МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное   
учреждение высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

**Разработка iOS-приложения для финансового планирования**

КУРСОВАЯ РАБОТА

по дисциплине «Программная инженерия»

ЮУрГУ – 09.03.02.2024.308-077.КР

|  |  |
| --- | --- |
| Нормоконтролер,  старший преподаватель кафедры СП  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Я.А. Краева  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | Научный руководитель:  старший преподаватель кафедры СП  \_\_\_\_\_\_\_\_\_\_ Я.А. Краева  Автор работы:  студент группы КЭ-304  \_\_\_\_\_\_\_\_\_\_И.Д. Варгунин  Работа защищена  с оценкой: \_\_\_\_\_\_\_\_\_\_\_  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

10.02.2024

**ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине «Проектирование и архитектура программных систем»

студенту группы КЭ-304 Варгунину Ивану Дмитриевичу,

обучающемуся по направлению 09.03.04 «Программная инженерия»

1. **Тема работы**Разработка iOS-приложения для финансового планирования.
2. **Срок сдачи студентом законченной работы:** 31.05.2024 г.
3. **Исходные данные к работе**
4. Apple. iOS Human Interface Guidelines-Designing for iOS. – 2014.
5. Усов В. Swift. Разработка приложений под iOS на основе фреймворка UIKit. – Москва: 2021. – 492 с.
6. Apple Developer Documentation. [Электронный ресурс] URL: https://devel- oper.apple.com/documentation (дата обращения: 19.02.2024 г.).
7. **Перечень подлежащих разработке вопросов**
8. Провести анализ предметной области и обзор аналогичных мобильных приложений.
9. Выполнить проектирование мобильного приложения для финансового планирования.
10. Реализовать и провести тестирование разработанного мобильного приложения.
11. **Дата выдачи задания:** 9 февраля 2024 г.

Научный руководитель , Я.А. Краева

старший преподаватель кафедры СП

Задание принял к исполнению И.Д. Варгунин

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc180503349)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc180503350)

[1.1. Обзор аналогичных проектов 6](#_Toc180503351)

[1.2. Обзор языков программирования 9](#_Toc180503352)

[2. АНАЛИЗ ТРЕБОВАНИЙ К СИСТЕМЕ 12](#_Toc180503353)

[2.1. Функциональные требования к проектируемой системе 12](#_Toc180503354)

[2.2. Нефункциональные требования к проектируемой системе 13](#_Toc180503355)

[2.3. Диаграмма вариантов использования 13](#_Toc180503356)

[3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ 15](#_Toc180503357)

[3.1. Архитектура системы 15](#_Toc180503358)

[3.3. Проектирование базы данных 18](#_Toc180503359)

[3.5. Проверка корректности при создании новой транзакции 19](#_Toc180503360)

[3.6. Проектирование интерфейса 20](#_Toc180503361)

[4. РЕАЛИЗАЦИЯ СИСТЕМЫ 22](#_Toc180503362)

[4.1. Реализация аутентификации пользователей 22](#_Toc180503363)

[4.2. Реализация облачной базы данных 24](#_Toc180503364)

[4.3. Реализация главного экрана 27](#_Toc180503365)

[4.4. Реализация экрана добавления новой транзакции 29](#_Toc180503366)

[4.5. Реализация экрана всех транзакций 31](#_Toc180503367)

[4.6. Реализация экрана пользователя 31](#_Toc180503368)

[4.7. Реализация перехода между главным экраном и экраном пользователя 33](#_Toc180503369)

[5. ТЕСТИРОВАНИЕ СИСТЕМЫ 34](#_Toc180503370)

[ЗАКЛЮЧЕНИЕ 36](#_Toc180503371)

[ЛИТЕРАТУРА 37](#_Toc180503372)

[ПРИЛОЖЕНИЕ 39](#_Toc180503373)

[Спецификация вариантов использования 39](#_Toc180503374)

# ВВЕДЕНИЕ

**Актуальность**

Мобильные приложения сопровождают миллионы людей ежедневно, выполняя не просто роль инструментов для получения информации или развлечений, они оказывают помощь в решении различных задач. Мобильные приложения призваны облегчить жизнь пользователей мобильных устройств и сделать ее лучше. Одной из важных областей, в которой мобильные приложения могут быть полезными, является финансовая сфера.

Сложно недооценивать значимость финансовой грамотности [1] в современном обществе, ведь ее отсутствие может нанести серьезный ущерб долгосрочному финансовому успеху человека. В сущности, финансовая грамотность – это способность человека принимать верные решения относительно своих доходов и расходов. Другими словами, это знания, помогающие делать обоснованные решения относительно финансов, которые способствуют достижению человеком финансового благополучия и экономической стабильности, помогают осознанно планировать свои будущие расходы.

Достижение финансовой грамотности человеком возможно при помощи отслеживания своих расходов и доходов, с чем может помочь мобильное приложения для финансового планирования. В мобильном приложении пользователь может вести историю своих транзакций и проводить анализ собственных трат и доходов с целью выявить прошлые неверные и определить будущие взвешенные с точки зрения финансовой грамотности решения относительно финансов.

**Постановка задачи**

Целью курсовой работы является разработка мобильного приложения для финансового планирования для платформы iOS.

Для достижения поставленной цели необходимо решить следующие задачи:

1. провести анализ предметной области и обзор существующих аналогичных мобильных приложений;
2. провести анализ требований к разрабатываемому приложению;
3. спроектировать мобильное приложение для мобильной операционной системы iOS;
4. реализовать и протестировать мобильное приложения для финансового планирования.

**Структура и содержание работы**

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 41 страницу, объем списка литературы – 16 источников.

В первой главе, «Анализ предметной области», производится анализ аналогичных приложений и обзор языков программирования для разработки мобильных приложений под операционную систему iOS.

Во второй главе, «Анализ требований к системе», определяются функциональные и нефункциональные требования к программной системе, а также приводится диаграмма вариантов использования приложения.

В третьей главе, «Проектирование системы», описываются архитектура и компоненты системы, схема базы данных и графический интерфейс приложения.

В четвертой главе, «Реализация системы», подробно описывается реализации мобильного приложения для финансового планирования.

В пятой главе, «Тестирование системы», приведены результаты функционального тестирования разработанной системы и тестирования пользовательского интерфейса приложения.

В приложении представлена спецификация вариантов использования.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Анализ предметной области – важный этап в начале проекта, целью которого является определение текущего состояния и основных аспектов предметной области. В процессе анализа собирается и изучается информация о текущих процессах, системах и технологиях, используемых в данной области. Выявление основных проблем и вызовов позволяет точно определить те аспекты, которые требуют внимания и улучшения в рамках проекта.

1.1. Обзор аналогичных проектов

В ходе анализа предметной области было выявлено, что разрабатываемое приложение имеет множество аналогов, схожих по функционалу.

**Money manager, expense tracker**

Money manager, expense tracker – одно из самых популярных приложений в App Store для финансового планирования и учета расходов и доходов. Данное приложение имеет высокие пользовательские оценки в магазине приложений App Store [2]: 53 тысячи отзывов, суммарный рейтинг которых составляет 5.0. В Money manager, expense tracker пользователь может записывать историю своих ежедневных расходов и доходов, а также анализировать соотношения объемов транзакций в различных категориях при помощи графиков. Пользователь может персонализировать приложение при помощи добавления собственных категорий трат и расходов. Скриншоты приложения Money manager, expense tracker представлены на рисунке 1.

К достоинствам приложения можно отнести автоматическое построение диаграмм, отображающих соотношение расходов и доходов в разных категориях транзакций, синхронизацию данных между устройствами пользователя и возможность постановки целей для бюджета. К недостаткам «Money manager, expense tracker» относится перегруженный элементами управления пользовательский интерфейс, также в отзывах в магазине мобильных приложений App Store пользователи отмечали нехватку возможности удалять неиспользуемые категории расходов и доходов, доступные с момента первой установки Money manager, expense tracker.

