Федеральное государственное образовательное бюджетное учреждение

высшего образования

**«Финансовый университет при Правительстве**

**Российской Федерации»**

**(Финансовый университет)**

Колледж информатики и программирования

(наименование структурного подразделения)

Дипломный проект

Тема «Разработка мобильного приложения “Анализ семейного бюджета”»

(наименование)

Студент (ка) Лебедева Александра Михайловна

(фамилия, имя, отчество полностью)

Учебная группа 4ПКС-420

Специальность 09.02.03 Программирование в компьютерных системах

(код и наименование специальности)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Руководитель  дипломного проекта |  |  |  | И.В. Сибирев |
|  |  | (подпись) |  | (инициалы, фамилия) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Консультант  дипломного проекта |  |  |  |  |
| (при наличии) |  | (подпись) |  | (инициалы, фамилия) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Председатель предметной  (цикловой) комиссии |  |  |  | Т.Г. Аксёнова |
|  |  | (подпись) |  | (инициалы, фамилия) |

Москва – 2024 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc168049251)

[Глава 1. Предпроектное исследование предметной области 5](#_Toc168049252)

[1.1. Актуальность проблемы управления семейным бюджетом 5](#_Toc168049253)

[1.2. Методы управления семейными финансами 9](#_Toc168049254)

[1.3. Обзор существующих мобильных приложений для анализа семейного бюджета 14](#_Toc168049255)

[1.4. Особенности разработки мобильных приложений на платформе Flutter 17](#_Toc168049256)

[Глава 2. Разработка программного обеспечения 19](#_Toc168049257)

[2.1. Анализ требований к мобильному приложению 19](#_Toc168049258)

[2.2. Проектирование архитектуры мобильного приложения 23](#_Toc168049259)

[2.3. Архитектура мобильного приложения 27](#_Toc168049260)

[2.4. Разработка кода приложения на языке программирования Dart 30](#_Toc168049261)

[2.5. Тестирование и отладка мобильного приложения 31](#_Toc168049262)

[2.6. Разработка руководства пользователя для мобильного приложения «Анализ семейного бюджета» 34](#_Toc168049263)

[ЗАКЛЮЧЕНИЕ 38](#_Toc168049264)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ИНТЕРНЕТ-РЕСУРСОВ 40](#_Toc168049265)

[ПРИЛОЖЕНИЕ 42](#_Toc168049266)

# **ВВЕДЕНИЕ**

В эпоху цифровизации экономики и социализации информационных технологий, мобильные приложения занимают прочное место в ежедневной жизни современного человека. Одной из значимых потребностей является управление семейным бюджетом, которое становится особенно актуальным в условиях глобальной экономической неопределенности и повышенных требований к финансовой грамотности населения. Эти обстоятельства определяют актуальность темы дипломного проекта - разработки мобильного приложения «Анализ семейного бюджета» на Flutter.

Объектом исследования данного проекта является семейный бюджет в контексте его планирования, контроля и анализа. Семейный бюджет представляет собой финансовую систему, формируемую доходами и расходами семьи, что делает его неотъемлемой частью ежедневной жизни каждого человека, стремящегося к рациональному распределению средств в условиях ограниченных ресурсов.

Предметом исследования выступает процесс анализа семейного бюджета, предусматривающий систематизацию данных о доходах и расходах с целью оптимизации финансового положения и достижения устойчивости и экономической независимости домохозяйства. В рамках дипломного проекта особое внимание уделяется разработке мобильной платформы на Flutter, как современного и эффективного инструмента для реализации данных задач.

Основной целью дипломного проекта является создание мобильного приложения, позволяющего пользователям в режиме реального времени анализировать свой семейный бюджет, оптимизировать расходы и планировать доходы, что, в свою очередь, способствовало бы повышению их финансовой грамотности и улучшению общего благосостояния.

Задачи исследования включают:

1. Исследование теоретических основ анализа и управления семейным бюджетом.

2. Обзор существующих мобильных приложений для управления финансами и выявление их достоинств и недостатков.

3. Проектирование архитектуры приложения на Flutter, уделяя внимание удобству интерфейса и функциональности.

4. Разработка и тестирование приложения.

В процессе дипломного проектирования использовались как теоретические, так и эмпирические методы научного исследования, включая анализ и синтез для обработки теоретического материала, индукцию и дедукцию в процессе обобщения полученных результатов, а также методы моделирования и прототипирования в рамках разработки программного продукта.

Теоретической значимостью дипломного проекта является глубокое изучение проблематики анализа и управления семейным бюджетом, а практическая значимость заключается в реализации функционального мобильного приложения, способного оказать существенную помощь пользователям в улучшении финансового состояния их семей.

Таким образом, дипломный проект включает в себя широкий круг рассматриваемых проблем от теоретических аспектов финансового планирования до практической реализации мобильного приложения, что делает его актуальным и значимым для широкого круга потенциальных пользователей.

# **Глава 1. Предпроектное исследование предметной области**

## **Актуальность проблемы управления семейным бюджетом**

Бюджет – это финансовый план, который включает в себя как доходы, так и расходы. В современном мире управление бюджетом стало важным аспектом обеспечения финансовой устойчивости и достижения финансовых целей.

Семейный бюджет представляет собой конкретный случай бюджета, который охватывает доходы и расходы семьи на определенный период времени. Это инструмент, который позволяет контролировать движение денег внутри семьи. Он не только помогает минимизировать необдуманные траты, но и отражает отношение членов семьи к финансам.

В семейном бюджете есть две составляющие – доходная часть и расходная часть:

Семья получает доходы и тратит их в виде потребительских расходов. Доходы – это деньги или материальные ценности, получаемые от предприятия, отдельного лица или какого-либо вида деятельности.

Все доходы семьи делятся на:

1. денежные;

Денежные доходы населения — суммы денег, получаемые и аккумулируемые населением. Основными источниками денежных доходов населения являются оплата труда, доход от предпринимательской деятельности, социальные выплаты, реализация продукции личного подсобного хозяйства, поступления из финансовой системы. Например:

* зарплаты вместе с различными начислениями и доплатами;
* пенсий, пособий, стипендий и других социальных и страховых

выплат;

* доходов от предпринимательской деятельности;
* доходов от операций с личным имуществом и денежными

накоплениями.

1. предоставление льгот;

Льгота — определённые преимущества, дополнительные права, полное или частичное освобождение от выполнения установленных правил, обязанностей, или облегчение условий их выполнения.

Льготы получают некоторые категории населения, определенные государственными органами. Например, льготные проездные билеты для студентов и учащихся

1. натуральные;

Натуральные доходы — это экономическая выгода, полученная физлицом в виде переданных ему товаров, работ или услуг. С подобных доходов, получаемых работниками, компания, являющаяся их источником, должна удержать НДФЛ и перечислить его в бюджет.

Натуральные доходы состоят в основном из материальных благ, получаемых на приусадебном участке (овощи, фрукты, мясо, яйца, мед и т. п.), подарков, выигрышей, продуктового довольствия и др.

Доходы расходуют на товары и услуги, необходимые для жизни человека и существования семьи, таким образом, доходы превращаются в расходы.

Расходы – это затраты, издержки, потребление чего-либо для определенных целей.

Все расходы семьи можно разделить на обязательные и произвольные:

1. Обязательные расходы включают в себя ежемесячные траты на питание, квартплату, коммунальные услуги, одежду, транспорт и другие неотложные нужды, которые семья должна удовлетворить регулярно.
2. Произвольные расходы менее структурированы и зависят от текущих потребностей семьи. Это могут быть различные развлечения, неожиданные траты и другие расходы, которые возникают по мере необходимости.

Теперь выделим плюсы ведения семейного бюджета для пользователя:

1. Наглядный вид:

Самая главная причина, по которой стоит начать учет личных финансов, это то, что доходы и расходы станут прозрачными и не придется в конце месяца вспоминать, на что была потрачена каждая сумма.

1. Возможность изыскать резервы для экономии:

Занимаясь учетом личных финансов, можно выяснить, какие расходы действительно важные, а от каких можно отказаться. Проанализировав расходы, можно понять, что, скорее всего, есть на что сэкономить.

1. Учет долгов:

Осуществляя такой учет, пользователь может держать под контролем активы и пассивы бюджета. Пользователю проще помнить, что было одолжено или взято в долг.

1. Возможность быстрее достичь поставленных целей:

Практически у каждого человека есть какие-либо финансовые цели, но без контроля над своими финансами достижение таких целей становится затруднительным, а иногда и вовсе невозможным.

1. Установление порядка в жизни:

Ведение домашнего бюджета способствует установлению порядка в жизни пользователя и достижению финансовой свободы. Это позволяет избавиться от необходимости держать в голове цифры, все доходы и расходы, депозиты и инвестиции, долги и кредиты. Все это возможно контролировать, научившись вести учет своих финансов.

Учитывая описанные выше аспекты управления семейным бюджетом, актуальность разработки мобильного приложения «Анализ семейного бюджета» на Flutter обусловлена рядом социально-экономических и технологических тенденций современного общества:

1. Возрастающая сложность финансовой среды. Современный мир характеризуется высокой степенью экономической нестабильности и изменчивостью доходов семей. Это требует от каждой семьи более активного подхода к планированию своего бюджета, учету расходов и доходов, что способствует более эффективному управлению своими средствами.

2. Цифровизация финансовых процессов. Общество становится всё более ориентированным на цифровые решения, которые способны значительно облегчить и оптимизировать привычные процессы, включая управление семейным бюджетом. Мобильное приложение для учета финансов может стать мощным инструментом в руках пользователей, предоставляя им под рукой все данные о своих финансах.

3. Растущее внимание к личным финансам. В атмосфере глобализации и информационного перенасыщения пользователи всё более стремятся контролировать свои финансы, рационализируя расходы и вкладывая в своё финансовое будущее. Прозрачность и контроль становятся ключевыми критериями в управлении личными и семейными бюджетами.

4. Необходимость индивидуального подхода. Семейный бюджет и потребности в управлении им отличаются от семьи к семье. Подход «один для всех» неэффективен, поэтому приложение, способное предложить гибкие настройки и адаптацию под конкретные потребности пользователя, имеет огромное значение.

5. Образовательный аспект. Приложение не только поможет в учёте финансов, но и способствует повышению финансовой грамотности пользователей. Вместе с аналитическими возможностями приложение может предлагать рекомендации и советы по оптимизации бюджета, основываясь на лучших практиках и поведенческих паттернах.

6. Технологические инновации. Использование Flutter как кроссплатформенного инструмента для разработки мобильного приложения позволяет создать продукт, который хорошо функционирует на Android. Это обеспечит широкий охват аудитории и упростит техническую поддержку и развитие продукта.

7. Важность пользовательского опыта. Современные пользователи ожидают не только функциональности от приложений, но и удобства их использования. Разработка на Flutter позволяет создать высокопроизводительный и визуально привлекательный интерфейс, который сделает процесс управления финансами не только полезным, но и приятным.

Необходимо дополнить, что разрабатываемое мобильное приложение предназначено для одной категории пользователей: членов семьи.

Основной целью приложения является предоставление пользователям инструментов для отслеживания доходов, расходов и управления общим бюджетом семьи.

Приложение ориентируется на легкость использования, а важность такого приложения диктуется как существующими вызовами управления семейным бюджетом, так и потенциалом для улучшения качества жизни пользователей через оптимизацию их финансов.

## **Методы управления семейными финансами**

Теперь рассмотрим аспекты эффективного распределения и использования семейного бюджета, различные методы, подходы к управлению финансами, которые являются актуальными в системе управления семейным бюджетом

1. Традиционный метод

Традиционный метод управления семейными финансами предполагает ведение записей доходов и расходов вручную. Это может осуществляться с использованием бумажных средств, таких как записные книжки или таблицы на бумаге. Особенность этого подхода заключается в его простоте и доступности, помимо этого, он не требует использования специализированного программного обеспечения или устройств. Однако он требует регулярных усилий по ведению записей и анализу данных, а также может приводить к ошибкам из-за человеческого фактора.

