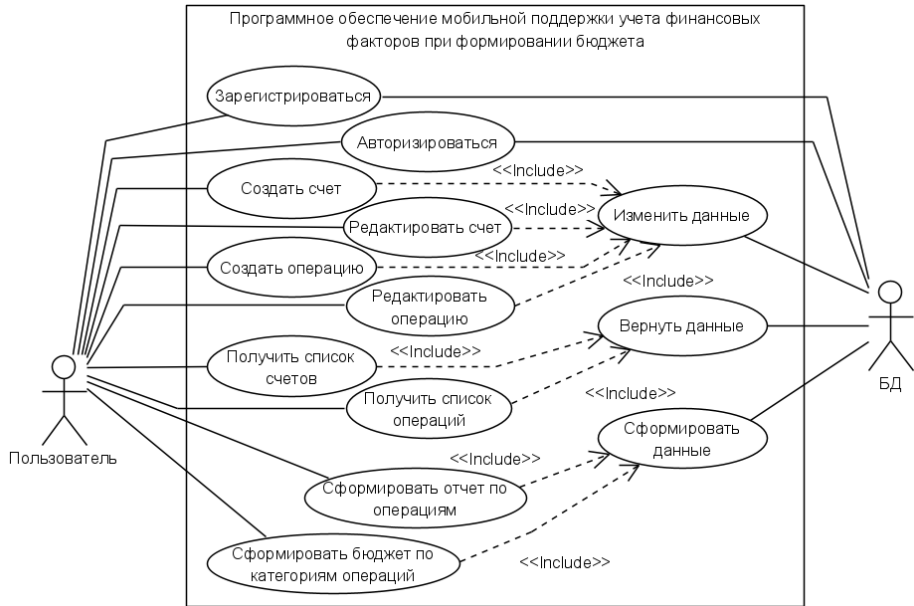


Рисунок 8 – Диаграмма вариантов использования мобильного приложения

Составим спецификацию прецедента «Создать счет» (рисунок 9).

Прецедент: Создать счет
ID: 1
Краткое описание: Создание счета
Главные актеры: Пользователь
Предусловие: Пользователь нажимает кнопку добавления операции
Основной поток: 1. Пользователь заполняет параметры счета 1.1. Пользователь указывает имя счета 1.2. Пользователь указывает тип счета 1.3. Пользователь указывает текущий баланс 1.4. Пользователь указывает описание счета 2. Пользователь посылает запрос системе на сохранение счета 3. Система сохраняет счет в базе данных

Рисунок 9 – Спецификация прецедента «Создать счет»

Составим спецификацию прецедента «Редактировать операцию» (рисунок 10).

Прецедент: Редактировать операцию
ID: 2
Краткое описание: Пользователь редактирует данные по операции
Главные актеры: Пользователь
Предусловие: Пользователь нажимает кнопку редактирования операции
Основной поток: <ol style="list-style-type: none"> 1. Пользователь редактирует информацию о операции <ol style="list-style-type: none"> 1.1. Пользователь редактирует название 1.2. Пользователь редактирует тип счета 1.3. Пользователь редактирует текущий баланс 1.4. Пользователь редактирует описание 2. Пользователь посылает запрос системе на сохранение информации о операциях 3. Система сохраняет операцию в базе данных

Рисунок 10 – Спецификация прецедента «Редактировать операцию»

Составим спецификацию прецедента «Посмотреть график по операциям» (рисунок 11).

Прецедент: Посмотреть график по операциям
ID: 3
Краткое описание: Пользователь просматривает график по операциям
Главные актеры: Пользователь.
Предусловие: Пользователь анализирует выполненные операции
Основной поток: <ol style="list-style-type: none"> 1. Пользователь задает параметры для построения графика <ol style="list-style-type: none"> 1.1. Пользователь выбирает тип операций 1.2. Пользователь выбирает временной промежуток 1.3. Пользователь выбирает учитываемые категории 2. Пользователь посылает запрос системе на построение графика по выбранным параметрам 3. Система строит график по заданным параметрам

Рисунок 11 – Спецификация прецедента «Посмотреть график по операциям»

Более подробное описание функционирования прецедентов можно представить в виде диаграммы деятельности. На рисунке 12 представлена диаграмма деятельности для прецедента «Создать операцию».

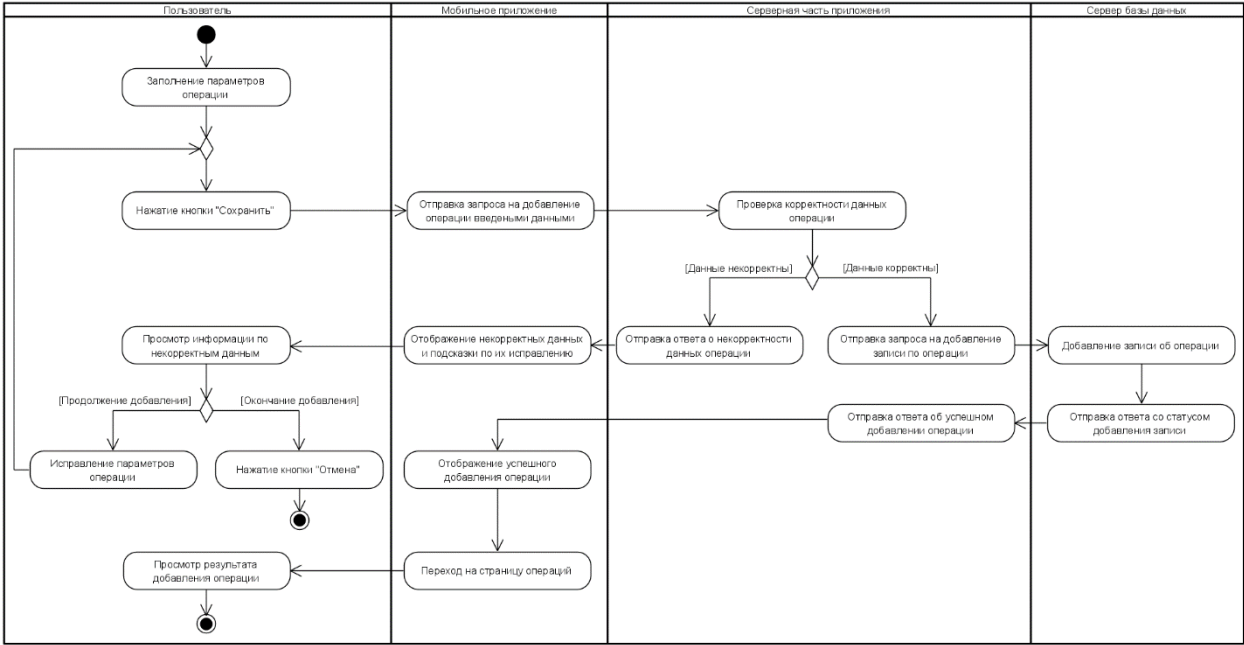


Рисунок 12 – Диаграмма деятельности прецедента «Создать операцию»

На рисунке 13 представлена диаграмма деятельности для прецедента «Зарегистрироваться».

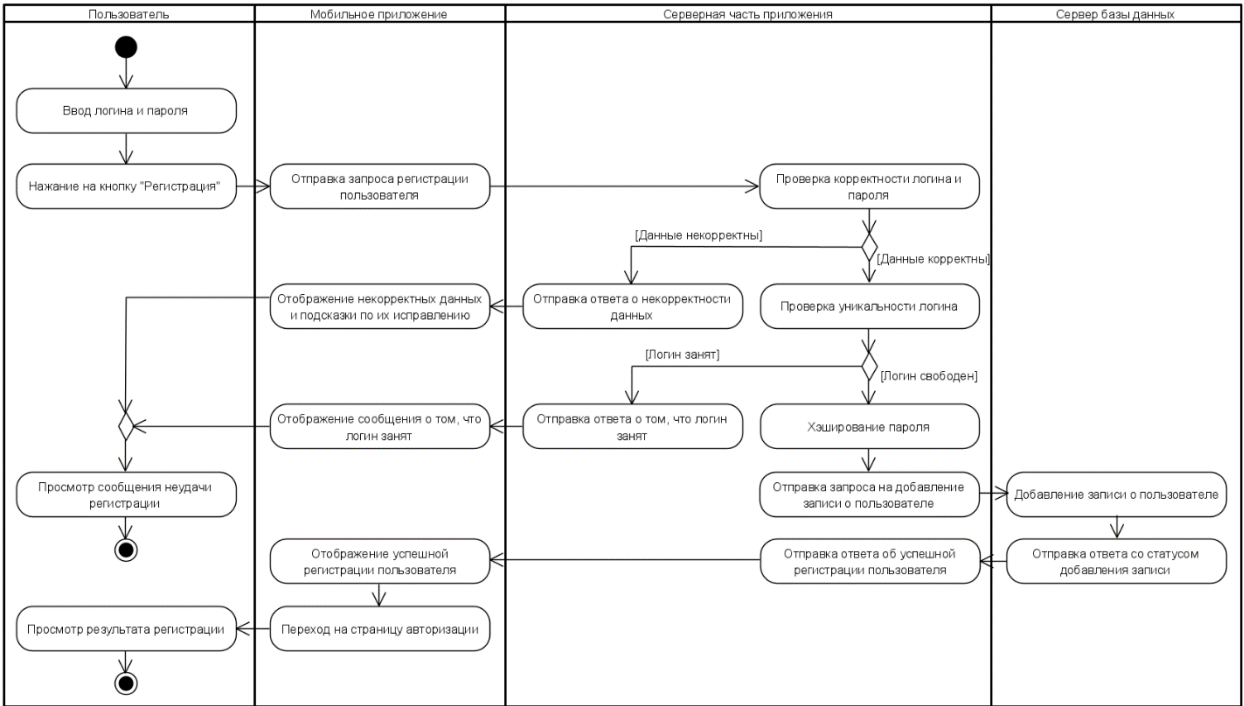


Рисунок 13 – Диаграмма деятельности прецедента «Зарегистрироваться»

Для прецедента «Получить список счетов» была составлена диаграмма последовательности, представленная на рисунке 14.

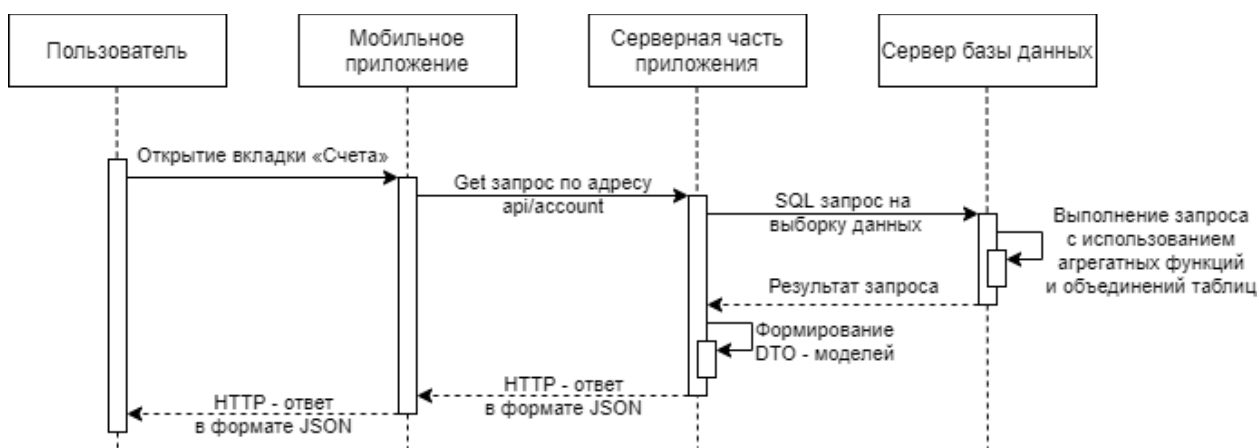


Рисунок 14 – Диаграмма последовательности прецедента «Получить список счетов»

В результате анализа функциональных требований были успешно выявлены основные функции разрабатываемого программного продукта. Для разрабатываемого программного продукта была построена диаграмма вариантов использования, а также для некоторых прецедентов были составлены спецификации и построена диаграмма последовательности.

1.5 Анализ нефункциональных требований

Исходя из поставленной задачи, был сформулирован ряд нефункциональных требований к разрабатываемому программному продукту. Будут использованы следующие технологии разработки: RUP, ООП, MVC, MVVM [13 – 15].

Серверная часть приложения должна быть выполнена на языке программирования C# с использованием платформы ASP.NET, в среде программирования Microsoft Visual Studio 2022. Данные на сервере хранятся в реляционной базе данных, реализованной с использованием СУБД MS SQL Server, запросы к базе данных осуществляются на языке SQL при помощи фреймворка Entity Framework Core. Серверная часть должна работать под управлением операционной системы Windows 10 и выше.

Клиентская часть приложения должна быть реализована на языке программирования C# с использованием платформы .NET MAUI, в среде

программирования Microsoft Visual Studio 2022. Она должна представлять собой мобильное приложение с удобным и интуитивно понятным пользовательским интерфейсом. Клиентское приложение должно быть отлажено и должно успешно работать под управлением операционной системы (ОС) Android 6.0 и выше [14 – 19].

Для планирования разработки и оценки бюджета будет использован Microsoft Project [20], для разработки диаграмм UML будет использован Visual Paradigm.

1.6 Концептуальное моделирование базы данных

На основе анализа предметной области была построена концептуальная модель базы данных (БД). Графическое представление концептуальной модели БД в виде диаграммы «сущность-связь» представлено на рисунке 15.

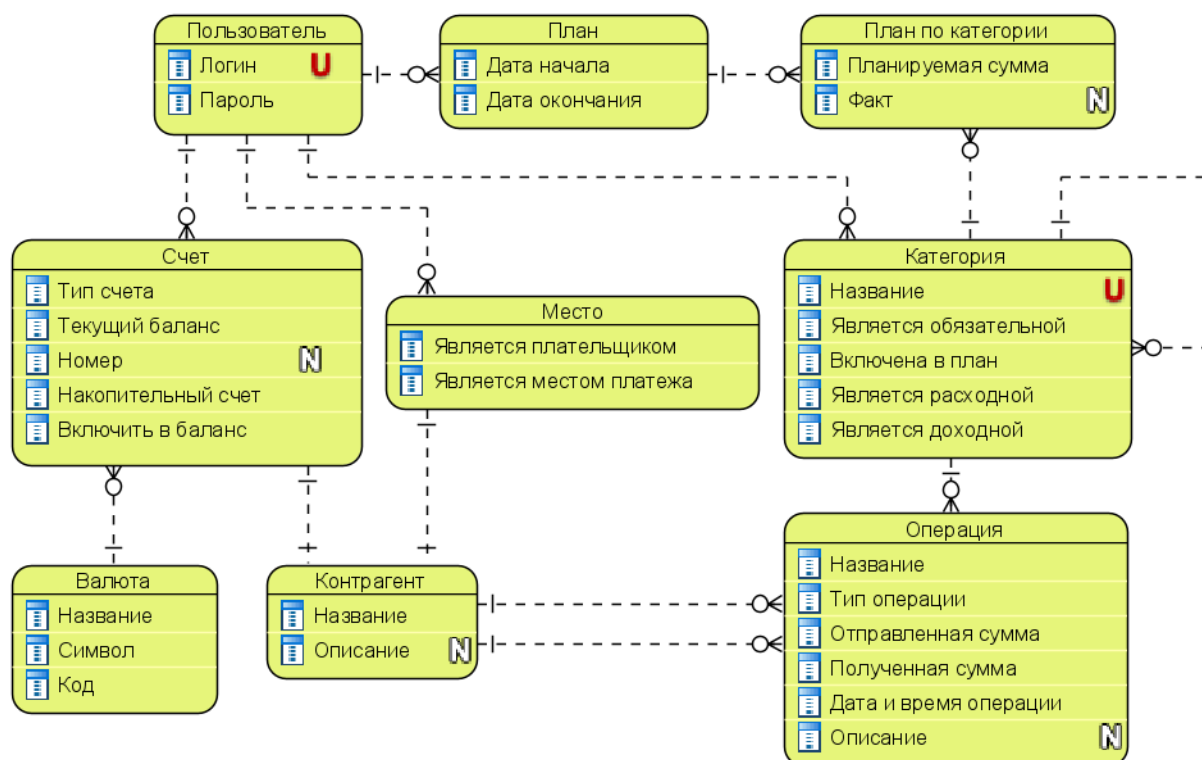


Рисунок 15 – Графическое представление концептуальной модели

В таблице 2 представлены сущности, сформированные на основании входных и выходных данных бизнес-процессов.

Таблица 2 – Описание сущностей

№	Название	Описание	Количество экземпляров	Режим обновления
1	Пользователь	Перечень пользователей	100	1-2 раза в год
2	Счет	Перечень счетов для конкретного пользователя	1-10 экземпляров на пользователя	1-2 раза в месяц
3	Операция	Перечень операций по конкретному счету	1-1000 экземпляров на счет	1-10 раз в день
4	Контрагент	Перечень контрагентов, с которыми осуществляются операции	В среднем 50 контрагентов на пользователя	В среднем 1 раз в месяц
5	Место	Перечень мест, с которыми осуществляются операции	В среднем 50 мест на пользователя	В среднем 1 раз в месяц
6	Категория	Иерархия категорий, созданных пользователем	В среднем 20 категорий на пользователя	1-10 раз в месяц
7	План	План по операциям, который выбрал пользователь	В среднем 10 планов на пользователя	Не чаще одного раза в месяц
8	План по категории	Перечень планов для каждой категории, сформированных на основе выбранного плана.	1 на пару план-категория	Не чаще нескольких раз в месяц
9	Валюта	Список валют	5	Пользователь не может изменять данные

В таблице 3 представлено описание связей между сущностями.

Таблица 3 – Описание связей

№	Связь	Описание	Мощность	Обязательность
1	Пользователь – Счет	Определяет список счетов пользователя	1:M	Да
2	Пользователь – Место	Определяет список мест, с которыми осуществляются операции	1:M	Да
3	Пользователь – Категория	Определяет список категорий операций, совершаемых пользователем	1:M	Да
4	Пользователь – План	Определяет список планов на операции	1:M	Да
5	Контрагент – Место	Дополняет данные контрагента информацией о месте	1:1	Да
6	Контрагент – Счет	Дополняет данные контрагента информацией о счете	1:1	Да
7	Операция – Контрагент	Определяет список операций перевода из контрагента	M:1	Да
8	Операция – Контрагент	Определяет список операций перевода в контрагент	M:1	Да
10	Категория – Операция	Определяет к какой категории относится операция	1:M	Нет
11	Категория – Категория	Определяет операции, являющиеся родителями другим	1:M	Нет
12	Категория – ПланДляКатегории	Определяет планируемую сумму по операциям данной категории	1:M	Да
13	План – ПланДляКатегории	Определяет список планируемых сумм по операциям	1:M	Да

На данном уровне моделирования были спроектированы только сущности с атрибутами и связями между ними, хранящими основные данные, без включения атрибутов первичных и внешних ключей.

1.7 Планирование разработки и оценка бюджета

Качественное и своевременное выполнение работ по любому проекту, в том числе и в сфере информационных технологий, невозможно при отсутствии грамотного планирования. Оно позволяет построить график выполнения работ над проектом таким образом, чтобы добиться минимизации времени и затрат на выполнение всех поставленных задач и целей с приемлемым качеством [21 – 24].

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project. Проект был начат 28 марта 2022 года. При планировании разработки было выделено 24 задачи. Перечень задач и их описание представлено на таблице 4.

Таблица 4 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ предметной области	1 день	1
4	Анализ программных продуктов-аналогов	2 дня	1
5	Анализ требований к функционалу	1 день	3;4
6	Анализ требований к интерфейсу	1 день	5
7	Постановка задачи	2 дня	6
8	Анализ требований завершен	-	7
9	Проектирование программных средств	-	-
10	Проектирование архитектуры приложения	4 дня	8
11	Проектирование базы данных	3 дня	8
12	Проектирование функционала	4 дня	10;11
13	Проектирование интерфейса пользователя	3 дня	10;11
14	Проектирование завершено	-	12;13
15	Реализация программных средств	-	-
16	Реализация базы данных	6 дней	10;11
17	Реализация серверной части приложения	14 дней	14;16
18	Реализация мобильного приложения	14 дней	14;16
19	Реализация завершена	-	17;18
20	Тестирование	-	-
21	Функциональное тестирование	2 дня	19
22	Модульное тестирование	5 дней	19
23	Тестирование завершено	-	21;22
24	Составление и оформление документации	7 дней	23
25	Конец проекта	-	24

В таблице 5 представлено описание затрачиваемых ресурсов.

Таблица 5 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист 1	Трудовой	250 руб/ч	500 руб/ч
Программист 2	Трудовой	250 руб/ч	500 руб/ч
Дизайнер	Трудовой	200 руб/ч	400 руб/ч
Аналитик	Трудовой	200 руб/ч	400 руб/ч
Тестировщик	Трудовой	150 руб/ч	300 руб/ч
Компьютер	Материальный	40000 руб	-
Интернет	Затраты	-	-

Распределение ресурсов по задачам представлено в таблице 6.

Таблица 6 – Назначение ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ предметной области	Аналитик Компьютер Интернет	30% 2 1000 руб
4	Анализ программных продуктов-аналогов	Аналитик	70%
5	Анализ требований к функционалу	Аналитик Программист 1	100% 100%
6	Анализ требований к интерфейсу	Аналитик Дизайнер	100% 100%
7	Постановка задачи	Аналитик Дизайнер Программист 1 Программист 2	100% 25% 25% 25%
8	Анализ требований завершен	-	-
9	Проектирование программных средств	-	-
10	Проектирование архитектуры приложения	Аналитик Программист 1	50% 50%
11	Проектирование базы данных	Аналитик Программист 1	50% 50%
12	Проектирование функционала	Аналитик Программист 2	50% 50%
13	Проектирование интерфейса пользователя	Аналитик Дизайнер Программист 2	50% 100% 50%
14	Проектирование завершено	-	-
15	Реализация программных средств	-	-
16	Реализация базы данных	Программист 1	100%
17	Реализация серверной части приложения	Программист 1	100%
18	Реализация мобильного приложения	Программист 1 Дизайнер	100% 100%
19	Реализация завершена	-	-
20	Тестирование	-	-
21	Функциональное тестирование	Тестировщик	20%
22	Модульное тестирование	Тестировщик Программист 2	80% 50%

Продолжение таблицы 6

№	Название задачи	Ресурс	Единицы
23	Тестирование завершено	-	-
24	Составление и оформление документации	Аналитик Программист 1 Программист 2	50% 50% 50%
25	Конец проекта	-	-

Таким образом, реализацией проекта будут заниматься программист 1, программист 2, аналитик и дизайнер, которым для работы нужно два компьютера и доступ в интернет. Для вычисления длительности реализации проекта и его критического пути построен сетевой график (рисунок 16).

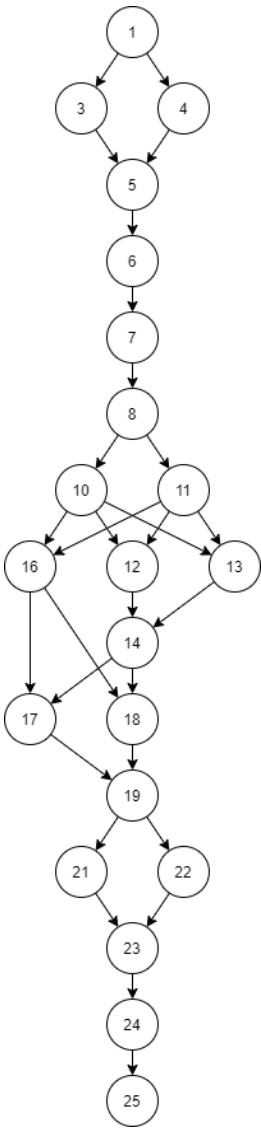


Рисунок 16 – Сетевой график проекта

Перечень задач в Microsoft Project представлен на рисунке 17.

зад	Название задачи	Длительнс	Начало	Окончани	Предшественн	Названия ресурсов
1	Начало проекта	0 дней	Пн 28.03.22	Пн 28.03.22		
2	Анализ требований	6 дней	Пн 28.03.22	Пн 04.04.22		
3	Анализ предметной области	1 день	Пн 28.03.22	Пн 28.03.22	1	Аналитик[30%];Комьютер[2];Интернет[1 000,00 Р]
4	Анализ программных продуктов-аналогов	2 дней	Пн 28.03.22	Вт 29.03.22	1	Аналитик[70%]
5	Анализ требований к функционалу	1 день	Ср 30.03.22	Ср 30.03.22	3;4	Аналитик;Программист 1
6	Анализ требований к интерфейсу	1 день	Чт 31.03.22	Чт 31.03.22	5	Аналитик;Дизайнер
7	Постановка задачи	2 дней	Пт 01.04.22	Пн 04.04.22	6	Аналитик;Дизайнер[25%];Программист 1[25%];Программист 2[25%]
8	Анализ требований завершен	0 дней	Пн 04.04.22	Пн 04.04.22	7	
9	Проектирование программных средств	8 дней	Вт 05.04.22	Чт 14.04.22		
10	Проектирование архитектуры приложения	4 дней	Вт 05.04.22	Пт 08.04.22	8	Аналитик[50%];Программист 1[50%]
11	Проектирование базы данных	3 дней	Вт 05.04.22	Чт 07.04.22	8	Аналитик[50%];Программист 1[50%]
12	Проектирование функционала	4 дней	Пн 11.04.22	Чт 14.04.22	10;11	Аналитик[50%];Программист 2[50%]
13	Проектирование интерфейса пользователя	3 дней	Пн 11.04.22	Ср 13.04.22	10;11	Аналитик[50%];Дизайнер;Программист 2[50%]
14	Проектирование завершенно	0 дней	Чт 14.04.22	Чт 14.04.22	12;13	
15	Реализация программных средств	20 дней	Пн 11.04.22	Пт 06.05.22		
16	Реализация базы данных	6 дней	Пн 11.04.22	Пн 18.04.22	10;11	Программист 1
17	Реализация серверной части приложения	14 дней	Вт 19.04.22	Пт 06.05.22	14;16	Программист 1
18	Реализация мобильного приложения	14 дней	Вт 19.04.22	Пт 06.05.22	14;16	Программист 2;Дизайнер
19	Реализация завершенна	0 дней	Пт 06.05.22	Пт 06.05.22	17;18	
20	Тестирование	5 дней	Пн 09.05.22	Пт 13.05.22		
21	Функциональное тестирование	2 дня	Пн 09.05.22	Вт 10.05.22	19	Тестировщик[20%]
22	Модульное тестирование	5 дней	Пн 09.05.22	Пт 13.05.22	19	Тестировщик[80%];Программист 2[50%]
23	Тестирование завершенно	0 дней	Пт 13.05.22	Пт 13.05.22	21;22	
24	Составление и оформление документации	7 дней	Пн 16.05.22	Вт 24.05.22	23	Аналитик[50%];Программист 1[50%];Программист 2[50%]
25	Конец проекта	0 дней	Вт 24.05.22	Вт 24.05.22	24	

Рисунок 17 – Перечень задач проекта

Диаграмма Ганта, спроектированная в Microsoft Project представлена на рисунке 18.

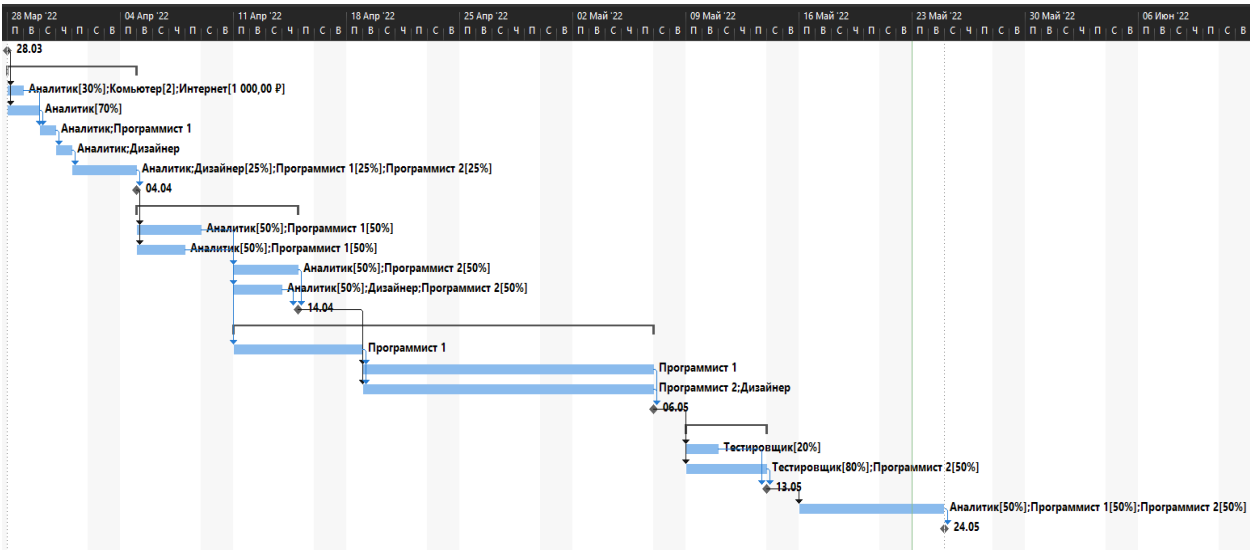


Рисунок 18 – Диаграмма Ганта

График загрузки ресурса «Программист 1» представлен на рисунке 19.