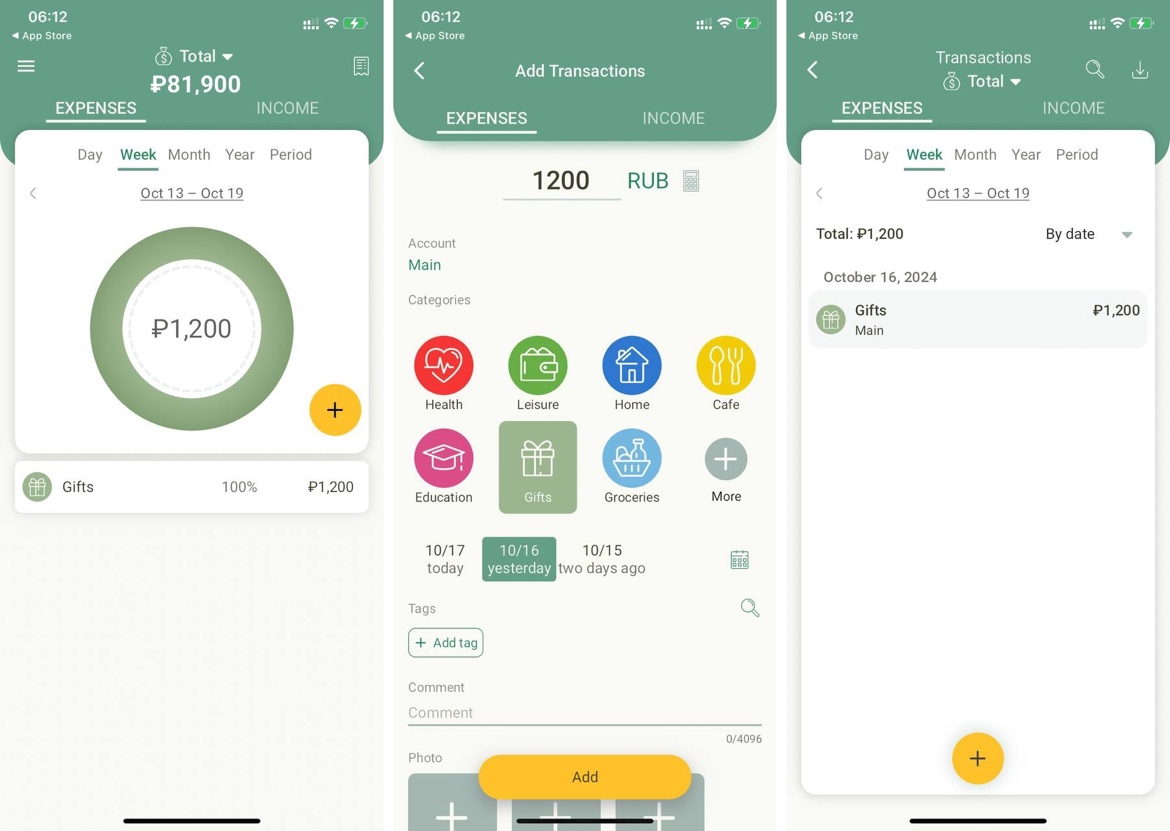


Рисунок 1 – Скриншоты приложения «Money manager, expense tracker»

**Buddy: Money & Budget planner**

Buddy: Money & Budget planner [3] – мобильное приложения для устройств на платформе iOS, позволяющее пользователю вести подсчет своих транзакций совместно с другими пользователями приложения, ведя совместный бюджет. Также доступна установка автоматических записей о регулярных платежах или поступлений доходов, таких как фиксированная заработная плата или платежи по кредитам. Скриншоты данного приложения представлены на рисунке 2.

Отличительной чертой и достоинством приложения Buddy: Money & Budget planner является возможность ведения совместного бюджета с другими пользователями, так же в приложении можно экспортировать историю транзакций в текстовом формате. К недостаткам мобильного приложения относится навязчивое предложение пользователю платных услуг, что часто отмечают пользователи в отзывах к приложению в App Store, также пользователи отмечают неудобство жесткого разделения историй транзакций по месяцам, когда остаток средств или накопленный долг с прошлого месяца не переносится на текущий месяц.

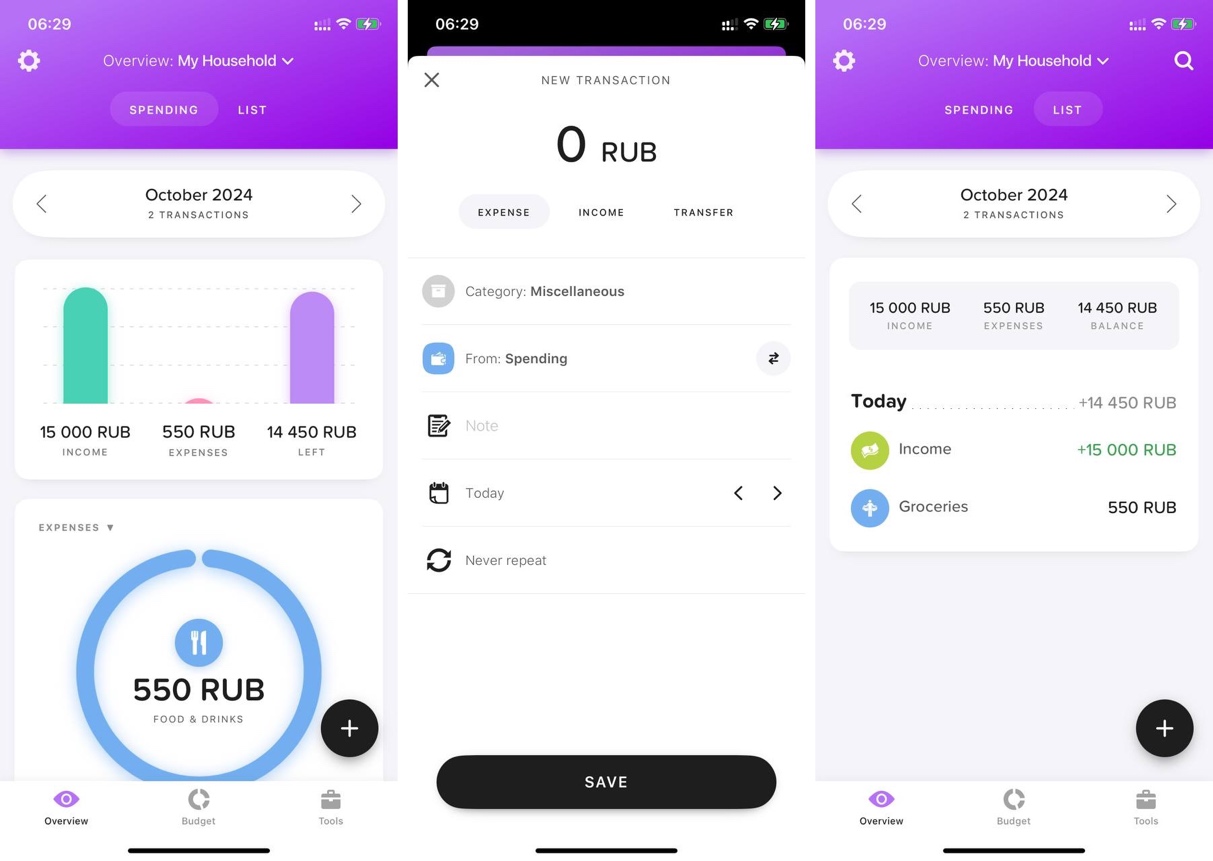


Рисунок 2 – «Скриншоты приложения Buddy: Money & Budget planner»

**Organizze**

Organizze – приложение для учета доходов и расходов пользователя, отличительной чертой которого является интеграция с банковскими приложениями, что дает возможность полностью автоматического сбора информации о транзакциях. Как заявляет разработчик в описании Organizze в магазине App Store [4], хранимые приложением данные являются защищенными от злоумышленников. Скриншоты данного приложения представлены на рисунке 3.

Приложение имеет множество плюсов: возможность настройки регулярных автоматических записях о доходах и расходах, простая авторизация в приложении с помощью аккаунта Apple или Google, удобный встроенный калькулятор на экране создания новой записи о доходе или расходе, однако в приложении отсутствует локализация интерфейса и возможность выбора валюты транзакций, что ограничивает аудиторию приложения лишь носителями языка приложения.