Анализ традиционного метода управления семейными финансами вдохновил меня на интеграцию определённых его аспектов в разрабатываемое мобильное приложение, сохраняя при это его основные преимущества, такие как наглядность и простота использования, и преодолевая его недостатки посредством автоматизации. Приложение предлагает пользователям удобный интерфейс для ввода данных о доходах и расходах с возможностью просмотра исторических данных в форме списка, графиков или диаграмм, что значительно упрощает процесс анализа и планирования семейного бюджета.

Таблица 1. Сравнительная таблица традиционного метода

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| 1. Доступность | 1. Вероятность ошибок |
| Для традиционного метода не требуются специальные устройства или программы — достаточно простой записной книжки и ручки. | Вручную введённые данные могут содержать ошибки из-за человеческого фактора. |
| 2. Наглядность | 2. Регулярное обновление |
| Ведение записей вручную помогает некоторым пользователям лучше осмысливать свои расходы и доходы, визуализируя финансовый поток на бумаге. | Традиционный метод требует постоянного и систематического обновления информации, что может оказаться время затратным. |
| 3. Простота | 3. Сложность анализа |
| Этот метод не требует особых навыков или знаний для его использования. | Для получения обобщённой информации и анализа трендов требуются дополнительные усилия для систематизации и суммирования данных. |

Таким образом, хотя традиционный метод и имеет свои ограничения, его основные принципы лежат в фундаменте концепции моего проекта и помогут обеспечить его актуальность и полезность для пользователей.

1. Метод 50/30/20

Метод 50/30/20 — это подход к управлению семейным бюджетом, который предлагает распределение доходов следующим образом: 50% дохода выделяется на необходимые расходы (жилье, питание, коммунальные услуги и т.д.), 30% — на желаемые траты (развлечения, хобби, отпуск), а оставшиеся 20% откладываются на сбережения и инвестиции. Этот метод помогает в установлении баланса между текущими потребностями и финансовыми целями на будущее.

Ключевая идея метода 50/30/20 заключается в разделении семейного дохода на три категории:

* 50% на «Потребности»

Эта категория включает в себя обязательные расходы, которые необходимы для поддержания базового уровня жизни, такие как жилье, питание, коммунальные услуги, транспорт и т.д.

* 30% на «Желания»

Данная категория охватывает расходы на развлечения, хобби, отпуск, рестораны и прочие необязательные траты, которые повышают качество жизни и удовлетворяют интересы членов семьи.

* 20% на «Будущее»

Эта категория предназначена для накопления средств на будущее, что может включать в себя сбережения на пенсию, инвестиции, погашение долгов или реализацию крупных финансовых целей.

Таблица 2. Сравнительная таблица метода 50/30/20

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| 1. Сбалансированность | 1. Ограниченность для низкодоходных семей |
| Метод 50/30/20 помогает семьям находить компромисс между текущими потребностями, желаниями и долгосрочными финансовыми целями. | Не всегда остается достаточно средств для других категорий после покрытия базовых потребностей. |
| 1. Гибкость и адаптивность | 1. Недостаточная детализация категорий |
| Метод 50/30/20 можно адаптировать под финансовые возможности семьи. | Требуется более детальное разделение расходов для эффективного управления финансами. |
| 1. Развитие финансовой дисциплины | 1. Сложность в определении приоритетов |
| Метод помогает развить осознанность в расходах и формировать полезные финансовые привычки. | Метод не предоставляет четких рекомендаций по определению приоритетов финансовых целей. |
| 1. Достижение финансовых целей | 1. Риск упрощения |
| Выделение 20% на сбережения помогает работать над долгосрочными целями. | Метод может создать иллюзию, что управление финансами сводится к механическому распределению доходов. |

Метод 50/30/20 реализовывается в мобильном приложении «Анализ семейного бюджета» при поддержке и желании пользователя. Приложение отслеживает расходы пользователей в реальном времени и предоставляет визуальные показатели расходов по каждой категории, что поможет пользователям не выходить за пределы установленных границ и интегрировать данный метод в свою финансовую жизнь.

1. Метод управления денежными потоками

Метод управления денежными потоками (Cash Flow Management) ставит своей целью анализ и оптимизацию притока и оттока денежных средств в течение определенного периода времени. Основная концепция метода заключается в достижении положительного денежного потока, предполагающего превышение доходов над расходами. В этом контексте метод предлагает пользователям данный функционал:

* Регулярный мониторинг притока и оттока денежных средств, включая поступления зарплаты, пенсии, инвестиций и прочих доходов, а также расходов на продукты питания, жилье, транспорт, образование, развлечения и прочие нужды.
* Идентификация статей расходов, которые не приносят существенной пользы или удовольствия, что позволяет сократить траты и высвободить дополнительные средства.
* Анализ возможностей оптимизации существующих расходов, например, поиск более выгодных предложений на товары и услуги, использование альтернативных способов транспортировки или применение принципов энергоэффективности для экономии на коммунальных платежах.

Таблица 3. Сравнительная таблица метода управления денежными потоками

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| 1. Полный контроль над финансами | 1. Трудоемкость и временные затраты |
| Возможность управлять всеми финансовыми транзакциями и оптимизировать расходы. | Ведение детальных записей требует много времени и усилий. |
| 2. Улучшение финансовой дисциплины | 2. Риск ошибок и неточностей |
| Регулярное отслеживание денежных потоков способствует более ответственному подходу к тратам. | Вручную вводимые данные могут содержать ошибки, искажающие общую картину финансов. |
| 3. Возможность прогнозирования и планирования | 3. Необходимость регулярного обновления данных |
| Анализ исторических данных помогает планировать будущие финансовые потоки и устанавливать реалистичные цели. | Для точного анализа необходимо постоянно обновлять информацию о транзакциях. |
| 4. Выявление возможностей для экономии | 4. Сложность анализа данных |

Продолжение таблицы 3

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| Детальное отслеживание расходов помогает выявить области для сокращения расходов. | Анализ больших объемов данных без специальных инструментов является сложным и затратным. |

Метод управления денежными потоками вносит существенный вклад в функционал приложения. Интегрируя в приложение инструменты автоматической или ручная организация расходов по категориям (еда, жилье, транспорт, развлечения и т.д.) для отслеживания динамики трат и идентификации потенциальных возможностей для экономии, а также предоставление пользователям наглядных и интерактивных графиков и диаграмм, иллюстрирующих динамику притока и оттока денежных средств, что облегчит выявление слабых мест в управлении финансами.

1. SMART-цели в планировании семейного бюджета

Ключевой аспект метода SMART - цели должны быть:

* Специфичными (Specific). Отвечают на вопрос, что именно вы хотите достичь, например, «Накопить на первый взнос по ипотеке».
* Измеримыми (Measurable). Должны иметь конкретную единицу измерения, какой результат считается достижением цели, например, «Накопить 500 000 рублей».
* Достижимыми (Attainable). Цель должна быть реальной, и вы должны обладать ресурсами и возможностями для ее достижения, например, исходя из доходов и расходов, необходимо определить реалистичную сумму ежемесячных накоплений.
* Реалистичными (Relevant). Цель должна быть актуальной для вашей текущей финансовой ситуации и соответствовать вашим финансовым приоритетам, например, прежде чем накопить на новый автомобиль, возможно необходимо создать или пополнить «подушку безопасности».
* С ограничением во времени (Time-bound). Каждой цели должен быть определён конкретный срок достижения, например, «Копить на первый взнос по ипотеке в течение 2 лет».

Таблица 4. Сравнительная таблица SMART-цели

|  |  |
| --- | --- |
| Преимущества | Недостатки |
| 1. Конкретность и ясность | 1. Негибкость и ограниченность |
| Четкое определение финансовых приоритетов и целей. | Трудности в адаптации к изменяющимся обстоятельствам. |
| 2. Измеримость и отслеживание прогресса | 2. Сложность в балансировании множества целей |
| Возможность оценить прогресс в достижении целей. | Требуется тщательное планирование и приоритезация. |
| 3. Реалистичность и достижимость | 3. Риск чрезмерной концентрации на количественных показателях |
| Установление реалистичных и осуществимых целей. | Возможность упустить качественные аспекты финансов. |
| 4. Временные рамки и срочность | 4. Необходимость регулярного пересмотра и адаптации |
| Создание ощущения срочности и мотивации к действию. | Требует дополнительных временных и умственных ресурсов. |

С развитием технологий стало возможным использование специализированных программ и мобильных приложений для управления семейными финансами. Эти инструменты автоматизируют процесс ввода, хранения и анализа финансовой информации, предлагая пользователям графики, отчеты и рекомендации по оптимизации бюджета. Данное разрабатываемое мобильное приложение «Анализ семейного бюджета» является одним из них.

## **Обзор существующих мобильных приложений для анализа семейного бюджета**

Функциональные возможности мобильного приложения FamilyBudget включают в себя:

Регистрация и аутентификация пользователей:

* Пользователи могут создавать учетные записи, предоставляя необходимую информацию.
* Система обеспечивает безопасную аутентификацию пользователей для доступа к персонализированным функциям.

Управление профилем:

* Пользователи могут добавлять, обновлять и управлять своими личными данными.
* Возможность вносить изменения в информацию о пользователе.

Учет доходов и расходов:

* Возможность регистрации и отслеживания всех доходов и расходов пользователей.
* Классификация транзакций по различным категориям, таким как питание, транспорт, развлечения и др.
* Указание дат транзакций для точного учета финансов.

Аналитика:

* Генерация статистики по финансовым операциям и бюджетированию.
* Использование диаграмм и графиков для визуализации общего бюджета, анализа тенденций расходов и доходов.
* Возможность сравнения данных за определенный промежуток времени.

Эти функциональности обеспечивают пользователям полноценный инструмент управления семейным бюджетом.

FamilyBudget — Это приложение предлагает обширный набор функциональных возможностей, позволяющих пользователям эффективно управлять своими финансами. Среди основных возможностей приложения: регистрация и аутентификация пользователей, управление профилем, учет доходов и расходов, бюджетирование и аналитика. Отличительной особенностью FamilyBudget является высокий уровень интерфейса и удобства использования, что делает его привлекательным выбором для пользователей, стремящихся к эффективному управлению своими финансами.

Рассмотрим уже существующие программы для управления семейным бюджетом:

1Money — приложение, где также реализованы основные функциональные возможности: регистрация и аутентификация пользователей, управление профилем, учет доходов и расходов, бюджетирование, отчетность и аналитика. Имеется доступ на разных платформах, что делает его более универсальным. Однако удобство использования оценивается как среднее, и присутствует платный контент [8].

Wallet — приложение с высоким уровнем удобства использования, предоставляющее регистрацию и аутентификацию, управление профилем, учет доходов и расходов, бюджетирование, отчетность и аналитику. Поддерживает многопользовательскую функциональность и уведомления о превышении бюджета. Интеграция с банковскими счетами и доступ на разных платформах сделали его привлекательным вариантом. Присутствует платный контент [9].

Мой бюджет - приложение с низким уровнем удобства использования, предоставляющее основные функциональные возможности, такие как регистрация и аутентификация пользователей, управление профилем, учет доходов и расходов. Однако у него отсутствует многопользовательская поддержка, уведомления о превышении бюджета, и стоит отметить, что приложение платное [10].

Таблица 5. Сравнительная таблица приложений ведения семейного бюджета

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Критерии | Family-  Budget | 1 Money | Wallet | Мой бюджет |
| 1 | Регистрация и аутентификация | Да | Да | Да | Да |
| 2 | Управление профилем | Да | Да | Да | Да |
| 3 | Учет доходов и расходов | Да | Да | Да | Да |
| 4 | Бюджетирование | Да | Да | Да | Да |
| 5 | Отчетность и аналитика | Да | Да | Да | Да |
| 6 | Многопользовательская поддержка | Нет | Нет | Да | Да |
| 7 | Уведомления о превышении бюджета | Нет | Нет | Да | Нет |
| 8 | Интерфейс и удобство использования | Высокий | Средний | Высокий | Низкий |
| 9 | Доступность на разных платформах | Нет | Да | Да | Да |
| 10 | Интеграция с банковскими счетами | Нет | Да | Нет | Да |
| 11 | Стоимость использования | Бесплатно | Присутствует  платный контент | Присутствует  платный контент | Платно |

Из данного раздела мы можем сделать вывод, что FamilyBudget обладает широкими функциональными возможностями и удобством использования, но отстает от конкурентов по поддержке многопользовательской функциональности и уведомлениям о превышении бюджета. Приложения 1Money и Wallet предоставляют конкурентоспособные функции, но Wallet выделяется высоким уровнем удобства использования и интеграцией с банковскими счетами, хотя они предоставляют платный контент. Мой бюджет, в свою очередь, ограничен функционалом, но является бесплатным вариантом. Выбор между ними зависит от конкретных потребностей пользователя и их готовности использовать платный контент.