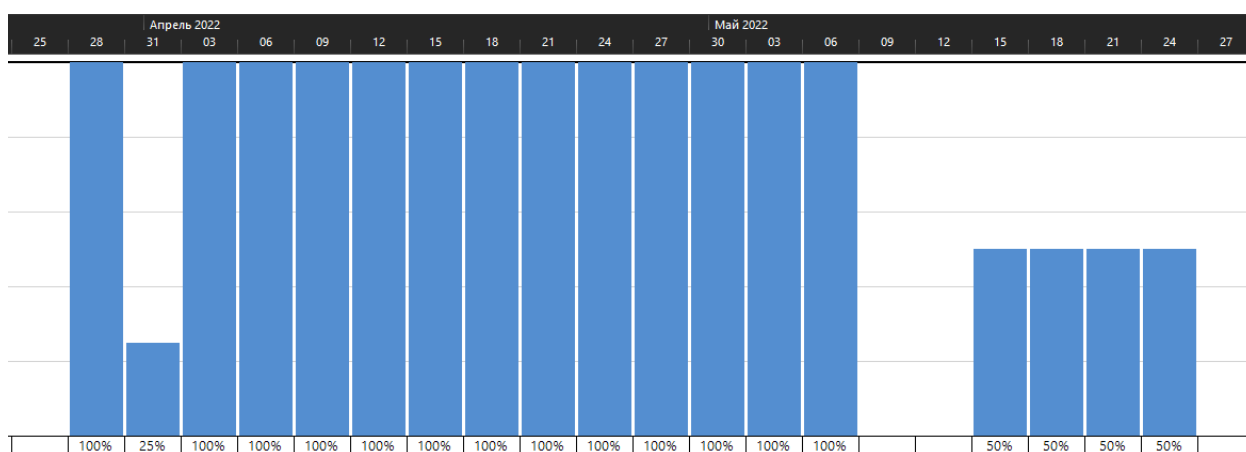


Рисунок 19 – График загрузки ресурса «Программист 1»

График загрузки ресурса «Программист 2» представлен на рисунке 20.

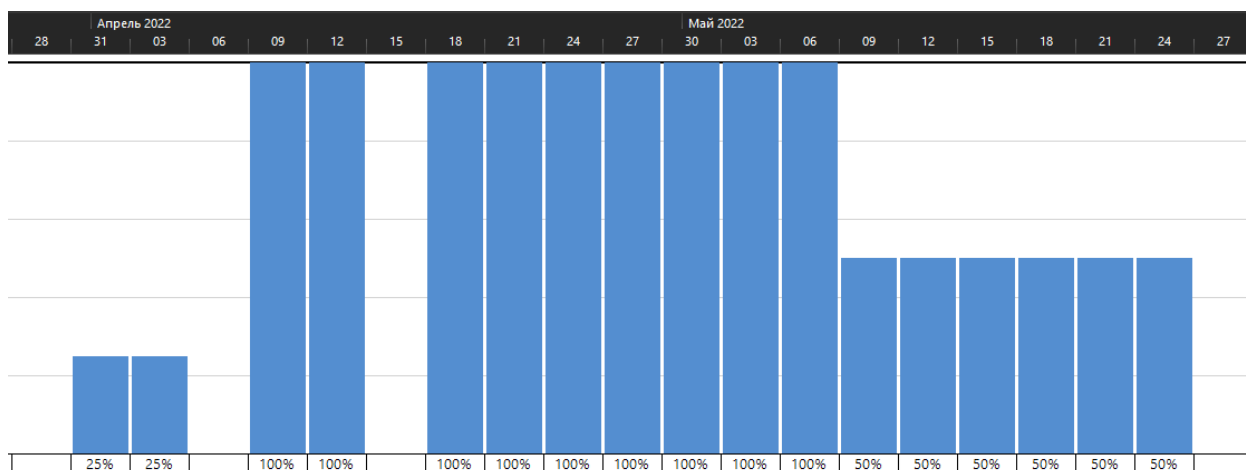


Рисунок 20 – График загрузки ресурса «Программист 2»

График загрузки ресурса «Аналитик» представлен на рисунке 21.

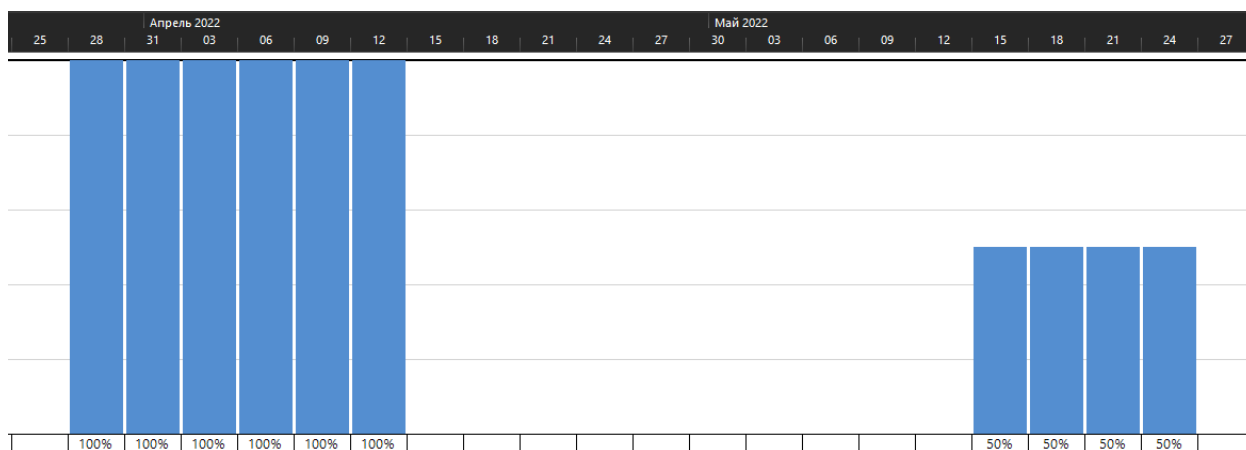


Рисунок 21 – График загрузки ресурса «Аналитик»

График загрузки ресурса «Дизайнер» представлен на рисунке 22.

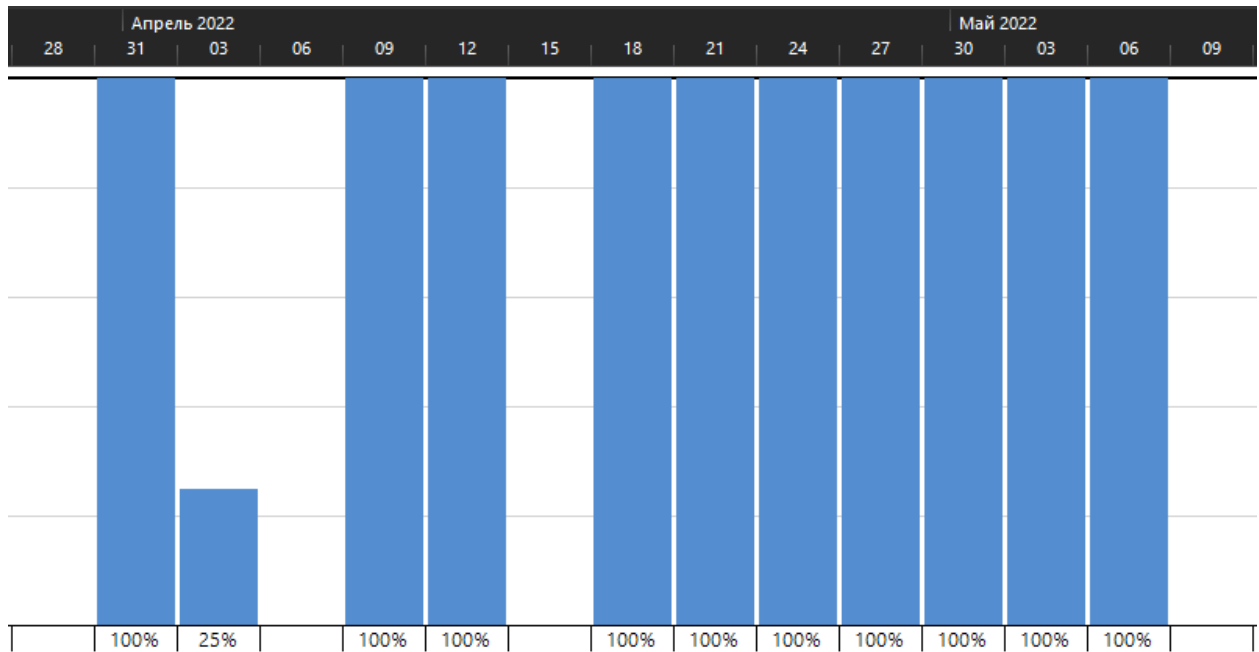


Рисунок 22 – График загрузки ресурса «Дизайнер»

График загрузки ресурса «Тестировщик» представлен на рисунке 23.

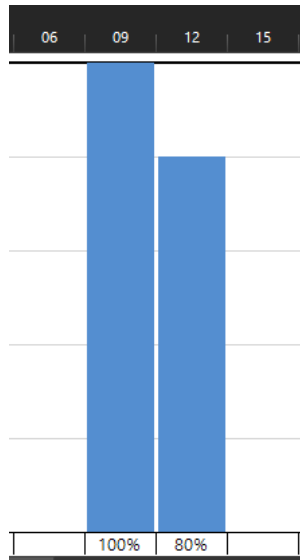


Рисунок 23 – График загрузки ресурса «Тестировщик»

На рисунке 24 представлен обзорный отчет проекта. Реализация проекта займет 42 дня, а предполагаемые затраты 246800 рублей.

<u>Даты</u>			
Начало:	Пн 28.03.22	Окончание:	Вт 24.05.22
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала	0 дней	Отклон. окончания	0 дней
<u>Длительность</u>			
Текущий план:	42 дней	Оставшаяся:	42 дней
Базовый план:	0 дней	Фактическая:	0 дней
Отклонение:	42 дней	Процент завершен	0%
<u>Трудозатраты</u>			
Текущий план:	732,8 ч	Оставшиеся:	732,8 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	732,8 ч	Процент завершен	0%
<u>Затраты</u>			
Текущий план:	246 800,00 Р	Оставшиеся:	246 800,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	246 800,00 Р		
<u>Состояние задачи</u>		<u>Состояние ресурса</u>	
Еще не начатые задачи	25	Трудовые ресурсы:	6
Выполняющиеся задачи	0	Трудовые ресурсы с прев. д	0
Завершенные задачи	0	Материальные ресурсы:	1
Итого задач:	25	Итого ресурсов:	7

Рисунок 24 – Обзорный отчет проекта

На основе произведенных оценок стоимости и сроков проекта можно сделать вывод о его экономической целесообразности.

2 Проектирование программного обеспечения

2.1 Выбор технологий и инструментальных средств разработки

В качестве методологии разработки выбрана модель Rational Unified Process (RUP). Разработка приложения в соответствии с методологией RUP носит итеративный характер. Вся разработка разделяется на четыре основных этапа, каждый из которых включает в себя одну или несколько итераций [25]:

- начальная стадия – производится сбор требований, оцениваются риски;
- стадия уточнения – производится уточнение предметной области и проектирование архитектуры продукта;
- стадия построения – реализуется спроектированный продукт;
- стадия внедрения – производится доработка функций, устранение выявленных ошибок.

В качестве базы данных выбрана MS SQL Server 2019, является одной из самых востребованных СУБД на рынке. Серверная часть взаимодействует с БД с помощью ORM-технологии Entity Framework Core 6. Данная технология позволяет управлять сущностями БД через их проекцию – классы C# описанные с использованием специальных атрибутов, а также предоставляет возможность выполнения SQL запросов через язык запросов Linq.

Серверная часть приложения будет реализована при помощи платформы ASP.NET, данная платформа предоставляет обширный базовый функционал для обработки HTTP запросов и методы управления поведением сервера. Сервер будет работать под управлением Windows 10.

Клиентская часть приложения будет реализована при помощи NET MAUI – мультиплатформенный фреймворк позволяющий разрабатывать приложения для Windows 10, Android, iOS, macOS при помощи единого описания интерфейса, ресурсов и функционала приложения. Целевой платформой в данной разработке является Android.

2.2 Проектирование архитектуры программного обеспечения

На основе поставленных требований было принято решение использовать в приложении трехуровневую архитектуру «клиент-сервер». Схема архитектуры представлена на рисунке 25.

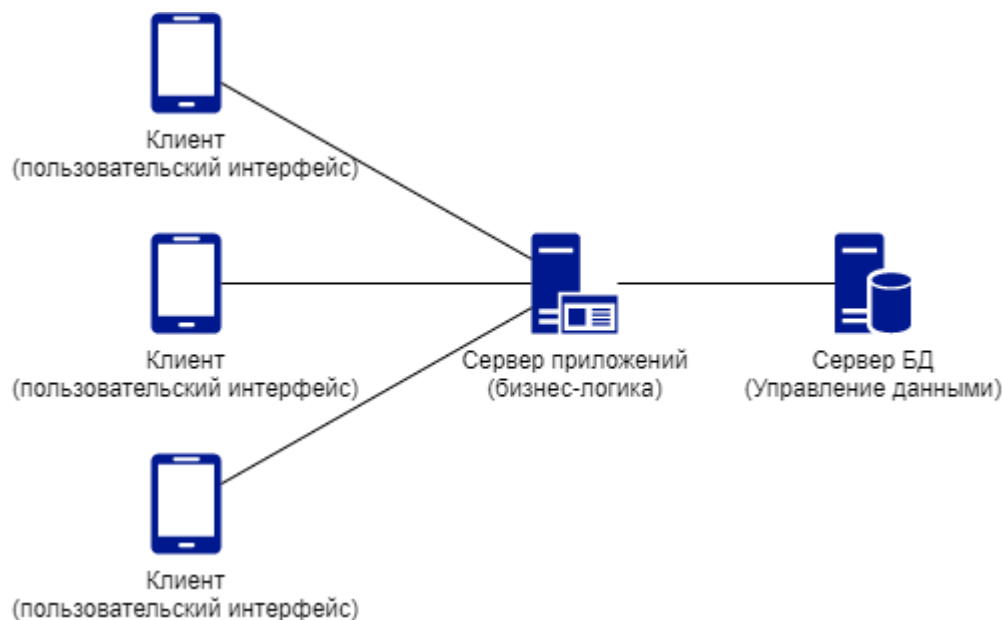


Рисунок 25 – Трехуровневая архитектура «клиент-сервер»

В данной архитектуре сервер базы данных занимается управлением данными. Сервер приложений реализует бизнес-логику (функционал) всего программного средства. Клиентская часть занимается отображением полученной информации с сервера приложений, а также отправкой запросов на получение и изменение информации [25].

Данная архитектура позволяет на каждом уровне абстрагироваться от поведения на других, что позволяет вести параллельную разработку.

2.3 Моделирование базы данных

2.3.1 Логическое моделирование базы данных

На основании концептуальной модели была спроектирована логическая модель базы данных в среде Visual Paradigm (рисунок 26).

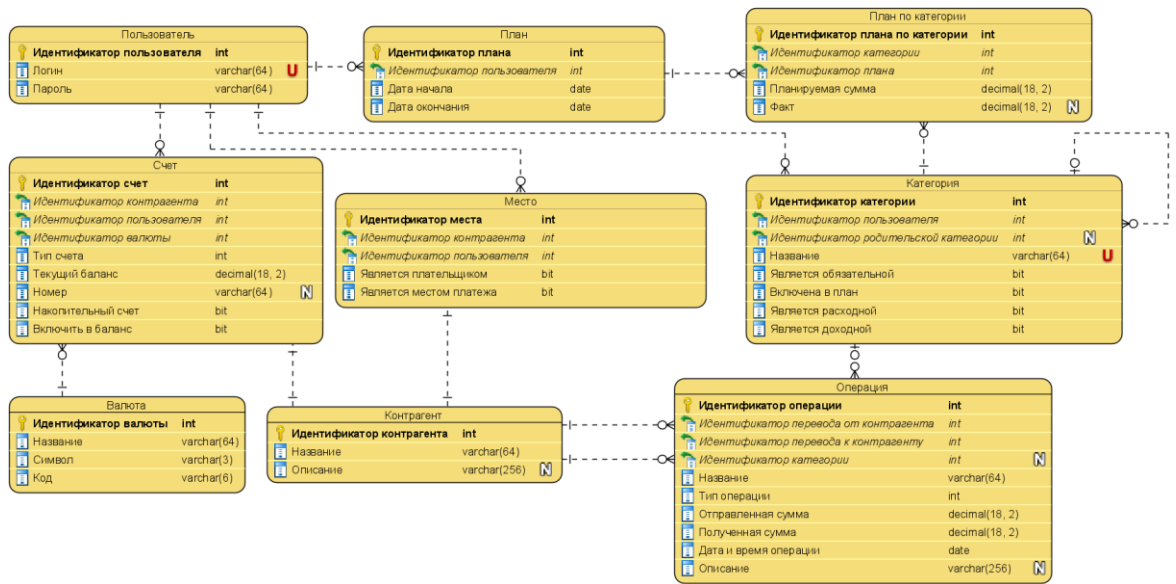


Рисунок 26 – Графическое представление логической модели базы данных

На представленной логической модели спроектированная структура БД отображена уже более подробно. Здесь определены описания и ограничения атрибутов сущностей, а также уточнен их состав и связи между ними.

2.3.2 Физическое моделирование базы данных

На основе концептуальной и логической модели в терминах СУБД MS SQL Server 2019 была разработана физическая модель базы данных (рисунок 27).

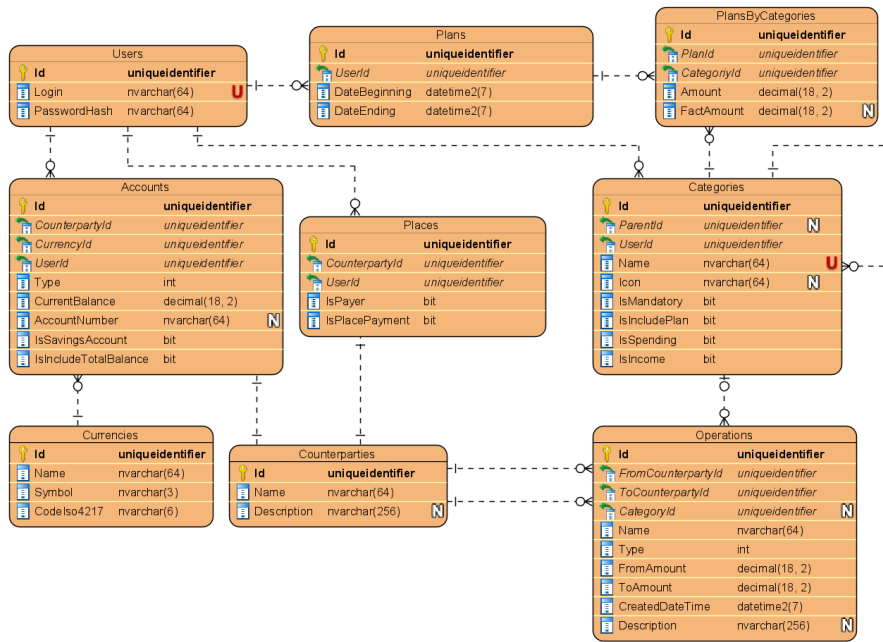


Рисунок 27 – Физическая модель базы данных

Описание атрибутов сущности «Пользователь» представлено в таблице 7.

Таблица 7 – Описание атрибутов сущности «Пользователь»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
Login	Логин	nvarchar(64)	Нет	Любое	-	-
PasswordHash	Пароль	nvarchar(64)	Нет	Любое	-	-

Описание атрибутов сущности «Счет» представлено в таблице 8.

Таблица 8 – Описание атрибутов сущности «Счет»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
UserId	Идентификатор пользователя	uniqueidentifier	Нет	Любое	-	FK
CurrencyId	Идентификатор валюты	uniqueidentifier	Нет	Любое	-	FK
CounterpartyId	Идентификатор контрагента	uniqueidentifier	Нет	Любое	-	FK
Type	Тип счета	int	Нет	0-5	-	-
CurrentBalance	Текущий баланс	decimal(18,2)	Нет	Любое	-	-
AccountNumber	Номер счета	nvarchar(64)	Да	Любое	-	-
IsSavingsAccount	Накопительный счет	bit	Нет	True, False	false	-
IsIncludeTotalBalance	Включить в баланс	bit	Нет	True, False	true	-

Описание атрибутов сущности «Операция» представлено в таблице 9.

Таблица 9 – Описание атрибутов сущности «Операция»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
FromCounterpartyId	Идентификатор перевода от контрагента	uniqueidentifier	Нет	Любое	-	FK
ToCounterpartyId	Идентификатор перевода к контрагенту	uniqueidentifier	Нет	Любое	-	FK

Продолжение таблицы 9

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
CategoryId	Идентификатор категории	uniqueidentifier	Да	Любое	-	-
Name	Название	nvarchar(64)	Нет	Любое	-	-
Type	Тип операции	int	Нет	0-2	-	-
FromAmount	Отправленная сумма	decimal(18,2)	Нет	> 0	-	-
ToAmount	Полученная сумма	decimal(18,2)	Нет	> 0	-	-
CreatedDateTime	Дата и время операции	datetime2	Нет	Любое	-	-
Description	Описание	nvarchar(256)	Да	Любое	-	-

Описание атрибутов сущности «Контрагент» представлено в таблице 10.

Таблица 10 – Описание атрибутов сущности «Контрагент»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
Name	Название	nvarchar(64)	Нет	Любое	-	-
Description	Описание	nvarchar(64)	Да	Любое	-	-

Описание атрибутов сущности «Место» представлено в таблице 11.

Таблица 11 – Описание атрибутов сущности «Место»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
UserId	Идентификатор пользователя	uniqueidentifier	Нет	Любое	-	FK
CounterpartyId	Идентификатор контрагента	uniqueidentifier	Нет	Любое	-	FK
IsPayer	Является плательщиком	bit	Нет	True, False	true	-
IsPlacePayment	Является местом платежа	bit	Нет	True, False	true	-

Описание атрибутов сущности «Категория» представлено в таблице 12.

Таблица 12 – Описание атрибутов сущности «Категория»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
UserId	Идентификатор пользователя	uniqueidentifier	Нет	Любое	-	FK
ParentId	Идентификатор родительской категории	uniqueidentifier	Нет	Любое	-	FK
Name	Название	nvarchar(64)	Нет	Любое	-	-
IsMandatory	Является обязательной	bit	Нет	True, False	true	-
IsIncludePlan	Включена в план	bit	Нет	True, False	true	-
IsSpending	Является расходной	bit	Нет	True, False	true	-
IsIncome	Является доходной	bit	Нет	True, False	true	-

Описание атрибутов сущности «План» представлено в таблице 13.

Таблица 13 – Описание атрибутов сущности «План»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
UserId	Идентификатор пользователя	uniqueidentifier	Нет	Любое	-	FK
DateBeginning	Дата начала	datetime2	Нет	Любое	-	-
DateEnding	Дата окончания	datetime2	Нет	Любое	-	-

Описание атрибутов сущности «План по категории» представлено в таблице 14.

Таблица 14 – Описание атрибутов сущности «План по категории»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
PlanId	Идентификатор плана	uniqueidentifier	Нет	Любое	-	FK
CategoryId	Идентификатор плана	uniqueidentifier	Нет	Любое	-	FK
Amount	Планируемая сумма	decimal(18,2)	Нет	> 0	-	-
FactAmount	Факт	decimal(18,2)	Да	Любое	-	-

Описание атрибутов сущности «Валюта» представлено в таблице 15.

Таблица 15 – Описание атрибутов сущности «Валюта»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор	uniqueidentifier	Нет	Любое	-	PK
Name	Название	nvarchar(64)	Нет	Любое	-	-
Symbol	Символ	nvarchar(3)	Нет	Любое	-	-
CodeIso4217	Код	nvarchar(6)	Нет	Любое	-	-

Описание атрибутов проводилось с учетом специфики типов MS SQL Server.

2.4 Проектирование серверной части приложения

Серверная часть выполнена на основе REST API. Взаимодействие с сервером осуществляется по протоколу HTTP при помощи стандартных методов (POST – создание нового ресурса, PUT – обновление существующего ресурса, GET – получение ресурса, DELETE – удаление ресурса).

Исходя из выбранных технологий реализации серверной части приложения, было решено использовать архитектурный шаблон MVC (Model-View-Controller). Данный шаблон разделяет приложение на три компонента представление (View), контроллер (Controller) и модель (Model).

Слой представления отвечает за взаимодействие с клиентом в данном случае с мобильным приложением. Слой контроллера отвечает за логику обработки запросов и доступ к модели. Слой модели отвечает за способ организацию доступа к данным.

Схема взаимодействия компонентов шаблона MVC представлена на рисунке 28.

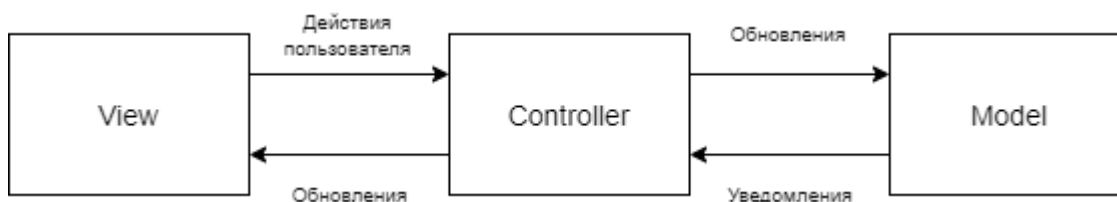


Рисунок 28 – Схема шаблона MVC

Выбор данного шаблона обусловлен технологией ASP.NET, которая нацелена на использование данного шаблона.

В таблице 16 приведена спецификация всех HTTP запросов.

Таблица 16 – Описание HTTP запросов

Метод и URL	Параметры	Возвращаемые данные	Краткое описание
POST api/user/register	Данные пользователя	Статус регистрации	Регистрация пользователя
POST api/user/authorize	Данные пользователя	Jwt-токен	Авторизация пользователя
GET api/user/name	-	Логин пользователя	Проверка авторизации пользователя
POST api/account	Данные счета	Статус добавления	Добавление счета
PUT api/account	Данные счета	Статус изменения	Изменение счета
GET api/account	-	Список счетов	Получение списка счетов
GET api/account/details/{id}	Идентификатор счета	Подробная информация о счете	Получение подробной информации о счете
GET api/account/info	-	Общая информация о счетах	Получение общей информации о счетах
DELETE api/account/{id}	Идентификатор счета	Статус удаления	Удаление счета
POST api/operation	Данные операции	Статус добавления	Добавление операции
PUT api/operation	Данные операции	Статус изменения	Изменение операции
POST api/operation/filter	Фильтр параметров операции	Список операций	Получение списка операций по фильтру
GET api/operation/details/{id}	Идентификатор операции	Подробная информация об операции	Получение подробной информации об операции
DELETE api/operation/{id}	Идентификатор операции	Статус удаления	Удаление операции
POST api/category	Данные категории	Статус добавления	Добавление категории
PUT api/category	Данные категории	Статус изменения	Изменение категории
GET api/category	-	Список категорий	Получение списка категорий
GET api/category/details/{id}	Идентификатор категории	Подробная информация о категории	Получение подробной информации о категории
DELETE api/category/{id}	Идентификатор категории	Статус удаления	Удаление категории
POST api/place	Данные места	Статус добавления	Добавление места
PUT api/place	Данные места	Статус изменения	Изменение места
GET api/place	-	Список мест	Получение списка мест
DELETE api/place/{id}	Идентификатор места	Статус удаления	Удаление места
GET api/plan/{dateMonth}	Год и месяц	Данные плана	Получение плана на месяц
PUT api/plan	Данные плана	Статус изменения	Изменение плана
GET api/currency	-	Список валют	Получение списка валют
POST api/analytic	Параметры выборки по аналитике	Данные по операциям	Получение данных по операциям

Все запросы спроектированы на основе функциональных требований и с учетом специфики платформы ASP.NET.

2.5 Проектирование мобильного приложения

Исходя из выбранных технологий реализации мобильного приложения, было решено использовать архитектурный шаблон MVVM (Model-View-ViewModel). Данный шаблон разделяет приложение на три компонента: представление (View), модель представления (ViewModel) и модель (Model).

Слой представления отвечает за отображение данных и отслеживание действий пользователя. Слой модели представления отвечает за бизнес-логику приложения и данные, отображаемые в представлении. Слой модели отвечает за организацию доступа к данным.

Схема взаимодействия компонентов шаблона MVVM представлена на рисунке 29.