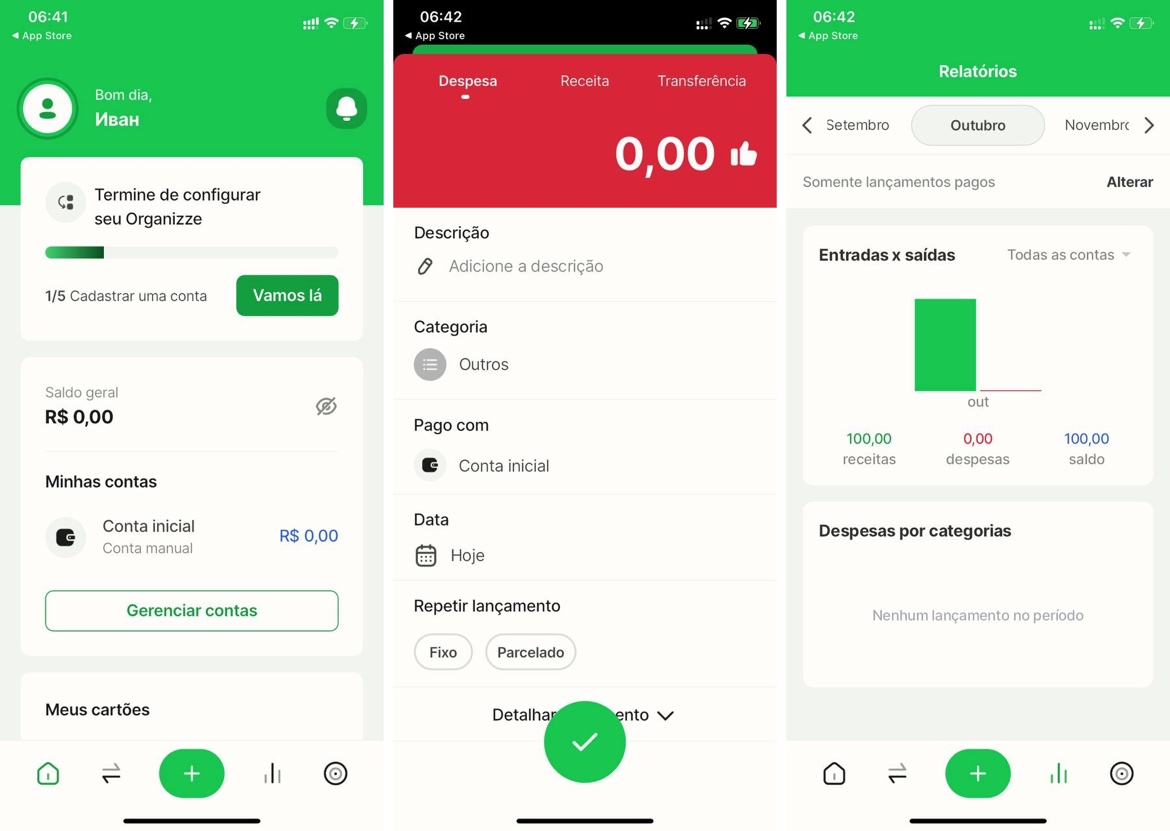


Рисунок – Скриншоты приложения Organizze

1.2. Обзор языков программирования

Еще один вопрос, который возникает, когда мы решаем создать приложение – какой язык использовать. Разработчиком операционной системы iOS – компанией Apple поддерживаются два нативных официальных языка – Objective C и Swift. Основной средой разработки для данных языков является Xcode. Рассмотрим преимущества и недостатки каждого языка.

**Objective C**

Objective C [5] – это язык программирования, который расширяет язык C, добавляя в него элементы объектно-ориентированного программирования. Он был разработан в начале 1980-х годов Брэдом Коксом и Томом Лавом и стал одним из основных языков для разработки под операционные системы macOS и iOS от Apple до появления Swift.

Преимуществами языка являются легкая интеграция с языками программирования C и С++, позволяющая использовать множество готовых библиотек и решать низкоуровневые задачи, автоматическое управление памятью, которое помогает избежать утечек памяти и упрощает управление ресурсами, динамическая типизация, благодаря которой ускоряется написание программ. Язык имеет развитую экосистему и долгую историю как основной язык для платформ Apple, что сопровождается обширной документацией и поддержкой, однако новые подходы и фреймворки в большей степени адаптируются для использования вместе с современным языком программирования Swift.

**Swift**

Swift [7] – это язык программирования, разработанный компанией Apple и представленный широкой аудитории в 2014 году на конференции Worldwide Developers Conference. Swift является прямым наследником Objective-C и в данный момент является основным языком разработки на платформе iOS.

Swift наследует большинство достоинств языка Objective C, однако при этом и обладает рядом уникальных достоинств, таких как более простой и понятный синтаксис в сравнении с Objective-C, высокая производительность за счет механизмов управления памятью и статической типизацией языка, официальная поддержка и развитие компанией Apple.

Swift полностью совместим с Objective C, то есть в рамках одного проекта можно использовать оба языка. Однако Swift обладает рядом преимуществ, которые делают написание кода проще, например, возможность создания интерфейса приложения с помощью современного декларативного фреймворка SwiftUI.

**Вывод по главе 1**

Таким образом, обзор перечисленных аналогов показал, что все рассмотренные мобильные приложения имеют как общие, так и уникальные функции. К обязательным функциям приложения можно отнести отображение записей о доходах и расходах, построение диаграмм, отображающих соотношение расходов и доходов в разных категориях по месяцам.

Среди преимуществ рассмотренных решений можно отметить синхронизацию данных между устройствами пользователя, использование простых в эксплуатации систем авторизации, поставляемых компаниями Google или Apple, когда пользователю приложения для авторизации нужно лишь подтвердить свою личность отпечатком пальца или сканом лица. К недостаткам можно отнести наличие навязчивой рекламы в приложениях, отсутствие локализации и возможности сменить валюту записей о доходах и расходах, а также перегруженный элементами управления пользовательский интерфейс. Кроме того, проанализировав два языка, для создания приложения было решено использовать современный Swift.

# 2. АНАЛИЗ ТРЕБОВАНИЙ К СИСТЕМЕ

Определение требований к проекту включает в себя выяснение функциональных и нефункциональных характеристик, необходимых для достижения поставленных целей.

Анализ технологической составляющей оценивает текущие технологии и системы, используемые в предметной области, и их пригодность для реализации поставленных задач. На основе анализа формулируются рекомендации и стратегии для улучшения процессов и решения проблем, что способствует скорейшему достижению целей проекта.

Для дальнейшего проектирования приложения были составлены два типа требований:

* функциональные требования, которые определяют, что приложение должно делать;
* нефункциональные требования, которые определяют, как приложение должно работать.

Оба типа требований играют важную роль в процессе проектирования приложения, поскольку помогают разработчикам понять, что от них ожидается и какие ограничения имеются.

2.1. Функциональные требования к проектируемой системе

Функциональные требования описывают, как продукт должен вести себя в различных ситуациях. Они определяют, какие функции и возможности должны быть реализованы разработчиками, чтобы пользователи могли в полной мере выполнять свои задачи в рамках бизнес – требований. Это важное соотношение между требованиями пользователя, бизнес-потребностями и функциональностью продукта является ключевым для успешного завершения проекта. Функциональные требования обычно формулируются в виде утверждений, использующих слова «должен» или «должна», и описывают конкретные функции или поведение продукта, необходимые для удовлетворения потребностей пользователей.

В рамках вышеописанной задачи были выявлены следующие функциональные требования:

1. приложение должно предоставлять возможность просматривать записи о доходах и расходах;
2. приложение должно предоставлять возможность сохранять и удалять записи о доходах и расходах;
3. приложение должно предоставлять отчет за определенное время о доходах и расходах бюджета по категориям;
4. приложение должно иметь локализированный интерфейс с возможностью выбора учитываемой валюты;
5. приложение должно иметь синхронизацию данных на нескольких устройствах.

2.2. Нефункциональные требования к проектируемой системе

Нефункциональные требования дополняют функциональные требования, определяя, как программная система должна выполнять определенные функции. Они определяют качества, характеристики и ограничения системы, а не ее конкретные особенности. По сути, нефункциональные требования устанавливают стандарты производительности, безопасности и удобства использования системы. Были выделены следующие нефункциональные требования:

1. приложение должно быть написано на языке Swift с использованием фреймворка SwiftUI [11];
2. пользовательский интерфейс должен быть неперегруженным элементами управления и интуитивно понятным пользователю;
3. приложение должно иметь поддержку различных языков и валют.

2.3. Диаграмма вариантов использования

Для проектирования описанных выше функциональных требований с помощью языка объектного моделирования UML была создана диаграмма вариантов использования, показывающая отношения между пользователями и прецедентами, представленная на рисунке 5.



Рисунок 5 – Диаграмма вариантов использования iOS-приложения   
для финансового планирования

Двумя основными актерами, взаимодействующими с системой**,** являются авторизированный пользователь и неавторизованный пользователь.

Неавторизованный пользователь – пользователь, у которого нет доступа к основному функционалу системы, кроме авторизации, после которой он станет авторизированным пользователем. Авторизованный пользователь – пользователь, имеющий доступ к функционалу приложения. Он взаимодействует с приложением для создания записей о доходах и расходах, их удаления и просмотра статистики по категориям. Спецификация основных вариантов использования приведена в приложении.

**Вывод по главе 2**

Таким образом, были сформированы функциональные и нефункциональные требования к проектируемой системе, была создана диаграмма вариантов использования приложения с двумя основными актерами: авторизованным и неавторизованным пользователем.

# 3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1. Архитектура системы

В мобильном приложении использован архитектурный паттерн MVVM (Model-View-ViewModel). Шаблон архитектуры клиентских приложений был предложен Джоном Госсманом как альтернатива шаблонам MVC и MVP. Его концепция заключается в отделении логики представления данных от бизнес-логики путем вынесения ее в отдельный класс для более четкого разграничения [13].

1. Модель (Model) – компонент, который отвечает за абстрагирование источников данных.
2. Вид (View) – компонент, целью которого является информирование модели представления о действии пользователя. Представление является пассивным и не содержит логики или состояния приложения.
3. Модель представления (ViewModel) – компонент, который служит связующим звеном между моделью и представлением.

На рисунке 6 представлена схема работы паттерна MVVM в мобильном приложении.

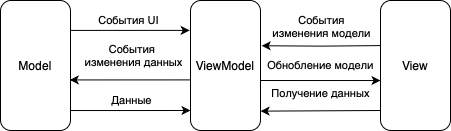


Рисунок 6 – Схема работы паттерна MVVM в мобильном приложении

На рисунке 7 представлена диаграмма основных компонентов системы разрабатываемого приложения, спроектированная на базе архитектурного паттерна MVVM.