## **Особенности разработки мобильных приложений на платформе Flutter**

В процессе разработки мобильных приложений важно учитывать ряд ключевых факторов, таких как эффективность работы приложения, его доступность на разных операционных системах, дизайн пользовательского интерфейса, а также поддержка со стороны сообщества разработчиков. В этой главе мы углубимся в анализ платформы Flutter и обсудим, какие преимущества она предлагает для создания мобильного приложения «Анализ семейного бюджета».

1. Компиляция и производительность

Использование языка Dart и компиляция в нативный код обеспечивают высокую производительность приложений на платформе Flutter. Это критически важно для приложения, так как оно будет работать с финансовыми данными и требует быстрого и отзывчивого интерфейса.

1. Разработка одного исходного кода для всех платформ

Flutter позволяет разрабатывать приложения для Android и iOS из одной кодовой базы, что упрощает процесс разработки и экономит время. Это особенно важно для нашего проекта, так как мы можем сконцентрироваться на разработке функционала, а не на адаптации под различные платформы.

1. Интерфейс и пользовательское оформление

Flutter предлагает множество инструментов для создания кастомизированного пользовательского интерфейса с использованием разнообразных виджетов.

1. Горячая перезагрузка

Функция горячей перезагрузки (Hot Reload) в Flutter позволяет мгновенно видеть результаты изменений в коде без необходимости перезапуска приложения. Это существенно сокращает время, затраченное на разработку и тестирование нового функционала, а также позволяет быстро реагировать на обратную связь от пользователей и вносить необходимые изменения.

1. Сообщество и поддержка

Flutter поддерживается Google и имеет активное сообщество разработчиков, что обеспечивает доступ к обширной документации, библиотекам и разбору ошибок при производстве.

В нашем дипломном проекте выбор Flutter для разработки мобильного приложения обусловлен его преимуществами в пользовательском оформлении, кроссплатформенности, производительности и удобстве. Это позволит создавать приложение с комфортом и максимальной экономией времени.

# **Глава 2. Разработка программного обеспечения**

## **Анализ требований к мобильному приложению**

**Диаграмма вариантов использования.**

Use Case Diagram – это диаграмма вариантов использования. Диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Актор (Use Case) – это роль объекта вне системы, который прямо взаимодействует с ее частью — конкретным элементом.

На изображении представлена диаграмма использования для мобильного приложения «Анализ семейного бюджета». Диаграмма показывает взаимодействие пользователя с приложением и различные функциональные возможности, которые предоставляет приложение.

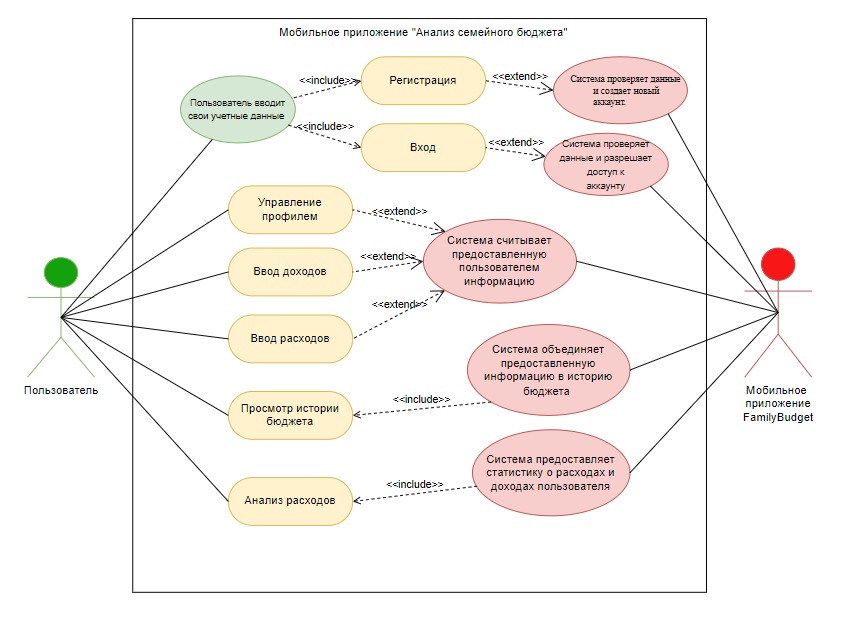


Рисунок 1. Диаграмма вариантов использования мобильного приложения

На диаграмме изображены два основных актора: «Пользователь» и «Мобильное приложение FamilyBudget». Они взаимодействуют друг с другом через различные функции, предоставляемые приложением.

Функциональные возможности приложения включают:

* Регистрация: пользователь может зарегистрироваться в системе, вводя необходимые учетные данные.
* Вход: процедура аутентификации пользователя в системе.
* Управление профилем: возможность для пользователя изменять свои учетные данные и другую информацию в своем профиле.
* Ввод доходов: пользователь может вводить информацию о своих доходах, указывая источники и суммы.
* Ввод расходов: функция для учета расходов пользователя или семьи.
* Просмотр истории бюджета: пользователь может просматривать предыдущие записи о доходах и расходах.
* Анализ расходов: возможность проводить детальный анализ семейных расходов с помощью приложения.

Взаимодействие между пользователем и приложением описывается сценариями использования:

* Регистрация: пользователь вводит учетные данные, система проверяет их и создает новый аккаунт.
* Вход: пользователь вводит логин и пароль, система проверяет данные и разрешает доступ к аккаунту.
* Управление профилем: пользователь изменяет информацию в своем профиле, система считывает и обрабатывает предоставленные данные.
* Ввод доходов и расходов: пользователь вводит информацию о доходах и расходах, система считывает их и сохраняет.
* Просмотр и анализ истории бюджета: пользователь может просматривать обзор своего бюджета и анализировать расходы с помощью предоставленных отчетов.

Таким образом, диаграмма демонстрирует функциональность и взаимодействие основных элементов приложения, необходимых для эффективного управления семейным бюджетом.

**Диаграмма бизнес-процессов.**

Business Process Model and Notation (нотация моделирования бизнес-процессов) — это система условных обозначений, которая отображает бизнес-процессы с помощью блок-схем. BPMN диаграмма показывает в какой последовательности совершаются рабочие действия и перемещаются потоки информации.

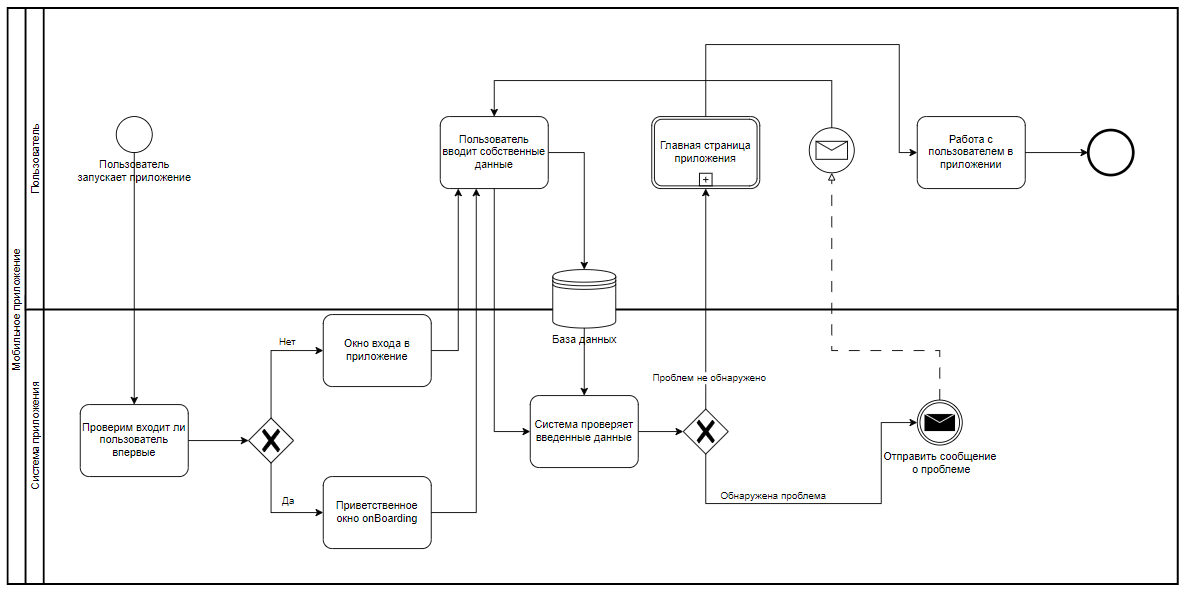


Рисунок 2. BPMN диаграмма

Это блок-схема, описывающая процесс работы с мобильным приложением. Вот последовательность шагов, представленных на диаграмме:

1. «Пользователь запускает приложение» – начало процесса.
2. «Проверим входит ли пользователь впервые» – условие, где система определяет, входит ли пользователь в приложение впервые.
   1. Если да, то процесс переходит к шагу «Приветственное окно onBoarding».
   2. Если нет, то процесс идет к шагу «Окно входа в приложение».
3. «Пользователь вводит свои данные» – пользователь вводит информацию для входа в систему.
4. «Система проверяет введенные данные» – здесь данные проверяются на соответствие с базой данных.
5. «Проблем не обнаружено» – условие, определяющее успешность проверки данных.
   1. Если данные верны и проблем не обнаружено, процесс переходит к шагу «Главная страница приложения», что соответствует успешному входу в приложение.
   2. Если же обнаружена проблема, процесс переходит к шагу «Отправить сообщение о проблеме», где, пользователь уведомляется о проблеме авторизации.
6. «Работа с пользователем в приложении» – финальный этап, когда пользователь успешно вошел в приложение и начинает им пользоваться.

Фигуры, используемые в диаграмме, включают в себя условные обозначения блок-схем, такие как прямоугольники для процессов, ромбы для принятия решений, параллелограммы для ввода/вывода данных и круги для начала и конца процесса. Также присутствует база данных, обозначенная схематичным изображением двух цилиндров. Сообщение об ошибке представлено в виде конверта с галочкой.

**Функциональная диаграмма верхнего уровня.**

Функциональная диаграмма верхнего уровня (FDVУ) — это графическое представление основных функций и их взаимосвязей в системе или процессе. Она служит для общего понимания структуры и целей системы на высоком уровне и предоставляет абстрактное отображение ключевых функциональных элементов.

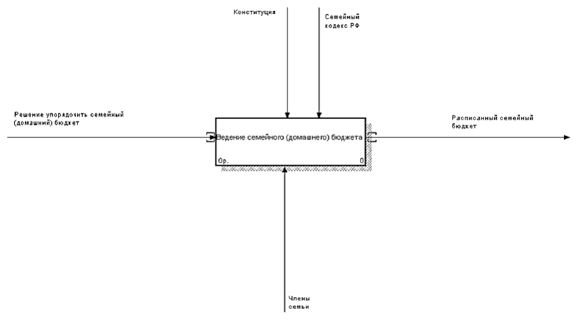


Рисунок 3. Функциональная диаграмма верхнего уровня

FDVУ предоставляет целостное представление о том, как функции взаимодействуют в системе на самом высоком уровне, в данной работе диаграмма представляет модель взаимодействия законодательства (конституции), семейного бюджета и уровня жизни в каком-то исследовании или аналитическом материале, связанном с экономикой или социологией.

## **Проектирование архитектуры мобильного приложения**

Построение ER-диаграммы

Диаграмма «Сущность-связь» (ER-диаграмма) – это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы.