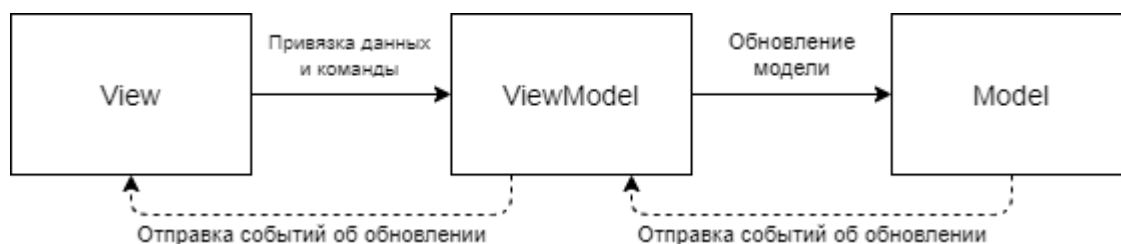


Рисунок 29 – Схема шаблона MVVM

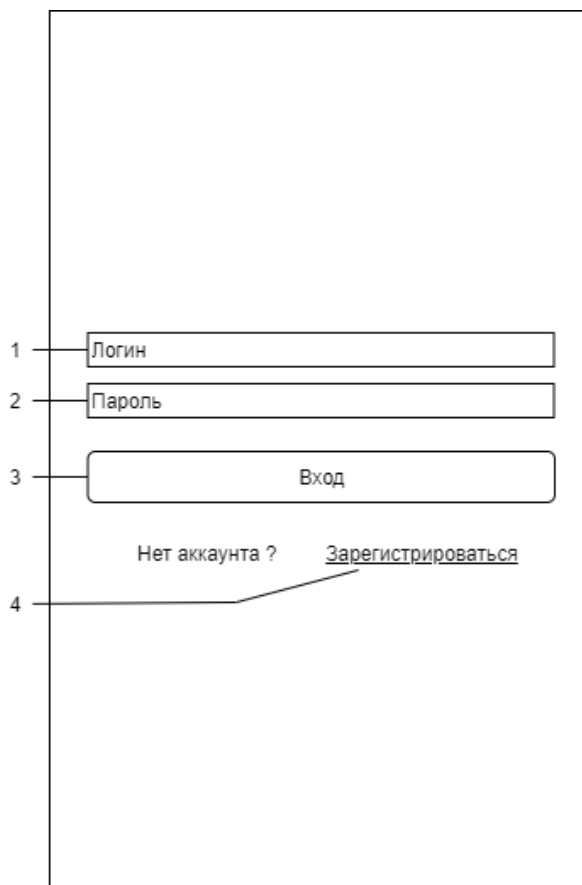
Выбор данного шаблона обусловлен технологией .NET MAUI, которая нацелена на использование данного шаблона.

2.6 Проектирование интерфейса пользователя

На основе выявленных функциональных требований был разработан интерфейс мобильного приложения. Он состоит из 18 страниц, 4 из которых являются вкладками. На вкладке «Счета» пользователь может управлять счетами (добавлять, редактировать, удалять) и просматривать информацию по ним. На вкладке «Операции» пользователь может управлять операциями и просматривать полную историю операций. На вкладке «Аналитика» пользователь может сформировать бюджет по категориям операций,

просматривать графики и таблицы, сформированные по внесенным операциям, для дальнейшего анализа и повышения эффективности управления средствами. На вкладке «Другое» имеется возможность редактирования списка категорий, мест.

Макет интерфейса страницы авторизации представлен на рисунке 30.



Макет интерфейса страницы авторизации. В центре расположены четыре элемента, пронумерованные слева:

- 1 – поле ввода «Логин»
- 2 – поле ввода «Пароль»
- 3 – кнопка «Вход»
- 4 – ссылка «Зарегистрироваться» (подчеркнутая), расположенная рядом с текстом «Нет аккаунта ?»

Рисунок 30 – Страница авторизации

На макете страницы авторизации имеются следующие элементы:

- 1 – поле ввода логина;
- 2 – поле ввода пароля;
- 3 – кнопка «Вход», при нажатии осуществляет попытку авторизации;
- 4 – кнопка «Зарегистрироваться», при нажатии открывает страницу регистрации пользователя.

Макет интерфейса страницы регистрации представлен на рисунке 31.

1 — поле ввода логина;

2 — поле ввода пароля;

3 — кнопка «Регистрация»;

Рисунок 31 – Страница регистрации

На макете страницы регистрации имеются следующие элементы:

1 – поле ввода логина;

2 – поле ввода пароля;

3 – кнопка «Регистрация», при нажатии осуществляет попытку регистрации пользователя.

Макет интерфейса страницы счетов представлен на рисунке 32.

1 — поле ввода баланса;

2 — кнопка «+»;

3 — заголовок списка операций;

4 — список операций;

5 — подвал списка операций;

Счета Операции Аналитика Другое

Рисунок 32 – Страница счетов

На макете страницы счетов имеются следующие элементы:

- 1 – поле отображения общего баланса с указанием валюты;
- 2 – кнопка «Добавить», при нажатии открывает страницу добавления нового счета;
- 3 – название счета;
- 4 – список счетов;
- 5 – остаток на счете с указанием валюты.

Макет интерфейса страницы просмотра информации о счете представлен на рисунке 33.

	1	2	3
←	Информация о счете		
Название			
Тип счета			
Текущий баланс			
Номер			
Описание			

Рисунок 33 – Страница просмотра информации о счете

На макете страницы просмотра информации о счете имеются следующие элементы:

- 1 – кнопка «Назад», при нажатии возвращает на предыдущую страницу;
- 2 – кнопка «Изменить», при нажатии открывает страницу изменения счета;

3 – кнопка «Удалить», при нажатии задает вопрос об удалении счета и удаляет при подтверждении;

4 – название счета;

5 – тип счета;

6 – текущий баланс с указанием валюты;

7 – номер счета;

8 – описание счета.

Макет интерфейса страницы создания/изменения счета представлен на рисунке 34.

Рисунок 34 – Страница создания/изменения счета

На макете страницы создания/изменения счета имеются следующие элементы:

1 – кнопка «Назад», при нажатии возвращает на предыдущую страницу;

2 – кнопка «Сохранить», при нажатии осуществляет попытку создания/изменения счета;

3 – поле ввода названия счета;

4 – поле выбора типа счета;

- 5 – поле выбора валюты;
- 6 – поле ввода текущего баланса на счету;
- 7 – поле ввода номера счета;
- 8 – поле выбора является ли счет накопительным;
- 9 – поле выбора включать ли счет в общий баланс;
- 10 – поле ввода описания счета.

Макет интерфейса страницы операций представлен на рисунке 35.

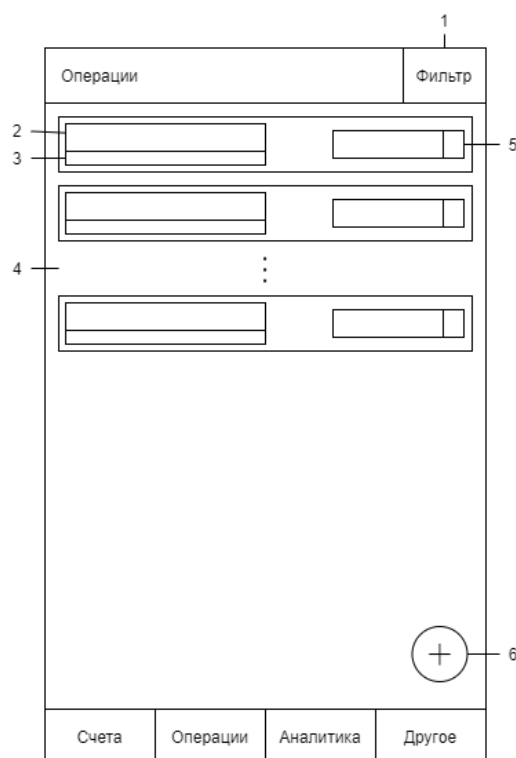


Рисунок 35 – Страница операций

На макете страницы операций имеются следующие элементы:

- 1 – кнопка «Фильтр», при нажатии открывает страницу настройки фильтра отображения операций;
- 2 – название операции;
- 3 – счет, по которому совершена операция;
- 4 – список операций;
- 5 – сумма по операции с указанием валюты;
- 6 – кнопка «Добавить», при нажатии открывает меню выбора типа добавляемой операции (расход, доход, перевод).

Макет интерфейса страницы просмотра информации об операции представлен на рисунке 36.

The wireframe shows a mobile application interface for viewing operation details. At the top is a header bar with three elements: a back arrow labeled '1', the title 'Информация об операции', and two buttons labeled '2' and '3'. Below the header, there are seven input fields, each with a label on the left and a corresponding number on the right: 'Тип операции' (4), 'Сумма' (5), 'Категория' (6), 'Дата' (7), 'Отправитель' (8), 'Получатель' (9), and 'Описание' (10). The 'Описание' field is a larger text area compared to the others.

Рисунок 36 – Страница просмотра информации об операции

На макете страницы просмотра информации об операции имеются следующие элементы:

- 1 – кнопка «Назад», при нажатии возвращает на предыдущую страницу;
- 2 – кнопка «Изменить», при нажатии открывает страницу изменения операции;
- 3 – кнопка «Удалить», при нажатии задает вопрос об удалении операции и удаляет при подтверждении;
- 4 – тип операций;
- 5 – сумма по операции с указанием валюты;
- 6 – категория операции;
- 7 – дата операции;

8 – при расходной операции показывает с какого счета произведен расход, при доходной операции показывает плательщик (место), при операции перевода показывает счет, с которого осуществлен перевод;

9 – при расходной операции показывает место платежа, при доходной операции показывает счет, на который поступил платеж, при операции перевода показывает счет, на который осуществлен перевод;

10 – описание операции.

Макет интерфейса страницы создания/изменения операции представлен на рисунке 37.

Рисунок 37 – Страница создания/изменения операции

На макете страницы создания/изменения операции имеются следующие элементы:

1 – кнопка «Назад», при нажатии возвращает на предыдущую страницу;

2 – кнопка «Сохранить», при нажатии осуществляет попытку создания/изменения операции;

3 – поле ввода суммы по операции с указанием валюты;

4 – поле выбора категории операции;

5 – поле выбора даты операции;

6 – при расходной операции выбирается с какого счета произведен расход, при доходной операции выбирается плательщик (место), при операции перевода выбирается счет, с которого осуществлен перевод;

7 – при расходной операции выбирается место платежа, при доходной операции выбирается счет, на который поступил платеж, при операции перевода выбирается счет, на который осуществлен перевод;

8 – поле ввода описания операции.

Макет интерфейса страницы аналитики представлен на рисунке 38.

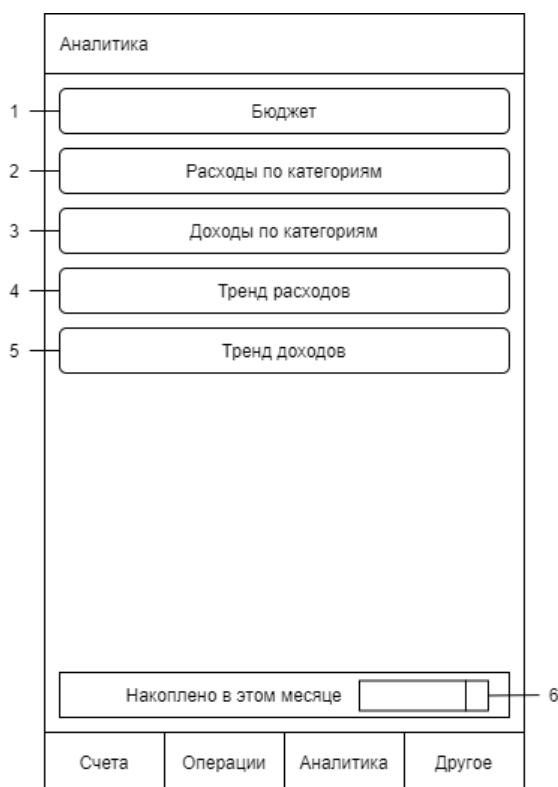


Рисунок 38 – Страница аналитики

На макете страницы аналитики имеются следующие элементы:

1 – кнопка «Бюджет», при нажатии открывает страницу просмотра бюджета на текущий месяц;

2 – кнопка «Расходы по категориям», при нажатии открывает страницу просмотра таблицы расходов по категориям;

3 – кнопка «Доходы по категориям», при нажатии открывает страницу просмотра таблицы доходов по категориям;

4 – кнопка «Тренд расходов», при нажатии открывает страницу графика расходов по месяцам за последний год;

5 – кнопка «Тренд доходов», при нажатии открывает страницу графика доходов по месяцам за последний год;

6 – изменение баланса по накопительным счетам в этом месяце.

Макет интерфейса страницы аналитики представлен на рисунке 39.

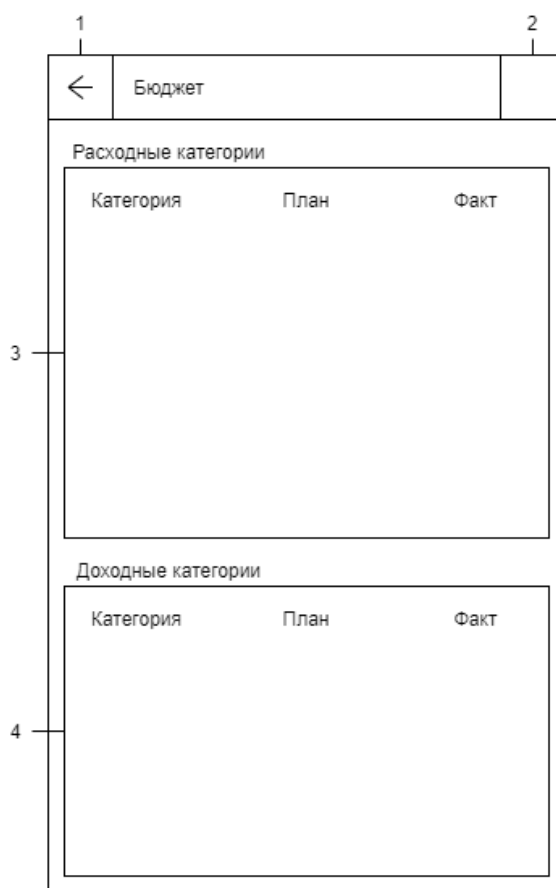


Рисунок 39 – Страница просмотра бюджета

На макете стриницы просмотра бюджета имеются следующие элементы:

1 – кнопка «Назад», при нажатии возвращает на предыдущую страницу;

2 – кнопка «Изменить», при нажатии открывает страницу изменения счета;

3 – таблица бюджета по расходным категориям (в колонке «План» – запланированная сумма, в колонке «Факт» – потраченная сумма);

4 – таблица бюджета по доходным категориям (в колонке «План» – запланированная сумма, в колонке «Факт» – заработанная сумма).

Макет интерфейса страницы «Другое» представлен на рисунке 40.

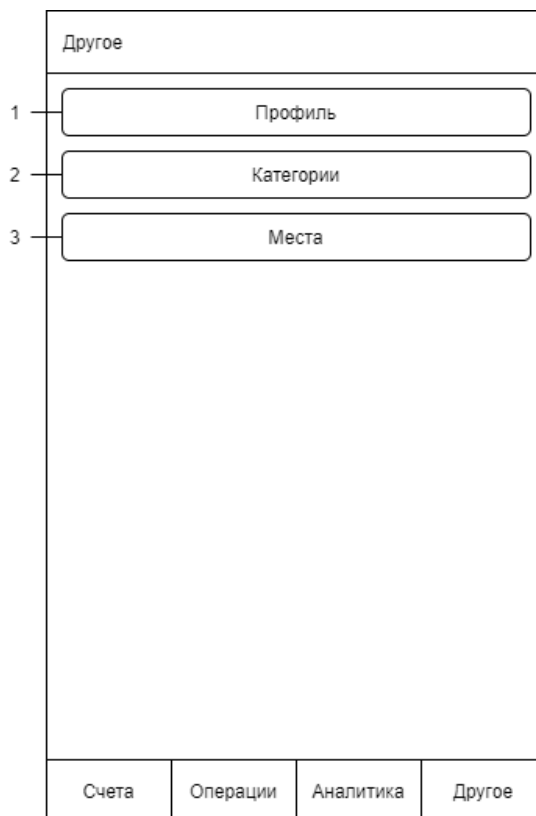


Рисунок 40 – Страница «Другое»

На макете страницы «Другое» имеются следующие элементы:

- 1 – кнопка «Профиль», при нажатии открывает страницу профиля;
- 2 – кнопка «Категории», при нажатии открывает страницу категорий;
- 3 – кнопка «Места», при нажатии открывает страницу мест.

Страницы просмотра информации о категориях и местах, а также их создания и изменения имеют аналогичный вид страниц управления операциями.

3 Реализация программного обеспечения

3.1 Реализация и развертывание базы данных

На основе физической модели были описаны классы представляющие сущности базы данных. При помощи технологии Entity Framework Core командой Add-Migration из описанных классов был сгенерирован скрипт миграции на языке C# для создания базы данных. Развертывание миграции базы данных выполнено с помощью команды Update-Database.

На рисунке 41 представлена диаграмма развернутой базы данных, построенная в среде MS SQL Management Studio.

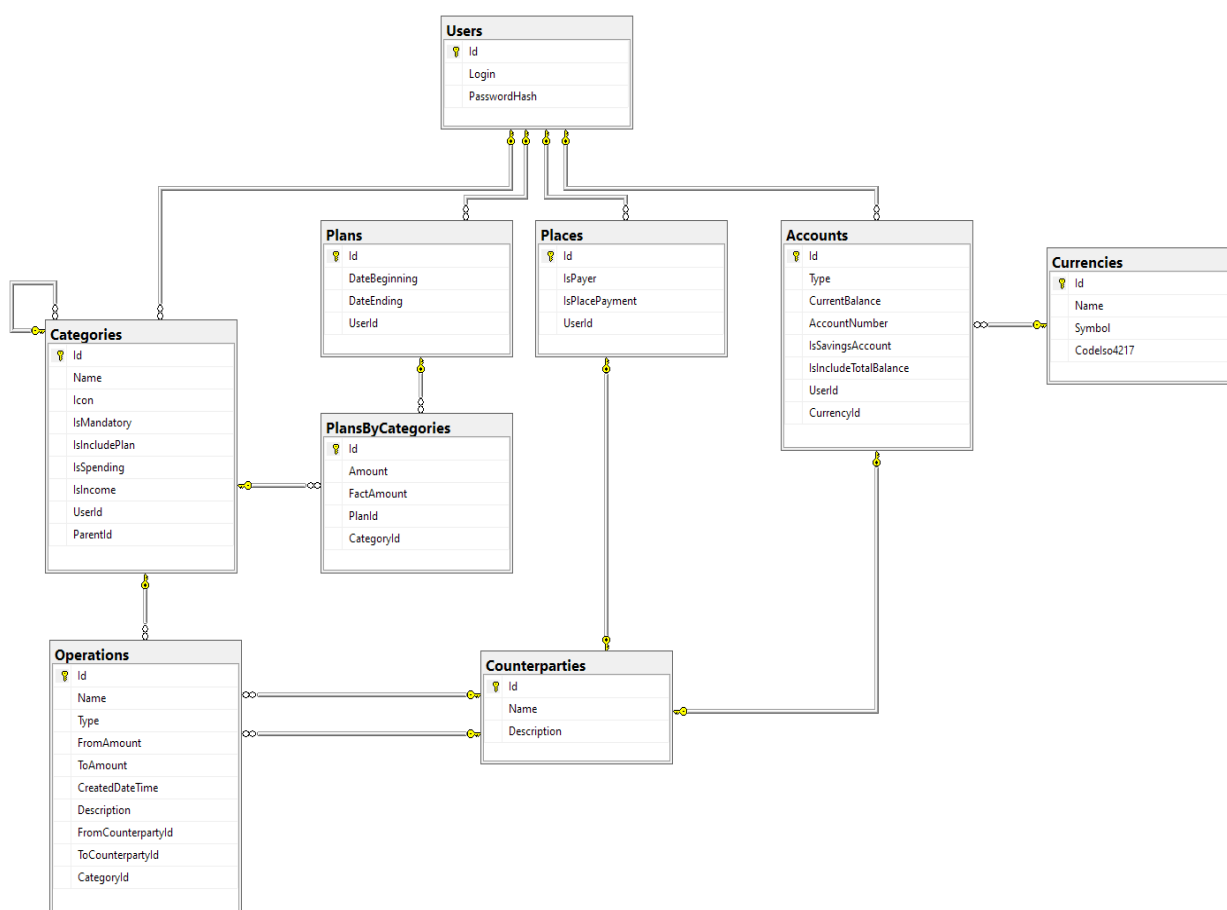


Рисунок 41 – Диаграмма развернутой базы данных

Развернутая БД содержит 9 сущностей и 13 связей между ними.

3.2 Реализация серверной части приложения

Исходя из результатов проектирования программного обеспечения и выбора технологий реализации, была реализована серверная часть приложения на языке C#. Исходный код представлен в приложении В.

Для регистрации, авторизации и аутентификации пользователя использована технология JwtBearer, она предоставляет простой интерфейс создания jwt-токенов, которые хранит пользователь системы, для доступа к защищённым ресурсам. При отправке запроса с этим токеном сервер понимает какой именно пользователь совершает запрос и обрабатывает его в соответствии с вышеперечисленным функционалом.

В соответствии с используемой архитектурой MVC все классы серверной части приложения можно разделить на два уровня: уровень модели, предназначенный для взаимодействия с БД, и уровень контроллеров, предназначенный для обработки HTTP-запросов к серверу.

Диаграмма классов уровня модели представлена на рисунке 42.

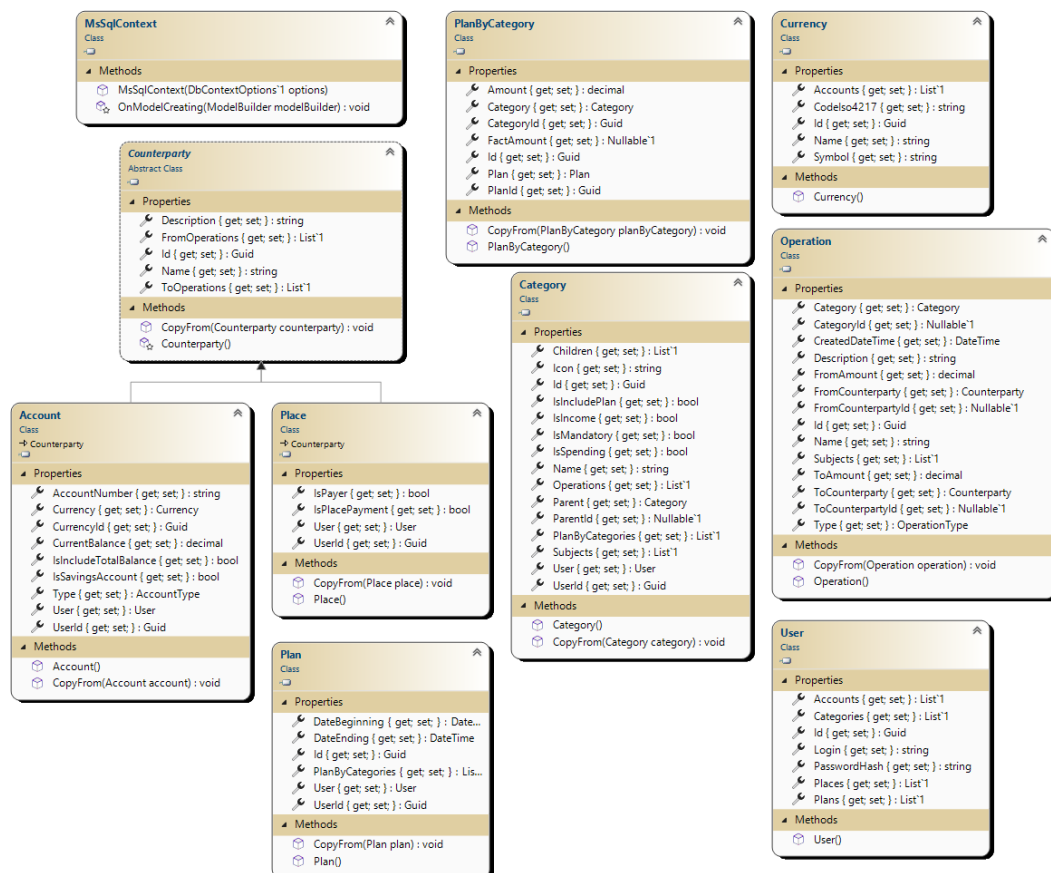


Рисунок 42 – Диаграмма классов уровня модели

Описание классов уровня модели приведено в таблице 17.

Таблица 17 – Описание классов уровня модели

Класс	Назначение
MsSqlContext	Предоставляет контекст базы данных, для выполнения запросов
User	Описывает сущность «Пользователь»
Counterparty	Описывает сущность «Контрагент»
Account	Описывает сущность «Счет»
Operation	Описывает сущность «Операция»
Category	Описывает сущность «Категория»
Place	Описывает сущность «Место»
Plan	Описывает сущность «План»
PlanByCategory	Описывает сущность «План по категориям»
Currency	Описывает сущность «Валюта»

Диаграмма классов уровня контроллеров представлена на рисунке 43.

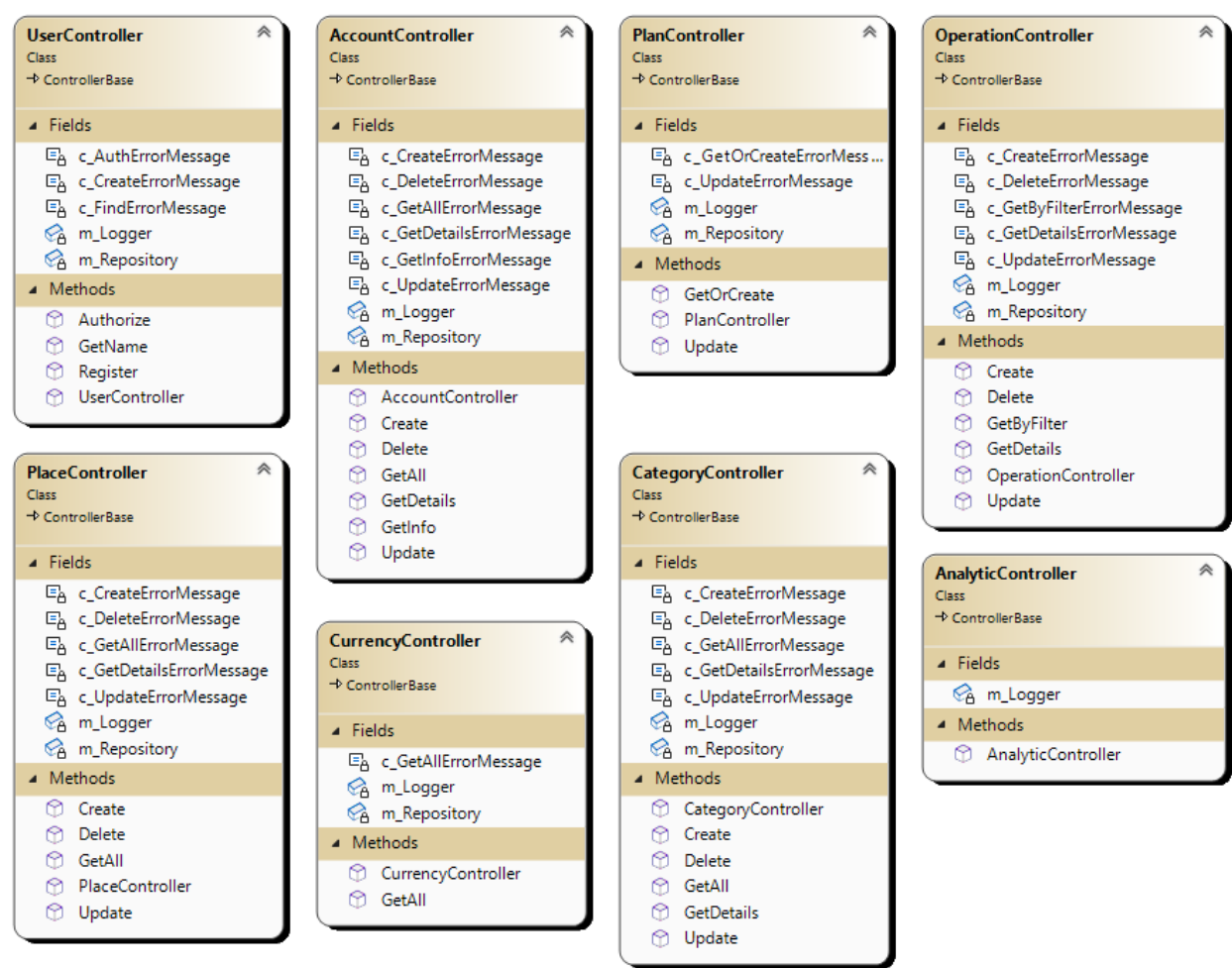


Рисунок 43 – Диаграмма классов уровня контроллеров

Описание классов уровня контроллеров приведено в таблице 18.

Таблица 18 – Описание классов уровня контроллеров

Класс	Назначение
UserController	Обрабатывает запросы авторизации, аутентификации и регистрации пользователя
AccountController	Обрабатывает запросы создания, изменения, получения и удаления счетов
OperationController	Обрабатывает запросы создания, изменения, получения и удаления операций
CategoryController	Обрабатывает запросы создания, изменения, получения и удаления категорий
PlaceController	Обрабатывает запросы создания, изменения, получения и удаления мест
AnalyticController	Обрабатывает запросы получения данных для формирования таблиц и графиков по внесенным операциям
PlanController	Обрабатывает запросы получения данных для формирования бюджета по категориям операций
CurrencyController	Обрабатывает запросы получения валют и информации о них

Все контроллеры взаимодействуют с базой данных через репозитории имеющие аналогичные названия.

3.3 Реализация мобильного приложения

Исходя из результатов проектирования программного обеспечения и выбора технологий реализации, было реализовано мобильное приложение на языке C#. Исходный код представлен в приложении Г.

Диаграмма компонентов, отражающая структуру программы, представлена на рисунке 44.