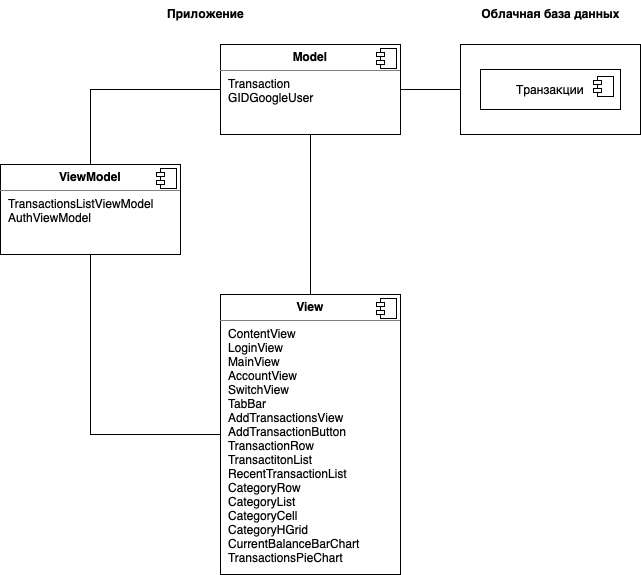


Рисунок – Диаграмма компонентов

Были выделены 2 основных модели приложения, перечисленные ниже.

1. Transaction – модель, представляющая запись о расходе или доходе.
2. GIDGoogleUser – модель, представляющая информацию о пользователе.

Были выделены 17 основных представлений для приложения, представленные ниже.

1. ContentView – представление, являющее контейнером для всех экранов приложения.
2. AuthView – экран аутентификации пользователя.
3. MainView – главный экран приложения.
4. AccountView – экран для отображения информации о пользователе.
5. SwitchView – представление для выбора типа отображаемых записей (доходы или расходы) на AccountView.
6. TabBar – представление для нижней панели навигации по экранам приложения.
7. AddTransactionsView – экран для добавления новой транзакции.
8. AddTransactionsButton – представление для добавления новой транзакции.
9. TransactionRow – представление, отображающее ячейку списка транзакций.
10. TransactionsList – экран, отображающий список транзакций.
11. RecentTransactionsList – экран, отображающий список последних пяти транзакций.
12. CategoryRow – представление, отображающее ячейку списка категорий транзакций.
13. CategoryList – представление, отображающее список категорий транзакций.
14. CategoryCell – представление, отображающее ячейку таблицы категорий транзакций.
15. CategotyHGrid – представление, отображающее таблицу категорий транзакций.
16. CurrentBalanceBarChart – представление, отображающее столбчатый график изменения баланса пользователя за последний месяц.
17. TransactionsPieChart – представление, отображающее круговую диаграмму транзакций пользователя, группируя их по месяцам и по категориям транзакций.

Ниже описаны модели представлений разрабатываемого приложения.

1. TransactionsListViewModel – модель представления для управления списком транзакций.
2. AuthViewModel – модель представления для управления авторизацией пользователя.

3.3. Проектирование базы данных

При разработке мобильного приложения для планирования финансов, стоит учитывать то, что данные вносимые в приложение необходимо хранить, чтобы пользователь, зайдя в приложение через некоторое время увидел свои данные и мог продолжить работу с приложением. Необходимо не забыть и про авторизацию, чтобы пользователь получал доступ именно к своим данным.

С помощью набора инструментов и сервисов для разработки мобильных и веб-приложений от Google – Firebase можно быстро подключить базы данных и настроить авторизацию пользователей. Платформа облачная, поэтому все ресурсы приложений, включая исходный код и базы данных, хранятся на серверах Google.

В основе Firebase лежит база данных реального времени, которая синхронизирует данные на всех подключенных устройствах в режиме реального времени. База данных использует документно-ориентированную модель данных NoSQL, что позволяет разработчикам хранить данные гибким и масштабируемым образом. Данные хранятся в формате JSON, база данных поддерживает атомарные транзакции и уведомления о событиях в реальном времени.

Для хранения данных приложения используется база данных реального времени. Firebase Database – облачная документно-ориентированная база данных NoSQL, позволяющая хранить и синхронизировать данные в реальном времени. Обеспечивает одновременную работу на разных устройствах и оптимизирована для автономного использования. На рисунке 8 представлена схема базы данных записей о доходах и расходах пользователей.

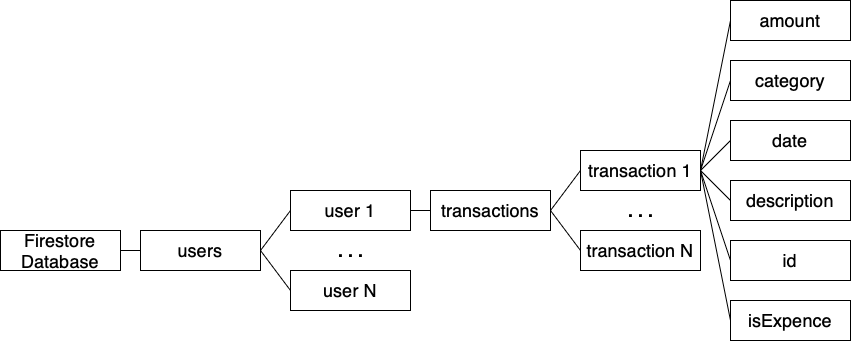


Рисунок 8 – Схема базы данных

В корне базы данных хранится каталог users, предназначенный для хранения списка пользователей приложения. При авторизации пользователя из каталога users, а затем из папки авторизированного пользователя загружаются данные. Каталог каждого пользователя содержит каталог с записями о доходах и расходах.

3.5. Проверка корректности при создании новой транзакции

На рисунке 8 видны поля структуры записи о доходе или расходе: поле amount и id имеют численный тип, date имеет тип даты, isExpense имеет булевый тип, а category и description имеют строковый тип.

При этом поля amount и description в приложении являются свободно определяемым пользовательским вводом в отличие от полей category, date и isExpence с ограниченным множеством значений и автоматически определяемого поля id. Это означает, что необходима проверка корректности пользовательского ввода при создании новой транзакции. Данная проверка отображена на рисунке 9.