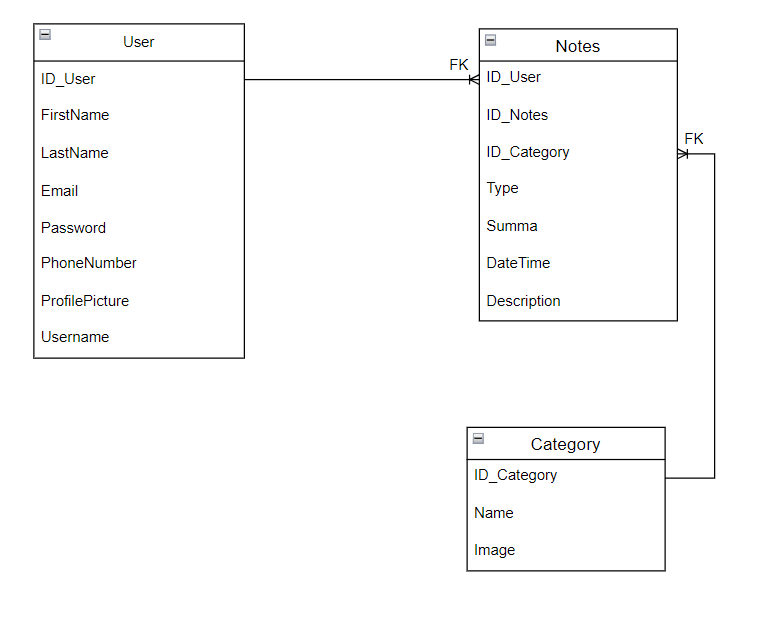


Рисунок 4. ER-диаграмма в нотации Баркера

На изображении представлена диаграмма, которая описывает структуру базы данных. Диаграмма состоит из трех сущностей: User, Notes и Category. Сущности связаны друг с другом линиями, обозначающими взаимосвязи между ними.

1. Сущность User имеет следующие атрибуты: ID\_User, FirstName, LastName, Email, Password, PhoneNumber, ProfilePicture, Username. Это могут быть данные пользователей системы.
2. Сущность Notes включает в себя атрибуты: ID\_Notes, ID\_User, ID\_Category, Type, Summa (вероятно, сумма), DateTime, Description. Это может быть таблица, содержащая записи или заметки, созданные пользователями, распределенные по категориям, с указанием времени создания и описания.
3. Сущность Category содержит атрибуты: ID\_Category, Name, Image. Это описание категорий, которые могут использоваться в рамках системы, например, для классификации заметок.

Линии с меткой "FK" обозначают внешние ключи (Foreign Keys), которые используются для установления связей между сущностями:

* Сущность User связана с Notes через внешний ключ ID\_User. Это означает, что каждая заметка в таблице Notes ассоциирована с определенным пользователем из таблицы User.
* Сущность Notes связана с Category через внешний ключ ID\_Category. Это показывает, что заметки могут быть отнесены к определенным категориям из таблицы Category.

Диаграмма описывает, как данные о пользователях, их заметках и разных категориях этих заметок организованы и связаны между собой в базе данных.

Данная диаграмма базы данных (рисунок 4) реализована в Firebase.

Firebase — это облачная платформа, предоставляемая компанией Google, которая предлагает различные инструменты для разработки мобильных и веб-приложений. Одним из ключевых сервисов Firebase является база данных в реальном времени, которая предоставляет возможность хранения и синхронизации данных между различными клиентами приложений.

Для реализации базы данных, используемой в мобильном приложении «Анализ семейного бюджета» на платформе Firebase, используется следующая структура:

1. Коллекция «Users»
2. Документы в этой коллекции содержат информацию о зарегистрированных пользователях приложения.
3. Каждый документ имеет уникальный идентификатор (ID\_User), который также является внешним ключом для связи с другими коллекциями.
4. Атрибуты документа соответствуют атрибутам сущности «User» из ER-диаграммы: FirstName, LastName, Email, Password, PhoneNumber, ProfilePicture, Username.

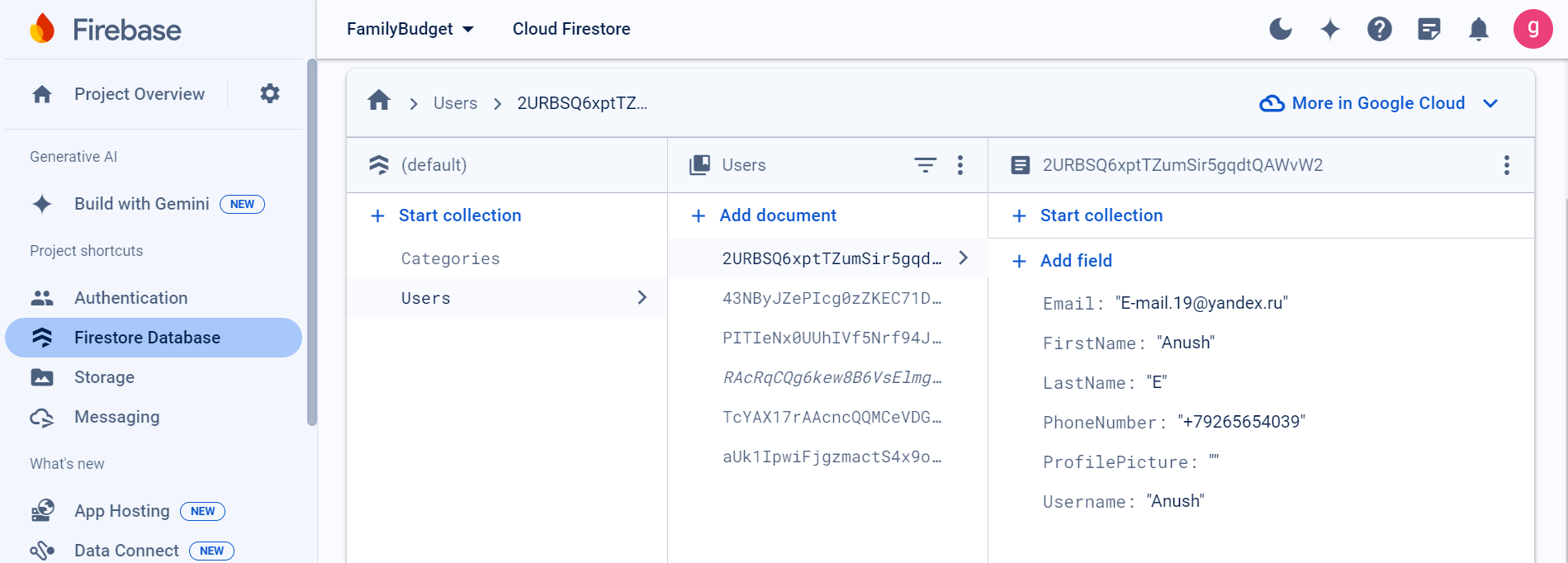


Рисунок 5. Коллекция «User» в Firebase

1. Коллекция «Notes»
   1. Документы в этой коллекции содержат информацию о заметках пользователей.
   2. Каждый документ имеет уникальный идентификатор (ID\_Notes).
   3. Атрибуты документа соответствуют атрибутам сущности «Notes» из ER-диаграммы: ID\_User, ID\_Category, Type, Summa, DateTime, Description.
   4. Для установления связи с пользователями и категориями используются внешние ключи ID\_User и ID\_Category.

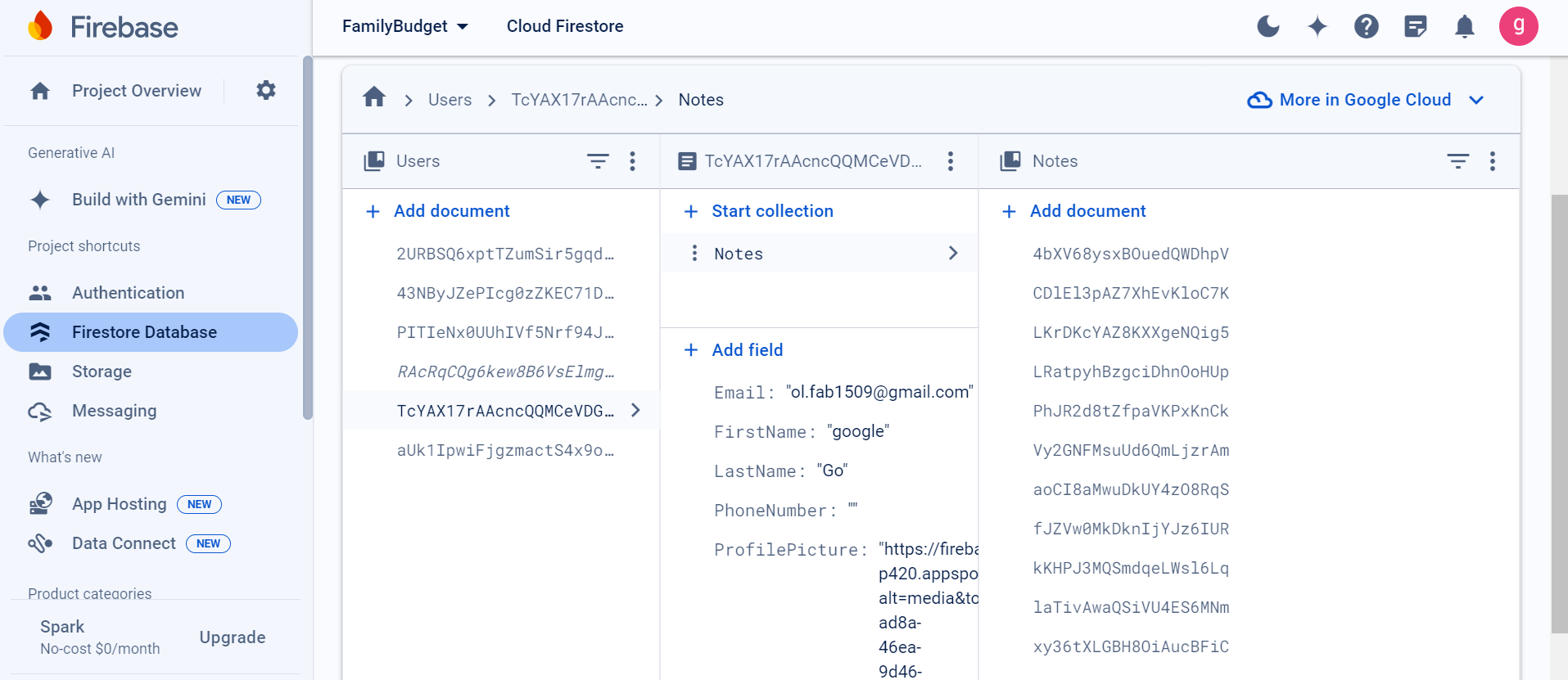


Рисунок 6. Коллекция «Notes» в Firebase

1. Коллекция «Categories»
   1. Документы в этой коллекции содержат информацию о категориях заметок.
   2. Каждый документ имеет уникальный идентификатор (ID\_Category).
   3. Атрибуты документа соответствуют атрибутам сущности «Category» из ER-диаграммы: Name, Image.

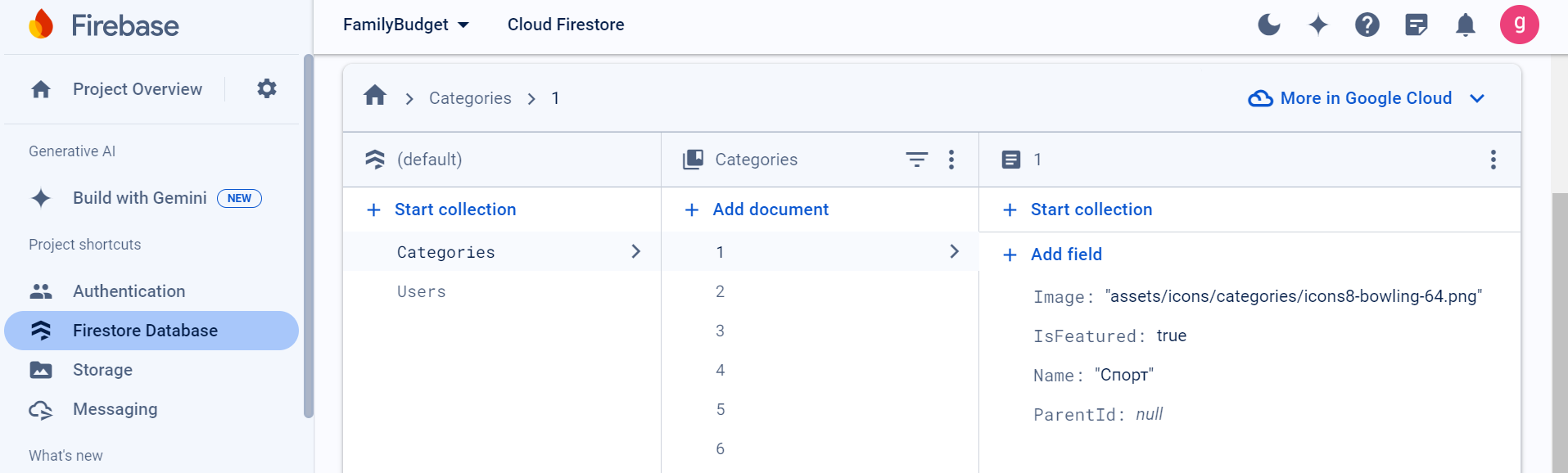


Рисунок 7. Коллекция «Categories» в Firebase

Связи между коллекциями установлены следующим образом:

Коллекция «Notes» имеет поле «ID\_User», которое ссылается на соответствующий документ в коллекции «Users».