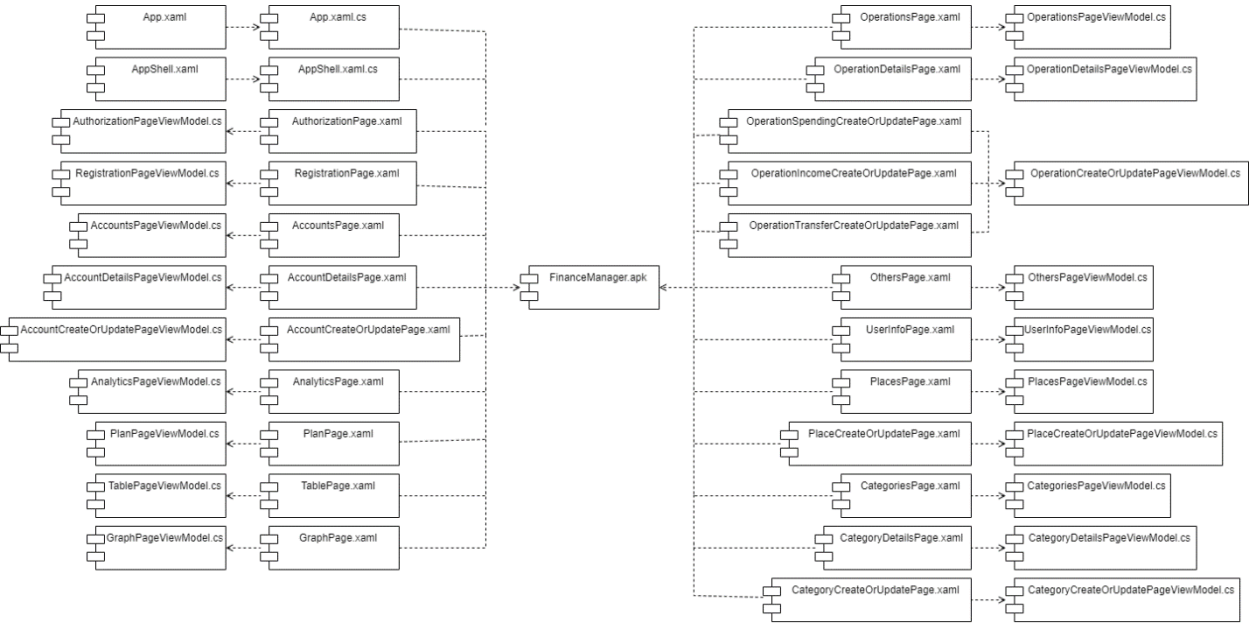


Рисунок 44 – Диаграмма компонентов мобильного приложения

Описание компонентов приведено в таблице 19.

Таблица 19 – Описание классов мобильного приложения

Компонент	Назначение
App.xaml	ХАМЛ-файл описания подключаемых ресурсов к приложению
App.xaml.cs	Является начальной точкой загрузки, закрытия приложения
AppShell.xaml	ХАМЛ-файл описания структуры интерфейса приложения
AppShell.xaml.cs	Регистрирует страницы приложения, не вошедшие в основную структуру
AuthorizationPage.xaml	ХАМЛ-файл описания интерфейса страницы авторизации
AuthorizationPageViewModel.cs	Управляет логикой и данными страницы авторизации
RegistrationPage.xaml	ХАМЛ-файл описания интерфейса страницы регистрации
RegistrationPageViewModel.cs	Управляет логикой и данными страницы регистрации
AccountsPage.xaml	ХАМЛ-файл описания интерфейса страницы счетов
AccountsPageViewModel.cs	Управляет логикой и данными страницы счетов
AccountDetailsPage.xaml	ХАМЛ-файл описания интерфейса страницы просмотра подробной информации о счете
AccountDetailsPageViewModel.cs	Управляет логикой и данными страницы просмотра подробной информации о счете
AccountCreateOrUpdatePage.xaml	ХАМЛ-файл описания интерфейса страницы добавления или изменения счета
AccountCreateOrUpdatePageViewModel.cs	Управляет логикой и данными страницы добавления или изменения счета
AnalyticsPage.xaml	ХАМЛ-файл описания интерфейса страницы аналитики
AnalyticsPageViewModel.cs	Управляет логикой и данными страницы аналитики
PlanPage.xaml	ХАМЛ-файл описания интерфейса страницы бюджета
PlanPageViewModel.cs	Управляет логикой и данными страницы бюджета
TablePage.xaml	ХАМЛ-файл описания интерфейса страницы таблиц отчета по операциям
TablePageViewModel.cs	Управляет логикой и данными страницы таблиц отчета по операциям
GraphPage.xaml	ХАМЛ-файл описания интерфейса страницы графиков отчета по операциям
GraphPageViewModel.cs	Управляет логикой и данными страницы графиков отчета по операциям
OperationsPage.xaml	ХАМЛ-файл описания интерфейса страницы операций
OperationsPageViewModel.cs	Управляет логикой и данными страницы операций
OperationDetailsPage.xaml	ХАМЛ-файл описания интерфейса страницы подробной информации об операции
OperationDetailsPageViewModel.cs	Управляет логикой и данными страницы подробной информации об операции
OperationSpendingCreateOrUpdatePage.xaml	ХАМЛ-файл описания интерфейса страницы создания или изменения расходной операции
OperationIncomeCreateOrUpdatePage.xaml	ХАМЛ-файл описания интерфейса страницы создания или изменения доходной операции
OperationTransferCreateOrUpdatePage.xaml	ХАМЛ-файл описания интерфейса страницы создания или изменения операции перевода
OperationCreateOrUpdatePageViewModel.cs	Управляет логикой и данными страницы создания или изменения операции
OthersPage.xaml	ХАМЛ-файл описания интерфейса страницы «Другое»
OthersPageViewModel.cs	Управляет логикой и данными страницы «Другое»
UserInfoPage.xaml	ХАМЛ-файл описания интерфейса страницы пользователя
UserInfoPageViewModel.cs	Управляет логикой и данными страницы пользователя

Продолжение таблицы 19

Компонент	Назначение
PlacesPage.xaml	ХАМЛ-файл описания интерфейса страницы мест
PlacesPageViewModel.cs	Управляет логикой и данными страницы мест
PlaceCreateOrUpdatePage.xaml	ХАМЛ-файл описания интерфейса страницы создания или изменения мест
PlaceCreateOrUpdatePageViewModel.cs	Управляет логикой и данными страницы создания или изменения мест
CategoriesPage.xaml	ХАМЛ-файл описания интерфейса страницы категорий
CategoriesPageViewModel.cs	Управляет логикой и данными страницы категорий
CategoryDetailsPage.xaml	ХАМЛ-файл описания интерфейса страницы подробной информации о категории
CategoryDetailsPageViewModel.cs	Управляет логикой и данными страницы подробной информации о категории
CategoryCreateOrUpdatePage.xaml	ХАМЛ-файл описания интерфейса страницы создания или изменения категории
CategoryCreateOrUpdatePageViewModel.cs	Управляет логикой и данными страницы создания или изменения категории

Диаграмма развертывания, на которой представлено расположение уровней программного продукта, представлена на рисунке 45.

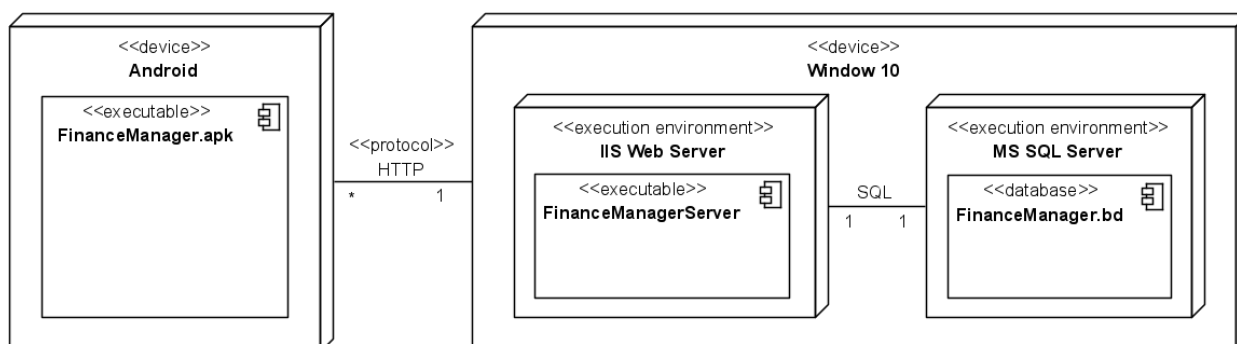


Рисунок 45 – Диаграмма развертывания программного продукта

Из диаграммы можно увидеть, что программный продукт состоит из трех уровней: мобильное приложение, работающее под управлением ОС Android, серверная часть и СУБД, работающие под управлением ОС Windows 10.

4 Тестирование программного обеспечения

4.1 Тестирование серверной части приложения

4.1.1 Функциональное тестирование

В первую очередь проведем функциональное тестирование корректности работы серверной части приложения. Результаты функционального тестирования мобильного приложения представлены в таблице 20.

Таблица 20 – Результаты функционального тестирования серверной части

№	Описание теста	Структура запроса	Результат
1	Регистрация пользователя	Метод: POST URL: api/user/register Body: { "login": "login", "password": "password123" }	Тест пройден. Пользователь успешно зарегистрирован (рисунок Е.1).
2	Авторизация пользователя	Метод: POST URL: api/user/authorize Body: { "login": "login", "password": "password123" }	Тест пройден. Пользователь успешно авторизован (рисунок Е.2).
3	Проверка аутентификации пользователя	Метод: GET URL: api/user/name	Тест пройден. Аутентификация выполнена (рисунок Е.3).
4	Получение списка валют	Метод: GET URL: api/currency	Тест пройден. Получен список валют (рисунок Е.4).
5	Создание счета	Метод: POST URL: api/user/account Body: { "name": "Наличные", ... }	Тест пройден. Счет успешно создан (рисунок Е.5).
6	Получение списка счетов	Метод: GET URL: api/user/account	Тест пройден. Получен список счетов (рисунок Е.6).
7	Изменение счета	Метод: PUT URL: api/user/account Body: { "name": "Наличные-2", ... }	Тест пройден. Счет успешно изменен (рисунок Е.7).
8	Получение детальной информации о счете	Метод: GET URL: api/user/account/details/6137f579-325a-4150-e407-08da4d354dd1	Тест пройден. Получена информация о счете (рисунок Е.8).
9	Получение общей информации о счетах	Метод: GET URL: api/user/account/info	Тест пройден. Получена общая информация о счетах (рисунок Е.9).
10	Удаление счета	Метод: DELETE URL: api/user/account/6137f579-325a-4150-e407-08da4d354dd1	Тест пройден. Счет успешно удален (рисунок Е.10).
11	Создание операции	Метод: POST URL: api/operation Body: { "fromAmount": 550, ... }	Тест пройден. Операция успешно создана (рисунок Е.11).

Продолжение таблицы 20

№	Описание теста	Структура запроса	Результат
13	Изменение операции	Метод: POST URL: api/operation/filter Body: { "fromAmount": 600, ... }	Тест пройден. Операция успешно изменена (рисунок Е.13).
14	Получение детальной информации об операции	Метод: GET URL: api/operation/details/d72d4a5f-d1fe-46f3-4b7a-08da4f501abc	Тест пройден. Получена детальная информация по операции (рисунок Е.14).
15	Удаление операции	Метод: DELETE URL: api/operation/d72d4a5f-d1fe-46f3-4b7a-08da4f501abc	Тест пройден. Операция успешно удалена (рисунок Е.15).
16	Добавление места	Метод: POST URL: api/place Body: { "name": "Пятерочка", ... }	Тест пройден. Место успешно добавлено (рисунок Е.16).
17	Получение списка мест	Метод: GET URL: api/place	Тест пройден. Получен список мест (рисунок Е.17).
18	Изменение места	Метод: PUT URL: api/place Body: { "name": "Магнит", ... }	Тест пройден. Место успешно изменено (рисунок Е.18).
19	Удаление места	Метод: DELETE URL: api/place/d32069f3-7def-4e26-dbfd-08da4f4f626d	Тест пройден. Место успешно удалено (рисунок Е.19).
20	Получение плана по категориям на месяц	Метод: GET URL: api/user/plan/2022-06-16T04:37:47.912Z	Тест пройден. Получен план по категориям на месяц (рисунок Е.20).
21	Изменение месячного плана	Метод: PUT URL: api/user/plan	Тест пройден. План успешно изменен (рисунок Е.21).
22	Получение суммы внесенных операций расхода по категориям	Метод: POST URL: api/place Body: { "operationType": "0", ... }	Тест пройден. Получены суммы внесённых операций расхода по категориям (рисунок Е.22).

Все функциональные тесты прошли успешно, это показывает корректность работы серверной части программного продукта.

4.1.2 Модульное тестирование

В дополнение к функциональному тестированию, на языке C# в среде разработки Visual Studio 2022 при использовании фреймворка MS Test были созданы модульные тесты регистрации и авторизации пользователя исходный код тестов представлен в приложении Д. Результаты тестирования представлены на рисунке 46.

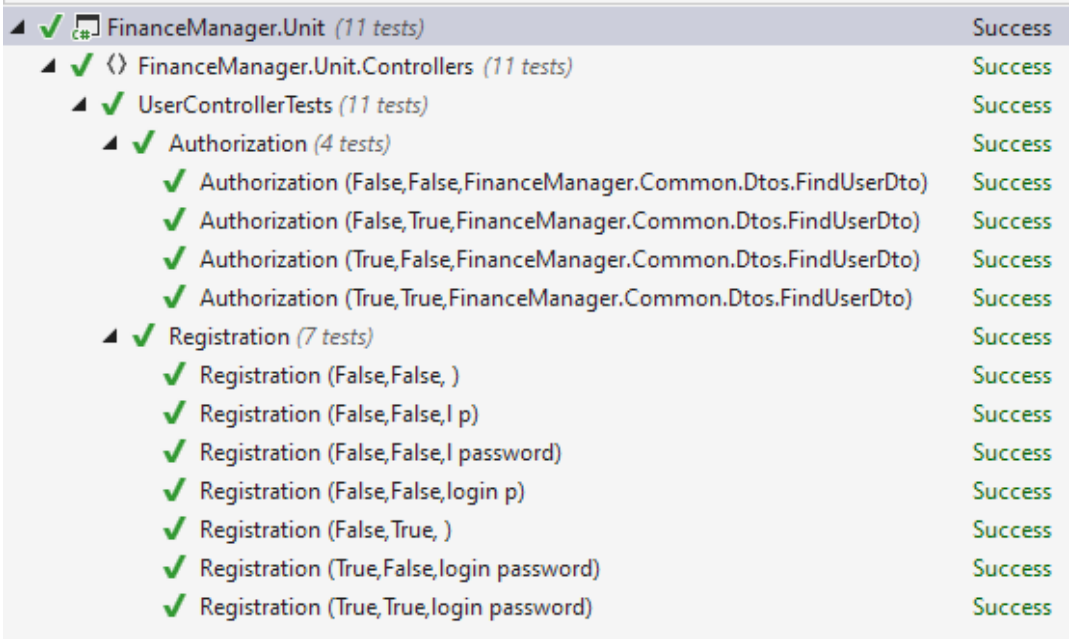


Рисунок 46 – Результаты модульного тестирования.

Все модульные тесты прошли успешно, это показывает корректность работы выбранного модуля серверной части программного продукта.

4.2 Тестирование мобильного приложения

Результаты функционального тестирования мобильного приложения представлены в таблице 21.

Таблица 21 – Результаты функционального тестирования мобильного приложения

Вариант использования	Параметры теста	Результат
Зарегистрироваться	Регистрация пользователя с корректными данными. Логин: ravel Пароль 123456	Тест пройден. Пользователь успешно зарегистрирован в системе (рисунок Ж.1).
	Регистрация пользователя с некорректными данными. Логин: av Пароль 13	Тест пройден. Показано сообщение с пояснением ошибки (рисунок Ж.2).
	Регистрация пользователя с тем же логином. Логин: ravel Пароль 456789	Тест пройден. Показано сообщение о том, что такой пользователь уже существует (рисунок Ж.3).

Продолжение таблицы 21

Вариант использования	Параметры теста	Результат
Авторизоваться	Авторизация пользователя с корректными данными. Логин: ravel Пароль 123456	Тест пройден. Авторизация выполнена. (рисунок Ж.4).
	Авторизация пользователя с некорректными данными. Логин: av Пароль 13	Тест пройден. Показано сообщение с пояснением ошибки. (рисунок Ж.5).
Создать счет	Создание счета	Тест пройден. Счет успешно создан. (рисунок Ж.6).
Редактировать счет	Изменение счета	Тест пройден. Счет успешно изменен. (рисунок Ж.7).
Создать операцию	Создание операции	Тест пройден. Операция успешно создана. (рисунок Ж.8).
Редактировать операцию	Изменение операции	Тест пройден. Операция успешно изменена. (рисунок Ж.9).
Получить список счетов	Получение списка счетов	Тест пройден. Показан список счетов. (рисунок Ж.10).
Получить список операций	Получение списка операций	Тест пройден. Показан список операций. (рисунок Ж.11).
Сформировать отчет по операциям	Просмотр отчета по операциям	Тест пройден. Показан отчет по операциям. (рисунок Ж.12).
Сформировать бюджет по категориям операций	Просмотр бюджета по операциям	Тест пройден. Показан бюджет по операциям. (рисунок Ж.13).

Тесты пройдены успешно, это показывает корректность работы мобильного приложения.

4.3 Метрики кода

Для оценки качества исходного кода написанного в результате реализации программного продукта были вычислены различные метрики кода. Для вычисления метрик возьмём метод `Authorize`, который реализует обработку HTTP запроса на авторизацию пользователя. Код данного метода приведен в листинге 1.

Листинг 1 – Исходный код метода авторизации пользователя

```
[HttpPost("authorize")]
public async Task<ActionResult<string>> Authorize(FindUserDto findUserDto)
{
    const int NumberDayValidToken = 30;

    try
    {
        findUserDto.Password = PasswordHasher.Hash(findUserDto.Password);
```

```

        var foundUser = await m_Repository.FindAsync(findUserDto);

        if (foundUser is null) return BadRequest(new { title = "Данные для
авторизации не верны" });

        var claims = new List<Claim>
        {
            new(ClaimTypes.NameIdentifier, $"{foundUser.Id}"),
            new(ClaimTypes.Name, foundUser.Login)
        };

        var jwt = new JwtSecurityToken(
            AuthOptions.Issuer,
            AuthOptions.Audience,
            claims,
            expires:
DateTime.UtcNow.Add(TimeSpan.FromDays (NumberDayValidToken)),
            signingCredentials: new
SigningCredentials (AuthOptions.GetSymmetricSecurityKey(),
SecurityAlgorithms.HmacSha256)
        );

        string? encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);

        return Ok(encodedJwt);
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_AuthErrorMessage);

        return BadRequest(new { title = $"{c_AuthErrorMessage}
({ex.Message})" });
    }
}

```

Данный метод содержится в серверной части приложения и выполняется при отправке запроса на авторизацию пользователя. В теле метода производится хеширование пароля, проверка существования пользователя с данным логином и хэшем пароля. В случае ошибки возвращается ее описание. Для данного метода будут вычислены две группы метрик: метрики Мак-Кейба и метрики Холстеда.

4.3.1 Метрика Мак-Кейба

Метрика Мак-Кейба, также известная, как цикломатическая метрика, позволяет оценить структурную сложность программы на основе управляющего графа программы.

Построенный управляющий граф анализируемого кода (рисунок 47).

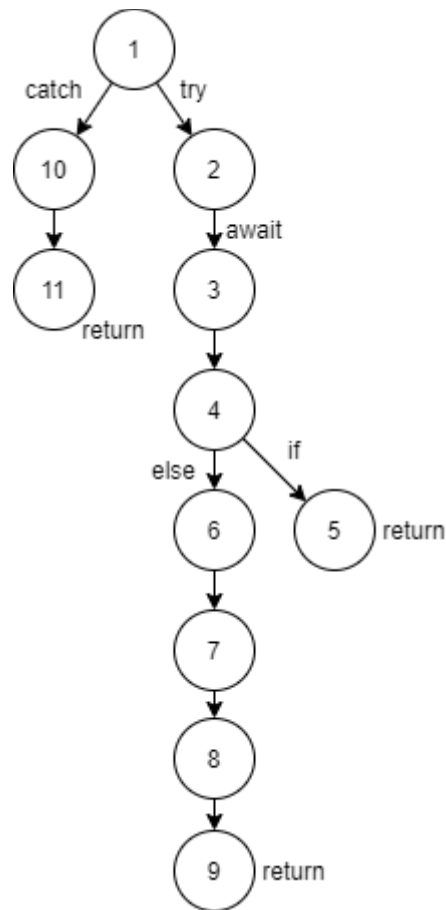


Рисунок 47 – Управляющий граф метода Authorize

Посчитаем количество вершин и дуг построенного графа: количество вершин $v = 11$, количество дуг $e = 10$. А также найдем компонент связности графа (p), для его нахождения необходимо посчитать количество таких подмножеств вершин графа, что каждые две вершины в этом подмножестве соединены хотя бы одной дугой. В данном графе $p = 1$.

Вычислим метрику Мак-Кейба:

$$C = e - v + 2p = 1.$$

Исходя из того, что полученное значение меньше 10, можно сделать вывод о том, что анализируемый программный код обладает достаточно простой структурой потока управления и легок для восприятия.

4.3.2 Метрика Холстеда

Для вычисления метрик Холстеда необходимо измерить следующие характеристики программы: число уникальных операторов, число уникальных

операндов, общее число операторов и общее число операндов. Для измерения вышеперечисленных характеристик составим словарь операторов (таблица 22) и словарь операндов (таблица 23).

Таблица 22 – Словарь операторов

Оператор	Число операторов
()	18
{}	9
;	10
.	16
const	1
=	8
var	3
await	1
if	1
is	1
return	3
new	6
“	4
<>	2
\$	2
,	9
:	2
?	1
catch	1

Таблица 23 – Словарь операндов

Операнд	Число операндов
findUserDto	4
NumberDayValidToken	2
foundUser	4
«строка»	3
claims	2
jwt	2
encodedJwt	2
c_AuthErrorMessage	2
null	1

Составленный словарь позволяет следующие значения: число уникальных операторов программы $n_1 = 19$, число уникальных операндов программы $n_2 = 9$, общее число операторов в программе $N_1 = 98$, общее число операндов в программе $N_2 = 22$.

На основе метрик Холстеда можно вычислить целый ряд характеристик программы. Вычислим некоторые из них:

- словарь программы (n):

$$n = n_1 + n_2 = 28.$$

- длина программы (N):

$$N = N_1 + N_2 = 120.$$

- теоретический объем программы (V):

$$V = N \log_2 n \approx 576.$$

- сложность программы (D):

$$D = \left(\frac{n_1}{2}\right) \cdot \left(\frac{N_2}{n_2}\right) \approx 12.$$

- оценка интеллектуальных усилий (E):

$$E = D \cdot V = 6912.$$

Вычисленные характеристики показывают, что программный код имеет достаточно высокий уровень качества и удобства для дальнейшего усовершенствования поддержки.

Заключение

В ходе выполнения выпускной квалификационной работы было создано мобильное приложение учета финансовых факторов при формировании бюджета.

На этапе анализа предметной области и формулирования требований был выполнен анализ программного обеспечения аналогичного назначения, в результате которого были выявлены их основные достоинства и недостатки, произведен анализ целевой аудитории и определены функциональные и нефункциональные требования, а так же разработана концептуальная модель БД.

Затем было осуществлено планирование разработки и оценки бюджета. По результатам планирования было выявлено, что реализация проекта займет 42 дня, а предполагаемые затраты будут 246800 рублей, из чего можно сделать вывод о его экономической целесообразности.

На этапе проектирования была разработана логическая и физическая модель базы данных. Выявлено что база данных будет развернута под управлением MS SQL Server 2019, серверная часть будет реализована на платформе ASP.NET Core, а мобильное приложение будет написано на языке C# на платформе .NET MAUI.

На этапе реализации была развернута база данных при помощи ORM-технологии Entity Framework Core. Реализован весь спроектированный функционал.

В перспективе планируется опубликовать разработанное приложение на мобильных площадках магазинов приложений.

Таким образом, в ходе выполнения выпускной квалификационной работы были успешно выполнены все поставленные задачи.

Список использованных источников

1. Цифровизация как тенденция развития современного общества: специфика научного дискурса. [Электронный ресурс]. <https://cyberleninka.ru/article/n/tsifrovizatsiya-kak-tendentsiya-razvitiya-sovremennogo-obschestva-spetsifika-nauchnogo-diskursa> (дата обращения: 13.04.2022).
2. Распоряжение Президента Российской Федерации от 9 мая 2017 г. № 203 «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы». [Электронный ресурс]. URL: http://znedu.hostedu.ru/files/ykaz_7668.pdf (дата обращения: 14.04.2022).
3. Программа «Цифровая экономика Российской Федерации». Утверждена распоряжением Правительства Российской Федерации от 28 июля 2017 г. № 1632-р. [Электронный ресурс]. URL: <http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf> (дата обращения: 15.04.2022).
4. Официальный сайт приложения «Money Manager». [Электронный ресурс]. URL: <https://realbyteapps.com> (дата обращения: 16.04.2022).
5. Официальный сайт приложения «Дзен-мани». [Электронный ресурс]. URL: <https://zenmoney.ru> (дата обращения: 17.04.2022).
6. Официальный сайт приложения «Monefy». [Электронный ресурс]. URL: <https://monefy.me> (дата обращения: 18.04.2022).
7. Гудков П.А. Методы сравнительного анализа. Учеб. пособие. / П.А. Гудков // Изд-во Пенз.гос. ун-та, 2008 – 81 с.
8. Глобальный рынок финансовых мобильных приложений. [Электронный ресурс]. URL: <https://plusworld.ru/daily/digital-banking/post-474780> (дата обращения: 19.04.2022).
9. Глобальный рынок мобильных устройств. [Электронный ресурс]. URL: <https://www.counterpointresearch.com/apple-achieves-record-june-quarter-shipments-xiaomi-becomes-second-largest-smartphone-brand-globally/> (дата обращения: 20.04.2022).

10. Android Releases | Android Developers. [Электронный ресурс]. URL: <https://developer.android.com/about/versions> (дата обращения: 21.04.2022).
11. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание / Д. Арлоу, А. Нейштадт // СПб: Символ-Плюс, 2007. – 624 с.
12. Прошкина Е. Н. Язык моделирования информационных процессов UML 2.0 (учебное пособие) / Е.Н. Прошкина, И.Ю. Балашова // Пенза, Издательство ПГУ, 2018. – 112 с.
13. Методология разработки программного обеспечения RUP. Рабочие потоки, действия и артефакты стадий. [Электронный ресурс]. URL: <https://skarlupka.ru/articles.php?id=38> (дата обращения: 02.05.2022).
14. Шаблон Model-View-ViewModel. [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/enterprise-application-patterns/mvvm> (дата обращения: 04.05.2022).
15. Шаблон Model-View-Controller. [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/aspnet/core/mvc> (дата обращения: 05.05.2022).
16. Microsoft Visual Studio | Microsoft вики. [Электронный ресурс]. URL: https://microsoft.fandom.com/ru/wiki/Microsoft_Visual_Studio (дата обращения: 06.05.2022).
17. Документация по Entity Framework. [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/ef> (дата обращения: 07.05.2022).
18. Документация по ASP.NET. [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/aspnet> (дата обращения: 08.05.2022).
19. Документация по MAUI. [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/maui> (дата обращения: 09.05.2022).
20. MS Project для удобного проектного планирования. [Электронный ресурс]. URL: <https://blog.ganttpro.com/ru/obzor-ms-microsoft-project/?ysclid=l2ekektrno> (дата обращения: 10.05.2022).
21. Ехлаков, Ю.П. Управление программными проектами. Стандарты, модели: Учебное пособие / Ю.П. Ехлаков. – СПб.: Лань, 2019. – 244 с.

22. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. – 102 с.
23. Гонтарева, И.В. Управление проектами / И.В. Гонтарева, Р.М. Нижегородцев, Д.А. Новиков. – М.: КД Либроком, 2018. – 384 с.
24. Вольфсон, Б.Л. Гибкое управление проектами и продуктами / Б.Л. Вольфсон. - СПб.: Питер, 2019. – 208 с.
25. Типовые архитектуры. [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/633/489/lecture/11073> (дата обращения: 11.05.2022).

Приложение А
ГЛОССАРИЙ
(справочное)

Финансовые факторы – к финансовым факторам относят доходные операции, расходные операции, задолженность.

Бюджет – финансовый план определенного субъекта установленный на определенный период времени.

RUP (Rational Unified Process) – методология разработки программного обеспечения, в основе которой лежит итеративная модель разработки.

Android – мобильная операционная система используемая в подавляющем большинстве мобильных устройств.

СУБД (Система управления базой данных) – специализированное программное обеспечение, предоставляющее программный интерфейс, который дает возможность создавать, модифицировать и использовать базу данных.

Jwt-токен – строка, состоящая из трех основных частей: заголовка, нагрузки и подписи.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных, основанный на технологии «клиент-сервер».

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

DTO (Data Transfer Object) – класс предназначенный для описания структуры хранения и передачи данных.

SQL (Structured Query Language) – декларативный язык программирования, применяемый для создания и модификации данных в реляционной базе данных.

REST API – архитектурный стиль взаимодействия компонентов распределенного приложения сети.

ООП – объектно ориентированное программирование, методология программирования, основанная на представлении программы в виде совокупности объектов.