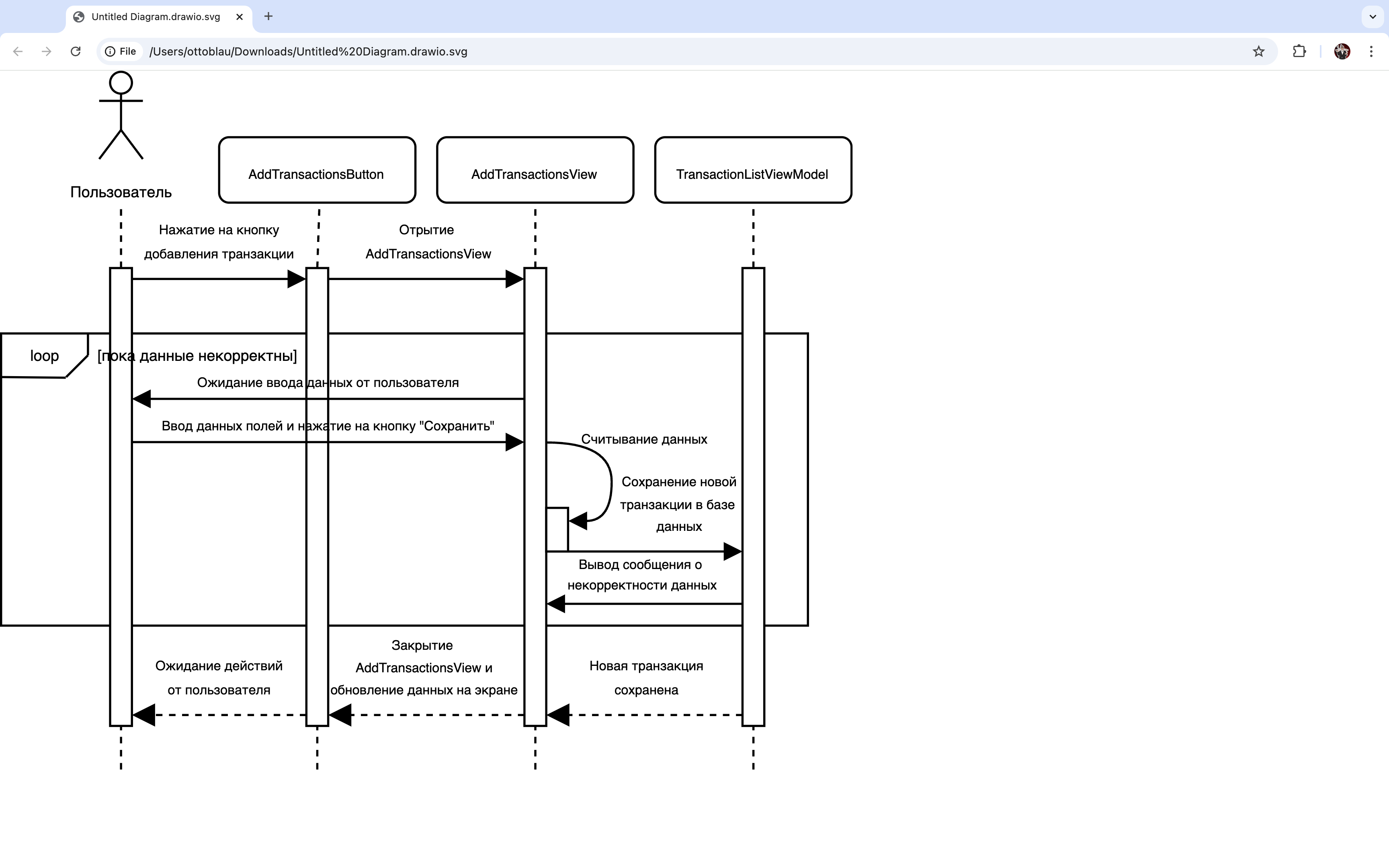


Рисунок 9 – Диаграмма последовательности создания новой транзакции

3.6. Проектирование интерфейса

Разработка макетов является неотъемлемой частью процесса разработки пользовательского интерфейса приложения, позволяющей визуализировать идеи и концепции перед тем, как приступать к финальному исполнению проекта [14].

На рисунке 10 представлены макеты главного экрана и экрана списка всех транзакций. На рисунке 11 представлены макеты экрана добавления новой транзакции и экрана пользователя.

На главном экране приложения, согласно макету, располагается график баланса пользователя за последний месяц, кнопки добавления новых доходов или расходов. Таблица недавних транзакций имеет переход на экран с расширенной историей записей о доходах и расходах.

Пользовательский экран содержит в себе настройки приложения и круговую диаграмму транзакций по категориям за месяц.

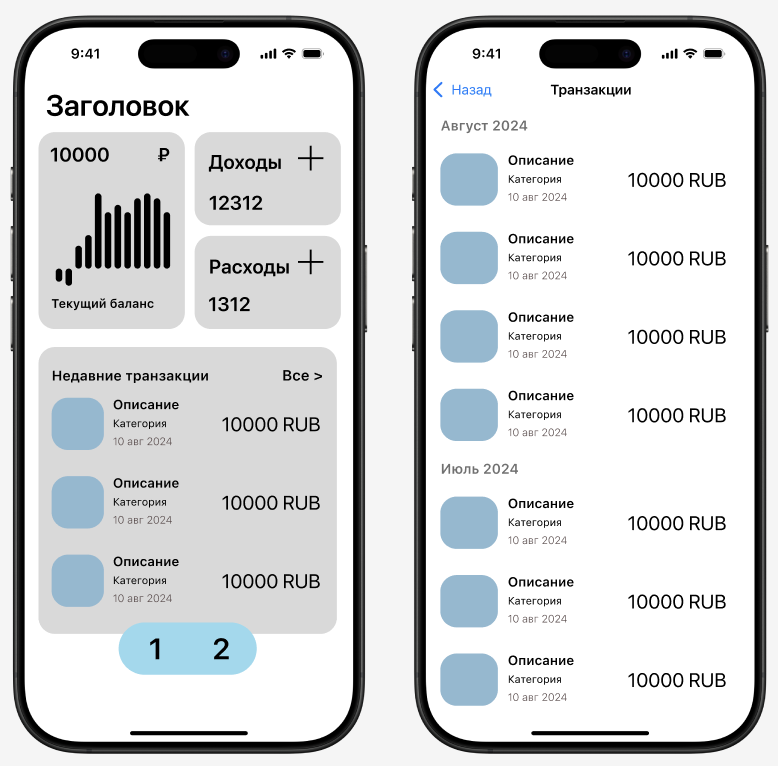


Рисунок – Макеты главного экрана и экрана списка всех транзакций

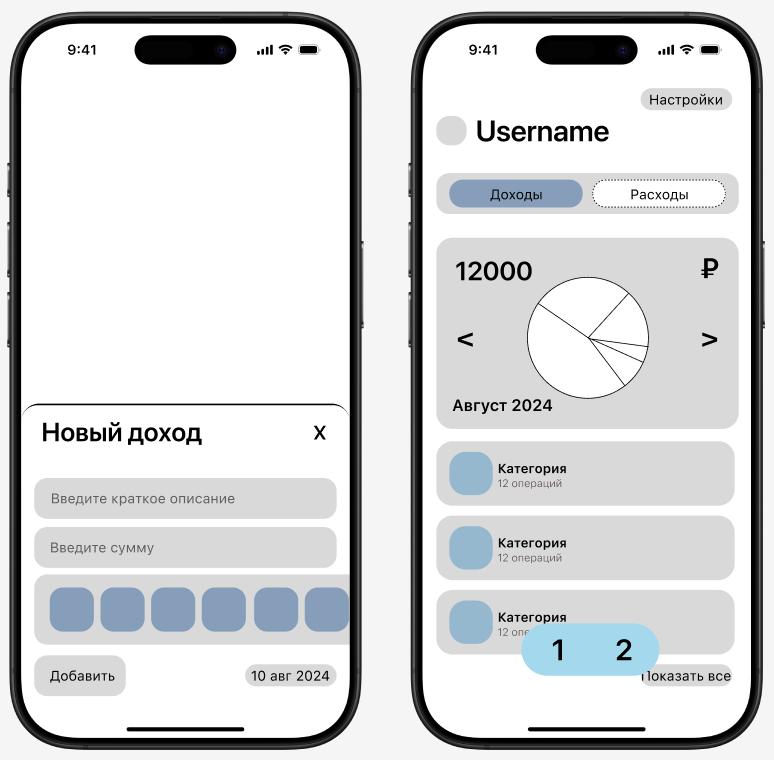


Рисунок – Макеты экрана добавления новой транзакции и экрана пользователя

**Выводы по главе 3**

В результате работы была спроектирована архитектура и описаны компоненты приложения, была разработана схема баз данных и были созданы макеты пользовательского интерфейса приложения.

# 4. РЕАЛИЗАЦИЯ СИСТЕМЫ

Для разработки приложения было решено использовать Xcode, интегрированную среду разработки для работы с платформой iOS, и язык программирования Swift.

* 1. Реализация аутентификации пользователей

Firebase Authentication, предоставляет различные способы аутентификации пользователей в приложении, такие как аутентификация с помощью электронной почты или пароля или с помощью компаний, предоставляющих средства аутентификации, например, аутентификация с помощью сервиса Apple, Google или Microsoft. В приложении будет использоваться аутентификация при помощи сервиса Google.

Пример отображения зарегистрированных пользователей в консоли Firebase показан на рисунке 12.

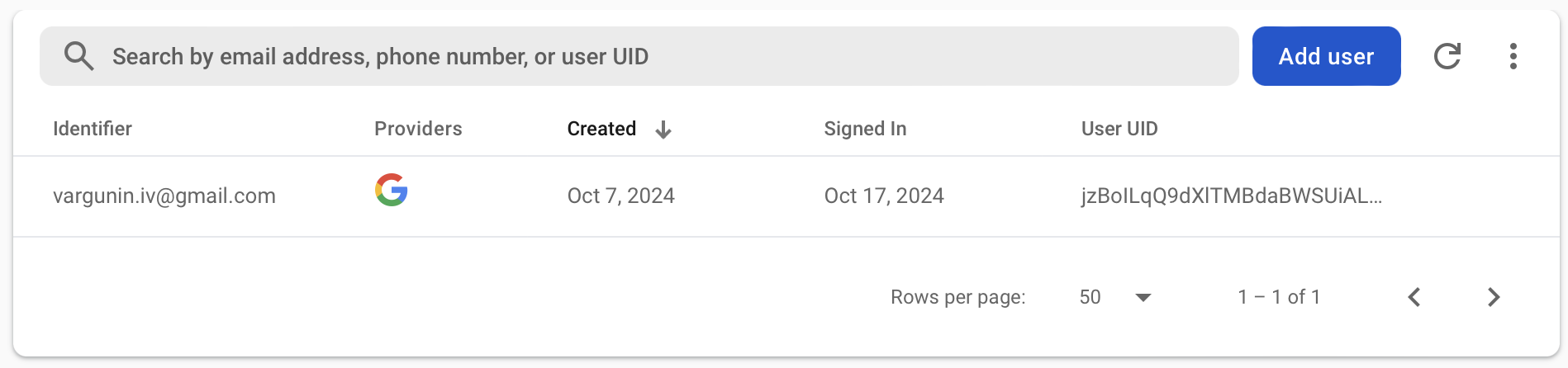


Рисунок – Список зарегистрированных пользователей в консоли Firebase

При запуске приложения происходит инициализация ContentView – представления, являющегося контейнером для других представлений, которое обращается к модели представления AuthViewModel, чтобы получить информацию, существует ли в момент запуска приложения аутентифицированный пользователь, и, исходя из этой информации, представляет пользователю экран аутентификации AuthView или главный экран приложения MainView.