Коллекция «Notes» также имеет поле «ID\_Category», которое ссылается на соответствующий документ в коллекции «Categories».

Таким образом, база данных Firebase организована так, чтобы хранить информацию о пользователях, их заметках и категориях этих заметок, с учетом связей между этими сущностями, как описано в ER-диаграмме (рисунок 4).

## **Архитектура мобильного приложения**

На данный момент существует множество разнообразных методов реализации архитектуры мобильных приложений, однако Google, официальный разработчик операционной системы Android, предоставляет официальное руководство по созданию эффективной архитектуры в мобильных приложениях. На рисунке 8 изображена схема предлагаемой архитектуры для мобильного приложения.

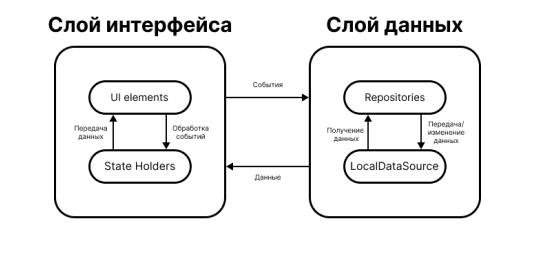


Рисунок 8. Архитектура мобильного приложения

Архитектура включает в себя два основных слоя: слой интерфейса и слой данных. Слой данных содержит всю бизнес-логику мобильного приложения. Он состоит из следующих компонентов:

1. Репозитории (Repositories) — это классы, которые взаимодействуют с источниками данных, используемыми в приложении. Они отвечают за предоставление данных слою пользовательского интерфейса, управление изменениями данных и разрешение конфликтов между различными источниками данных.
2. Источники данных (Data Sources) — это классы, которые обрабатывают данные из различных источников. Они отличаются от репозиториев тем, что являются оболочкой над источниками и отвечают только за получение необработанных данных и передачу их в репозиторий. Чаще всего используются два типа источников данных: сетевой источник данных (Network Data Source), который взаимодействует с сервером, и локальный источник данных (Local Data Source), который работает с локальными файлами или базой данных.

Слой интерфейса содержит логику отображения данных приложения на экране. Каждый раз, когда данные изменяются из-за действий пользователя или внешнего ввода, пользовательский интерфейс обновляется, чтобы отразить эти изменения.

Слой интерфейса включает следующие компоненты:

1. Элементы пользовательского интерфейса (UI Elements) — это классы, которые отображают пользовательский интерфейс. Все элементы создаются с использованием класса View или функций Jetpack Compose.
2. Хранители состояния (State Holders) — это классы, которые сохраняют состояния экранов и обрабатывают логику событий, такие как ViewModels.

В мобильном приложении будет использоваться архитектурный паттерн Model-View-ViewModel (MVVM). MVVM — это архитектурный паттерн, который разделяет пользовательский интерфейс (View) от бизнес-логики (Model) с помощью промежуточного слоя (ViewModel).

Рассмотрим основные компоненты MVVM:

* Модель (Model)

Представляет бизнес-логику и данные приложения. Модель может быть независимым компонентом или отражать структуру базы данных. Она содержит данные и методы для работы с ними.

* Представление (View)

Отвечает за отображение данных и взаимодействие с пользователем. Это может быть графический интерфейс пользователя (GUI) или другой способ представления информации. Представление должно быть пассивным и не содержать логики или состояния приложения.

* Модель-представление (ViewModel)

Служит прослойкой между моделью и представлением. Он содержит логику приложения, преобразует данные из модели в формат, удобный для представления, и обрабатывает пользовательские действия, передавая соответствующие команды и запросы обратно в модель. ViewModel обычно реализует интерфейс, доступный для представления.

На рисунке 9 представлена схема работы паттерна MVVM в мобильном приложении.

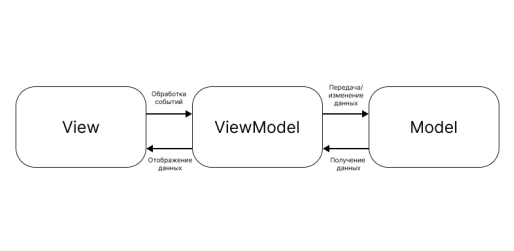


Рисунок 9. Схема Model-View-ViewModel

## **Разработка кода приложения на языке программирования Dart**

В рамках моего дипломного проекта, посвященного разработке мобильного приложения «Анализ семейного бюджета» на платформе Flutter, я занималась созданием ключевых компонентов приложения с использованием языка программирования Dart. Одним из важных аспектов приложения является его способность проводить анализ семейного бюджета, путем вычисления общей суммы, доходов и расходов пользователей. Сейчас мы поговрим о разработанном классе NotesCalculator, который выполняет указанные расчеты.

**Структура класса NotesCalculator**

NotesCalculator – это класс, который отвечает за выполнение технических расчетов бюджета на основе предоставленных данных о транзакциях. Он принимает список транзакций (notes), каждая из которых представлена моделью NotesModel, содержащей информацию о типе операции (type) и сумме (summa).

**Расчет общей суммы**

Метод totalSum вычисляет общую сумму всех транзакций. Для этого он проходит по списку транзакций и суммирует суммы доходов, вычитая расходы. Таким образом, пользователи могут видеть текущий баланс своего бюджета.

**Расчет суммы доходов**

Метод incomeSum вычисляет сумму всех доходов пользователей за определенный период. Он фильтрует список транзакций, оставляя только те, которые относятся к доходам, и затем суммирует их значения.

**Расчет суммы расходов**

Метод expenseSum аналогично вычисляет сумму всех расходов пользователей за определенный период времени. Он фильтрует список транзакций, оставляя только те, которые относятся к расходам, и затем суммирует их значения.

Листинг 1. Класс NotesCalculator

import '../models/notes\_model.dart';

/// calculate total sum, income sum, and expense sum

class NotesCalculator {

double totalSum(notes) {

return notes.fold(0.0, (previousValue, note) {

if (note.type == 'Доход') {

return previousValue + double.parse(note.summa);

} else {

return previousValue - double.parse(note.summa);

}

});

}

double incomeSum(List<NotesModel> notes) {

return notes

.where((note) => note.type == 'Доход')

.fold(0.0, (previousValue, note) => previousValue + double.parse(note.summa));

}

double expenseSum(List<NotesModel> notes) {

return notes

.where((note) => note.type == 'Расход')

.fold(0.0, (previousValue, note) => previousValue + double.parse(note.summa));

}

}

## **Тестирование и отладка мобильного приложения**

В ходе разработки моего дипломного проекта было проведено тестирование пользовательского интерфейса (UI), чтобы гарантировать стабильное и удобное взаимодействие для конечных пользователей. Вот подробное описание этого этапа тестирования:

1. Тестирование совместимости на различных устройствах
2. Цель тестирования: проверить, что приложение корректно отображается и работает на различных устройствах с разными размерами экранов и версиями операционной системы.
3. Методика: приложение устанавливалось и запускалось на различных устройствах, включая Xiaomi Redmi 8T с Android 11.0 и эмулятор Pixel 2 с Android 14, чтобы убедиться в адаптивности интерфейса и функциональности.
4. Результаты: Приложение успешно адаптировалось под разные размеры экранов и надежно работало на всех тестируемых версиях Android, начиная с 11.0.
5. Тестирование текстовых элементов
6. Цель тестирования: гарантировать высокое качество текстовых элементов, включая правильность орфографии, грамматики и пунктуации.
7. Методика: все тексты в приложении были внимательно проверены на наличие орфографических, грамматических и пунктуационных ошибок.
8. Результаты: тексты в приложении были грамотно составлены, что обеспечило профессиональный и приятный интерфейс для пользователей.
9. Тестирование пользовательского взаимодействия и интуитивности интерфейса
10. Цель тестирования: убедиться, что интерфейс приложения понятен и удобен.
11. Методика: несколько пользователей были приглашены для выполнения типичных задач, например, добавление транзакции, просмотр истории и изменении данных пользователя.
12. Результаты: большинство пользователей успешно справились с основными задачами, что подтверждает высокую интуитивность интерфейса и навигации приложения.

Проведенное тестирование пользовательского интерфейса на различных этапах разработки помогло выявить и успешно устранить ряд недочетов, связанных с совместимостью, текстовым содержимым и взаимодействием пользователя. Полученные результаты подтверждают готовность пользовательского интерфейса мобильного приложения «Анализ семейного бюджета» к более широкому использованию и дальнейшему развитию. Тем не менее, процесс тестирования и улучшения интерфейса будет продолжаться для обеспечения наивысшего уровня пользовательского опыта.

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определенных условиях решать задачи, нужные пользователям. Для тестирования приложения были использованы устройства, описанные в разделе тестирования пользовательского интерфейса. Набор тестов представлен в таблице 6.

Таблица 6. Таблица функциональных тестов

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название теста | Шаги | Ожидаемый результат |
| 1 | Тестирование приветственных экранов | Запуск приложения  Переход между приветственными экранами  Использование кнопки «Пропустить» | Корректное отображение экранов, возможность пропуска приветствия |
| 2 | Тестирование процесса входа в приложение через ввод данных | Ввод электронной почты и пароля  Установка галочки «Запомнить меня»  Восстановление пароля через кнопку «Забыли пароль?» | Успешный вход в приложение, возможность восстановления пароля |
| 3 | Тестирование процесса входа в приложение через Google Account | Нажатие на иконку Google  Выбор учетной записи или добавление новой | Успешный вход через Google, предоставление необходимых данных |
| 4 | Тестирование процесса создания аккаунта | Нажатие на кнопку «Создать аккаунт»  Ввод необходимой информации (имя, фамилия, эл. почта, телефон, пароль)  Согласие с политикой конфиденциальности  Подтверждение электронной почты | Успешное создание аккаунта, возможность входа в приложение |
| 5 | Тестирование главного экрана приложения | Просмотр кошелька и истории транзакций  Добавление новой транзакции  Редактирование и удаление транзакций  Просмотр графиков анализа бюджета | Корректное отображение данных, возможность взаимодействия с транзакциями и графиками |
| 6 | Тестирование фильтрации транзакций | Нажатие кнопки фильтрации на главном экране  Ввод тестовых значений для фильтрации  Применение фильтра | Отображение транзакций с установленным фильтром |
| 7 | Тестирование функций управления пользователем | Переход в настройки пользователя  Редактирование данных пользователя  Удаление аккаунта пользователя  Выход из системы | Отображение данных пользователя, их редактирование, удаление аккаунта и выхода из системы |

## **Разработка руководства пользователя для мобильного приложения «Анализ семейного бюджета»**

Первый раз входя в приложение "Анализ семейного бюджета", вы увидите серию приветственных экранов, которые расскажут вам о ключевых функциях приложения (рисунок 10).



Рисунок 10. Приветственное окно мобильного приложения

Для перехода между приветственными экранами используйте свайп влево или вправо, либо нажмите кнопку внизу экрана, чтобы быстро перейти к соответствующему разделу, нажмите «Пропустить», чтобы сразу приступить к использованию приложения.