Приложение Б
ИСХОДНЫЙ КОД МИГРАЦИИ БАЗЫ ДАННЫХ
(рекомендуемое)

Листинг Б.1 – Исходный код скрипта миграции базы данных

```

using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace FinanceManager.DataAccess.Migrations
{
    public partial class InitialCreate : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Counterparties",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
                    Name = table.Column<string>(type: "nvarchar(64)",
maxLength: 64, nullable: false),
                    Description = table.Column<string>(type: "nvarchar(256)",
maxLength: 256, nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Counterparties", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Currencies",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
                    Name = table.Column<string>(type: "nvarchar(64)",
maxLength: 64, nullable: false),
                    Symbol = table.Column<string>(type: "nvarchar(3)",
maxLength: 3, nullable: false),
                    CodeIso4217 = table.Column<string>(type: "nvarchar(6)",
maxLength: 6, nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Currencies", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Users",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
                    Login = table.Column<string>(type: "nvarchar(64)",
maxLength: 64, nullable: false),
                    PasswordHash = table.Column<string>(type: "nvarchar(64)",
maxLength: 64, nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Users", x => x.Id);
                });
        }
    }
}

```

```

migrationBuilder.CreateTable(
    name: "Accounts",
    columns: table => new
    {
        Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
        Type = table.Column<int>(type: "int", nullable: false),
        CurrentBalance = table.Column<decimal>(type:
"decimal(18,2)", precision: 18, scale: 2, nullable: false),
        AccountNumber = table.Column<string>(type:
"nvarchar(64)", maxLength: 64, nullable: true),
        IsSavingsAccount = table.Column<bool>(type: "bit",
nullable: false),
        IsIncludeTotalBalance = table.Column<bool>(type: "bit",
nullable: false),
        UserId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
        CurrencyId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Accounts", x => x.Id);
        table.ForeignKey(
            name: "FK_Accounts_Counterparties_Id",
            column: x => x.Id,
            principalTable: "Counterparties",
            principalColumn: "Id");
        table.ForeignKey(
            name: "FK_Accounts_Currencies_CurrencyId",
            column: x => x.CurrencyId,
            principalTable: "Currencies",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_Accounts_Users_UserId",
            column: x => x.UserId,
            principalTable: "Users",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateTable(
    name: "Categories",
    columns: table => new
    {
        Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
        Name = table.Column<string>(type: "nvarchar(64)",
maxLength: 64, nullable: false),
        Icon = table.Column<string>(type: "nvarchar(64)",
maxLength: 64, nullable: true),
        IsMandatory = table.Column<bool>(type: "bit", nullable:
false),
        IsIncludePlan = table.Column<bool>(type: "bit", nullable:
false),
        IsSpending = table.Column<bool>(type: "bit", nullable:
false),
        IsIncome = table.Column<bool>(type: "bit", nullable:
false),
        UserId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
        ParentId = table.Column<Guid>(type: "uniqueidentifier",
nullable: true)
    },

```

```

constraints: table =>
{
    table.PrimaryKey("PK_Categories", x => x.Id);
    table.ForeignKey(
        name: "FK_Categories_Categories_ParentId",
        column: x => x.ParentId,
        principalTable: "Categories",
        principalColumn: "Id");
    table.ForeignKey(
        name: "FK_Categories_Users_UserId",
        column: x => x.UserId,
        principalTable: "Users",
        principalColumn: "Id",
        onDelete: ReferentialAction.Cascade);
});

migrationBuilder.CreateTable(
    name: "Places",
    columns: table => new
    {
        Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
        IsPayer = table.Column<bool>(type: "bit", nullable:
false),
        IsPlacePayment = table.Column<bool>(type: "bit",
nullable: false),
        UserId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Places", x => x.Id);
        table.ForeignKey(
            name: "FK_Places_Counterparties_Id",
            column: x => x.Id,
            principalTable: "Counterparties",
            principalColumn: "Id");
        table.ForeignKey(
            name: "FK_Places_Users_UserId",
            column: x => x.UserId,
            principalTable: "Users",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateTable(
    name: "Plans",
    columns: table => new
    {
        Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
        DateBeginning = table.Column<DateTime>(type: "datetime2",
nullable: false),
        DateEnding = table.Column<DateTime>(type: "datetime2",
nullable: false),
        UserId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Plans", x => x.Id);
        table.ForeignKey(
            name: "FK_Plans_Users_UserId",
            column: x => x.UserId,
            principalTable: "Users",

```

```

        principalColumn: "Id",
        onDelete: ReferentialAction.Cascade);
    });

    migrationBuilder.CreateTable(
        name: "Operations",
        columns: table => new
        {
            Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
            Name = table.Column<string>(type: "nvarchar(64)",
maxLength: 64, nullable: false),
            Type = table.Column<int>(type: "int", nullable: false),
            FromAmount = table.Column<decimal>(type: "decimal(18,2)",
precision: 18, scale: 2, nullable: false),
            ToAmount = table.Column<decimal>(type: "decimal(18,2)",
precision: 18, scale: 2, nullable: false),
            CreatedDateTime = table.Column<DateTime>(type:
"datetime2", nullable: false),
            Description = table.Column<string>(type: "nvarchar(256)",
maxLength: 256, nullable: true),
            FromCounterpartyId = table.Column<Guid>(type:
"uniqueidentifier", nullable: true),
            ToCounterpartyId = table.Column<Guid>(type:
"uniqueidentifier", nullable: true),
            CategoryId = table.Column<Guid>(type: "uniqueidentifier",
nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Operations", x => x.Id);
            table.ForeignKey(
                name: "FK_Operations_Categories_CategoryId",
                column: x => x.CategoryId,
                principalTable: "Categories",
                principalColumn: "Id");
            table.ForeignKey(
                name:
"FK_Operations_Counterparties_FromCounterpartyId",
                column: x => x.FromCounterpartyId,
                principalTable: "Counterparties",
                principalColumn: "Id");
            table.ForeignKey(
                name:
"FK_Operations_Counterparties_ToCounterpartyId",
                column: x => x.ToCounterpartyId,
                principalTable: "Counterparties",
                principalColumn: "Id");
        });

    migrationBuilder.CreateTable(
        name: "PlansByCategories",
        columns: table => new
        {
            Id = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
            Amount = table.Column<decimal>(type: "decimal(18,2)",
precision: 18, scale: 2, nullable: false),
            FactAmount = table.Column<decimal>(type: "decimal(18,2)",
precision: 18, scale: 2, nullable: true),
            PlanId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
            CategoryId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false)
        },

```

```

constraints: table =>
{
    table.PrimaryKey("PK_PlansByCategories", x => x.Id);
    table.ForeignKey(
        name: "FK_PlansByCategories_Categories_CategoryId",
        column: x => x.CategoryId,
        principalTable: "Categories",
        principalColumn: "Id",
        onDelete: ReferentialAction.Cascade);
    table.ForeignKey(
        name: "FK_PlansByCategories_Plans_PlanId",
        column: x => x.PlanId,
        principalTable: "Plans",
        principalColumn: "Id");
});

migrationBuilder.InsertData(
    table: "Currencies",
    columns: new[] { "Id", "CodeIso4217", "Name", "Symbol" },
    values: new object[,]
    {
        { new Guid("17e7f1ef-f3a4-4154-99cf-a2e5c343cafb"),
        "USD", "Доллар США", "$" },
        { new Guid("30cf7770-3d10-445b-84b4-13636190d71f"),
        "JPY", "Иена", "¥" },
        { new Guid("8425aeb4-7192-41c8-bf97-b31eb7dc778a"),
        "CNY", "Китайский юань", "¥" },
        { new Guid("b5aa1362-369f-48b1-97dd-9958974468a6"),
        "EUR", "Евро", "€" },
        { new Guid("fcd7b9f7-7029-4286-997c-81c485ea8f24"),
        "GBP", "Фунт стерлингов", "£" },
        { new Guid("fda08c69-466f-47b7-8e22-cba398dbbbd8"),
        "RUB", "Российский рубль", "₽" }
    });

migrationBuilder.CreateIndex(
    name: "IX_Accounts_CurrencyId",
    table: "Accounts",
    column: "CurrencyId");

migrationBuilder.CreateIndex(
    name: "IX_Accounts_UserId",
    table: "Accounts",
    column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_Categories_Name",
    table: "Categories",
    column: "Name",
    unique: true);

migrationBuilder.CreateIndex(
    name: "IX_Categories_ParentId",
    table: "Categories",
    column: "ParentId");

migrationBuilder.CreateIndex(
    name: "IX_Categories_UserId",
    table: "Categories",
    column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_Operations_CategoryId",
    table: "Operations",
    column: "CategoryId");

```

```

migrationBuilder.CreateIndex(
    name: "IX_Operations_FromCounterpartyId",
    table: "Operations",
    column: "FromCounterpartyId");

migrationBuilder.CreateIndex(
    name: "IX_Operations_ToCounterpartyId",
    table: "Operations",
    column: "ToCounterpartyId");

migrationBuilder.CreateIndex(
    name: "IX_Places_UserId",
    table: "Places",
    column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_Plans_UserId",
    table: "Plans",
    column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_PlansByCategories_CategoryId",
    table: "PlansByCategories",
    column: "CategoryId");

migrationBuilder.CreateIndex(
    name: "IX_PlansByCategories_PlanId",
    table: "PlansByCategories",
    column: "PlanId");

migrationBuilder.CreateIndex(
    name: "IX_Subjects_CategoryId",
    table: "Subjects",
    column: "CategoryId");

migrationBuilder.CreateIndex(
    name: "IX_Subjects_OperationId",
    table: "Subjects",
    column: "OperationId");

migrationBuilder.CreateIndex(
    name: "IX_Users_Login",
    table: "Users",
    column: "Login",
    unique: true);
}

protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Accounts");

    migrationBuilder.DropTable(
        name: "Places");

    migrationBuilder.DropTable(
        name: "PlansByCategories");

    migrationBuilder.DropTable(
        name: "Subjects");

    migrationBuilder.DropTable(
        name: "Currencies");

```

```
migrationBuilder.DropTable(  
    name: "Plans");  
  
migrationBuilder.DropTable(  
    name: "Operations");  
  
migrationBuilder.DropTable(  
    name: "Categories");  
  
migrationBuilder.DropTable(  
    name: "Counterparties");  
  
migrationBuilder.DropTable(  
    name: "Users");  
    }  
}  
}
```

Приложение В
ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг В.1 – Файл AccountController.cs

```

namespace FinanceManager.Controllers;

[ApiController]
[Route("api/[controller]")]
public class AccountController : ControllerBase
{
    private const string c_CreateErrorMessage = "Неудача добавления  
счета";
    private const string c_UpdateErrorMessage = "Неудача изменения  
счета";
    private const string c_GetAllErrorMessage = "Неудача получения списка  
счетов";
    private const string c_GetDetailsErrorMessage = "Неудача получения  
детальной информации о счете";
    private const string c_GetInfoErrorMessage = "Неудача получения общей  
информации о счетах";
    private const string c_DeleteErrorMessage = "Неудача удаления счета";

    private readonly ILogger<AccountController> m_Logger;
    private readonly IAccountRepository m_Repository;

    public AccountController(ILogger<AccountController> logger,
IAccountRepository repository)
    {
        m_Logger = logger;
        m_Repository = repository;
    }

    [HttpPost]
    [Authorize]
    public async Task<ActionResult> Create(CreateAccountDto createAccountDto)
    {
        try
        {
            var userId = this.GetUserId();

            await m_Repository.CreateForUserAsync(userId, createAccountDto);

            return Ok();
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_CreateErrorMessage);

            return BadRequest(new { title = $"{c_CreateErrorMessage}  
{ex.Message}" });
        }
    }

    [HttpPut]
    [Authorize]
    public async Task<ActionResult> Update(UpdateAccountDto updateAccountDto)
    {
        try
        {
            var userId = this.GetUserId();

```

```

        await m_Repository.UpdateForUserAsync(userId, updateAccountDto);

        return Ok();
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_UpdateErrorMessage);

        return BadRequest(new { title = $"{c_UpdateErrorMessage}
({ex.Message})" });
    }
}

[HttpGet]
[Authorize]
public async Task<ActionResult<List<GetAccountDto>>> GetAll()
{
    try
    {
        var userId = this.GetUserId();

        var foundAccounts = await
m_Repository.GetAllForUserAsync(userId);

        return Ok(foundAccounts);
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_GetAllErrorMessage);

        return BadRequest(new { title = $"{c_GetAllErrorMessage}
({ex.Message})" });
    }
}

[HttpGet("details/{id:guid}")]
[Authorize]
public async Task<ActionResult<List<GetAccountDetailsDto>>>
GetDetails(Guid id)
{
    try
    {
        var userId = this.GetUserId();

        var foundAccount = await
m_Repository.GetDetailsForUserAsync(userId, id);

        return Ok(foundAccount);
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_GetDetailsErrorMessage);

        return BadRequest(new { title = $"{c_GetDetailsErrorMessage}
({ex.Message})" });
    }
}

[HttpGet("info")]
[Authorize]
public async Task<ActionResult<GetAccountsInfoDto>> GetInfo()
{
    try
    {
        var userId = this.GetUserId();

```

```

        var info = await m_Repository.GetInfoForUserAsync(userId);

        return Ok(info);
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_GetInfoErrorMessage);

        return BadRequest(new { title = $"{c_GetInfoErrorMessage}
({ex.Message})" });
    }
}

[HttpDelete("{id:guid}")]
[Authorize]
public async Task<ActionResult> Delete(Guid id)
{
    try
    {
        var userId = this.GetUserId();

        await m_Repository.DeleteForUserAsync(userId, id);

        return Ok();
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_DeleteErrorMessage);

        return BadRequest(new { title = $"{c_DeleteErrorMessage}
({ex.Message})" });
    }
}
}

```

Листинг В.2 – Файл CategoryController.cs

```

namespace FinanceManager.Controllers;

[ApiController]
[Route("api/[controller]")]
public class CategoryController : ControllerBase
{
    private const string c_CreateErrorMessage = "Неудача добавления
категории";
    private const string c_UpdateErrorMessage = "Неудача изменения
категории";
    private const string c_GetAllErrorMessage = "Неудача получения списка
категорий";
    private const string c_GetDetailsErrorMessage = "Неудача получения
детальной информации о категории";
    private const string c_DeleteErrorMessage = "Неудача удаления
категории";

    private readonly ILogger<CategoryController> m_Logger;
    private readonly ICategoryRepository m_Repository;

    public CategoryController(ILogger<CategoryController> logger,
ICategoryRepository repository)

```

```

{
    m_Logger      = logger;
    m_Repository = repository;
}

[HttpPost]
[Authorize]
public async Task<ActionResult> Create(CreateCategoryDto
createCategoryDto)
{
    try
    {
        var userId = this.GetUserId();

        await m_Repository.CreateForUserAsync(userId, createCategoryDto);

        return Ok();
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_CreateErrorMessage);

        return BadRequest(new { title = $"{c_CreateErrorMessage}
({ex.Message})" });
    }
}

[HttpPut]
[Authorize]
public async Task<ActionResult> Update(UpdateCategoryDto
updateCategoryDto)
{
    try
    {
        var userId = this.GetUserId();

        await m_Repository.UpdateForUserAsync(userId, updateCategoryDto);

        return Ok();
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_UpdateErrorMessage);

        return BadRequest(new { title = $"{c_UpdateErrorMessage}
({ex.Message})" });
    }
}

[HttpGet]
[Authorize]
public async Task<ActionResult<List<GetCategoryDto>>> GetAll()
{
    try
    {
        var userId = this.GetUserId();

        var foundCategories = await
m_Repository.GetAllForUserAsync(userId);

        return Ok(foundCategories);
    }
    catch (Exception ex)
    {

```

```

        m_Logger.LogError(ex, c_GetAllErrorMessage);

        return BadRequest(new { title = $"{c_GetAllErrorMessage}
({ex.Message})" });
    }

    [HttpGet("details/{id:guid}")]
    [Authorize]
    public async Task<ActionResult<List<GetCategoryDetailsDto>>>
GetDetails(Guid id)
    {
        try
        {
            var userId = this.GetUserId();

            var foundCategory = await
m_Repository.GetDetailsForUserAsync(userId, id);

            return Ok(foundCategory);
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_GetDetailsErrorMessage);

            return BadRequest(new { title = $"{c_GetDetailsErrorMessage}
({ex.Message})" });
        }
    }

    [HttpDelete("{id:guid}")]
    [Authorize]
    public async Task<ActionResult> Delete(Guid id)
    {
        try
        {
            var userId = this.GetUserId();

            await m_Repository.DeleteForUserAsync(userId, id);

            return Ok();
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_DeleteErrorMessage);

            return BadRequest(new { title = $"{c_DeleteErrorMessage}
({ex.Message})" });
        }
    }
}

```

Листинг В.3 – Файл CurrencyController.cs

```

namespace FinanceManager.Controllers;

[ApiController]
[Route("api/[controller]")]
public class CurrencyController : ControllerBase
{
    private const string c_GetAllErrorMessage = "Неудача получения списка
валют";
}

```

```

private readonly ILogger<CurrencyController> m_Logger;
private readonly ICurrencyRepository m_Repository;

public CurrencyController(ILogger<CurrencyController> logger,
ICurrencyRepository repository)
{
    m_Logger = logger;
    m_Repository = repository;
}

[HttpGet]
[Authorize]
public async Task<ActionResult<List<GetCurrencyDto>>> GetAll()
{
    try
    {
        var foundCurrencies = await m_Repository.GetAllAsync();

        return Ok(foundCurrencies);
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_GetAllErrorMessage);

        return BadRequest(new { title = $"{c_GetAllErrorMessage}
({ex.Message})" });
    }
}

```

Листинг В.4 – Файл OperationController.cs

```

namespace FinanceManager.Controllers;

[ApiController]
[Route("api/[controller]")]
public class OperationController : ControllerBase
{
    private const string c_CreateErrorMessage = "Неудача добавления операции";
    private const string c_UpdateErrorMessage = "Неудача изменения операции";
    private const string c_GetByFilterErrorMessage = "Неудача получения списка операций по фильтру";
    private const string c_GetDetailsErrorMessage = "Неудача получения детальной информации о операции";
    private const string c_DeleteErrorMessage = "Неудача удаления операции";

    private readonly ILogger<OperationController> m_Logger;
    private readonly IOperationRepository m_Repository;

    public OperationController(ILogger<OperationController> logger,
IOperationRepository repository)
    {
        m_Logger = logger;
    }
}

```

```

        m_Repository = repository;
    }

    [HttpPost]
    [Authorize]
    public async Task<ActionResult> Create(CreateOperationDto
createOperationDto)
    {
        try
        {
            var userId = this.GetUserId();

            await m_Repository.CreateForUserAsync(userId,
createOperationDto);

            return Ok();
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_CreateErrorMessage);

            return BadRequest(new { title = $"{c_CreateErrorMessage}
({ex.Message})" });
        }
    }

    [HttpPut]
    [Authorize]
    public async Task<ActionResult> Update(UpdateOperationDto
updateOperationDto)
    {
        try
        {
            var userId = this.GetUserId();

            await m_Repository.UpdateForUserAsync(userId,
updateOperationDto);

            return Ok();
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_UpdateErrorMessage);

            return BadRequest(new { title = $"{c_UpdateErrorMessage}
({ex.Message})" });
        }
    }

    [HttpPost("filter")]
    [Authorize]
    public async Task<ActionResult<List<GetOperationDto>>>
GetByFilter(FilterOperationDto filter)
    {
        try
        {
            var userId = this.GetUserId();

            var foundOperations = await
m_Repository.GetByFilterForUserAsync(userId, filter);

            return Ok(foundOperations);
        }
        catch (Exception ex)

```

```

        {
            m_Logger.LogError(ex, c_GetByFilterErrorMessage);

            return BadRequest(new { title = $"{c_GetByFilterErrorMessage}
({ex.Message})" });
        }
    }

    [HttpGet("details/{id:guid}")]
    [Authorize]
    public async Task<ActionResult<List<GetOperationDetailsDto>>>
    GetDetails(Guid id)
    {
        try
        {
            var userId = this.GetUserId();

            var foundOperation = await
m_Repository.GetDetailsForUserAsync(userId, id);

            return Ok(foundOperation);
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_GetDetailsErrorMessage);

            return BadRequest(new { title = $"{c_GetDetailsErrorMessage}
({ex.Message})" });
        }
    }

    [HttpDelete("{id:guid}")]
    [Authorize]
    public async Task<ActionResult> Delete(Guid id)
    {
        try
        {
            var userId = this.GetUserId();

            await m_Repository.DeleteForUserAsync(userId, id);

            return Ok();
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_DeleteErrorMessage);

            return BadRequest(new { title = $"{c_DeleteErrorMessage}
({ex.Message})" });
        }
    }
}

```

Листинг В.5 – Файл PlaceController.cs

```

namespace FinanceManager.Controllers;

[ApiController]
[Route("api/[controller]")]
public class PlaceController : ControllerBase
{

```

```

        private const string c_CreateErrorMessage      = "Неудача добавления
места";
        private const string c_UpdateErrorMessage     = "Неудача изменения
места";
        private const string c_GetAllErrorMessage     = "Неудача получения списка
мест";
        private const string c_DeleteErrorMessage     = "Неудача удаления места";

        private readonly ILogger<PlaceController> m_Logger;
        private readonly IPlaceRepository           m_Repository;

        public PlaceController(ILogger<PlaceController> logger, IPlaceRepository
repository)
        {
            m_Logger      = logger;
            m_Repository = repository;
        }

        [HttpPost]
        [Authorize]
        public async Task<ActionResult> Create(CreatePlaceDto createPlaceDto)
        {
            try
            {
                var userId = this.GetUserId();

                await m_Repository.CreateForUserAsync(userId, createPlaceDto);

                return Ok();
            }
            catch (Exception ex)
            {
                m_Logger.LogError(ex, c_CreateErrorMessage);

                return BadRequest(new { title = $"{c_CreateErrorMessage}
({ex.Message})" });
            }
        }

        [HttpPut]
        [Authorize]
        public async Task<ActionResult> Update(UpdatePlaceDto updatePlaceDto)
        {
            try
            {
                var userId = this.GetUserId();

                await m_Repository.UpdateForUserAsync(userId, updatePlaceDto);

                return Ok();
            }
            catch (Exception ex)
            {
                m_Logger.LogError(ex, c_UpdateErrorMessage);

                return BadRequest(new { title = $"{c_UpdateErrorMessage}
({ex.Message})" });
            }
        }

        [HttpGet]
        [Authorize]

```

```

public async Task<ActionResult<List<GetPlaceDto>>> GetAll()
{
    try
    {
        var userId = this.GetUserId();

        var foundPlaces = await m_Repository.GetAllForUserAsync(userId);

        return Ok(foundPlaces);
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_GetAllErrorMessage);

        return BadRequest(new { title = $"{c_GetAllErrorMessage}
({ex.Message})" });
    }
}

[HttpDelete("{id:guid}")]
[Authorize]
public async Task<ActionResult> Delete(Guid id)
{
    try
    {
        var userId = this.GetUserId();

        await m_Repository.DeleteForUserAsync(userId, id);

        return Ok();
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_DeleteErrorMessage);

        return BadRequest(new { title = $"{c_DeleteErrorMessage}
({ex.Message})" });
    }
}
}

```

Листинг В.6 – Файл PlanController.cs

```

namespace FinanceManager.Controllers;

[ApiController]
[Route("api/[controller]")]
public class PlanController : ControllerBase
{
    private const string c_GetOrCreateErrorMessage = "Неудача получения или
создания плана";
    private const string c_UpdateErrorMessage = "Неудача изменения
плана";

    private readonly ILogger<AccountController> m_Logger;
    private readonly IPlanRepository m_Repository;

    public PlanController(ILogger<AccountController> logger, IPlanRepository
repository)
    {

```

```

        m_Logger      = logger;
        m_Repository = repository;
    }

    [HttpGet("{dateMonth:datetime}")]
    [Authorize]
    public async Task<ActionResult<GetPlanDto>> GetOrCreate(DateTime
dateMonth)
    {
        try
        {
            var userId = this.GetUserId();

            var plan = await m_Repository.GetOrCreateForUserAsync(userId,
dateMonth);

            return Ok(plan);
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_GetOrCreateErrorMessage);

            return BadRequest(new { title = $"{c_GetOrCreateErrorMessage}
({ex.Message})" });
        }
    }

    [HttpPut]
    [Authorize]
    public async Task<ActionResult> Update(UpdatePlanDto updatePlanDto)
    {
        try
        {
            var userId = this.GetUserId();

            await m_Repository.UpdateForUserAsync(userId, updatePlanDto);

            return Ok();
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_UpdateErrorMessage);

            return BadRequest(new { title = $"{c_UpdateErrorMessage}
({ex.Message})" });
        }
    }
}

```

Листинг В.7 – Файл UserController.cs

```

using System.IdentityModel.Tokens.Jwt;

using FinanceManager.Auth;

using Microsoft.IdentityModel.Tokens;

namespace FinanceManager.Controllers;

[ApiController]

```

```

[Route("api/[controller]")]
public class UserController : ControllerBase
{
    private const string c_CreateErrorMessage = "Неудача добавления
пользователя";
    private const string c_AuthErrorMessage = "Неудача авторизации
пользователя";
    private const string c_FindErrorMessage = "Неудача поиска
пользователя";

    private readonly ILogger<UserController> m_Logger;
    private readonly IUserRepository m_Repository;

    public UserController(ILogger<UserController> logger, IUserRepository
repository)
    {
        m_Logger = logger;
        m_Repository = repository;
    }

    [HttpPost("register")]
    public async Task<ActionResult> Register(CreateUserDto createUserDto)
    {
        try
        {
            createUserDto.Password =
PasswordHasher.Hash(createUserDto.Password);

            bool IsContains = await
m_Repository.ContainsAsync(createUserDto.Login);

            if (IsContains) return BadRequest(new { title = "Пользователь с
данным логином уже существует" });

            await m_Repository.CreateAsync(createUserDto);

            return Ok();
        }
        catch (Exception ex)
        {
            m_Logger.LogError(ex, c_CreateErrorMessage);

            return BadRequest(new { title = $"{c_CreateErrorMessage}
({ex.Message})" });
        }
    }

    [HttpPost("authorize")]
    public async Task<ActionResult<string>> Authorize(FindUserDto
findUserDto)
    {
        const int NumberDayValidToken = 30;

        try
        {
            findUserDto.Password = PasswordHasher.Hash(findUserDto.Password);

            var foundUser = await m_Repository.FindAsync(findUserDto);

            if (foundUser is null) return BadRequest(new { title = "Данные
для авторизации не верны" });

```

```

        var claims = new List<Claim>
        {
            new(ClaimTypes.NameIdentifier, $"{foundUser.Id}"),
            new(ClaimTypes.Name, foundUser.Login)
        };

        var jwt = new JwtSecurityToken(
            AuthOptions.Issuer,
            AuthOptions.Audience,
            claims,
            expires:
DateTime.UtcNow.Add(TimeSpan.FromDays (NumberDayValidToken)),
            signingCredentials: new
SigningCredentials(AuthOptions.GetSymmetricSecurityKey(),
SecurityAlgorithms.HmacSha256)
        );

        string? encodedJwt = new
JwtSecurityTokenHandler().WriteToken(jwt);

        return Ok(encodedJwt);
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_AuthErrorMessage);

        return BadRequest(new { title = $"{c_AuthErrorMessage}
({ex.Message})" });
    }
}

[HttpGet("name")]
[Authorize]
public async Task<ActionResult<GetUserDto>> GetName()
{
    try
    {
        var userId = this.GetUserId();

        var foundUser = await m_Repository.GetAsync(userId);

        return foundUser is { Login: { } }
            ? Ok(foundUser)
            : NotFound(new { title = "Ошибка авторизации" });
    }
    catch (Exception ex)
    {
        m_Logger.LogError(ex, c_FindErrorMessage);

        return BadRequest(new { title = $"{c_FindErrorMessage}
({ex.Message})" });
    }
}
}

```

Листинг В.8 – Файл AccountRepository.cs

```
using AutoMapper;
```

```
namespace FinanceManager.DataAccess.Repositories;
```

```

public class AccountRepository : BaseRepository, IAccountRepository
{
    public AccountRepository(MsSqlContext context, IMapper mapper) :
    base(context, mapper)
    {
    }

    public async Task CreateForUserAsync(Guid userId, CreateAccountDto
createAccountDto)
    {
        var account = Mapper.Map<Account>(createAccountDto);

        account.UserId = userId;

        await Context.Accounts.AddAsync(account);

        await Context.SaveChangesAsync();
    }

    public async Task UpdateForUserAsync(Guid userId, UpdateAccountDto
updateAccountDto)
    {
        var account = Mapper.Map<Account>(updateAccountDto);

        var foundAccount = await Context.Accounts.FirstAsync(x =>
            x.UserId.Equals(userId)
            && x.Id.Equals(account.Id));

        foundAccount.CopyFrom(account);

        await Context.SaveChangesAsync();
    }

    public async Task<List<GetAccountDto>> GetAllForUserAsync(Guid userId)
    {
        var account = await Context.Accounts.Where(x =>
            x.UserId.Equals(userId))
            .ToListAsync();

        return Mapper.Map<List<GetAccountDto>>(account);
    }

    public async Task<GetAccountDetailsDto> GetDetailsForUserAsync(Guid
userId, Guid id)
    {
        var date = DateTime.Now.Date;

        var filter = new FilterOperationDto
        {
            BeginDate = date,
            EndDate    = date.AddDays(1)
        };

        var foundAccount = await Context.Accounts.FirstAsync(x =>
            x.UserId.Equals(userId)
            && x.Id.Equals(id));

        return Mapper.Map<GetAccountDetailsDto>(foundAccount);
    }

    public async Task<GetAccountsInfoDto> GetInfoForUserAsync(Guid userId)
    {
        //TODO: currency
        var currency = await Context.Currencies.FirstAsync(x =>

```

```

        x.CodeIso4217.Equals("RUB"));

decimal balance = await Context.Accounts.Where(x =>
    x.UserId.Equals(userId)
    && x.IsIncludeTotalBalance
    && x.Currency.Id.Equals(currency.Id))
    .SumAsync(x => x.CurrentBalance);

var info = new GetAccountsInfoDto
{
    Balance = balance,
    CurrencySymbol = currency.Symbol
};

return info;
}

public async Task DeleteForUserAsync(Guid userId, Guid id)
{
    var foundAccount = await Context.Accounts.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(id));

    Context.Accounts.Remove(foundAccount);

    await Context.SaveChangesAsync();
}
}

```

Листинг В.9 – Файл BaseRepository.cs

```

using AutoMapper;

namespace FinanceManager.DataAccess.Repositories;

public class BaseRepository
{
    public MsSqlContext Context { get; }
    public IMapper Mapper { get; }

    public BaseRepository(MsSqlContext context, IMapper mapper)
    {
        Context = context;
        Mapper = mapper;
    }
}

```

Листинг В.10 – Файл CategoryRepository.cs

```

using AutoMapper;

namespace FinanceManager.DataAccess.Repositories;

public class CategoryRepository : BaseRepository, ICategoryRepository
{
    public CategoryRepository(MsSqlContext context, IMapper mapper) :
    base(context, mapper)

```

```

{
}

public async Task CreateForUserAsync(Guid userId, CreateCategoryDto
createCategoryDto)
{
    var category = Mapper.Map<Category>(createCategoryDto);

    category.UserId = userId;

    await Context.Categories.AddAsync(category);

    await Context.SaveChangesAsync();
}

public async Task UpdateForUserAsync(Guid userId, UpdateCategoryDto
updateCategoryDto)
{
    var category = Mapper.Map<Category>(updateCategoryDto);

    var foundCategory = await Context.Categories.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(category.Id));

    foundCategory.CopyFrom(category);

    await Context.SaveChangesAsync();
}

public async Task<List<GetCategoryDto>> GetAllForUserAsync(Guid userId)
{
    var categories = await Context.Categories.Where(x =>
        x.UserId.Equals(userId))
        .ToListAsync();

    return Mapper.Map<List<GetCategoryDto>>(categories);
}

public async Task<GetCategoryDetailsDto> GetDetailsForUserAsync(Guid
userId, Guid id)
{
    var foundCategory = await Context.Categories.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(id));

    var categoryDetails =
Mapper.Map<GetCategoryDetailsDto>(foundCategory);

    categoryDetails.Childrens =
Mapper.Map<List<GetCategoryDto>>(foundCategory.Children);

    return categoryDetails;
}

public async Task DeleteForUserAsync(Guid userId, Guid id)
{
    var foundCategory = await Context.Categories.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(id));

    Context.Categories.Remove(foundCategory);

    await Context.SaveChangesAsync();
}

```

```
}
```

Листинг В.11 – Файл CurrencyRepository.cs

```
using AutoMapper;

namespace FinanceManager.DataAccess.Repositories;

public class CurrencyRepository : BaseRepository, ICurrencyRepository
{
    public CurrencyRepository(MsSqlContext context, IMapper mapper) :
    base(context, mapper)
    {
    }

    public async Task<List<GetCurrencyDto>> GetAllAsync()
    {
        var currencies = await Context.Currencies
            .ToListAsync();

        return Mapper.Map<List<GetCurrencyDto>>(currencies);
    }
}
```

Листинг В.12 – Файл ExtensionsRepository.cs

```
namespace FinanceManager.DataAccess.Repositories;

public static class ExtensionsRepository
{
    public static Task<List<Guid>> GetIdAccountsByFilter(this BaseRepository
    repository, Guid userId, FilterOperationDto filter)
    {
        return filter switch
        {
            { IncludeAccountTypes: { Count: > 0 } } =>
                repository.Context.Accounts.Where(x =>
                    x.UserId.Equals(userId)
                    && filter.IncludeAccountTypes.Contains(x.Id))
                    .Select(x => x.Id)
                    .ToListAsync(),
            { ExcludeAccountTypes: { Count: > 0 } } =>
                repository.Context.Accounts.Where(x =>
                    x.UserId.Equals(userId)
                    && !filter.ExcludeAccountTypes.Contains(x.Id))
                    .Select(x => x.Id)
                    .ToListAsync(),
            _ => repository.Context.Accounts.Where(x =>
                x.UserId.Equals(userId))
                .Select(x => x.Id)
                .ToListAsync()
        };
    }

    public static Task<List<Guid>> GetIdPlacesByFilter(this BaseRepository
    repository, Guid userId, FilterOperationDto filter)
    {
        return filter switch
```

```

    {
        { IncludePlacesTypes: { Count: > 0 } } =>
            repository.Context.Places.Where(x =>
                x.UserId.Equals(userId)
                && filter.IncludePlacesTypes.Contains(x.Id))
                .Select(x => x.Id)
                .ToListAsync(),
        { ExcludePlacesTypes: { Count: > 0 } } =>
            repository.Context.Places.Where(x =>
                x.UserId.Equals(userId)
                && !filter.ExcludePlacesTypes.Contains(x.Id))
                .Select(x => x.Id)
                .ToListAsync(),
        _ => repository.Context.Places.Where(x =>
            x.UserId.Equals(userId))
            .Select(x => x.Id)
            .ToListAsync()
    };
}

public static Task<List<Operation>> GetOperationsByFilter(this
BaseRepository repository, List<Guid>? accounts, List<Guid>? places,
FilterOperationDto filter)
{
    var query = repository.Context.Operations.AsQueryable();

    query = accounts switch
    {
        { Count: > 0 } =>
            query.Where(x =>
                (x.FromCounterpartyId.HasValue &&
accounts.Contains(x.FromCounterpartyId.Value))
                || (x.ToCounterpartyId.HasValue &&
accounts.Contains(x.ToCounterpartyId.Value))),
        _ => query
    };

    query = places switch
    {
        { Count: > 0 } =>
            query.Where(x =>
                (x.FromCounterpartyId.HasValue &&
places.Contains(x.FromCounterpartyId.Value))
                || (x.ToCounterpartyId.HasValue &&
places.Contains(x.ToCounterpartyId.Value))),
        _ => query
    };

    query = filter switch
    {
        { BeginDate: { }, EndDate: { } } =>
            query.Where(x =>
                DateTime.Compare(filter.BeginDate.Value,
x.CreatedDateTime) <= 0
                && DateTime.Compare(filter.EndDate.Value,
x.CreatedDateTime) > 0),
        { BeginDate: { } } =>
            query.Where(x =>
                DateTime.Compare(filter.BeginDate.Value,
x.CreatedDateTime) <= 0),
        { EndDate: { } } =>
            query.Where(x =>
                DateTime.Compare(filter.EndDate.Value, x.CreatedDateTime)
> 0),
        _ => query
    };
}

```

```

};

query = filter switch
{
    { OperationsTypes: { Count: > 0 } } =>
        query.Where(x =>
            filter.OperationsTypes.Contains(x.Type)),
    _ => query
};

return query
    .OrderBy(x => x.CreatedDateTime)
    .ToListAsync();
}
}

```

Листинг В.13 – Файл OperationRepository.cs

```

using AutoMapper;

namespace FinanceManager.DataAccess.Repositories;

public class OperationRepository : BaseRepository, IOperationRepository
{
    public OperationRepository(MsSqlContext context, IMapper mapper) :
    base(context, mapper)
    {
        public async Task CreateForUserAsync(Guid userId, CreateOperationDto
        createOperationDto)
        {
            var operation = Mapper.Map<Operation>(createOperationDto);

            await EnsureCorrectAccessCounterparty(userId, operation);

            UpdateName(operation);

            await Context.Operations.AddAsync(operation);

            await Context.SaveChangesAsync();
        }

        public async Task UpdateForUserAsync(Guid userId, UpdateOperationDto
        updateOperationDto)
        {
            var operation = Mapper.Map<Operation>(updateOperationDto);

            await EnsureCorrectAccessCounterparty(userId, operation);

            UpdateName(operation);

            var foundOperation = await Context.Operations.FirstAsync(x =>
                x.Id.Equals(operation.Id));

            foundOperation.CopyFrom(operation);

            await Context.SaveChangesAsync();
        }
    }
}

```

```

    public async Task<List<GetGroupOperationsDto>>
    GetByFilterForUserAsync(Guid userId, FilterOperationDto filter)
    {
        var foundAccounts = await this.GetIdAccountsByFilter(userId, filter);

        var foundPlaces = await this.GetIdPlacesByFilter(userId, filter);

        var foundOperations = await this.GetOperationsByFilter(foundAccounts,
foundPlaces, filter);

        return foundOperations
            .GroupBy(x => x.CreatedDateTime.Date)
            .Select(x => new GetGroupOperationsDto
            {
                Date = x.Key,
                Operations = Mapper.Map<List<GetOperationDto>>(x)
            })
            .ToList();
    }

    public async Task<GetOperationDetailsDto> GetDetailsForUserAsync(Guid
userId, Guid id)
    {
        var foundOperation = await Context.Operations.FirstAsync(x =>
x.Id.Equals(id));

        await EnsureCorrectAccessCounterparty(userId, foundOperation);

        var operationDetails =
Mapper.Map<GetOperationDetailsDto>(foundOperation);

        operationDetails.FromCounterparty =
Mapper.Map<GetCounterpartyDto>(foundOperation.FromCounterparty);
        operationDetails.ToCounterparty =
Mapper.Map<GetCounterpartyDto>(foundOperation.ToCounterparty);
        operationDetails.Category =
Mapper.Map<GetCategoryDto>(foundOperation.Category);
        operationDetails.Subjects =
Mapper.Map<List<GetSubjectDto>>(foundOperation.Subjects);

        return operationDetails;
    }

    public async Task DeleteForUserAsync(Guid userId, Guid id)
    {
        var foundOperation = await Context.Operations.FirstAsync(x =>
x.Id.Equals(id));

        await EnsureCorrectAccessCounterparty(userId, foundOperation);

        Context.Operations.Remove(foundOperation);

        await Context.SaveChangesAsync();
    }

    private async Task EnsureCorrectAccessCounterparty(Guid userId, Operation
operation)
    {
        switch (operation.Type)
        {
            case OperationType.Spending:
            {
                var fromCounterparty = await Context.Accounts.FirstAsync(x =>
x.UserId.Equals(userId)

```

```

        && x.Id.Equals(operation.FromCounterpartyId));

        var toCounterparty = await Context.Places.FirstAsync(x =>
            x.UserId.Equals(userId)
            && x.Id.Equals(operation.ToCounterpartyId));
    }
    break;

case OperationType.Income:
{
    var fromCounterparty = await Context.Places.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(operation.FromCounterpartyId));

    var toCounterparty = await Context.Accounts.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(operation.ToCounterpartyId));
}
    break;

case OperationType.Transfer:
{
    var fromCounterparty = await Context.Accounts.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(operation.FromCounterpartyId));

    var toCounterparty = await Context.Accounts.FirstAsync(x =>
        x.UserId.Equals(userId)
        && x.Id.Equals(operation.ToCounterpartyId));
}
    break;

//case OperationType.Debt:
//    break;

default:
    throw new ArgumentOutOfRangeException(nameof(OperationType));
}
}

private static void UpdateName(Operation operation)
{
    operation.Name = "Нет категории";

    foreach (var subject in operation.Subjects) subject.Name = "Нет
категории";
}
}

```

Листинг В.14 – Файл MsSqlContext.cs

```

namespace FinanceManager.DataAccess.Context;

public class MsSqlContext : DbContext
{
    public MsSqlContext(DbContextOptions<MsSqlContext> options) :
base(options)
    {
    }

    internal DbSet<Account> Accounts { get; set; } = null!;
}

```

```

internal DbSet<Category>          Categories          { get; set; } = null!;
internal DbSet<Counterparty>     Counterparties      { get; set; } = null!;
internal DbSet<Currency>         Currencies          { get; set; } = null!;
internal DbSet<Operation>        Operations          { get; set; } = null!;
internal DbSet<Place>            Places              { get; set; } = null!;
internal DbSet<Subject>          Subjects            { get; set; } = null!;
internal DbSet<User>             Users               { get; set; } = null!;
internal DbSet<Plan>             Plans               { get; set; } = null!;
internal DbSet<PlanByCategory>   PlansByCategories   { get; set; } = null!;

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Category>()
        .HasMany(p => p.PlanByCategories)
        .WithOne(t => t.Category)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<Plan>()
        .HasMany(p => p.PlanByCategories)
        .WithOne(t => t.Plan)
        .OnDelete(DeleteBehavior.NoAction);

    modelBuilder.Entity<Currency>().HasData(
        new Currency[]
        {
            new() { Id = Guid.NewGuid(), Name = "Российский рубль",
Symbol = "₽", CodeIso4217 = "RUB" },
            new() { Id = Guid.NewGuid(), Name = "Доллар США", Symbol
= "$", CodeIso4217 = "USD" },
            new() { Id = Guid.NewGuid(), Name = "Евро", Symbol
= "€", CodeIso4217 = "EUR" },
            new() { Id = Guid.NewGuid(), Name = "Китайский юань", Symbol
= "¥", CodeIso4217 = "CNY" },
            new() { Id = Guid.NewGuid(), Name = "Фунт стерлингов", Symbol
= "£", CodeIso4217 = "GBP" },
            new() { Id = Guid.NewGuid(), Name = "Иена", Symbol
= "¥", CodeIso4217 = "JPY" }
        });
}
}

```

Приложение Г
ИСХОДНЫЙ КОД МОБИЛЬНОГО ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг Г.1 – Файл MauiProgram.cs

```

using CommunityToolkit.Maui;

using FinanceManager.Profiles;
using FinanceManager.Services;
using FinanceManager.ViewModels;
using FinanceManager.Views;

namespace FinanceManager;

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .UseMauiCommunityToolkit()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            });

        builder.Services.AddAutoMapper(options =>
        {
            options.AddProfile<AccountMapperProfile>();
            options.AddProfile<CategoryMapperProfile>();
            options.AddProfile<PlaceMapperProfile>();
        });

        builder.Services.AddSingleton<IConnectivity>(Connectivity.Current);
        builder.Services.AddSingleton<IAuthService, AuthService>();
        builder.Services.AddSingleton<ICurrencyService, CurrencyService>();
        builder.Services.AddSingleton<IAccountService, AccountService>();
        builder.Services.AddSingleton<IOperationService, OperationService>();
        builder.Services.AddSingleton<ICategoryService, CategoryService>();
        builder.Services.AddSingleton<IPlaceService, PlaceService>();

        builder.Services.AddSingleton<IEmptyPageViewModel,
EmptyPageViewModel>();

        builder.Services.AddSingleton<IAuthorizationPageViewModel,
AuthorizationPageViewModel>();
        builder.Services.AddSingleton<IRegistrationPageViewModel,
RegistrationPageViewModel>();

        builder.Services.AddSingleton<IAccountsPageViewModel,
AccountsPageViewModel>();
        builder.Services.AddSingleton<IAccountCreateOrUpdatePageViewModel,
AccountCreateOrUpdatePageViewModel>();
        builder.Services.AddSingleton<IAccountDetailsPageViewModel,
AccountDetailsPageViewModel>();

        builder.Services.AddSingleton<IOperationsPageViewModel,
OperationsPageViewModel>();
        builder.Services.AddSingleton<ICreateOperationPanelViewModel,
CreateOperationPanelViewModel>();
        builder.Services.AddSingleton<IOperationCreateOrUpdatePageViewModel,
OperationCreateOrUpdatePageViewModel>();
    }
}

```

```

        builder.Services.AddSingleton<IOperationDetailsPageViewModel,
        OperationDetailsPageViewModel>();

        builder.Services.AddSingleton<IAalyticsPageViewModel,
        AnalyticsPageViewModel>();

        builder.Services.AddSingleton<IOthersPageViewModel,
        OthersPageViewModel>();
        builder.Services.AddSingleton<IUserInfoPageViewModel,
        UserInfoPageViewModel>();

        builder.Services.AddSingleton<ICategoriesPageViewModel,
        CategoriesPageViewModel>();
        builder.Services.AddSingleton<ICategoryCreateOrUpdatePageViewModel,
        CategoryCreateOrUpdatePageViewModel>();
        builder.Services.AddSingleton<ICategoryDetailsPageViewModel,
        CategoryDetailsPageViewModel>();

        builder.Services.AddSingleton<IPlacesPageViewModel,
        PlacesPageViewModel>();
        builder.Services.AddSingleton<IPlaceCreateOrUpdatePageViewModel,
        PlaceCreateOrUpdatePageViewModel>();

        builder.Services.AddTransient<EmptyPage>();

        builder.Services.AddTransient<AuthorizationPage>();
        builder.Services.AddTransient<RegistrationPage>();

        builder.Services.AddTransient<AccountsPage>();
        builder.Services.AddTransient<AccountCreateOrUpdatePage>();
        builder.Services.AddTransient<AccountDetailsPage>();

        builder.Services.AddTransient<OperationsPage>();
        builder.Services.AddTransient<OperationSpendingCreateOrUpdatePage>();
        builder.Services.AddTransient<OperationIncomeCreateOrUpdatePage>();
        builder.Services.AddTransient<OperationTransferCreateOrUpdatePage>();
        builder.Services.AddTransient<OperationDetailsPage>();

        builder.Services.AddTransient<AnalyticsPage>();

        builder.Services.AddTransient<OthersPage>();
        builder.Services.AddTransient<UserInfoPage>();

        builder.Services.AddTransient<CategoriesPage>();
        builder.Services.AddTransient<CategoryCreateOrUpdatePage>();
        builder.Services.AddTransient<CategoryDetailsPage>();

        builder.Services.AddTransient<PlacesPage>();
        builder.Services.AddTransient<PlaceCreateOrUpdatePage>();

        return builder.Build();
    }
}

```

Листинг Г.2 – Файл App.xaml.cs

```

namespace FinanceManager;

public record AuthData(string Login, string Token);

public partial class App : Application

```

```

{
    private readonly IAuthService m_Service;

    private bool m_IsBusy;

    public static AuthData? CurrentAuthData { get; private set; }

    public App(IAuthService service)
    {
        InitializeComponent();

        m_Service = service;

        MainPage = new AppShell();
    }

    public static void SetNewAuthData(AuthData? authData)
    {
        CurrentAuthData = authData;

        string? AuthDataString = CurrentAuthData is { }
            ? JsonSerializer.Serialize(CurrentAuthData)
            : null;

        Preferences.Set(nameof(CurrentAuthData), AuthDataString);
    }

    protected override void OnStart()
    {
        string? AuthDataString = Preferences.Get(nameof(CurrentAuthData),
null);

        if (AuthDataString is {}) CurrentAuthData =
JsonSerializer.Deserialize<AuthData>(AuthDataString);

        CheckAuthorize();
    }

    protected override void OnSleep()
    {
        string? AuthDataString = CurrentAuthData is { }
            ? JsonSerializer.Serialize(CurrentAuthData)
            : null;

        Preferences.Set(nameof(CurrentAuthData), AuthDataString);
    }

    protected override void OnResume()
    {
        string? AuthDataString = Preferences.Get(nameof(CurrentAuthData),
null);

        if (AuthDataString is {}) CurrentAuthData =
JsonSerializer.Deserialize<AuthData>(AuthDataString);

        CheckAuthorize();
    }

    private async void CheckAuthorize()
    {

```

```

        if (m_IsBusy) return;

        m_IsBusy = true;

        bool isTryContinue = true;

        try
        {
            while (isTryContinue)
            {
                try
                {
                    if (!m_Service.CheckNetworkAccess())
                    {
                        await AlertHelper.DisplayNotNetworkAccess();
                    }

                    bool isAuthorize = await m_Service.CheckAuthorizeAsync();

                    isTryContinue = false;

                    if (isAuthorize)
                    {
                        await Shell.Current.GoToAsync($"//AccountsPage");
                    }
                    else
                    {
                        await
Shell.Current.GoToAsync($"//AuthorizationPage");
                    }
                }
                catch (Exception ex)
                {
                    await AlertHelper.DisplayException(ex);
                }
            }
        }
        finally
        {
            m_IsBusy = false;
        }
    }
}

```

Листинг Г.3 – Файл AppShell.xaml

```

<?xml version="1.0" encoding="UTF-8"?>

<Shell
    x:Class="FinanceManager.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:FinanceManager"
    xmlns:view="clr-namespace:FinanceManager.Views"
    Shell.FlyoutBehavior="Disabled">

    <ShellContent Route="EmptyPage"
        ContentTemplate="{DataTemplate view:EmptyPage}" />

    <TabBar>
        <ShellContent Title="Счета"
            Icon="accounts.png"
            Route="AccountsPage"

```

```

        ContentTemplate="{DataTemplate view:AccountsPage}" />
<ShellContent Title="Операции"
    Icon="operations.png"
    Route="OperationsPage"
    ContentTemplate="{DataTemplate view:OperationsPage}" />
<ShellContent Title="Аналитика"
    Icon="analytics.png"
    Route="AnalyticsPage"
    ContentTemplate="{DataTemplate view:AnalyticsPage}" />
<ShellContent Title="Другое"
    Icon="others.png"
    Route="OthersPage"
    ContentTemplate="{DataTemplate view:OthersPage}" />
</TabBar>

<ShellContent Route="AuthorizationPage"
    ContentTemplate="{DataTemplate view:AuthorizationPage}" />
</Shell>

```

Листинг Г.4 – Файл AppShell.xaml.cs

```

using System.Diagnostics;

using FinanceManager.Views;

namespace FinanceManager;

public partial class AppShell : Shell
{
    public AppShell()
    {
        InitializeComponent();

        Routing.RegisterRoute(nameof(RegistrationPage),
            typeof(RegistrationPage));

        Routing.RegisterRoute(nameof(AccountCreateOrUpdatePage),
            typeof(AccountCreateOrUpdatePage));
        Routing.RegisterRoute(nameof(AccountDetailsPage),
            typeof(AccountDetailsPage));

        Routing.RegisterRoute(nameof(OperationSpendingCreateOrUpdatePage),
            typeof(OperationSpendingCreateOrUpdatePage));
        Routing.RegisterRoute(nameof(OperationIncomeCreateOrUpdatePage),
            typeof(OperationIncomeCreateOrUpdatePage));
        Routing.RegisterRoute(nameof(OperationTransferCreateOrUpdatePage),
            typeof(OperationTransferCreateOrUpdatePage));
        Routing.RegisterRoute(nameof(OperationDetailsPage),
            typeof(OperationDetailsPage));

        Routing.RegisterRoute(nameof(UserInfoPage), typeof(UserInfoPage));

        Routing.RegisterRoute(nameof(CategoriesPage),
            typeof(CategoriesPage));
        Routing.RegisterRoute(nameof(CategoryCreateOrUpdatePage),
            typeof(CategoryCreateOrUpdatePage));
        Routing.RegisterRoute(nameof(CategoryDetailsPage),
            typeof(CategoryDetailsPage));

        Routing.RegisterRoute(nameof(PlacesPage),
            typeof(PlacesPage));
    }
}

```

```

        Routing.RegisterRoute(nameof(PlaceCreateOrUpdatePage),
        typeof(PlaceCreateOrUpdatePage));
    }

    protected override void OnNavigating(ShellNavigatingEventArgs args)
    {
        Debug.WriteLine($"Source:{args.Source} '{args.Current?.Location}' =>
        '{args.Target?.Location}'");

        base.OnNavigating(args);
    }
}

```

Листинг Г.5 – Файл AccountsPage.xaml

```

<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:behaviors="clr-namespace:FinanceManager.Behaviors"
    xmlns:viewmodel="clr-namespace:FinanceManager.ViewModels"
    xmlns:model="clr-namespace:FinanceManager.Common.Dtos"
    xmlns:controls="clr-namespace:FinanceManager.Views.Controls"
    x:Class="FinanceManager.Views.AccountsPage"
    x:DataType="viewmodel:AccountsPageViewModel"
    Title="{Binding Title}">
    <ContentPage.Behaviors>
        <behaviors:ContentPageCommandBehavior NavigatedToCommand="{Binding
        ReloadAccountsCommand}" />
    </ContentPage.Behaviors>

    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Добавить"
            IconImageSource="add.png"
            Command="{Binding GoToAccountCreateCommand}" />
    </ContentPage.ToolbarItems>

    <Shell.TitleView>
        <Grid RowDefinitions="Auto,Auto"
            VerticalOptions="Center">
            <controls:FinanceLabel Style="{StaticResource WhiteMediumLabel}"
                Value="{Binding Info.Balance}"
                CurrencySymbol="{Binding
        Info.CurrencySymbol}" />
            <Label Grid.Row="1"
                Text="Баланс"
                Style="{StaticResource WhiteSmallLabel}" />
        </Grid>
    </Shell.TitleView>

    <Grid>
        <CollectionView ItemsSource="{Binding Accounts}"
            SelectionMode="None"
            EmptyView="Счетов нет">
            <CollectionView.ItemTemplate>
                <DataTemplate x:DataType="model:GetAccountDto">
                    <Grid Padding="10,10,10,0">
                        <Grid.GestureRecognizers>
                            <TapGestureRecognizer
                                Command="{Binding GoToAccountDetailsCommand,
        Source={RelativeSource AncestorType={x:Type
        viewmodel:AccountsPageViewModel}}}"

```

```

        CommandParameter="{Binding .}" />
</Grid.GestureRecognizers>
<Grid Padding="5" HeightRequest="50">
    <Label VerticalOptions="Center"
        Text="{Binding Name}" />
    <controls:FinanceLabel VerticalOptions="Center"
        HorizontalOptions="End"
        Value="{Binding
CurrentBalance}"
        CurrencySymbol="{Binding
CurrencySymbol}" />
</Grid>
</Grid>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>

<ActivityIndicator HorizontalOptions="FillAndExpand"
    VerticalOptions="CenterAndExpand"
    IsRunning="{Binding IsBusy}"
    IsVisible="{Binding IsBusy}" />
</Grid>
</ContentPage>

```

Листинг Г.6 – Файл AccountsPageViewModel.cs

```

namespace FinanceManager.ViewModels;

internal partial class AccountsPageViewModel : BaseViewModel,
IAccountsPageViewModel
{
    private readonly IAccountService m_Service;
    private readonly ICurrencyService m_CurrencyService;

    [ObservableProperty]
    private GetAccountsInfoDto? m_Info;

    public ObservableCollection<GetAccountDto> Accounts { get; } = new();

    public AccountsPageViewModel(IAccountService service, ICurrencyService
currencyService)
    {
        m_Service = service;
        m_CurrencyService = currencyService;

        Title = "Счета";
    }

    [ICommand]
    private async Task ReloadAccounts()
    {
        if (IsBusy) return;

        IsBusy = true;

        try
        {
            if (!m_Service.CheckNetworkAccess())

```

```

        {
            await AlertHelper.DisplayNotNetworkAccess();
            return;
        }

        if (!CurrencyTypes.IsInitialize)
        {
            var currencies = await m_CurrencyService.GetAllAsync();

            CurrencyTypes.Initialize(currencies);
        }

        var accounts = await m_Service.GetAllAsync();
        var info      = await m_Service.GetInfoAsync();

        Accounts.Clear();

        foreach (var account in accounts) Accounts.Add(account);

        Info = info;
    }
    catch (Exception ex)
    {
        await AlertHelper.DisplayException(ex);
    }
    finally
    {
        IsBusy = false;
    }
}

[ICommand]
private async Task GoToAccountCreate()
{
    if (IsBusy) return;

    IsBusy = true;

    try
    {
        var accountCreate = new CreateAccountDto
        {
            CurrencyId = CurrencyTypes.Default.Id
        };

        await Shell.Current.GoToAsync("AccountCreateOrUpdatePage", true,
new Dictionary<string, object>
        {
            ["IsCreate"]      = true,
            ["AccountCreate"] = accountCreate
        });
    }
    finally
    {
        IsBusy = false;
    }
}

[ICommand]
private async Task GoToAccountDetails(GetAccountDto? account)
{
    if (account is null) return;

    if (IsBusy) return;

```

```

        IsBusy = true;

        try
        {
            await Shell.Current.GoToAsync("AccountDetailsPage", true, new
Dictionary<string, object>
            {
                { "Account", account }
            });
        }
        finally
        {
            IsBusy = false;
        }
    }
}

```

Листинг Г.7 – Файл AccountService.cs

```

namespace FinanceManager.Services;

internal class AccountService : BaseHttpService, IAccountService
{
    private const string c_AddPath          = "api/account";
    private const string c_EditPath         = "api/account";
    private const string c_GetAllPath       = "api/account";
    private const string c_GetDetailsPath  = "api/account/details/";
    private const string c_GetInfoPath     = "api/account/info";
    private const string c_DeletePath      = "api/account/";

    public AccountService(IConnectivity connectivity) : base(connectivity)
    {
    }

    public async Task CreateAsync(CreateAccountDto createAccountDto)
    {
        ReinitializeAuthorization();

        var response = await Client.PostAsJsonAsync(new Uri(BaseUrl,
c_AddPath), createAccountDto);

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);
    }

    public async Task UpdateAsync(UpdateAccountDto updateAccountDto)
    {
        ReinitializeAuthorization();

        var response = await Client.PutAsJsonAsync(new Uri(BaseUrl,
c_EditPath), updateAccountDto);

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);
    }

    public async Task<List<GetAccountDto>> GetAllAsync()
    {
        ReinitializeAuthorization();
    }
}

```

```

        var response = await Client.GetAsync(new Uri(BaseUrl, c_GetAllPath));

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);

        var data = await
response.Content.ReadFromJsonAsync<List<GetAccountDto>>();

        return data ?? new();
    }

    public async Task<GetAccountDetailsDto> GetDetailsAsync(Guid id)
    {
        ReinitializeAuthorization();

        var response = await Client.GetAsync(new Uri(BaseUrl,
$"{c_GetDetailsPath}{id}"));

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);

        var data = await
response.Content.ReadFromJsonAsync<GetAccountDetailsDto>();

        return data ?? new();
    }

    public async Task<GetAccountsInfoDto> GetInfoAsync()
    {
        ReinitializeAuthorization();

        var response = await Client.GetAsync(new Uri(BaseUrl,
c_GetInfoPath));

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);

        var data = await
response.Content.ReadFromJsonAsync<GetAccountsInfoDto>();

        return data ?? new();
    }

    public async Task DeleteAsync(Guid id)
    {
        ReinitializeAuthorization();

        var response = await Client.DeleteAsync(new Uri(BaseUrl,
$"{c_DeletePath}{id}"));

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);
    }
}

```

Листинг Г.8 – Файл BaseHttpService.cs

```

using System.Text.Json.Serialization;

namespace FinanceManager.Services;

```

```

internal abstract class BaseHttpService : IHttpService
{
    protected static readonly Uri HttpUrl =
new("http://192.168.0.100:5212");
    protected static readonly Uri HttpsUrl =
new("https://192.168.0.100:7212");

    protected readonly IConnectivity Connectivity;

    protected readonly HttpClientHandler ClientHandler;
    protected readonly HttpClient Client;

    public virtual Uri BaseUrl => HttpUrl;

    protected BaseHttpService(IConnectivity connectivity)
    {
        Connectivity = connectivity;

        ClientHandler = new HttpClientHandler();
        Client = new(ClientHandler);

        Client.Timeout = TimeSpan.FromSeconds(15);

        ReinitializeAuthorization();
    }

    public bool CheckNetworkAccess()
    {
        return Connectivity.NetworkAccess == NetworkAccess.Internet;
    }

    protected void ReinitializeAuthorization()
    {
        const string Name = "Authorization";

        string value = $"Bearer {App.CurrentAuthData?.Token ??
string.Empty}";

        Client.DefaultRequestHeaders.Remove(Name);
        Client.DefaultRequestHeaders.Add(Name, value);
    }

    protected async Task ThrowErrorsResponse(HttpResponseMessage response)
    {
        try
        {
            var errorMessage = await
response.Content.ReadFromJsonAsync<ErrorsMessage>(new
JsonSerializerOptions(JsonSerializerDefaults.Web)
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });

            switch (errorMessage)
            {
                case { Errors: { } }:
                {
                    string errorsString = string.Join(Environment.NewLine,
errorsMessage.Errors.Select(x =>
string.Join(Environment.NewLine, x.Value)));

                    throw new HttpRequestException(errorsString);
                }
            }
        }
    }
}

```

```

    }

    case { Title: { } }:
        throw new HttpRequestException(errorsMessage.Title);

    default:
        throw new HttpRequestException("Что-то пошло не так");
    }
}
catch (HttpRequestException)
{
    throw;
}
catch (Exception ex)
{
    throw new HttpRequestException("Что-то пошло не так", ex);
}
}

private class ErrorMessage
{
    public string? Title { get; set; }
    public Dictionary<string, string[]>? Errors { get; set; }
}
}

```

Листинг Г.9 – Файл ContentPageCommandBehavior.cs

```

using System.Windows.Input;

namespace FinanceManager.Behaviors;

internal class ContentPageCommandBehavior : Behavior<ContentPage>
{
    public static BindableProperty AppearingCommandProperty =
        BindableProperty.Create(nameof(AppearingCommand), typeof(ICommand),
        typeof(ContentPageCommandBehavior));

    public static BindableProperty DisappearingCommandProperty =
        BindableProperty.Create(nameof(DisappearingCommand),
        typeof(ICommand), typeof(ContentPageCommandBehavior));

    public static BindableProperty LoadedCommandProperty =
        BindableProperty.Create(nameof(LoadedCommand), typeof(ICommand),
        typeof(ContentPageCommandBehavior));

    public static BindableProperty UnloadedCommandProperty =
        BindableProperty.Create(nameof(UnloadedCommand), typeof(ICommand),
        typeof(ContentPageCommandBehavior));

    public static BindableProperty NavigatedFromCommandProperty =
        BindableProperty.Create(nameof(NavigatedFromCommand),
        typeof(ICommand), typeof(ContentPageCommandBehavior));

    public static BindableProperty NavigatedToCommandProperty =
        BindableProperty.Create(nameof(NavigatedToCommand), typeof(ICommand),
        typeof(ContentPageCommandBehavior));

    public static BindableProperty NavigatingFromCommandProperty =

```

```
BindableProperty.Create(nameof(NavigatingFromCommand),
typeof(ICommand), typeof(ContentPageCommandBehavior));
```

```
public ICommand AppearingCommand
{
    get => (ICommand)GetValue(AppearingCommandProperty);
    set => SetValue(AppearingCommandProperty, value);
}

public ICommand DisappearingCommand
{
    get => (ICommand)GetValue(DisappearingCommandProperty);
    set => SetValue(DisappearingCommandProperty, value);
}

public ICommand LoadedCommand
{
    get => (ICommand)GetValue(LoadedCommandProperty);
    set => SetValue(LoadedCommandProperty, value);
}

public ICommand UnloadedCommand
{
    get => (ICommand)GetValue(UnloadedCommandProperty);
    set => SetValue(UnloadedCommandProperty, value);
}

public ICommand NavigatedFromCommand
{
    get => (ICommand)GetValue(NavigatedFromCommandProperty);
    set => SetValue(NavigatedFromCommandProperty, value);
}

public ICommand NavigatedToCommand
{
    get => (ICommand)GetValue(NavigatedToCommandProperty);
    set => SetValue(NavigatedToCommandProperty, value);
}

public ICommand NavigatingFromCommand
{
    get => (ICommand)GetValue(NavigatingFromCommandProperty);
    set => SetValue(NavigatingFromCommandProperty, value);
}

protected override void OnAttachedTo(ContentPage bindable)
{
    base.OnAttachedTo(bindable);

    bindable.BindingContextChanged += BindingContextChangedHandler;
    bindable.Appearing += ContentPageAppearingHandler;
    bindable.Disappearing += ContentPageDisappearingHandler;
    bindable.Loaded += ContentPageLoadedHandler;
    bindable.Unloaded += ContentPageUnloadedHandler;
    bindable.NavigatedFrom += ContentPageNavigatedFromHandler;
    bindable.NavigatedTo += ContentPageNavigatedToHandler;
    bindable.NavigatingFrom += ContentPageNavigatingFromHandler;
}

protected override void OnDetachingFrom(ContentPage bindable)
{
    base.OnDetachingFrom(bindable);
```

```

        bindable.BindingContextChanged -= BindingContextChangedHandler;
        bindable.Appearing -= ContentPageAppearingHandler;
        bindable.Disappearing -= ContentPageDisappearingHandler;
        bindable.Loaded -= ContentPageLoadedHandler;
        bindable.Unloaded -= ContentPageUnloadedHandler;
        bindable.NavigatedFrom -= ContentPageNavigatedFromHandler;
        bindable.NavigatedTo -= ContentPageNavigatedToHandler;
        bindable.NavigatingFrom -= ContentPageNavigatingFromHandler;
    }

    private void BindingContextChangedHandler(object? sender, EventArgs e)
    {
        BindingContext = (sender as ContentPage)?.BindingContext;
    }

    private void ContentPageAppearingHandler(object? sender, EventArgs e)
    {
        CommandHelper.Execute(AppearingCommand);
    }

    private void ContentPageDisappearingHandler(object? sender, EventArgs e)
    {
        CommandHelper.Execute(DisappearingCommand);
    }

    private void ContentPageLoadedHandler(object? sender, EventArgs e)
    {
        CommandHelper.Execute(LoadedCommand);
    }

    private void ContentPageUnloadedHandler(object? sender, EventArgs e)
    {
        CommandHelper.Execute(UnloadedCommand);
    }

    private void ContentPageNavigatedFromHandler(object? sender, EventArgs e)
    {
        CommandHelper.Execute(NavigatedFromCommand);
    }

    private void ContentPageNavigatedToHandler(object? sender, EventArgs e)
    {
        CommandHelper.Execute(NavigatedToCommand);
    }

    private void ContentPageNavigatingFromHandler(object? sender, EventArgs
e)
    {
        CommandHelper.Execute(NavigatingFromCommand);
    }
}

```

Листинг Г.10 – Файл AccountDetailsPage.xaml

```

<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:behaviors="clr-namespace:FinanceManager.Behaviors"
    xmlns:viewmodel="clr-namespace:FinanceManager.ViewModels"
    xmlns:model="clr-namespace:FinanceManager.Common.Dtos"
    xmlns:controls="clr-namespace:FinanceManager.Views.Controls"
    x:Class="FinanceManager.Views.AccountDetailsPage"

```

```

        x:DataType="viewModel:AccountDetailsPageViewModel"
        Title="{Binding Title}"
        Shell.TabBarIsVisible="False">
    <ContentPage.Behaviors>
        <behaviors:ContentPageCommandBehavior NavigatedToCommand="{Binding
ReloadDetailsCommand}" />
    </ContentPage.Behaviors>

    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Изменить"
            IconImageSource="edit.png"
            Command="{Binding GoToAccountUpdateCommand}" />
        <ToolbarItem Text="Удалить"
            IconImageSource="delete.png"
            Command="{Binding DeleteAccountCommand}" />
    </ContentPage.ToolbarItems>

    <Grid VerticalOptions="FillAndExpand">
        <Grid ColumnDefinitions="*,*"
            ColumnSpacing="5"
            RowDefinitions="Auto,Auto,Auto,Auto,*,Auto"
            RowSpacing="20"
            Padding="30">
            <Label Text="Название" />
            <Label Grid.Row="1"
                Text="Тип счета" />
            <Label Grid.Row="2"
                Text="Текущий баланс" />
            <Label Grid.Row="3"
                Text="Номер" />
            <Label Grid.Row="4"
                Text="Описание" />
            <Label Grid.Column="1"
                Text="{Binding AccountView.Name}" />
            <Label Grid.Column="1"
                Grid.Row="1"
                Text="{Binding AccountView.Type, Converter={StaticResource
AccountTypeValueToTitleConverter}}" />
            <controls:FinanceLabel Grid.Column="1"
                Grid.Row="2"
                Value="{Binding
AccountView.CurrentBalance}"
                CurrencySymbol="{Binding
AccountView.CurrencySymbol}" />
            <Label Grid.Column="1"
                Grid.Row="3" Text="{Binding AccountView.AccountNumber}" />
            <ScrollView Grid.Column="1"
                Grid.Row="4">
                <Label Text="{Binding AccountView.Description}"/>
            </ScrollView>
        </Grid>

        <ActivityIndicator HorizontalOptions="FillAndExpand"
            VerticalOptions="CenterAndExpand"
            IsRunning="{Binding IsBusy}"
            IsVisible="{Binding IsBusy}" />
    </Grid>
</ContentPage>

```

Листинг Г.11 – Файл AccountDetailsPageViewModel.cs

```
using AutoMapper;
```

```

namespace FinanceManager.ViewModels;

[QueryProperty(nameof(Account), "Account")]
internal partial class AccountDetailsPageViewModel : BaseViewModel,
IAccountDetailsPageViewModel
{
    private readonly IAccountService m_Service;
    private readonly IMapper m_Mapper;

    [ObservableProperty]
    private GetAccountDto m_Account = null!;

    [ObservableProperty]
    private GetAccountDetailsDto m_AccountView = null!;

    public AccountDetailsPageViewModel(IAccountService service, IMapper
mapper)
    {
        m_Service = service;
        m_Mapper = mapper;

        Title = "Информация о счете";
    }

    [ICommand]
    private async Task ReloadDetails()
    {
        if (IsBusy) return;

        IsBusy = true;

        try
        {
            if (!m_Service.CheckNetworkAccess())
            {
                await AlertHelper.DisplayNotNetworkAccess();
                return;
            }

            var accountDetails = await
m_Service.GetDetailsAsync(m_Account.Id);

            AccountView = accountDetails;
        }
        catch (Exception ex)
        {
            await AlertHelper.DisplayException(ex);
        }
        finally
        {
            IsBusy = false;
        }
    }

    [ICommand]
    private async Task GoToAccountUpdate()
    {
        if (IsBusy) return;

        IsBusy = true;
    }
}

```

```

        try
        {
            var accountUpdate =
m_Mapper.Map<UpdateAccountDto>(m_AccountView);

            await Shell.Current.GoToAsync("AccountCreateOrUpdatePage", true,
new Dictionary<string, object>
            {
                ["IsCreate"]      = false,
                ["AccountUpdate"] = accountUpdate
            });
        }
        finally
        {
            IsBusy = false;
        }
    }

    [ICommand]
    private async Task GoToOperationDetails(GetOperationDto? operation)
    {
        if (operation is null) return;

        if (IsBusy) return;

        IsBusy = true;

        try
        {
            await Shell.Current.GoToAsync("OperationDetailsPage", true, new
Dictionary<string, object>
            {
                { "Operation", operation }
            });
        }
        finally
        {
            IsBusy = false;
        }
    }

    [ICommand]
    private async Task DeleteAccount()
    {
        if (!await AlertHelper.DisplayDeleteQuestion("Удалить счет?"))
return;

        if (IsBusy) return;

        IsBusy = true;

        try
        {
            if (!m_Service.CheckNetworkAccess())
            {
                await AlertHelper.DisplayNotNetworkAccess();
                return;
            }

            await m_Service.DeleteAsync(m_AccountView.Id);

            await Shell.Current.GoToAsync("../", true);
        }
        catch (Exception ex)

```

```

        {
            await AlertHelper.DisplayException(ex);
        }
        finally
        {
            IsBusy = false;
        }
    }

    partial void OnAccountChanged(GetAccountDto value)
    {
        AccountView = m_Mapper.Map<GetAccountDetailsDto>(value);
    }
}

```

Листинг Г.14 – Файл AuthService.cs

```

namespace FinanceManager.Services;

internal class AuthService : BaseHttpService, IAuthService
{
    private const string c_RegisterPath = "api/User/register";
    private const string c_AuthorizePath = "api/user/authorize";
    private const string c_GetNamePath = "api/user/name";

    public AuthService(IConnectivity connectivity) : base(connectivity)
    {
    }

    public async Task RegisterAsync(string? login, string? password)
    {
        var data = new CreateUserDto { Login = login!, Password = password! };

        var response = await Client.PostAsJsonAsync(new Uri(BaseUrl,
c_RegisterPath), data);

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);
    }

    public async Task AuthorizeAsync(string? login, string? password)
    {
        var data = new FindUserDto { Login = login!, Password = password! };

        var response = await Client.PostAsJsonAsync(new Uri(BaseUrl,
c_AuthorizePath), data);

        if (!response.IsSuccessStatusCode) await
ThrowErrorsResponse(response);

        string token = await response.Content.ReadAsStringAsync();

        App.SetNewAuthData(new AuthData(login!, token));
    }

    public async Task<bool> CheckAuthorizeAsync()
    {
        if (App.CurrentAuthData is null) return false;
    }
}

```

```
        ReinitializeAuthorization();

        var response = await Client.GetAsync(new Uri(BaseUrl,
c_GetNamePath));

        if (!response.IsSuccessStatusCode) return false;

        var data = await response.Content.ReadFromJsonAsync<GetUserDto>();

        return string.Equals(App.CurrentAuthData.Login, data?.Login);
    }
}
```

Приложение Д
ИСХОДНЫЙ КОД ФУНКЦИЙ ТЕСТИРОВАНИЯ
(рекомендуемое)

Листинг Д.1 – Файл UserControllerTests.cs

```

using System.ComponentModel.DataAnnotations;

#pragma warning disable CS8625

namespace FinanceManager.Unit.Controllers;

[TestClass]
public class UserControllerTests
{
    public static IEnumerable<object[]> GetRegistrationSets()
    {
        yield return new object[] { false, false, new CreateUserDto() };
        yield return new object[] { false, true, new CreateUserDto() };
        yield return new object[] { true, false, new CreateUserDto { Login =
"login", Password = "password" } };
        yield return new object[] { true, true, new CreateUserDto { Login =
"login", Password = "password" } };
        yield return new object[] { false, false, new CreateUserDto { Login =
"1", Password = "p" } };
        yield return new object[] { false, false, new CreateUserDto { Login =
"login", Password = "p" } };
        yield return new object[] { false, false, new CreateUserDto { Login =
"1", Password = "password" } };
    }

    [DataTestMethod]
    [DynamicData(nameof(GetRegistrationSets), DynamicDataSourceType.Method)]
    public async Task Registration(bool isValid, bool isContains,
CreateUserDto data)
    {
        // Arrange
        var mockLogger = new Mock<ILogger<UserController>>();
        var mockRepository = new Mock<IUserRepository>();
        mockRepository.Setup(x => x.ContainsAsync(data.Login!))
            .ReturnsAsync(isContains);
        mockRepository.Setup(x => x.CreateAsync(data!))
            .Callback<CreateUserDto>(x => { });
        var controller = new UserController(mockLogger.Object,
mockRepository.Object);

        // Act
        bool resultValid = ValidationTestsHelper.IsValidObject(data);

        // Assert
        Assert.AreEqual(isValid, resultValid);

        if (!isValid) return;

        // Act
        var result = await controller.Register(data);

        // Assert
        if (!isContains)
        {
            Assert.IsInstanceOfType(result, typeof(StatusCodeResult));
            Assert.AreEqual(200, ((StatusCodeResult)result).StatusCode);
        }
        else
    }

```

```

        {
            Assert.IsInstanceOfType(result, typeof(ObjectResult));
            Assert.AreEqual(400, ((ObjectResult)result).StatusCode);
        }
    }

    public static IEnumerable<object[]> GetAuthorizationSets()
    {
        yield return new object[] { false, false, new FindUserDto() };
        yield return new object[] { false, true, new FindUserDto() };
        yield return new object[] { true, false, new FindUserDto { Login =
"login", Password = "password" } };
        yield return new object[] { true, true, new FindUserDto { Login =
"login", Password = "password" } };
    }

    [DataTestMethod]
    [DynamicData(nameof(GetAuthorizationSets), DynamicDataSourceType.Method)]
    public async Task Authorization(bool isValid, bool isFound, FindUserDto
data)
    {
        // Arrange
        var mockLogger = new Mock<ILogger<UserController>>();
        var mockRepository = new Mock<IUserRepository>();
        mockRepository.Setup(x => x.FindAsync(data))
            .ReturnsAsync(isFound ? new FoundUserDto { Id = new Guid(), Login
= data.Login } : null);
        var controller = new UserController(mockLogger.Object,
mockRepository.Object);

        // Act
        bool resultValid = ValidationTestsHelper.IsValidObject(data);

        // Assert
        Assert.AreEqual(isValid, resultValid);

        if (!isValid) return;

        // Act
        var result = await controller.Authorize(data);

        // Assert
        if (isFound)
        {
            Assert.IsInstanceOfType(result.Result!, typeof(ObjectResult));
            Assert.AreEqual(200, ((ObjectResult)result.Result!).StatusCode);
        }
        else
        {
            Assert.IsInstanceOfType(result.Result!, typeof(ObjectResult));
            Assert.AreEqual(400, ((ObjectResult)result.Result!).StatusCode);
        }
    }
}

public class ValidationTestsHelper
{
    internal static bool IsValidObject<T>(T obj) where T : class
    {
        try
        {
            ValidateObject(obj);
        }

        return true;
    }
}

```

```

    }
    catch (Exception)
    {
        return false;
    }
}

internal static void ValidateObject<T>(T obj) where T : class
{
    var type = typeof(T);
    var meta =
type.GetCustomAttributes(false).OfType<MetadataTypeAttribute>().FirstOrDefault();
    if (meta is not null) type = meta.MetadataClassType;
    var propertyInfo = type.GetProperties();
    foreach (var info in propertyInfo)
    {
        var attributes =
info.GetCustomAttributes(false).OfType<ValidationAttribute>();
        foreach (var attribute in attributes)
        {
            var objPropInfo = obj.GetType().GetProperty(info.Name);
            attribute.Validate(objPropInfo!.GetValue(obj, null),
info.Name);
        }
    }
}
}

```

Приложение Е
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СЕРВЕРНОЙ ЧАСТИ
ПРИЛОЖЕНИЯ
(рекомендуемое)

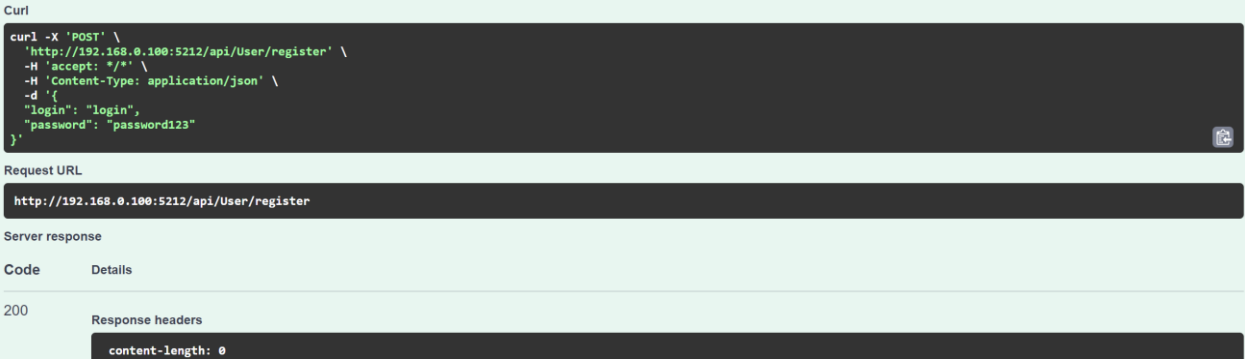


Рисунок Е.1 – Регистрации пользователя



Рисунок Е.2 – Авторизация пользователя

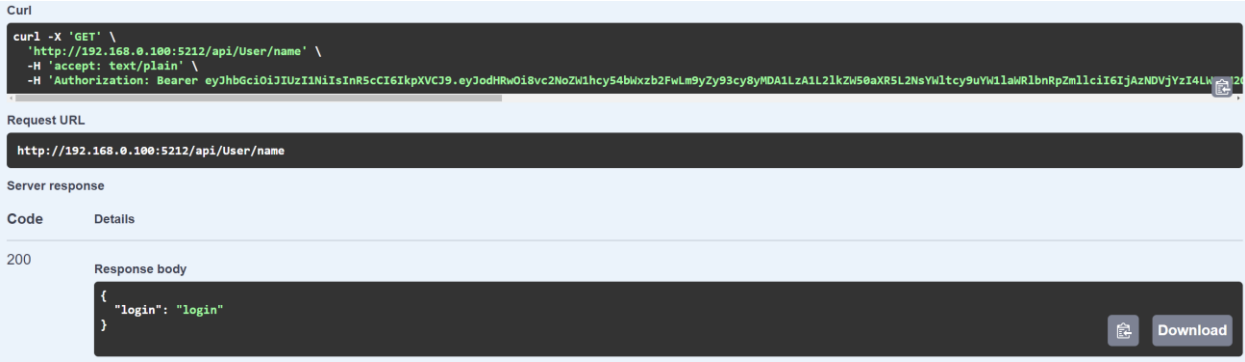


Рисунок Е.3 – Проверка аутентификации пользователя

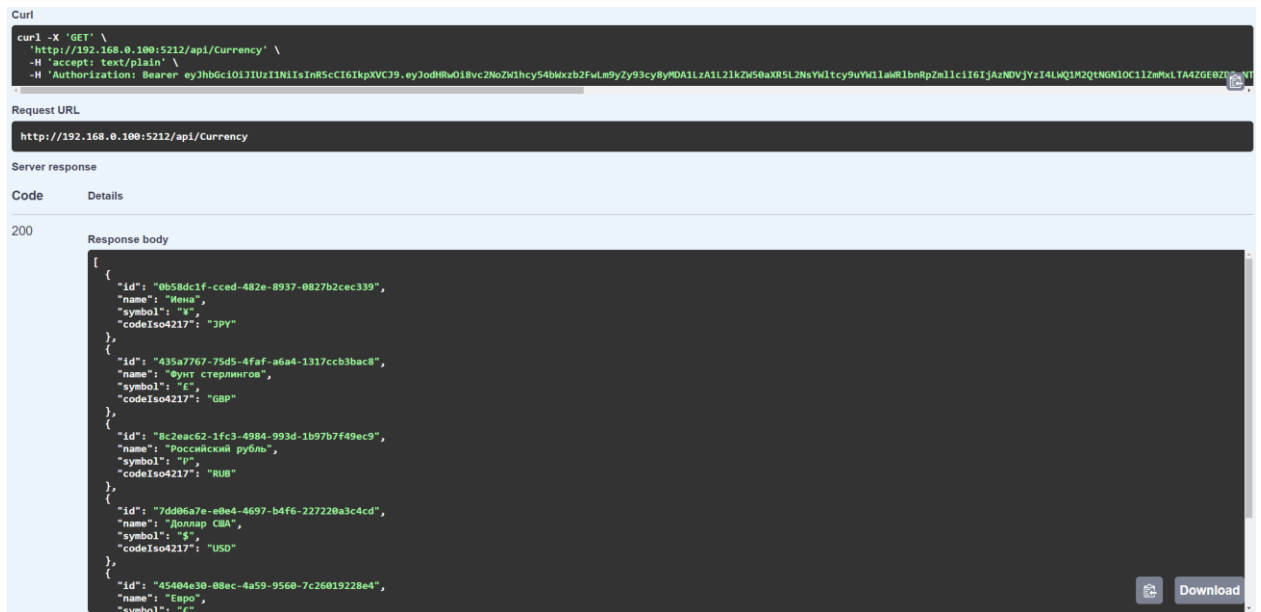


Рисунок Е.4 – Получение списка валют

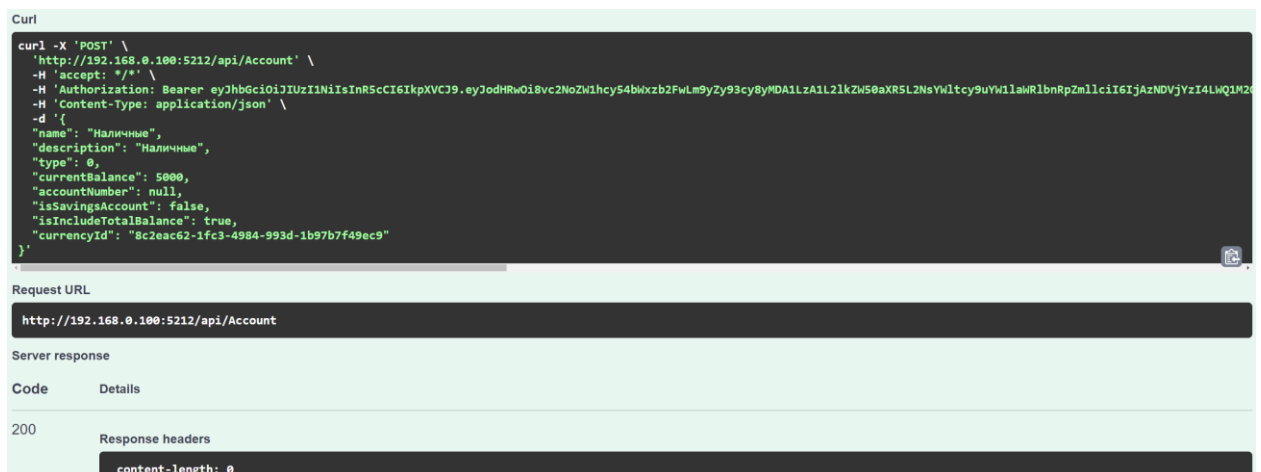


Рисунок Е.5 – Создание счета



Рисунок Е.6 – Получение списка счетов



Рисунок Е.7 – Изменение счета

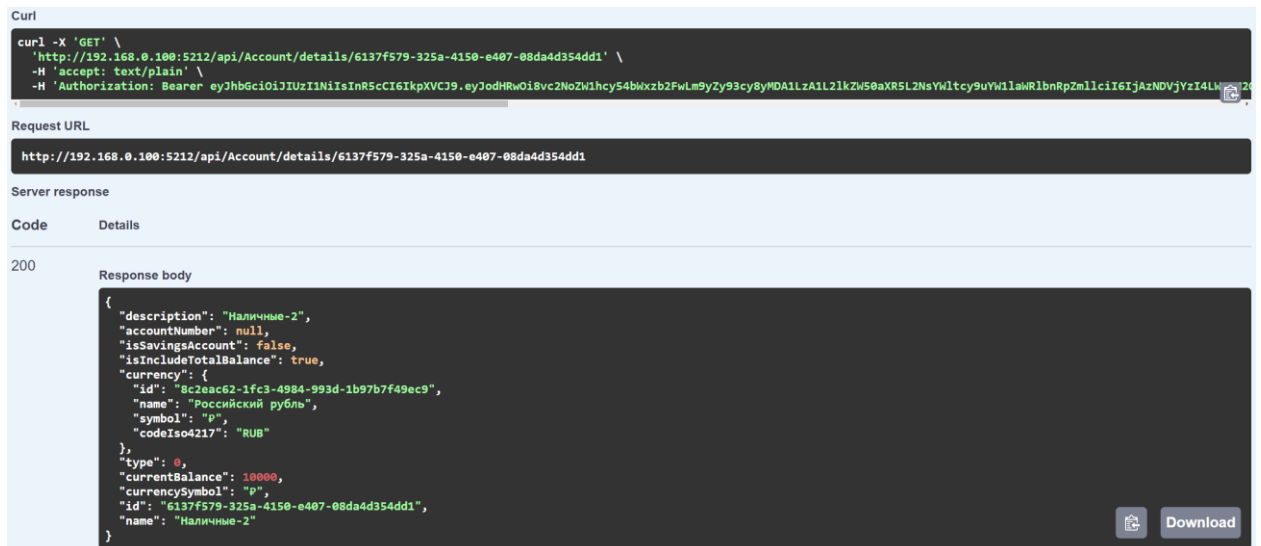


Рисунок Е.8 – Получение детальной информации о счете

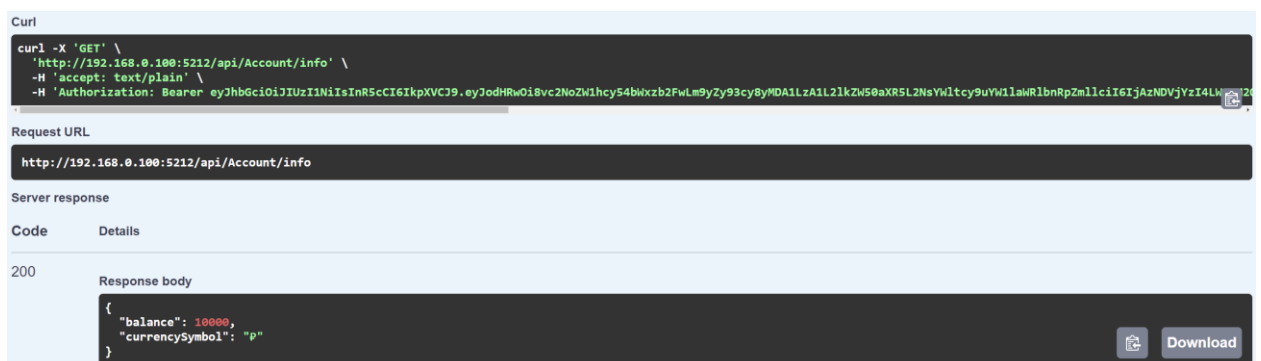


Рисунок Е.9 – Получение общей информации о счетах



Рисунок Е.10 – Удаление счета

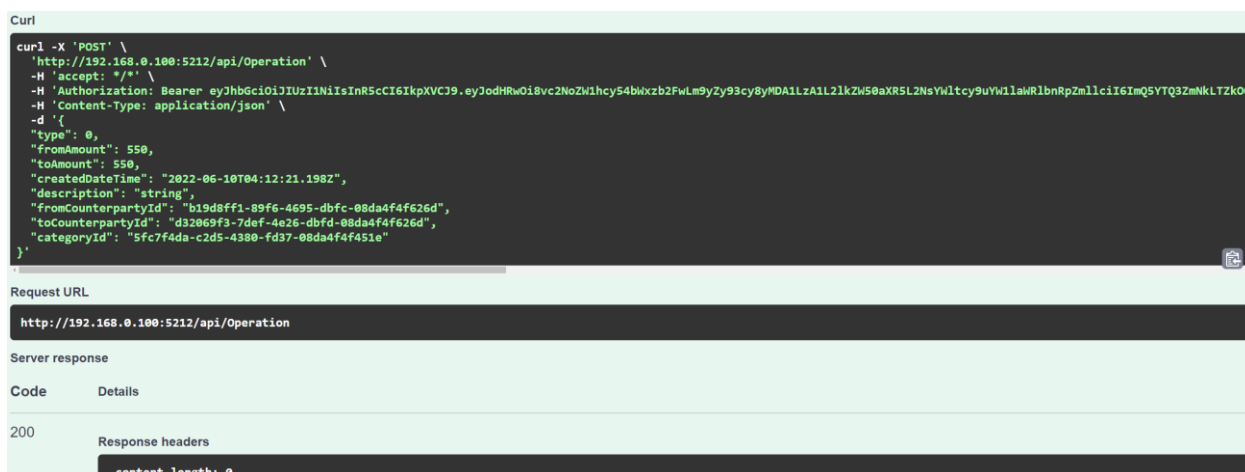


Рисунок Е.11 – Создание операции

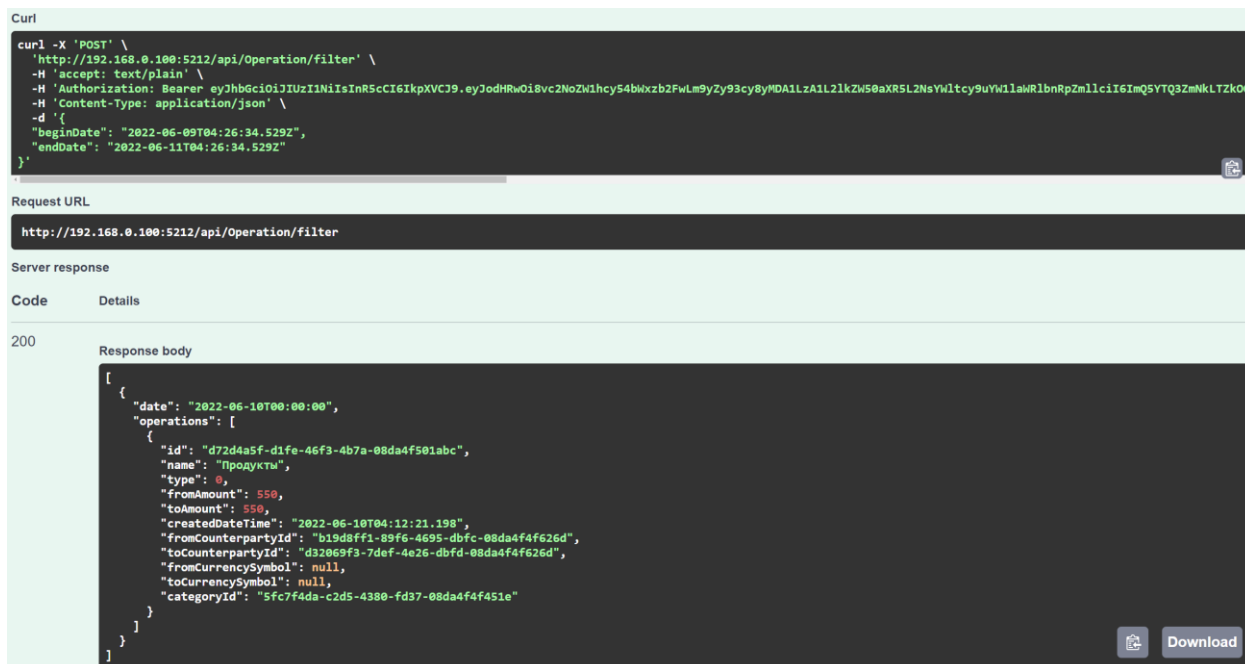


Рисунок Е.12 – Получен список операций

The screenshot displays a REST client interface with the following details:

- curl**:

```
curl -X 'GET' \  
'http://192.168.0.100:5212/api/Operation/details/d72da5f-d1fe-46f3-4b7a-08da4f501abc' \  
-H 'accept: text/plain' \  
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC99.eYJodHw018vc2NoZmlhc3Y4bWxz2FwLm9yZy99cyB0A11ZAl12kZW50aXSL2NoYnd1cy9uYnd1aWRlbnRpZmllc1I6Iiw9QSYTQ3ZW50aXSL2kOGt0NDUyNS1mZG9lTA4ZGF0Z2
```
- Request URL**:

```
http://192.168.0.100:5212/api/Operation/details/d72da5f-d1fe-46f3-4b7a-08da4f501abc
```
- Server response**:

```
200
```
- Code**: Details
- Response body**:

```
{  
  "description": null,  
  "fromCounterparty": {  
    "id": "b19d8ff1-89fe-4695-dbfc-08da4f4f626d",  
    "name": "Наличные"  
  },  
  "toCounterparty": {  
    "id": "d32069f3-7def-4e26-dbfd-08da4f4f626d",  
    "name": "Маргит"  
  },  
  "fromCurrency": null,  
  "toCurrency": null,  
  "category": {  
    "id": "5fc7f4da-c2d5-4380-fd37-08da4f4f451e",  
    "name": "Продукты",  
    "icon": null,  
    "isMandatory": true,  
    "isIncludePlan": true  
  },  
  "subjects": [],  
  "id": "d72da5f-d1fe-46f3-4b7a-08da4f501abc",  
  "name": "Нов категория",  
  "type": 0,  
  "fromAmount": 600,  
  "toAmount": 600,  
  "createdDateTime": "2022-06-10T04:12:21.198",  
  "fromCounterpartyId": "b19d8ff1-89fe-4695-dbfc-08da4f4f626d",  
  "toCounterpartyId": "d32069f3-7def-4e26-dbfd-08da4f4f626d",  
}
```

```
Curl  
curl -X 'DELETE' \  
  'http://192.168.0.100:5212/api/Operation/d72d4a5f-d1fe-46f3-4b7a-08da4f501abc' \  
  -H 'accept: */*' \  
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1keY54bnxzczZlLnN5Zy9ycyBMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYVllawRlbnRpZmllci6ImQ5VTQ3ZmkKLT
```

Request URL

```
http://192.168.0.100:5212/api/Operation/d72d4a5f-d1fe-46f3-4b7a-08da4f501abc
```

Server response

Code	Details
200	Response headers

Рисунок Е.15 – Операция успешно удалена



Рисунок Е.16 – Место успешно добавлено



Рисунок Е.17 – Получен список мест



Рисунок Е.18 – Место успешно изменено



Рисунок Е.19 – Место успешно удалено

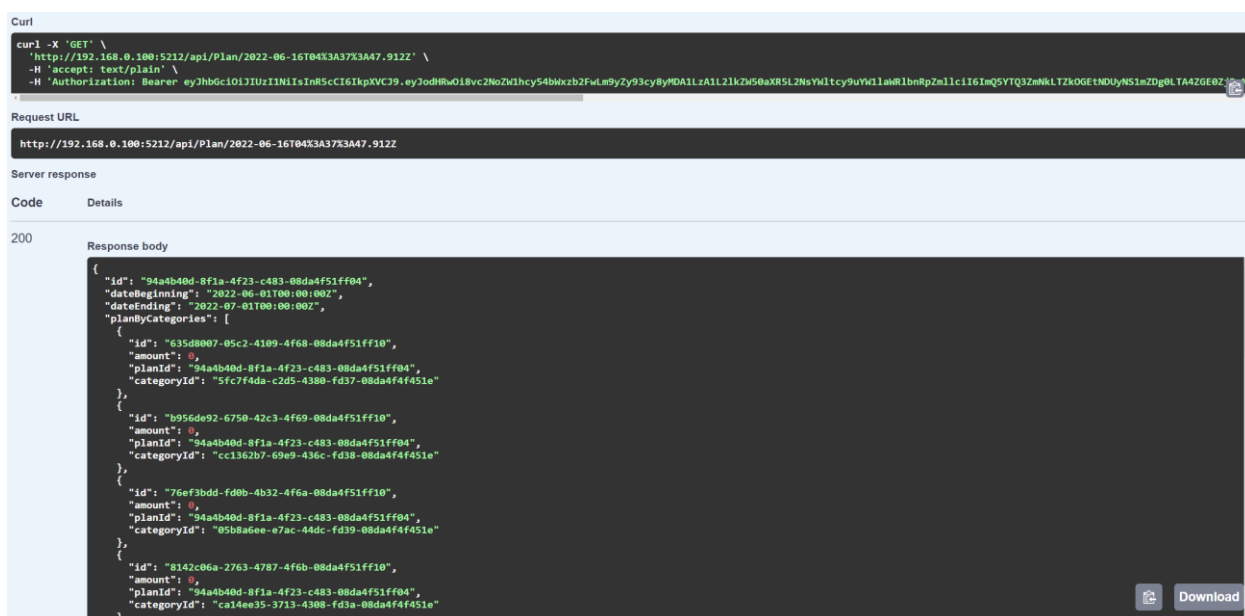


Рисунок Е.20 – Получен план по категориям на месяц

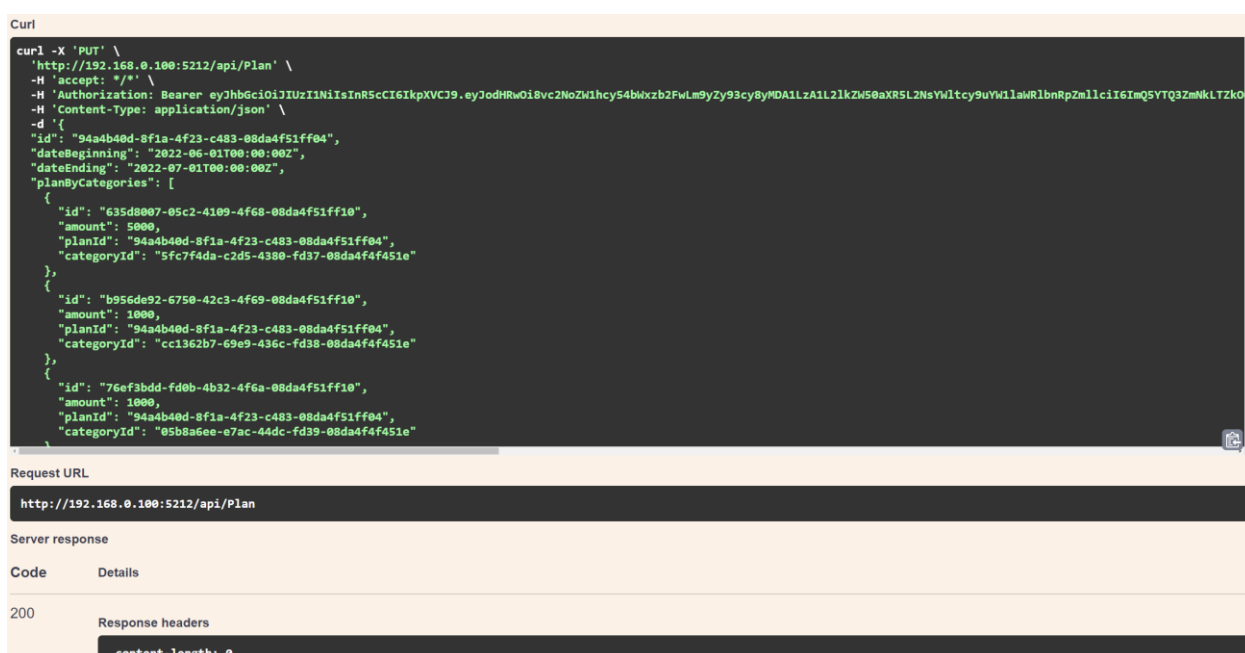


Рисунок Е.21 – План успешно изменен

```

Curl
curl -X 'POST' \
  'http://192.168.0.100:5212/api/Analytic' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ3b3RlbnR1b3R5cyBpdDA1LzA1L2lkZm50aXR5L2NsYWltcy9uYX11aW1mR1bnRpdZml1ci16ImQ5YTQ3ZmNkLTZkOGctNDUyNS1mZDg0LTA4ZGE0ZjRmMD' \
  -H 'Content-Type: application/json' \
  -d '{
    "operationType": 0,
    "analyticType": 0,
    "beginDate": "2022-06-01T04:56:03.928Z",
    "endDate": "2022-07-01T04:56:03.928Z"
  }'

Request URL
http://192.168.0.100:5212/api/Analytic

Server response

Code    Details
200

Response body
{
  "sumByCategory": [
    {
      "category": {
        "id": "1abc9b87-5f6e-4f21-9d91-6af35cb679bd",
        "name": "Продукты",
        "icon": null,
        "isMandatory": false,
        "isIncludePlan": false
      },
      "amount": 7504.5
    },
    {
      "category": {
        "id": "45d94ddf-9c5a-478d-af85-a564d762b781",
        "name": "Транспорт",
        "icon": null,
        "isMandatory": false,
        "isIncludePlan": false
      },
      "amount": 1132
    }
  ]
}

```

Рисунок Е.22 – Получены суммы внесённых операций расхода по категориям

Приложение Ж
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ
(рекомендуемое)

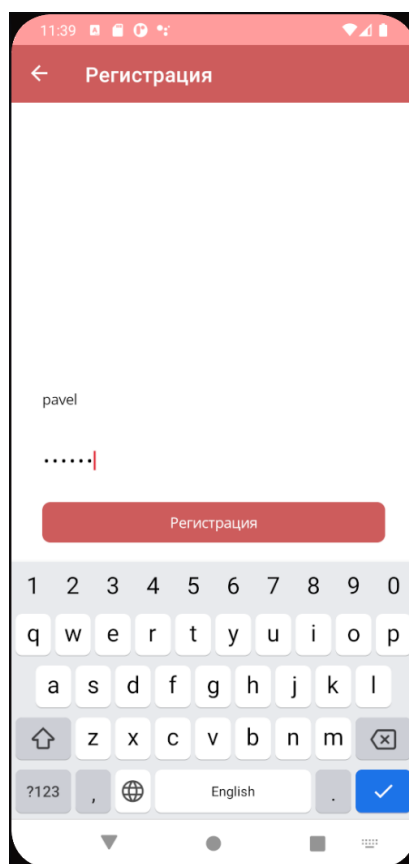


Рисунок Ж.1 – Пользователь успешно зарегистрирован в системе

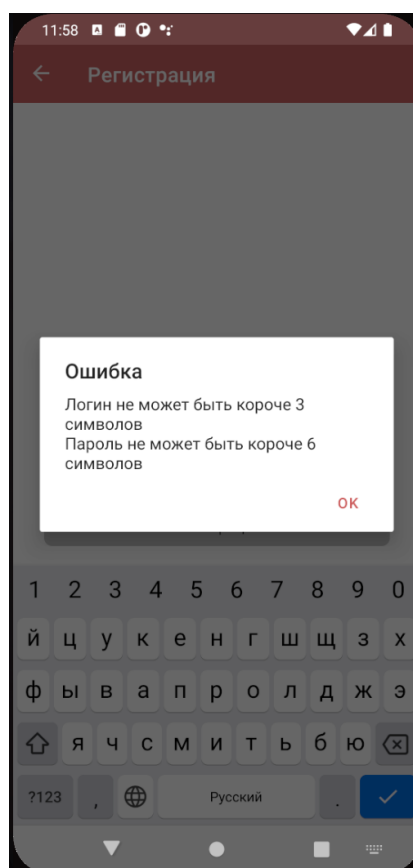


Рисунок Ж.2 – Показано сообщение с пояснением ошибки

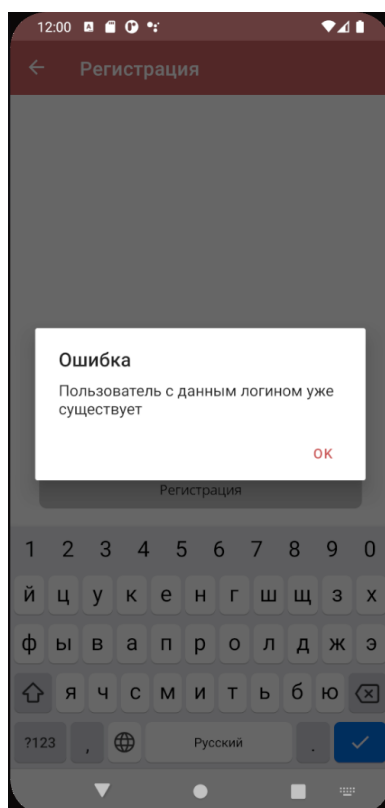


Рисунок Ж.3 – Показано сообщение о том, что такой пользователь уже существует

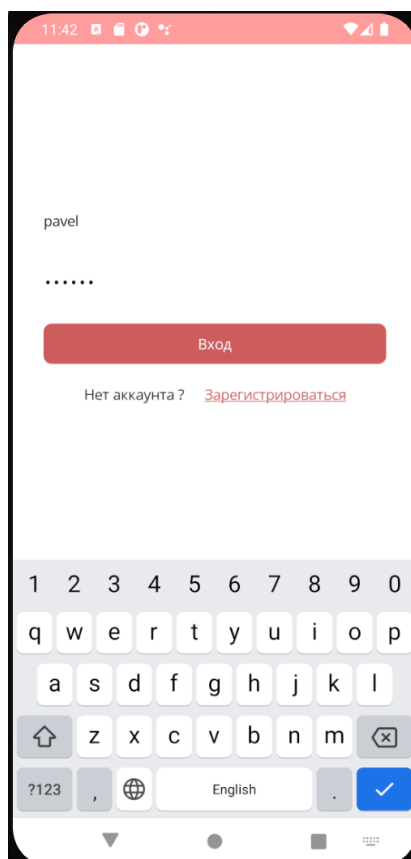


Рисунок Ж.4 – Авторизация выполнена

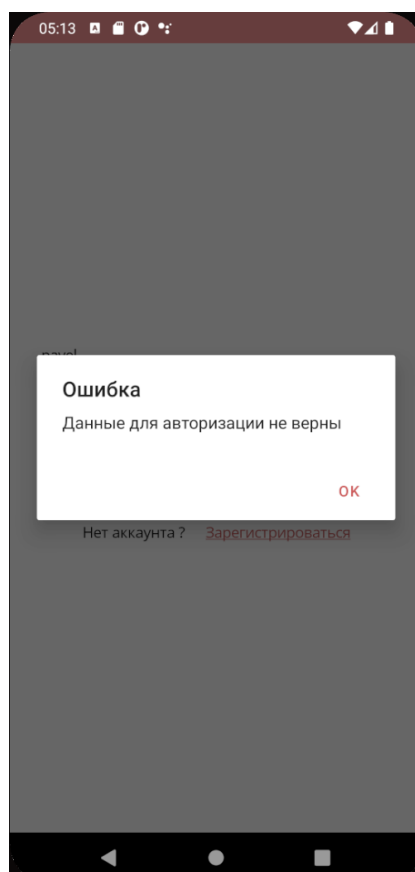


Рисунок Ж.5 – Показано сообщение с пояснением ошибки

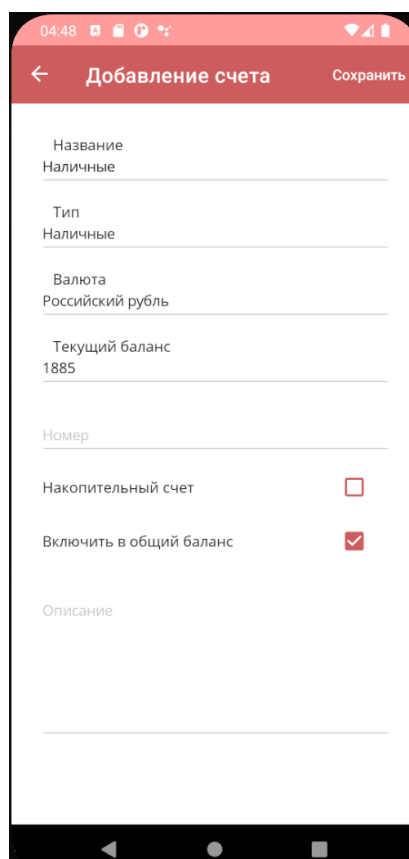


Рисунок Ж.6 – Счет успешно создан

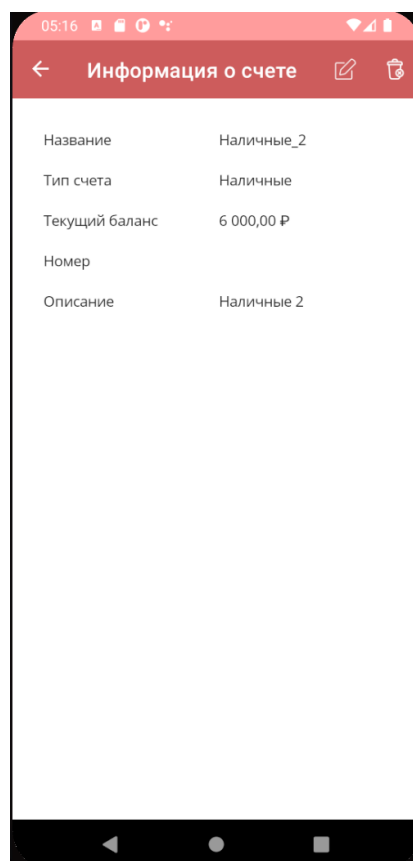


Рисунок Ж.7 – Счет успешно изменен

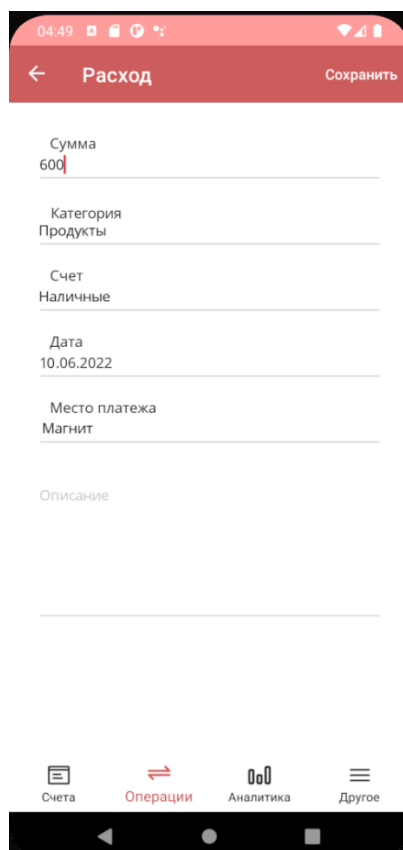


Рисунок Ж.8 – Операция успешно создана

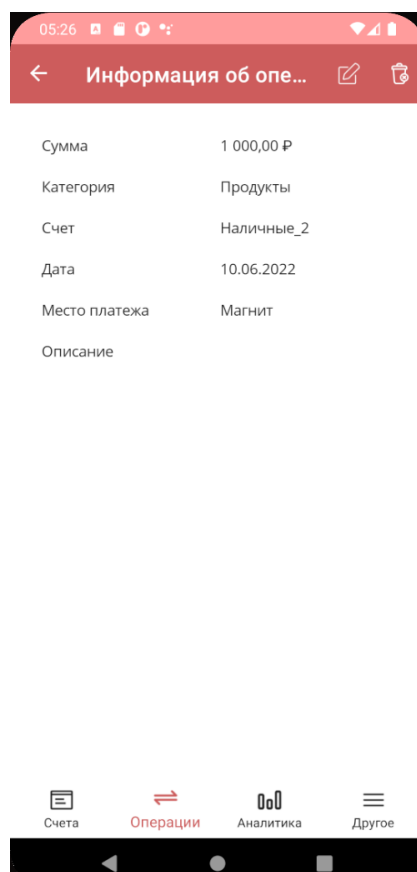


Рисунок Ж.9 – Операция успешно изменена

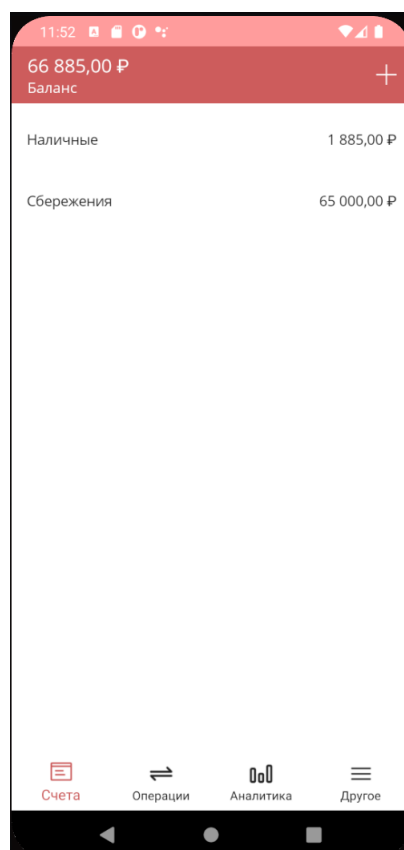


Рисунок Ж.10 – Показан список счетов

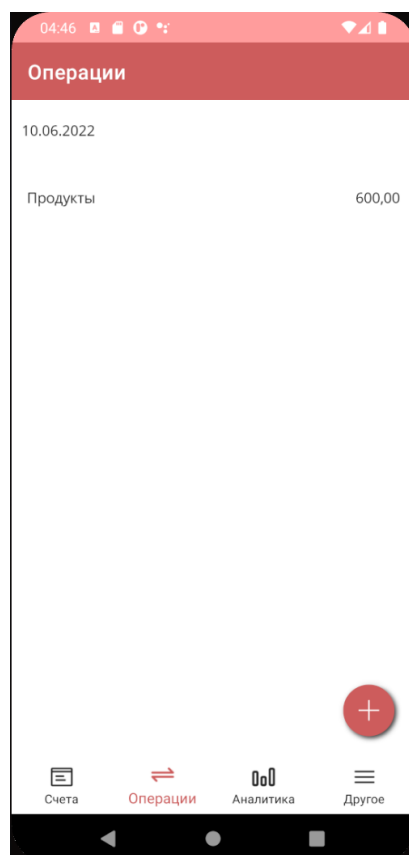


Рисунок Ж.11 – Показан список операций



Рисунок Ж.12 – Показан отчет по операциям

06:26

← Бюджет

Расходные категории

Категория	План	Факт
Продукты	12000	9560
Транспорт	4000	1575
Здоровье	4000	1100
Развлечения	3000	1800

Доходные категории

Категория	План	Факт
Зарплата	40000	40000

Счета Операции **Аналитика** Другое

Рисунок Ж.13 – Показан бюджет по операциям