Информация в AuthViewModel о текущем пользователе обновляется при аутентификации пользователя при помощи аккаунта Google, при выходе из аккаунта в приложении, а также при запуске приложения.

На листинге 1 представлен код аутентификации пользователя в AuthViewModel при запуске приложения. Если на момент запуска приложения аутентифицированного пользователя нет, то представляется экран аутентификации, на котором располагается единственная кнопка аутентификации при помощи сервисов Google.

Листинг 1 – Код аутентификации пользователя при запуске приложения

init() {

self.isUserLoggedIn = gidSignIn.hasPreviousSignIn()

if isUserLoggedIn {

gidSignIn.restorePreviousSignIn { [weak self] user, error in

self?.authenticate(user, with: error)

}

}

}

private func authenticate(\_ user: GIDGoogleUser?, with error: Error?) {

if let error = error {

print(error.localizedDescription)

return

}

guard let idToken = user?.idToken?.tokenString, let accessToken = user?.accessToken.tokenString else { return }

let credential = GoogleAuthProvider.credential(withIDToken: idToken, accessToken: accessToken)

Auth.auth().signIn(with: credential) { [unowned self] (\_, error) in

if let error = error {

print(error.localizedDescription)

} else {

self.isUserLoggedIn = true

self.currentUser = user

}

}

}

На листинге 2 представлен код входа пользователя в приложение при помощи сервисов Google. Для выхода из аккаунта на экране пользователя в шторке настроек необходимо нажать на кнопку выхода из аккаунта.

На листинге 3 представлен код выхода пользователя из аккаунта в приложении.

Листинг 2 – Код входа пользователя в приложение

func signIn() {

if gidSignIn.hasPreviousSignIn() {

gidSignIn.restorePreviousSignIn { [weak self] user, error in

self?.authenticate(user, with: error)

}

}

do {

guard let clientID = FirebaseApp.app()?.options.clientID else { return }

let config = GIDConfiguration(clientID: clientID)

let viewController = try UIApplication.getRootViewController()

gidSignIn.configuration = config

gidSignIn.signIn(withPresenting: viewController) { [weak self] result, error in

self?.authenticate(result?.user, with: error)

}

} catch {

print(error.localizedDescription)

}

}

Листинг 3 – Код выхода пользователя из аккаунта в приложении

func signOut() {

gidSignIn.signOut()

isUserLoggedIn = false

currentUser = nil

}

* 1. Реализация облачной базы данных

В разрабатываемом приложении необходимо считывать данные, добавлять и удалять их. Для этого отлично подойдет Firebase Firestore Database, которая позволяет обновлять данные приложения в реальном времени на всех устройствах, которые будут использовать приложение.

С каждым пользователем ассоциирован уникальный автоматически генерируемый идентификационный номер, который является частью пути к директории с транзакциями в документно-ориентированной базе данных Firebase Firestore.

На листинге 4 представлена функция получения пути до пользовательской директории с записями о доходах и расходах на сервере.

Листинг 4 – Код получения пути до пользовательской директории с транзакциями

extension Firestore {

public func referenceToTransactions() -> CollectionReference {

guard let clientID = FirebaseApp.app()?.options.clientID else {

fatalError("No firebase app")

}

Return self.collection("users").document(clientID).collection("transactions")

}

}

При инициализации модель представления TransactionListViewModel подписывается на событие изменения данных на сервере – данные модели, а затем представления обновляются автоматически (по аналогии представления подписаны на изменение TransactionListViewModel). В TransactionListViewModel реализованы функции добавления и удаления транзакций, которые вызывают представления. На листинге 5 представлена функция подписки TransactionListViewModel на событие изменения пользовательских данных на сервере.

Листинг 5 – функция подписки на событие изменения данных на сервере

private func subscribe() {

guard listenerRegistration == nil else {

return

}

let reference = firestore.referenceToTransactions()

listenerRegistration = reference.addSnapshotListener { [weak self] querySnaphot, error in

guard let documents = querySnaphot?.documents else {

print("No documents")

return

}

self?.objectWillChange.send()

self?.transactions = documents.compactMap {

do {

return try $0.data(as: Transaction.self)

} catch {

print(error.localizedDescription)

return nil

}

}

}

На листинге 6 представлены функции добавления и удаления транзакций в TransactionListViewModel. Для построения отображения информации о истории транзакций необходимы функции для подсчета суммы доходов и суммы расходов, функцию для группировки данных о транзакциях по дате, функцию подсчета префиксной суммы баланса пользователя по дням.

Листинг 6 – Функции добавления и удавления транзакций

func add(\_ transaction: Transaction) {

self.objectWillChange.send()

let reference = firestore.referenceToTransactions()

do {

try reference.document(transaction.id).setData(from: transaction)

} catch {

print(error.localizedDescription)

}

}

func remove(at indexSet: IndexSet) {

self.objectWillChange.send()

let reference = firestore.referenceToTransactions()

let IDs = indexSet.map { transactions[$0].id }

IDs.forEach { id in

reference.document(id).delete()

}

}

На листинге 7 и листинге 8 представлены функции для расчета сумм расходов и доходов пользователя и для группировки данных о транзакциях по дате соответственно.

Листинг 7 – Функции для расчета сумм доходов и расходов

/// Cумма доходов

func incomesSum() -> Double {

self.filter{ $0.isExpense == false }.reduce(0) { $0 + $1.amount }

}

/// Сумма расходов

func expensesSum() -> Double {

self.filter{ $0.isExpense == true }.reduce(0) { $0 + $1.amount }

}

Листинг 8 – Функция для группировки данных о транзакциях по дате

// Создает словарь транзакций, сгрупированных по месяцу и году

func makeTransactionGroupByDate(ascending: Bool = false) -> TransactionGroup { guard self.isNotEmpty else {

return [:]

}

let sorted = self.sorted(by: { ascending ? $0.date < $1.date : $0.date > $1.date })

return TransactionGroup(grouping: sorted) { $0.monthAndYear }

}

На листинге 9 представлена функция для подсчета префиксной суммы баланса пользователя по дням.

Листинг 9 – Функция для подсчета префиксной суммы баланса пользователя по дням

/// Создает прификсную сумму транзакций для создания к графиков

func makeTransactionPrefixSum() -> TransactionPrefixSum {

guard self.isNotEmpty else {

return []

}

let today = Date()

let day: TimeInterval = 60 \* 60 \* 24

let month: TimeInterval = day \* 30

let dateInterval = DateInterval(start: today.addingTimeInterval(-month), duration: month)

var sum: Double = 0

var cumulativeSum = TransactionPrefixSum()

for date in stride(from: dateInterval.start, through: today, by: day) {

let formattedDate = date.formatted(date: .numeric, time: .omitted)

let dailyTotal = self.filter{ $0.numericFormattedDate == formattedDate }.reduce(0) { $0 + $1.signedAmount }

sum += dailyTotal

sum = sum.roundedTo2Digits()

cumulativeSum.append((date, sum))

}

if let startDate = cumulativeSum.first(where: { $0.amount != 0 })?.date { cumulativeSum.removeAll(where: { $0.date < startDate })

}

return cumulativeSum

}

* 1. Реализация главного экрана

На главном экране – представлении MainView – отображается столбчатый график истории баланса пользователя за последний месяц, список из пяти последних транзакций и кнопки отображающие текущие суммы доходов и расходов. На листинге 10 показано тело представления MainView.

На листинге 11 показан код функции, создающей дочернее для MainView представление с кнопками добавления транзакций и графиком баланса пользователя. При нажатиях на кнопки добавления транзакций меняются значения флагов isAddingIncome или isAddingExpence, хранимых в MainView, при значении true которых пользователю модально представляется экран добавления транзакции, при этом главный экран, хоть он и виден на заднем плане, становится недоступен для нажатий пользователя до скрытия экрана добавления транзакций.

Листинг 10 – Тело представления MainView

var body: some View {

NavigationStack {

ScrollView {

VStack(alignment: .leading, spacing: 24) {

Header()

ChartWithButtons()

RecentTransactionsList(

transactions: transactionListViewModel.transactions,

onDelete: transactionListViewModel.remove

)

}

.padding()

.frame(maxWidth: .infinity)

}

.toolbarBackground(.hidden, for: .tabBar)

.toolbarBackground(.hidden, for: .automatic)

.scrollIndicators(.never)

.background(

LinearGradient(

gradient: Gradient(colors: [.assetsBackground, .gray]),

startPoint: .top,

endPoint: .bottom

)

)

.navigationBarTitleDisplayMode(.inline)

}

.tint(Color.assetsText)

.presentationCornerRadius(20)

.sheet(isPresented: $isAddingIncome) {

AddTransactionView(

isExpense: false,

onTapSave: { transaction in

transactionListViewModel.add(transaction)

}

)

}

.sheet(isPresented: $isAddingExpense) {

AddTransactionView(

isExpense: true,

onTapSave: { transaction in

transactionListViewModel.add(transaction)

}

)

}}

Листинг 11 – Представление с кнопками добавления транзакций и графиком баланса пользователя

private func ChartWithButtons() -> some View {

HStack(spacing: 12) {

CurrentBalanceBarChart(

data: prefixSum,

currency: .rub

)

.allowsHitTesting(prefixSum.isNotEmpty)

VStack(spacing: 12) {

AddTransactionButton(

amount: imcomesAmount,

currency: .rub,

type: .incomes

) {

withAnimation {

isAddingIncome = true

}

}

AddTransactionButton(

amount: expensesAmount,

currency: .rub,

type: .outcomes

) {

withAnimation {

isAddingExpense = true

}

}

}

}

}

* 1. Реализация экрана добавления новой транзакции

Экран для добавления новой транзакции служит для доступа пользователя к созданию новых записей о доходах или расходах. На экране присутствуют 2 формы для ввода краткого описания транзакции и для ввода ее суммы. Кроме того, на экране добавления новой транзакции есть виджет календаря, позволяющий указать дату транзакции, таблица с категориями транзакций и кнопка сохранения введенных данных.