При входе в мобильное приложение возможны два варианта:

1. Ввод данных
   * Просто введите свой адрес электронной почты и пароль.
   * При желании, чтобы приложение зафиксировало ваши данные для входа, поставьте галочку «Запомнить меня».
   * Забыли пароль? Нажмите на кнопку «Забыли пароль?» и следуйте инструкциям для восстановления.
   * Когда все данные введены, нажмите на кнопку "Войти".
2. Вход через Google Account
   * Просто нажмите на иконку Google (G).
   * Выберите свою учетную запись из списка или добавьте новую.
   * Перед входом, ознакомьтесь с тем, какие данные будут предоставлены приложению Google.
   * Как только все готово, следуйте инструкциям для завершения входа.

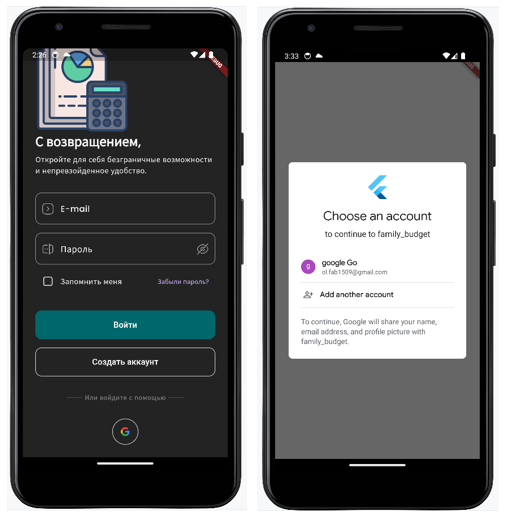


Рисунок 11. Вход в мобильное приложение

Если у вас еще нет аккаунта, вы можете создать его, нажав на кнопку «Создать аккаунт» на начальном экране. Затем вам потребуется ввести такую информацию, как ваше имя, фамилия, электронная почта и номер телефона. Не забудьте придумать и ввести надежный пароль, который будет защищать ваш аккаунт от взлома, он должен содержать заглавную букву, специальный символ и быть в длину не менее 6 символов.

После этого необходимо согласиться с нашей политикой конфиденциальности и нажать кнопку «Создать аккаунт». Если вы предпочитаете использовать свой аккаунт Google для регистрации, просто нажмите соответствующую кнопку. Далее на почту вам придет электронное письмо для подтверждения вашего регистрационного адреса. Просто откройте его и перейдите по ссылке, завершив процесс регистрации.

Поздравляем! После подтверждения электронной почты (рисунок 12) вы можете войти в приложение и начать управлять своим семейным бюджетом.

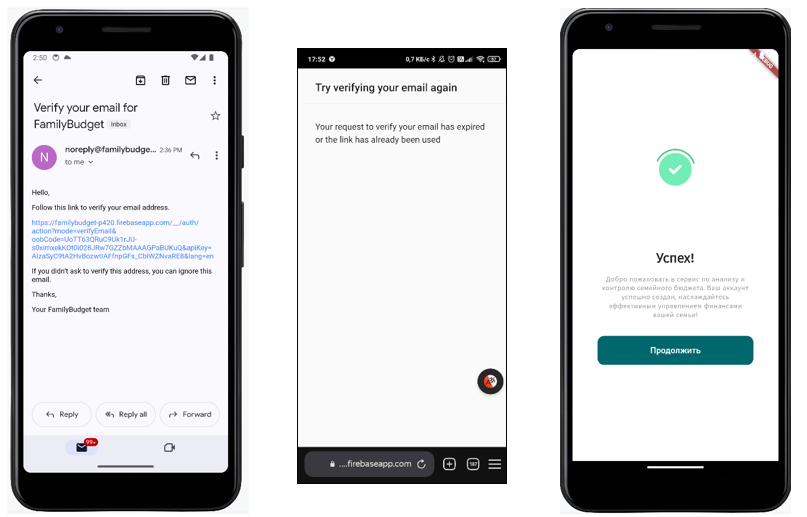


Рисунок 12. Подтверждение электронной почты

После входа вы окажетесь на главном экране, где сразу же сможете увидеть ваш «Кошелёк» и историю ваших финансовых транзакций. Эта история отображает каждую транзакцию с указанием типа, суммы, даты, категории и описание.

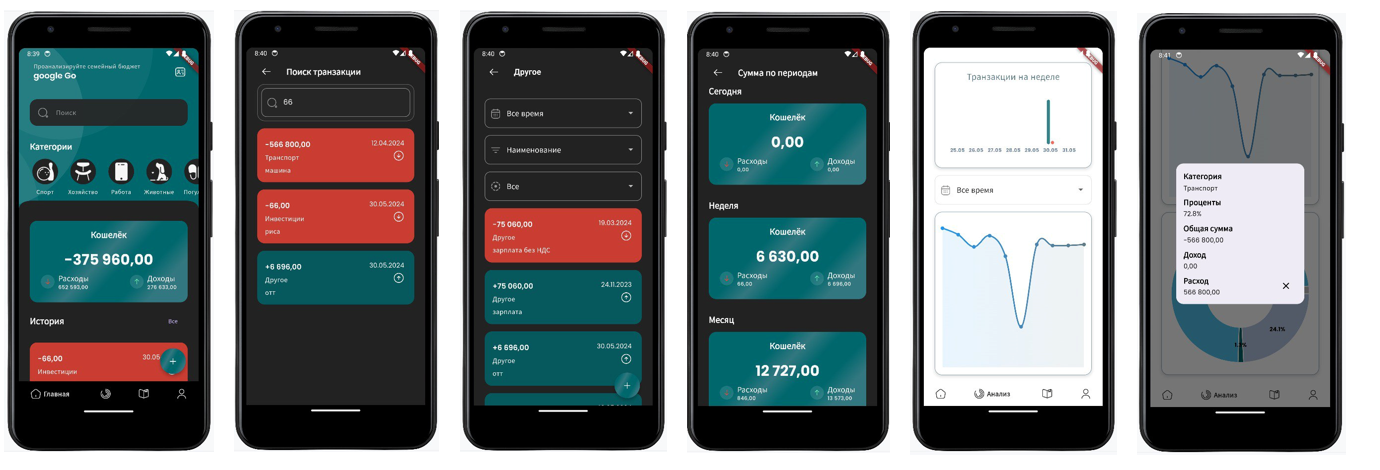


Рисунок 13. Главная страница, история и фильтрация транзакций

Когда наступит время добавить новую транзакцию, просто нажмите кнопку (+) в правом нижнем углу. Затем введите все необходимые данные: тип транзакции (доход или расход), сумму, дату, категорию, метод оплаты и, если нужно, добавьте описание. Когда все будет готово, нажмите «Создать», чтобы сохранить эту транзакцию в вашей истории.

Если у вас появится необходимость изменить или удалить какую-то транзакцию, просто найдите ее в вашей истории и свайпните влево для редактирования или удаления.

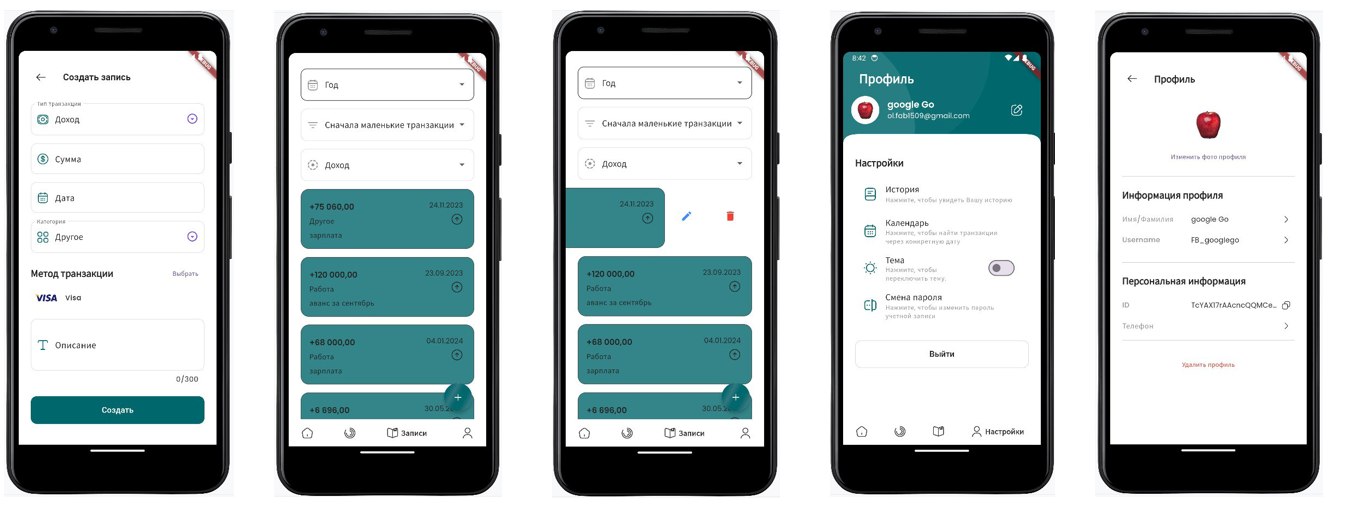


Рисунок 14. Создание, редактирование, удаление транзакции

И не забывайте использовать графики для более глубокого понимания вашего семейного бюджета на основе ваших финансовых транзакций.

1. График за неделю (столбчатая диаграмма доходов и расходов в течении недели от текущей даты)
2. Линейный график (меняется в зависимости от выбранного периода, при наведении на точку графика предоставляется информация о ней)
3. Круговая диаграмма (меняется в зависимости от выбранного периода, строится в зависимости от общей суммы транзакций в какой-либо категории)
4. Горизонтальная столбчатая диаграмма (дополняет круговую диаграмму, при нажатии на график отображает информацию о нём)

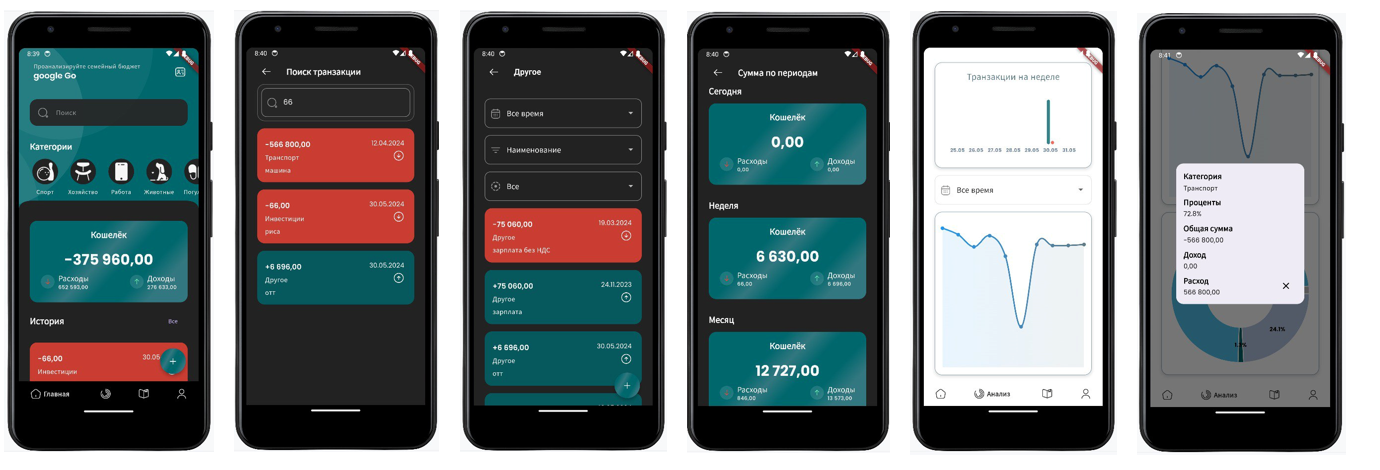


Рисунок 15. Построение графиков на основе ваших транзакций

# **ЗАКЛЮЧЕНИЕ**

Завершая работу над дипломным проектом «Разработка мобильного приложения «Анализ семейного бюджета»», хочется подчеркнуть его значимость в современном контексте цифровой трансформации. Этот проект направлен на предоставление людям инструмента для эффективного анализа и контроля своих финансов, обеспечивая им удобство и практичность в использовании. Приложение будет способствовать повышению уровня финансового планирования и даст возможность пользователям более осознанно управлять своими средствами, достигая при этом лучшего качества жизни и экономической стабильности.

В рамках данного проекта были достигнуты следующие цели:

* Проведено глубокое теоретическое исследование основ управления семейным бюджетом, что помогло выявить ключевые аспекты финансового планирования.
* Изучены существующие мобильные приложения для управления бюджетом, что позволило выявить их сильные и слабые стороны и определить направления для улучшения нашего проекта.
* Разработана архитектура приложения с учетом удобства пользовательского интерфейса и функциональности, что способствует повышению его привлекательности и эффективности использования.

Приложение было успешно реализовано и протестировано, подтвердив свою функциональность и практическую ценность.

Основным результатом данного дипломного проекта является создание мобильного приложения, предоставляющего пользователям удобный инструмент для анализа и управления семейным бюджетом. Это приложение поможет пользователям улучшить свой финансовый контроль и позволит им более осознанно планировать свои финансовые ресурсы.

Теоретическая значимость данного проекта заключается в исследовании и систематизации знаний в области управления семейным бюджетом, а также в применении современных технологий разработки мобильных приложений. Практическая же значимость проекта проявляется в создании конкретного продукта, способствующего повышению финансовой грамотности и улучшению управления семейными финансами.

Таким образом, результаты данного дипломного проекта являются важным шагом в области разработки мобильных приложений для управления семейным бюджетом, и предоставляют основу для дальнейших исследований и разработок в этой области.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ИНТЕРНЕТ-РЕСУРСОВ**

1. Методические рекомендации по подготовке и защите дипломного проекта по специальности 09.02.03 Программирование в компьютерных системах / сост. Т.Г. Аксёнова. – Москва, 2023. – 41 с.
2. Рябоконь О.С., Кукарцев В.В. Новый язык структурного веб-программирования Dart. // Актуальные проблемы Авиации и Космонавтики, 2013. – № 9. – С. 436–437.
3. Коркин Р.И. Финансовая грамотность населения как фактор обеспечения экономической безопасности государства // Теория и практика общественного развития. – 2017. – № 12.
4. Эрих Г., Ридчард Х., Ральф Д., Джон В. Design Patterns Elements of Reusable Object-Oriented Software. // Addison Wesley, 1994. – 395 с.
5. Карл В., Джой Б. Разработка требований к программному обеспечению. // Русская редакция, 2014. – 737 с.
6. Карпович Е. Е. Методы тестирования и отладки программного обеспечения: учебник. – Москва: МИСИС, 2020. – 136 с.
7. Санникова Л.В., Сичинава З.И. Мобильное приложение для автоматизации семейного бюджета [Электронный ресурс] // Инновационная наука. – 2017. – №5. – С. 124 – 126. URL: [https://aeterna-ufa.ru/sbornik/IN\_5-2017.pdf](https://aeterna-ufa.ru/sbornik/IN_5-2017.pdf%20) (дата обращения 27.09.2023)
8. Раздел приложения «1Money – расходы, финансы, бюджет» сайта «Google play» [Электронный ресурс]. URL: [https://play.google.com/store/apps/details?id=org.pixelrush.moneyiq](https://play.google.com/store/apps/details?id=org.pixelrush.moneyiq%20) (дата обращения 11.10.2023).
9. Раздел приложения «Wallet – деньги, бюджет» сайта «Google play» [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=com.droid4you.application.wallet> (дата обращения 11.10.2023).
10. Раздел приложения «Мой бюджет» сайта «Google play» [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=com.onetwoapps.mh> (дата обращения 07.12.2023).
11. IDE и редакторы кода для разработчиков. Подборка. [Электронный ресурс] URL: [https://habr.com/ru/company/serverspace/blog/693374](https://habr.com/ru/company/serverspace/blog/693374%20) (дата обращения: 09.02.2023 г.).
12. Android vs iOS market share – 2022 Statistics [Infographic]. [Электронный ресурс] URL: [https://en.downloadastro.com/magazine/android-vs-ios-market-share](https://en.downloadastro.com/magazine/android-vs-ios-market-share%20) (дата обращения: 23.01.2024 г.).
13. Android Studio: среда разработки мобильных приложений. [Электронный ресурс] URL: <https://arduinoplus.ru/android-studio> (дата обращения: 07.02.2024 г.).
14. Firebase Guides. [Электронный ресурс] URL: [https://firebase.google.com/docs/guides](https://firebase.google.com/docs/guides%20) (дата обращения: 25.02.2024).
15. Flutter Documentation. [Электронный ресурс] URL: https://flutter.dev/docs (дата обращения: 14.04.2024).
16. Библиотека firecloud\_firebase [Электронный ресурс] URL: [https://pub.dev/packages/cloud\_firestore](https://pub.dev/packages/cloud_firestore%20) (дата обращения: 10.04.2024).
17. Библиотека http запросов [Электронный ресурс] URL: [https://pub.dartlang.org/packages/http](https://pub.dartlang.org/packages/http%20) (дата обращения: 15.03.2024).
18. Библиотека визуализации данных charts\_flutter. [Электронный ресурс] URL: [https://pub.dartlang.org/packages/charts\_flutter](https://pub.dartlang.org/packages/charts_flutter%20) (дата обращения: 27.03.2024).
19. Документация Android SDK. [Электронный ресурс] URL: [https://developer.android.com/studio](https://developer.android.com/studio%20) (дата обращения: 23.01.2024)
20. Основы архитектуры приложений на Flutter: Vanilla, Scoped Model, BLoC. [Электронный ресурс] URL: [https://habr.com/ru/post/438574/](https://habr.com/ru/post/438574/%20) (дата обращения: 10.03.2024).

# **ПРИЛОЖЕНИЕ**

Приложение № 1

Листинг 2. Виджет навигационного меню (NavigationMenu)

|  |
| --- |
| import 'package:family\_budget/features/\_familybudget/screens/home/home.dart'; import 'package:family\_budget/features/personalization/screens/settings/setting.dart'; import 'package:family\_budget/utils/helpers/helper\_functions.dart'; import 'package:flutter/material.dart'; import 'package:get/get.dart'; import 'package:google\_nav\_bar/google\_nav\_bar.dart'; import 'package:iconsax/iconsax.dart';  import 'features/\_familybudget/screens/notes/widgets/sort\_all.dart';   class NavigationMenu extends StatelessWidget {  const NavigationMenu({super.key});   @override  Widget build(BuildContext context) {  final controller = Get.put(NavigationController());  final darkMode = THelperFunctions.*isDarkMode*(context);   return Scaffold(  bottomNavigationBar: Obx(  () => GNav(  rippleColor: darkMode ? Colors.*white*.withOpacity(0.25) :  Colors.*black*.withOpacity(0.25),  haptic: true,  tabBorderRadius: 100,  curve: Curves.*easeInOutCubic*,  duration: const Duration(milliseconds: 700),  gap: 5,  color: darkMode ? Colors.*white*.withOpacity(0.8) : Colors.*black*.withOpacity(0.8),  activeColor: darkMode ? Colors.*white*.withOpacity(0.8) : Colors.*black*.withOpacity(0.8),  backgroundColor: darkMode ? Colors.*black* : Colors.*white*,  padding: const EdgeInsets.symmetric(horizontal: 25, vertical: 15),  selectedIndex: controller.selectedIndex.value,  onTabChange: (index) => controller.selectedIndex.value = index,  tabs: const [  GButton(  icon: Iconsax.*home*,  text: 'Главная',  ),  GButton(  icon: Iconsax.*chart\_34*,  text: 'Анализ',  ),  GButton(  icon: Iconsax.*book\_saved*,  text: 'Записи',  ),  GButton(  icon: Iconsax.*user*,  text: 'Профиль',  ),  ],  ),  ),  body: Obx(() => controller.screens[controller.selectedIndex.value]),  );  } }  class NavigationController extends GetxController {  final Rx<int> selectedIndex = 0.obs;   final screens = [  const HomeScreen(),  const ChartsScreen(),  const NotesScreen(),  const SettingsScreen(),  ]; } |

Приложение № 2

Листинг 3. Виджет фильтрации транзакций для всех записей и для записей под конкретную категорию (AllSortNotes).

|  |
| --- |
| import 'package:cloud\_firestore/cloud\_firestore.dart';  import 'package:family\_budget/features/\_familybudget/models/notes\_model.dart';  import 'package:family\_budget/features/\_familybudget/screens/notes/widgets/single\_notes.dart';  import 'package:flutter/material.dart';  import 'package:get/get.dart';  import 'package:iconsax/iconsax.dart';  import '../../../../common/styles/floating\_button.dart';  import '../../../../common/widgets/appbar/appbar.dart';  import '../../../../common/widgets/grid\_layout.dart';  import '../../../../utils/constants/sizes.dart';  import '../../../../utils/helpers/cloud\_helper\_functions.dart';  import '../../controllers/all\_notes\_controller.dart';  import 'widgets/add\_new\_notes.dart';  class AllSortNotes extends StatelessWidget {  const AllSortNotes({  super.key,  this.title,  this.query,  this.futureMethod,  });  final TAppBar? title;  final Query? query;  final Future<List<NotesModel>>? futureMethod;  @override  Widget build(BuildContext context) {  final controller = Get.put(AllNotesController());  final cat = futureMethod == null ? false : true;  return PopScope(  canPop: true,  onPopInvoked: (didPop) {  if (didPop) {  controller.clearNotes();  controller.selectedSortOption.value = 'Наименование';  controller.selectedPeriodOption.value = 'Все время';  controller.selectedFilterOption.value = 'Все';  controller.filterNotesByPeriod(controller.selectedPeriodOption.value, controller.notes);  controller.sortNotes(controller.selectedSortOption.value);  controller.filterNotesByType(controller.selectedFilterOption.value, false);  }  },  child: Scaffold(  appBar: title,  body: SingleChildScrollView(  child: Padding(  padding: const EdgeInsets.all(TSizes.defaultSpace),  child: Column(  children: [  const SizedBox(height: 7),  FutureBuilder(  future: futureMethod ?? controller.fetchNotesByQuery(query),  builder: (context, snapshot) {  const loader = Center(child: CircularProgressIndicator());  final widget = TCloudHelperFunctions.checkMultiRecordState(snapshot: snapshot, loader: loader);  if (widget != null) return widget;  final notes = snapshot.data!;  WidgetsBinding.instance.addPostFrameCallback((\_) {  controller.assignNotes(notes, cat);  });  return Column(  children: [  DropdownButtonFormField(  decoration: const InputDecoration(prefixIcon: Icon(Iconsax.calendar\_1)),  value: controller.selectedPeriodOption.value,  onChanged: (value) {  controller.filterNotesByPeriod(value!, notes);  },  items: [  'Все время', 'Год', 'Три месяца','Месяц', 'Неделя', 'Сегодня'  ].map((option) => DropdownMenuItem(value: option, child: Text(option))).toList(),  ),  const SizedBox(height: TSizes.spaceBtwItems),  DropdownButtonFormField(  decoration: const InputDecoration(prefixIcon: Icon(Iconsax.sort)),  value: controller.selectedSortOption.value,  onChanged: (value) {  controller.sortNotes(value!);  },  items: [  'Наименование', 'Дата по убыванию', 'Дата по возрастанию', 'Сначала большие транзакции', 'Сначала маленькие транзакции'  ].map((option) => DropdownMenuItem(value: option, child: Text(option))).toList(),  ),  const SizedBox(height: TSizes.spaceBtwItems),  DropdownButtonFormField<String>(  decoration: const InputDecoration(prefixIcon: Icon(Icons.filter\_tilt\_shift)),  value: controller.selectedFilterOption.value,  onChanged: (value) {  controller.filterNotesByType(value!, cat);  },  items: ['Все', 'Доход', 'Расход']  .map((option) => DropdownMenuItem(value: option, child: Text(option)))  .toList(),  ),  ],  );  },  ),  const SizedBox(height: TSizes.spaceBtwItems),  Obx(()=> TGridLayout(  itemCount: controller.notesFilter.length,  itemBuilder: (\_, index) => SingleNotes(notes: controller.notesFilter[index],),  )),  ],  ),  ),  ),  floatingActionButtonLocation: FloatingActionButtonLocation.miniEndDocked,  floatingActionButton: FloatingButton(  onPressed: () => Get.to(() => const AddNewNotes()),  ),  ),  );  }  } |

Приложение № 3

Листинг 4. Контроллер транзакций (AllNotesController)

|  |
| --- |
| import 'package:cloud\_firestore/cloud\_firestore.dart';  import 'package:family\_budget/features/\_familybudget/models/notes\_model.dart';  import 'package:flutter/material.dart';  import 'package:get/get.dart';  import 'package:intl/intl.dart';  import '../../../data/repositories/notes/notes\_r.dart';  import '../../../navigation\_menu.dart';  import '../../../utils/constants/image\_strings.dart';  import '../../../utils/constants/sizes.dart';  import '../../../utils/helpers/network\_manager.dart';  import '../../../utils/popups/full\_screen\_loader.dart';  import '../../../utils/popups/loaders.dart';  class AllNotesController extends GetxController {  static AllNotesController get instance => Get.find();  final type = TextEditingController();  final summa = TextEditingController();  final category = TextEditingController();  final description = TextEditingController();  final dateTime = TextEditingController();  GlobalKey<FormState> notesFormKey = GlobalKey<FormState>();  final isLoading = false.obs;  RxBool refreshData = true.obs;  final notesRepository = Get.put(NotesRepository());  final RxString selectedSortOption = 'Наименование'.obs;  final RxString selectedPeriodOption = 'Все время'.obs;  final RxString selectedPeriod = 'Все время'.obs;  final RxString selectedFilterOption = 'Все'.obs;  RxList<NotesModel> notes = <NotesModel>[].obs;  RxList<NotesModel> notesFilter = <NotesModel>[].obs;  RxList<NotesModel> selectedDateNotes = <NotesModel>[].obs;  @override  void onInit() {  super.onInit();  init();  }  Future<void> init() async {  isLoading.value = true;  notes.assignAll(await fetchAllNotes());  fetchNotesByDate(DateTime.now());  isLoading.value = false;  }  void clearNotes() {  notes.clear();  }  Future<void> fetchNotesByDate(DateTime date) async {  try {  List<NotesModel> allNotes = await allUserNotes();  notes.assignAll(allNotes);  selectedDateNotes.assignAll(notes.where((note) {  final noteDate = note.dateTime;  return noteDate != null && noteDate.year == date.year && noteDate.month == date.month && noteDate.day == date.day;  }).toList());  } catch (e) {  TLoaders.errorSnackBar(title: 'Ошибка!', message: e.toString());  }  }  Future<void> fetchNotesRecord(String noteId) async {  try {  isLoading.value = true;  final note = await notesRepository.fetchNoteDetails(noteId);  notes([note]);  } catch (e) {  notes(<NotesModel>[]);  } finally {  isLoading.value = false;  }  }  Future<List<NotesModel>> allUserNotes() async {  try {  final notesses = await notesRepository.fetchUserNotes();  return notesses;  } catch (e) {  TLoaders.errorSnackBar(title: 'Транзакции не найдены', message: e.toString());  return [];  }  }  ///search ????  List<NotesModel> searchNotes(List<NotesModel> notes, String query) {  final suggestions = notes.where((note) {  final noteDescription = note.description.toLowerCase();  final noteSumma = note.summa.toLowerCase();  final input = query.toLowerCase();  if (RegExp(r'^\d+$').hasMatch(input)) {  return noteSumma.contains(input);  } else {  return noteDescription.contains(input);  }  }).toList();  return suggestions;  }  /// Add  Future addNewNotesController() async {  try {  TFullScreenLoader.openLoadingDialog('Сохраняем транзакцию...', TImages.docerAnimation);  final isConnected = await NetworkManager.instance.isConnected();  if (!isConnected) {  TFullScreenLoader.stopLoading();  return;  }  DateFormat dateFormat = DateFormat('dd.MM.yyyy');  // save  final notes = NotesModel(  id: '',  description: description.text.trim(),  type: type.text.trim(),  category: category.text.trim(),  summa: summa.text.trim(),  dateTime: dateFormat.parse(dateTime.text.trim()),  );  final id = await notesRepository.addNotes(notes);  notes.id = id;  TFullScreenLoader.stopLoading();  TLoaders.successSnackBar(title: 'Поздравляем', message: 'Ваша запись транзакции была сохранена!');  //refresh history  refreshData.toggle();  // reset fields  resetFormFields();  // redirect  Get.off(const NavigationMenu());  } catch(e) {  TFullScreenLoader.stopLoading();  TLoaders.errorSnackBar(title: 'Транзакция не найдена', message: e.toString());  }  }  void resetFormFields() {  category.clear();  type.clear();  dateTime.clear();  summa.clear();  description.clear();  notesFormKey.currentState?.reset();  }  void resetNotes() {  notes.clear();  refreshData.toggle();  }  Future<List<NotesModel>> fetchNotesByQuery(Query? query) async {  try {  if (query == null) return [];  final notes = await notesRepository.fetchNotesByQuery(query);  return notes;  } catch (e) {  TLoaders.errorSnackBar(title: 'Ошибка!', message: e.toString());  return [];  }  }  ///delete notes  void deleteNoteWarningPopup(NotesModel note) {  Get.defaultDialog(  contentPadding: const EdgeInsets.all(TSizes.md),  title: 'Удалить запись',  middleText: 'Вы уверены, что хотите удалить эту запись? Она будет удалена безвозвратно.',  confirm: ElevatedButton(  onPressed: () async => deleteNoteRecord(note),  style: ElevatedButton.styleFrom(  backgroundColor: Colors.red,  side: const BorderSide(color: Colors.red),  ),  child: const Padding(  padding: EdgeInsets.symmetric(horizontal: TSizes.lg),  child: Text('Удалить'),  ),  ),  cancel: OutlinedButton(  child: const Text('Отмена'),  onPressed: () => Navigator.of(Get.overlayContext!).pop(),  ),  );  }  void deleteNoteRecord(NotesModel note) async {  try {  TFullScreenLoader.openLoadingDialog('Удаляем запись...', TImages.docerAnimation);  final notesController = Get.find<AllNotesController>();  notesController.notesRepository.deleteNoteById(note.id);  notesController.clearNotes();  notesController.fetchAllNotes();  TFullScreenLoader.stopLoading();  TLoaders.successSnackBar(title: 'Успешно', message: 'Запись успешно удалена');  Get.off(() => const NavigationMenu());  } catch (e) {  TFullScreenLoader.stopLoading();  TLoaders.errorSnackBar(title: 'Ошибка!', message: e.toString());  }  }  void filterByPeriod(String period, List<NotesModel> allNotes) {  selectedPeriod.value = period;  DateTime now = DateTime.now();  List<NotesModel> filteredNotes;  switch (period) {  case 'Неделя':  DateTime weekAgo = now.subtract(const Duration(days: 7));  filteredNotes = allNotes.where((note) => note.dateTime!.isAfter(weekAgo)).toList();  break;  case 'Месяц':  DateTime monthAgo = DateTime(now.year, now.month - 1, now.day);  filteredNotes = allNotes.where((note) => note.dateTime!.isAfter(monthAgo)).toList();  break;  case 'Три месяца':  DateTime threeMonthsAgo = DateTime(now.year, now.month - 3, now.day);  filteredNotes = allNotes.where((note) => note.dateTime!.isAfter(threeMonthsAgo)).toList();  break;  case 'Год':  DateTime yearAgo = DateTime(now.year - 1, now.month, now.day);  filteredNotes = allNotes.where((note) => note.dateTime!.isAfter(yearAgo)).toList();  break;  default:  filteredNotes = allNotes;  break;  }  notes.assignAll(filteredNotes);  }  void assignPeriodNotes(List<NotesModel> notes) {  this.notes.assignAll(notes);  filterByPeriod(selectedPeriod.value, notes);  }  void filterNotesByPeriod(String period, List<NotesModel> note) async {  selectedPeriodOption.value = period;  DateTime now = DateTime.now();  List<NotesModel> filterNotes;  switch (period) {  case 'Сегодня':  filterNotes = note.where((note) {  DateTime noteDate = note.dateTime!;  return noteDate.year == now.year && noteDate.month == now.month && noteDate.day == now.day;  }).toList();  break;  case 'Неделя':  DateTime weekAgo = now.subtract(const Duration(days: 7));  filterNotes = note.where((note) => note.dateTime!.isAfter(weekAgo)).toList();  break;  case 'Месяц':  DateTime monthsAgo = DateTime(now.year, now.month - 1, now.day);  filterNotes = note.where((note) => note.dateTime!.isAfter(monthsAgo)).toList();  break;  case 'Три месяца':  DateTime threeMonthsAgo = DateTime(now.year, now.month - 3, now.day);  filterNotes = note.where((note) => note.dateTime!.isAfter(threeMonthsAgo)).toList();  break;  case 'Год':  DateTime yearAgo = DateTime(now.year - 1, now.month, now.day);  filterNotes = note.where((note) => note.dateTime!.isAfter(yearAgo)).toList();  break;  default:  filterNotes = note;  break;  }  notes.assignAll(filterNotes);  if (filterNotes.isEmpty) {  notesFilter.clear();  } else {  sortNotes(selectedSortOption.value);  filterNotesByType(selectedFilterOption.value, false);  }  }  void sortNotes(String sortOption) {  selectedSortOption.value = sortOption;  switch (sortOption) {  case 'Наименование':  notesFilter.sort((a, b) => a.description.compareTo(b.description));  break;  case 'Дата по убыванию':  notesFilter.sort((a, b) => b.dateTime!.compareTo(a.dateTime!));  break;  case 'Дата по возрастанию':  notesFilter.sort((a, b) => a.dateTime!.compareTo(b.dateTime!));  break;  case 'Сначала большие транзакции':  notesFilter.sort((a, b) {  double summaA = a.type == 'Расход' ? -double.parse(a.summa) : double.parse(a.summa);  double summaB = b.type == 'Расход' ? -double.parse(b.summa) : double.parse(b.summa);  return summaB.compareTo(summaA);  });  break;  case 'Сначала маленькие транзакции':  notesFilter.sort((a, b) {  double summaA = a.type == 'Расход' ? -double.parse(a.summa) : double.parse(a.summa);  double summaB = b.type == 'Расход' ? -double.parse(b.summa) : double.parse(b.summa);  return summaA.compareTo(summaB);  });  break;  default:  notesFilter.sort((a, b) => b.dateTime!.compareTo(a.dateTime!));  }  }  Future<void> filterNotesByType(String filterOption, bool cat) async {  selectedFilterOption.value = filterOption;  switch (filterOption) {  case 'Доход':  notesFilter.assignAll(notes.where((note) => note.type == 'Доход').toList());  break;  case 'Расход':  notesFilter.assignAll(notes.where((note) => note.type == 'Расход').toList());  break;  default:  notesFilter.assignAll(notes);  break;  }  if (notes.isEmpty && cat==false) {  List<NotesModel> allNotes = await allUserNotes();  if (allNotes.isNotEmpty) {  filterNotesByPeriod(selectedPeriodOption.value, allNotes);  sortNotes(selectedSortOption.value);  }  }  }  void assignNotes(List<NotesModel> notes, bool cat) {  this.notes.assignAll(notes);  filterNotesByPeriod(selectedPeriodOption.value, notes);  filterNotesByType(selectedFilterOption.value, cat);  }  void updateNotesList(List<NotesModel> searchResults) {  notes.assignAll(searchResults);  }  ///all filter  Future<List<NotesModel>> fetchAllNotes() async {  try {  // fetch  final notes = await notesRepository.getAllNotes();  return notes;  } catch (e) {  TLoaders.errorSnackBar(title: 'Ошибка!', message: e.toString());  return [];  }  }  ///category filter  Future<List<NotesModel>> fetchCategoryNotes(String category) async {  try {  // fetch  final notes = await notesRepository.getCategoryNotes(category);  return notes;  } catch (e) {  TLoaders.errorSnackBar(title: 'Ошибка!', message: e.toString());  return [];  }  }  } |