На листинге 12 представлен код тела представления для добавления новых записей о доходах или расходах. Экран добавления новой транзакции создается в теле главного экрана приложения, при этом в его инициализатор передается анонимная функция сохранения новой транзакции. Вызов функции сохранения происходит при нажатии на кнопку сохранений при соблюдении корректности ввода численного значения новой записи о доходе или расходе.

Листинг 12 – Код тела представления для добавления новых записей о доходах или расходах

var body: some View {

VStack(alignment: .center) {

topBar()

descriptionField()

amountField()

CategoryHGrid(

categories: isExpense ? Category.expenses : Category.incomes,

selectedCategory: $category

)

HStack {

addButton()

datePicker()

}

}

.padding()

.overlay {

GeometryReader { geometry in

Color.clear.preference(key: InnerHeightPreferenceKey.self, value: geometry.size.height)

}

}

.onPreferenceChange(InnerHeightPreferenceKey.self) { newHeight in

sheetHeight = newHeight

}

.presentationDetents([.height(sheetHeight)])

.presentationBackground(

LinearGradient(

gradient: Gradient(colors: [.assetsBackground, .gray]),

startPoint: .top,

endPoint: .bottom

)

)

.presentationCornerRadius(20)

}

Вызов функции сохранения (addAction) виден на листинге 13.

Листинг 13 – Функция создания кнопки сохранения для экрана добавления новой транзакции

private func addButton() -> some View {

Button {

addAction()

label: {

Text("Добавить")

.padding()

.background(Color.systemBackground)

.clipShape(.rect(cornerRadius: 16))

.tint(.assetsText)

}

.buttonStyle(.plain)

}

* 1. Реализация экрана всех транзакций

Экран всех транзакций представляет собой таблицу ячеек записей о доходах и расходах, упорядоченных по дате создания. Таблица поддерживает удаление записей с помощью правого сдвига ячеек, а действие удаления – анонимная функция – передается в инициализатор экрана всех транзакций. Вызов анонимной функции удаления транзакции onDelete() виден на листинге 14.

Листинг 14 – Тело экрана всех транзакций

var body: some View {

VStack {

List {

ForEach(Array(transactions.makeTransactionGroupByDate()), id: \.key)

{ month, transactions in

Section {

// MARK: Transaction list

ForEach(transactions) { transaction in

TransactionRow(transaction: transaction)

}

} header: {

// MARK: Transaction month

Text(month)

}

.listSectionSeparator(.hidden)

}

.onDelete(perform: onDelete)

}

.listStyle(.plain)

}

.navigationTitle("Transactions")

.navigationBarTitleDisplayMode(.inline)

}

* 1. Реализация экрана пользователя

Экран пользователя служит для отображения круговых диаграмм – отчетов о доходах и расходах пользователя по месяцам. Так же на экране есть всплывающее по кнопке меню настроек, в котором можно изменить язык или валюту приложения, а также выйти из аккаунта. Вспомогательным дочерним представлением для экрана пользователя является переключатель SwitchView, позволяющий выбирать вид транзакций – доходы или расходы – отображаемых на круговых диаграммах.

TransactionsPieChart – дочернее представление для экрана пользователя, которое содежит в себе вышеупомянутые диаграммы и таблицы категорий транзакций, содержащихся в диаграммах. Таблицы категорий аналогичны таблице представления истории всех транзакций, а ее ячейки отсортированы по сумме всех транзакций в категории за месяц. На листинге 15 показано тело представления экрана пользователя.

Листинг 15 – Представление экрана пользователя

var body: some View {

NavigationStack {

ScrollView(.vertical) {

VStack(spacing: 0) {

Title(

loginViewModel.currentUser?.profile?.name ?? "loading...",

imageUrl: loginViewModel.currentUser?.profile?.imageURL(withDimension: 32)

)

SwitchView(showsExpenses: $showsExpenses)

.padding(.horizontal)

TransactionsPieChart(

data: transactionsGroup,

currency: .rub,

showsExpenses: showsExpenses

)

Spacer()

}

}

.containerRelativeFrame([.horizontal, .vertical])

.background(

LinearGradient(

gradient: Gradient(colors: [.assetsBackground, .gray]),

startPoint: .top,

endPoint: .bottom

)

)

.toolbarBackground(.hidden, for: .tabBar)

.toolbarBackground(.hidden, for: .automatic)

.scrollIndicators(.never)

.toolbar {

ToolbarItem {

Menu {

Button {

switchLanguage()

} label: {

Label("Сменить язык", systemImage: "book.and.wrench")

}

Button {

switchCurrency()

} label: {

Label("Сменить валюту", systemImage: "arrow.down.left.arrow.up.right.square")

}

Button {

loginViewModel.signOut()

} label: {

Label("Выйти из аккаунта", systemImage: "logOut")

}

} label: {

Image(systemName: "gearshape")

.symbolRenderingMode(.palette)

.foregroundStyle(Color.assetsIcon, .primary)

}

}

}

}

}

* 1. Реализация перехода между главным экраном и экраном пользователя

Переход между главным экраном и экранном пользователя осуществляется с помощью нижней панели с кнопками вкладок приложения. Для реализации данной панели используется нативное представление TabView, которое служит механизмом навигации в приложении, а также контейнером для представления TabBar, которое представляет собой парящую над другими представлениями панель с двумя кнопками вкладок, переключающими состояние текущей вкладки приложения, которое хранится в ContentView.

На листинге 16 показана реализация тела представления ContentView при авторизованном состоянии пользователя, на ней явно видно устройство механизма перехода между главным экраном и экраном пользователя.

Листинг 16 – Тело представления ContentView

private func Content() -> some View {

TabView(selection: $selectedTab) {

MainView(transactionListViewModel: transactionListViewModel)

.tag(Tab.home)

AccountView(loginViewModel: loginViewModel,

transactionListViewModel: transactionListViewModel)

.tag(Tab.account)

}

.overlay(alignment: .bottom) { TabBar(selectedTab: $selectedTab)

.frame(width: 200) }}

**Вывод по главе 4**

Таким образом, была реализована система мобильного iOS-приложения для финансового планирования. Были приведены листинги различных частей приложения, в полной мере раскрывающие особенности его работы.

# 5. ТЕСТИРОВАНИЕ СИСТЕМЫ

Тестирование программного обеспечения – процесс оценки соответствия качества созданного программного продукта ожиданиям от него, поиск ошибок и несоответствий, требующих исправления для улучшения характеристик программы.

Функциональное тестирование – процесс тестирования с целью проверить отсутствие несоответствий функциональных требований приложения и его спецификаций. Функциональное тестирование необходимо чтобы убедится, что приложение работает так, как запланировал разработчик и как ожидает пользователь приложения, что приложение выполняет свои функции корректно.

Для функционального тестирования и тестирования пользовательского интерфейса получившегося приложения, было использовано устройство iPhone 11 и эмулятор iPhone 15 pro с версиями iOS 17.4. Набор тестов и их результаты представлены в таблице 1.

Таблица 1 – Протоколы тестирования системы

| **№** | **Функция** | **Шаги** | **Ожидаемый результат** |
| --- | --- | --- | --- |
| 1 | Корректный запуск приложения | Запустить приложение. | Приложение запустилось. |
| 2 | Аутентификация | 1. Запустить приложение. 2. Пройти аутентификацию. | Пользователь вошел в аккаунт. |
| 3 | Переход на экраны с помощью нижней панели навигации | 1. Запустить приложение. 2. Авторизоваться. 3. Нажать на кнопки нижней панели навигации | Главный экран и экран пользователя сменяют друг друга. |
| 4 | Корректное отображение экранов | 1. Запустить приложение. 2. Авторизоваться. 3. Перейти на разные экраны приложения. | Экраны приложения отображаются корректно. |
| 5 | На главном экране отображается баланс пользователя. | 1. Запустить приложение. 2. Авторизоваться. | На главном экране отображается баланс пользователя. |
| 6 | Добавить новую запись о доходе | 1. Запустить приложение. 2. Авторизоваться 3. Нажать на кнопку добавления нового дохода 4. Заполнить информацию о транзакции на экране добавления дохода 5. Нажать кнопку сохранить | Добавлена новая запись о доходе. |
| 7 | Добавить новую запись о расходе | 1. Запустить приложение. 2. Авторизоваться 3. Нажать на кнопку добавления нового дохода 4. Заполнить информацию о транзакции на экране добавления дохода   Нажать кнопку сохранить | Добавлена новая запись о расходе. |
| 8 | Удалить транзакцию | 1. Запустить приложение. 2. Авторизоваться 3. Перейти на экран всех транзакций 4. Сдвигом ячейки транзакции удалить транзакцию | Удалена транзакций. |
| 9 | Выход из аккаунта | 1. Запустить приложение. 2. Авторизоваться 3. Перейти на экран пользователя 4. Открыть меню 5. Нажать кнопку выхода из аккаунта | Показан экран аутентификации. |
| 10 | Смена языка | 1. Запустить приложение. 2. Авторизоваться 3. Перейти на экран пользователя 4. Открыть меню 5. Нажать кнопку смены языка | Изменился язык приложения. |
| 11 | Смена валюты | 1. Запустить приложение. 2. Авторизоваться 3. Перейти на экран пользователя 4. Открыть меню 5. Нажать кнопку смены валюты | Изменилась валюта приложения. | |
| 12 | Смена типа отображаемых транзакций на графике на экране пользователя | 1. Запустить приложение. 2. Авторизоваться 3. Перейти на экран пользователя 4. Нажать несколько раз на кнопки выбора типа транзакций | Изменяется тип отображаемых транзакций на графике на экране пользователя. | |

**Вывод по главе 5**

По результатам проведенного тестирования приложения для финансового планирования несоответствий работы программы с ожидаемым результатом не выявлено, все функции приложения работают корректно.

# ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано iOS-приложение для финансового планирования. При этом были решены следующие задачи.

1. Проведен анализ предметной области и обзор существующих аналогичных мобильных приложений.
2. Проведен анализ требований к разрабатываемому приложению.
3. Спроектировано мобильное iOS-приложение.
4. Реализовано и протестировано мобильное iOS-приложения для финансового планирования.

В ходе выполнения курсовой работы были изучены способы разработки мобильных приложений для платформы iOS и язык программирования Swift.

Планируется дальнейшее развитие проекта, включающее в себя добавление следующих новых функций в приложение: регулярные автоматических транзакции, создаваемые пользователем, совместное ведение бюджета вместе с другими пользователями, постановка целей на определенный период с периодическим напоминанием пользователю о них с помощью уведомлений.

# ЛИТЕРАТУРА

1. Финансовая грамотность [Электронный ресурс] URL: <https://www.vtb.ru/articles/chto-takoe-finansovaya-gramotnost/> (дата обращения: 10.10.2024г.)
2. Money manager, expense tracker [Электронный ресурс] URL: <https://apps.apple.com/ru/app/money-manager-expense-tracker/id1510997753?l=en-GB> (дата обращения: 10.10.2024г.)
3. Buddy: Money & Budget planner [Электронный ресурс] URL: <https://apps.apple.com/ru/app/buddy-money-budget-planner/id936422955?l=en-GB> (дата обращения: 10.10.2024г.)
4. Organizze [Электронный ресурс] URL: <https://apps.apple.com/ru/app/organizze-budget-planner/id677699286?l=en-GB> (дата обращения: 10.10.2024г.)
5. Усов В. У76 Swift. Основы разработки приложений под iOS, iPadOS и macOS. 6-е изд. дополненное и переработанное. – СПб.: Питер, 2021. – 544 с.: ил. (Серия «Библиотека программиста») (дата обращения: 10.10.2024г.).
6. Пайлон Т. Программируем для iPhone и iPad. – СПб.: Питер, 2014. – 336 с. (дата обращения: 10.10.2024г.).
7. Swift: Компилируемый язык программирования общего назначения: [Электронный ресурс] URL: <https://developer.apple.com/swift/> (дата обращения: 10.10.2024г.).
8. Харазян А. Язык Swift. Самоучитель. – СПб.: БХВ-Петербург, 2016. – 176 с. (дата обращения: 10.10.2024г.).
9. Грей Э. Swift. Карманный справочник. Программирование в среде iOS и ОS X. – М.: Вильямс, 2016. – 288 с. (дата обращения: 10.10.2024г.).
10. Марк Д. Swift. Разработка приложений в среде Xcode для iPhone и iPad с использованием iOS SDK. – М.: Вильямс, 2016. – 816 с. (дата обращения: 10.10.2024г.).
11. Документация SwiftUI. [Электронный ресурс] URL: https://developer.apple.com/documentation/swiftui (дата обращения: 10.10.2024г.)
12. Wals D. Mastering iOS 10 Programming. – Birmingham: Packt Publishing, 2016. – 543 с. (дата обращения: 10.10.2024г.).
13. Bessarabova E. MVP vs. MVC vs. MVVM vs.VIPER. What is Better For iOS Development? [Электронный ресурс] URL: https://themindstudios.com/blog/mvp-vs-mvc-vs-mvvm-vs-viper/ (дата обращения: 10.10.2024г.).
14. Human Interface Guidelines: Рекомендации по интерфейсу для iOS: [Электронный ресурс] URL: https://developer.apple.com/design/human-interface-guidelines/ (дата обращения: 10.10.2024г.).
15. UIKit: Набор элементов, из которых состоит интерфейс приложения: [Электронный ресурс] URL: https://developer.apple.com/documentation/uikit. (дата обращения: 10.10.2024г.).
16. App Store: магазин приложений для iOS устройств. [Электронный ресурс] URL: https://www.apple.com/app-store/ (дата обращения: 10.10.2024г.).

# ПРИЛОЖЕНИЕ

Спецификация вариантов использования

**Таблица 1**. Спецификация вариантов использования «Создание записи о доходе или расходе»

|  |
| --- |
| Прецедент**:** Создание записи о доходе или расходе. |
| ID: 1 |
| Аннотация: Пользователь создает запись о доходе или расходе. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Пользователь нажал на главном экране на кнопку добавления дозода или кнопку добавления расхода. |
| Основной поток:   1. Пользователь вводит описание записи о доходе или расходе. 2. Пользователь вводит численное значение суммы записи. 3. Пользователь выбирает категорию дохода или расхода. 4. Пользователь выбирает дату записи. |
| Постусловия: Добавлена запись о доходе или расходе. |
| Альтернативные потоки: Нет. |

**Таблица 2.** Спецификация вариантов использования «Просмотр последних пяти записей о доходах или расходах»

|  |
| --- |
| Прецедент**:** Просмотр последних пяти записей о доходах или расходах |
| ID: 2 |
| Аннотация: Пользователь просматривает список последних пяти записей о доходах или расходах. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Пользователь находится на главном экране. |
| Основной поток:   1. Пользователь на главном экране просматривает список последних пяти записей о доходах или расходах. |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |

**Таблица 3.** Спецификация вариантов использования «Просмотр всех записей о доходах или расходах»

|  |
| --- |
| Прецедент**:** Просмотр всех записей о доходах или расходах |
| ID: 3 |
| Аннотация: Пользователь просматривает список всех записей о доходах или расходах. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Пользователь нажал на главном экране кнопку просмотра всех записей о доходах и расходах. |
| Основной поток:   1. Пользователь на экране всех записей доходов и расходов просматривает список всех записей о доходах или расхода. |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |

**Таблица 4.** Спецификация вариантов использования «Удалить запись о доходе или расходе»

|  |
| --- |
| Прецедент**:** Удалить запись о доходе или расходе. |
| ID: 3.1 |
| Аннотация: Пользователь удаляет запись о доходе или расходе. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Пользователь просматривает список всех записей о доходах или расходах. |
| Основной поток:   1. Правым сдвигом ячейки дохода или расхода пользователь удаляет запись о доходе или расходе |
| Постусловия: Удалена запись о доходе или расходе. |
| Альтернативные потоки: Нет. |

**Таблица 5.** Спецификация вариантов использования «Просмотр статистики расходов и доходов по категориям»

|  |
| --- |
| Прецедент**:** Просмотр статистики расходов и доходов по категориям. |
| ID: 4 |
| Аннотация: Пользователь просматривает статистик расходов и доходов по категориям. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Пользователь находится на экране аккаунта. |
| Основной поток:   1. Пользователь просматривает на экране аккаунта статистику расходов и доходов по категориям |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |

**Таблица 6.** Спецификация вариантов использования «Авторизация»

|  |
| --- |
| Прецедент: Авторизация. |
| ID: 5 |
| Аннотация: Неавторизованный пользователь авторизуется. |
| Главные актеры: Неавторизованный пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Пользователь является неавторизованным. |
| Основной поток:   1. Пользователь нажимает на кпопку авторизации на экране авторизации. 2. Пользователь следует указаниям на экране. |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |