СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc479705628)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc479705629)

[1.1 Функциональные требования к программному продукту 4](#_Toc479705630)

[1.2 Нефункциональные требования к программному продукту 5](#_Toc479705631)

[1.3 Осмотр и анализ аналогов разрабатываемого продукта 6](#_Toc479705632)

[2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ И ИСПОЛЬЗУЕМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 10](#_Toc479705633)

[2.1 Целевая платформа 10](#_Toc479705634)

[2.2 Язык программирования 12](#_Toc479705635)

[2.3 Система управления базами данных 14](#_Toc479705636)

[3 МОДЕЛИРОВАНИЕ ПРОЦЕССОВ 16](#_Toc479705637)

[3.1 Методология IDEF0 17](#_Toc479705638)

[3.2 IDEF0-диаграмма для разрабатываемого продукта 19](#_Toc479705639)

[3.2.1 Обращение к системе 20](#_Toc479705640)

[3.2.2 Обработка действий пользователя 22](#_Toc479705641)

[3.2.3 Изменение базы данных 23](#_Toc479705642)

[4 СТРУКТУРА ДАННЫХ И РЕСУРСОВ ПРОГРАММЫ 25](#_Toc479705643)

[4.1 Класс MainActivity 25](#_Toc479705644)

[4.2 Класс UsingDataBaseListFragment 26](#_Toc479705645)

[4.3 Класс FragmentTransactions 26](#_Toc479705646)

[4.4 Класс UsingDataBaseActivity 28](#_Toc479705647)

[4.5 Класс SingleEntityActivity 28](#_Toc479705648)

[4.6 Класс AccountsActivity 29](#_Toc479705649)

[4.7 Класс SingleAccountActivity 30](#_Toc479705650)

[4.8 Класс CategoriesActivity 30](#_Toc479705651)

[4.9 Класс FragmentCategories 31](#_Toc479705652)

[4.10 Класс SingleCategoryActivity 32](#_Toc479705653)

[4.10 Класс SingleTransactionActivity 33](#_Toc479705654)

[4.11 Класс SQLiteHandler 34](#_Toc479705655)

[4.12 Описание базы данных 35](#_Toc479705656)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 37](#_Toc479705657)

[5.1 Начало роботы с приложением 37](#_Toc479705658)

[5.2 Окно «Журнал» 37](#_Toc479705659)

[5.3 Окно «Платеж» 40](#_Toc479705660)

[5.4 Окна «Счета» и «Счет» 41](#_Toc479705661)

[5.5 Окно «Категории» 43](#_Toc479705662)

[5.6 Окно «Категория» 44](#_Toc479705663)

[5.7 Окно «Круговая диаграмма» и «Столбчатая диаграмма» 45](#_Toc479705664)

[ЗАКЛЮЧЕНИЕ 47](#_Toc479705665)

# ВВЕДЕНИЕ

Особенно актуальной сегодня является тема рационального расхода и сбережения финансовых средств. Необходимо тщательно анализировать тенденцию расходов и доходов, распределять собственный бюджет, понимать на какие категории товаров уходит часть заработка.

Для достижения выше поставленных целей, необходимо вести домашнюю бухгалтерию. Контролировать расходы и доходы можно старыми способами, то есть в тетради, так и более современными методами, которые имеют неоспоримые преимущества.

Целью данной работы является проектирование и разработка мобильного приложения для ведения домашней бухгалтерии. Пользователю будет предоставляться возможность внесения расходов/доходов, ведение учета их категорий, создание собственных счетов и работа с ними. Также, возможен просмотр истории для конкретного промежутка времени и просмотр статистики в виде круговой и столбчатой диаграмм.

В пояснительной записке будут описаны основные этапы проектирования и разработки, используемое программное обеспечение, руководства программиста и пользователя.

# 1 ПОСТАНОВКА ЗАДАЧИ

## 1.1 Функциональные требования к программному продукту

Функциональные требования определяют основной «фронт» работ разработчика, регламентируют поведение приложения в той или иной ситуации. Они ставят задачи, которые должна выполнять система.

Необходимо создать программное обеспечение для ведения домашней бухгалтерии, предоставляющее достаточный функционал для анализа расходов/ доходов и получения наглядной статистики.

С точки зрения функциональных характеристик выделяются ниже перечисленные требования:

Работа с платежами:

* Выбор даты и времени
* Ввод суммы платежа
* Выбор категории, для которой производится платеж
* Выбор счёта, для которого производится платеж
* Ввод заметки
* Сохранение нового платежа
* Редактирование и удаление существующих платежей
* Внесение платежа посредством анализа входящих смс на основе шаблонов

Работа с категориями:

* Ввод названия категории
* Выбор типа категории: расход, доход
* Выбор логотипа категории
* Сохранение новой категории
* Редактирование и удаление существующих категорий

Работа со счетами:

* Ввод названия счёта
* Ввод начального баланса счета
* Сохранение нового счета
* Редактирование и удаление существующего счёта

Просмотр статистики в виде диаграмм:

* Круговая диаграмма
* Столбчатая диаграмма

Основной диалог программы:

* Просмотр журнала платежей отдельно для трат, пополнений
* Просмотр журнала для разных временных диапазонов: день, неделя, месяц, год
* Просмотр диаграмм для разных временных диапазонов
* Навигационное меню

## 1.2 Нефункциональные требования к программному продукту

Нефункциональные требования, определяют требования к системе в целом, а не в отдельных вариантах использования, описывают свойства и характеристики, которые должна демонстрировать система, а также ограничения, которые должны быть соблюдены в отдельных случаях.

Выделим нефункциональные требования для следующих групп: Внешние интерфейсы, атрибуты качества, ограничения.

Внешние интерфейсы:

* Интерфейс приложения должен иметь привлекательный дизайн
* Интерфейс должен иметь интуитивны характер
* Интерфейс должен быть эстетичным
* Интерфейс приложения должен соответствовать основным правилам юзабилити
* Дизайн приложений должен быть в стиле Material design

Атрибуты качества:

* Надежность – приложение должно правильно реагировать на любые данные, введенные пользователем, не зависать во время работы. Приложение должно быть защищено от ошибок и гарантировать целостность данных
* Производительность – приложение должно выполнять все предусмотренные операции без долгих ожиданий и зависаний. Выполнение каждой операции должно проходить в кратчайший срок.
* Эксплуатационная пригодность – приложение должно выполнять все необходимые требования и быть работоспособным не зависимо от времени его использования

Ограничение:

* Приложение исполняется на мобильных устройствах под управлением операционной системы Android
* Минимальный API является 18

## 1.3 Осмотр и анализ аналогов разрабатываемого продукта

Рассмотрим некоторые ранее созданные программные продукты.

HomeBank – общедоступная программа для ведения домашней бухгалтерии. Предоставляется возможность контролировать свои доходы и расходы, анализировать бюджет. Программа позволяет упорядочивать платежи по категориям, планировать транзакции для автоматического вноса в базу данных (например, ежедневные счета). В программе так же есть функция одновременного редактирования нескольких полей.

С помощью HomeBank можно анализировать финансовое состояние и составлять отчеты. Программа предоставляет возможность генерировать отчеты по текущему состоянию бюджета. Также, возможно настроить фильтрацию транзакций и параметров отчета. Присутствует функционал визуализации данных в виде графиков и диаграмм.

Можно сделать вывод, что программа HomeBank хорошо справляется с поставленными задачами. Но, также, в ней присутствуют минусы: программа распространяется исключительно на компьютеры, что не дает возможности оперативно занести платежи. Также в программе HomeBank слишком загроможден интерфейс и присутствует чрезмерный функционал.

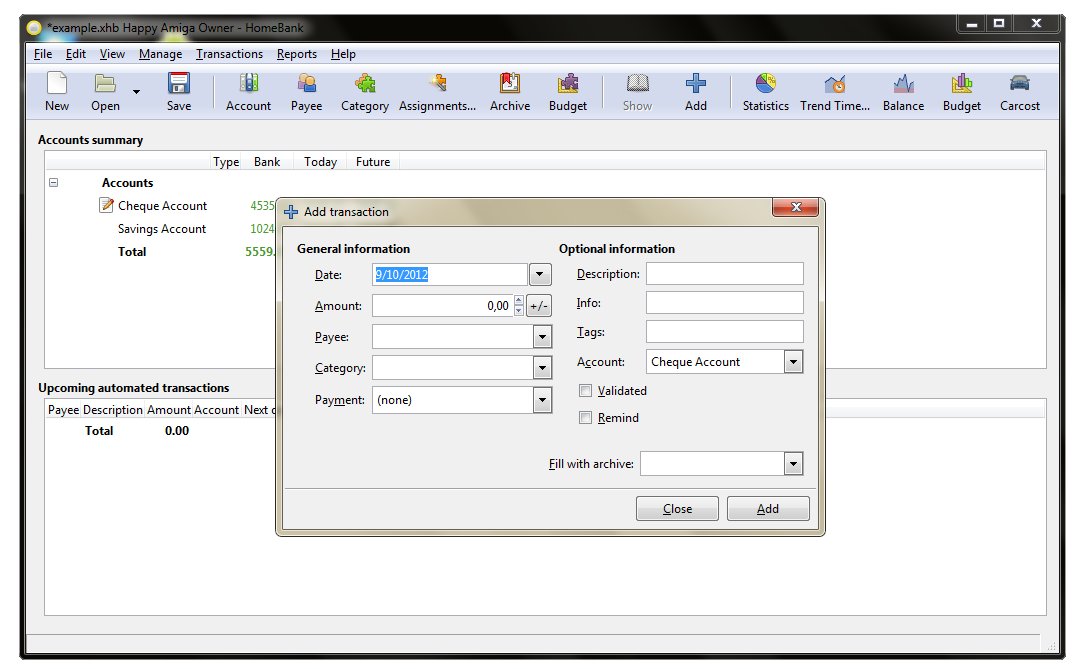


Рисунок 1.1 – Приложение HomeBank

Далее, рассмотрим приложение «Keepsoft домашняя бухгалтерия»

Данное приложение включает в себя следующий функционал:

* Учет расходов/доходов
* Возможность ведения счетов
* Возможность распределения расходов/доходов по категориям
* Учет долгов
* Составление бюджета расходов и доходов
* Планирование бюджета
* Параметризированный поиск
* Возможность применения фильтров
* Погашение кредитов и долгов по частям

Главным минусом данной программы является то, что бесплатного функционала не достаточно для того, чтобы удобно пользоваться программой. Также отсутствует функция визуального представления данных: в виде диаграмм.

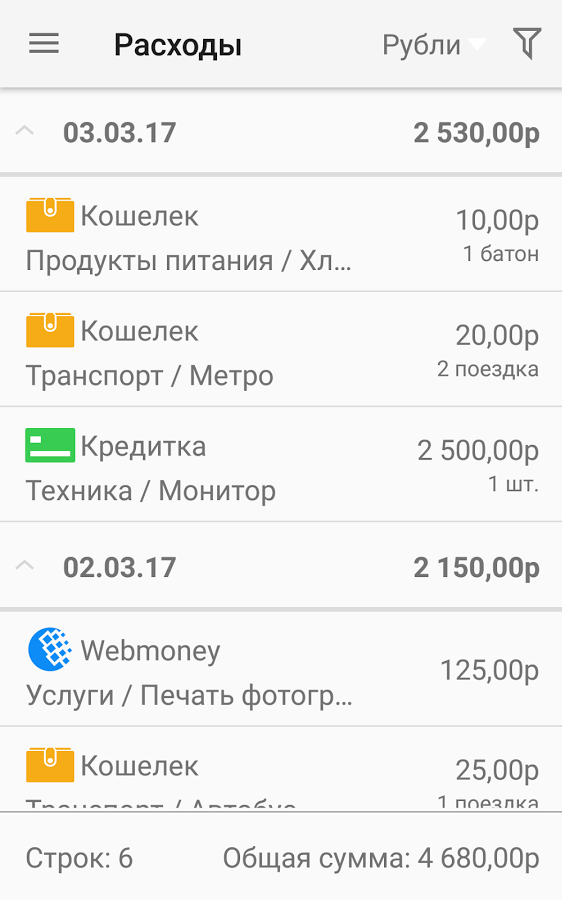


Рисунок 1.2 – Приложение «Keepsoft домашняя бухгалтерия»

Следующая рассматриваемая программа – Finance PM. Данная программа реализует следующий функционал:

* Возможность управления кошельками
* Возможность добавления таких операций как доходы, расходы и переводы
* Возможность редактировать категории
* Возможность работы с несколькими валютами
* Возможность регулярных платежей
* Возможность синхронизации данных
* Возможность создания шаблонных операций

В целом, как и ранее рассматриваемое приложение «Keepsoft домашняя

бухгалтерия», данный вариант справляется с поставленными для него задачами, но имеет аналогичный минус – необходимость приобретения дорогостоящей лицензии.



Рисунок 1.3 – Приложение «Finance PM»

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ И ИСПОЛЬЗУЕМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

## 2.1 Целевая платформа

Цель данной работы – создать приложение по ведению домашней бухгалтерии для мобильного устройства. Поэтому, необходимо выбрать операционную систему, на которой оно будет исполняться.

Среди главных кандидатов можно выделить Android и iOS. Рассмотрим данные операционные системы по отдельности.

Android – операционная система, разработанная для мобильных устройств, электронных книг, смарт-часов, телевизоров и других устройств. Операционная система Android имеет ядро Linux и имеет собственную реализацию Java Virtual Machine – Dalvik.

Компания Google предоставляет возможность писать собственные приложения на языке программирования Java, используя библиотеки Android Native Development Kit. Также, предоставляется возможность импортировать библиотеки и компоненты приложений языка C и других.

Среди достоинств Android можно выделить:

* Android имеет наилучшие показатели в рядя функций, как интернет-серфинг, взаимодействие с приложениями Google,
* Android является платформой с открытым исходным кодом, что предоставляет широкий круг возможностей
* Android предоставляет устанавливать программы без интернета, а также поддерживает отладку приложений по USB, что позволяет писать и тестировать собственные приложения на своём Android-устройстве
* Android доступен для различных аппаратных платформ, таких как ARM, MIPS, x86.

Недостатками Android являются:

* Android имеет ряд уязвимостей в безопасности
* В связи с тем, что можно устанавливать приложения из «недоверенных источников», развивается пиратство
* Android имеет чрезмерную фрагментацию, что влечет за собой немалые проблемы для разработчиков

Далее, рассмотрим операционную систему IOS.

iOS – Операционная система, выпускаемая компанией Apple. Предназначена для смартфонов, планшетов и проигрывателей

Для написания приложений под iOS используется язык программирования Objective-C и среда разработки Xcode . Также, можно разрабатывать на C и C++.

Среди достоинств iOS можно выделить

* Интерфейс, понятный на интуитивном уровне
* Высокая, по сравнению с другими ОС, скорость работы
* Высокая практичность приложений
* Отсутствие уязвимостей безопасности
* Многозадачность (малое потребление ресурсов свёрнутыми приложениями).

Среди недостатков iOS можно выделить:

* Отсутствует возможность прямого доступа к файловой систему
* Отсутствие возможности писать приложения без платного софта
* Отсутствие по-своему настроить устройство.
* Ограничения на устанавливаемые приложения

На данный момент около 60% всех мобильных устройств находятся под управлением операционной системы Android, когда под управленим iOS – меньше 20%.

Проанализировав операционные системы Android и iOS, принято решение, что целевая платформа для разрабатываемого приложения будет Android.

Данное решение было принято по далее описанным причинам. Во-первых, Android – открытая платформа, в связи с чем, на ней можно реализовать больше функций. Во-вторых, Android предоставляет широкий, по сравнению с iOS, выбор API и различных библиотек. В-третьих, Android имеет более широкую аудиторию, чем другие операционные системы.

## 2.2 Язык программирования

В предыдущем пункте, целевой операционной системой был выбран Android. Для Android необходимо написать приложение на языке программирования, компиляторы которых способны генерировать байт-код для Dalvik.

На данный момент самым популярным и рациональным способом является написание Android-приложения на строго типизированном объектно-ориентированном языке программирования Java.

Java – строго типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной Java-машины.

Основные особенности языка:

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) – программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

* применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде,
* широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках,
* аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM).

Среди основных возможностей Java можно выделить:

* автоматическое управление памятью.
* расширенные возможности обработки исключительных ситуаций.
* богатый набор средств фильтрации ввода-вывода.
* набор стандартных коллекций: массив, список, стек и т. п.
* наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI).
* наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы.
* встроенные в язык средства создания многопоточных приложений, которые потом были портированны на многие языки.
* унифицированный доступ к базам данных: на уровне отдельных SQL-запросов – на основе JDBC, SQLJ; на уровне концепции объектов, обладающих способностью к хранению в базе данных – на основе Java Data Objects и Java Persistence API;
* поддержка обобщений.
* поддержка лямбда выражений, замыканий, встроенные возможности функционального программирования.

Так как Java является объектно-ориентированным языком программирования, то тут реализуются три основные парадигмы.

Инкапсуляция – свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе.

Наследование – свойство системы, позволяющее описать новый класс на основе уже существующего, с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником, дочерним или производным классом.

Полиморфизм – свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

## 2.3 Система управления базами данных

В разрабатываемом приложении необходимо постоянно упорядоченно хранить набор данных, с которыми работает пользователь. Для этой цели самым рациональным решением будет использование базы данных. Для управления создания базы данных будет использоваться система управления базами данных.

Так как Android предоставляет API-интерфейсы, необходимые для использования базы данных SQLite, то в разрабатываемом приложении будет использоваться именно эта система управления базами данных.

SQLite – встраиваемая реляционная система управления базами данных.

«Встраиваемая» означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

В комплекте поставки идёт также функциональная клиентская часть в виде исполняемого файла sqlite3, с помощью которого демонстрируется реализация функций основной библиотеки. Клиентская часть работает из командной строки, позволяет обращаться к файлу БД на основе типовых функций ОС.

API-интерфейсы, необходимые для использования базы данных на платформе Android, доступны в составе пакета android.database.sqlite.

# 3 МОДЕЛИРОВАНИЕ ПРОЦЕССОВ

Моделирование процессов является одним из методов улучшения качества и эффективности работы организации. В основе этого метода лежит описание процесса через различные элементы (действия, данные, события, материалы и пр.) присущие процессу. Как правило, моделирование процессов описывает логическую взаимосвязь всех элементов процесса от его начала до завершения в рамках организации. В более сложных ситуациях моделирование может включать в себя внешние по отношению к организации процессы или системы.

Моделирование процессов позволяет понять работу и провести анализ организации. Это достигается за счет того, что модели могут быть составлены по различным аспектам и уровням управления. В больших организациях моделирование бизнес процессов выполняется более подробно и многограннее, чем в малых, что связано с большим количеством кросс-функциональных связей.

Цели моделирования:

* Во-первых, это цель описания процессов. За счет моделирования можно проследить, что происходит в процессах от начала, до завершения. Моделирование позволяет получить «внешний» взгляд на процессы и определить улучшения, которые повысят их эффективность.
* Во-вторых, нормирование процессов. Моделирование бизнес процессов задает правила выполнения процессов, т.е. то, каким образом они должны быть выполнены. Если следовать установленным в моделях правилам, руководящим указаниям или требованиям, то можно достичь желаемой производительности процессов.
* В-третьих, установление взаимосвязей в процессах. Моделирование бизнес процессов устанавливает четкую связь между процессами и требованиями, которые они должны выполнять.

В данной работе будет рассмотрено функциональное моделирование – вид моделирования, подразумевающий описание процессов в виде взаимосвязанных, четко структурированных функций. При этом строгая временная последовательность функций, в том виде, как она существует в реальных процессах, не обязательна.

Функциональное моделирование будет выполнено с помощью методологии IDEF0.

## 3.1 Методология IDEF0

Методология IDEF0 предписывает построение иерархической системы диаграмм - единичных описаний фрагментов системы. Сначала проводится описание системы в целом и ее взаимодействия с окружающим миром (контекстная диаграмма), после чего проводится функциональная декомпозиция - система разбивается на подсистемы и каждая подсистема описывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности.

Каждая IDEF0-диаграмма содержит блоки и дуги. Блоки изображают функции моделируемой системы. Дуги связывают блоки вместе и отображают взаимодействия и взаимосвязи между ними.

Функциональные блоки (работы) на диаграммах изображаются прямоугольниками, означающими поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Имя работы должно быть выражено отглагольным существительным, обозначающим действие.

Блоки в IDEF0 размещаются по степени важности, как ее понимает автор диаграммы. Этот относительный порядок называется доминированием. Доминирование понимается как влияние, которое один блок оказывает на другие блоки диаграммы. Например, самым доминирующим блоком диаграммы может быть либо первый из требуемой последовательности функций, либо планирующая или контролирующая функция, влияющая на все другие.

Взаимодействие работ с внешним миром и между собой описывается в виде стрелок, изображаемых одинарными линиями со стрелками на концах. Стрелки представляют собой некую информацию и именуются существительными.

В IDEF0 различают несколько типов стрелок.

* Вход – входит в работу слева и показывает информационные и материальные потоки, которые преобразуются в бизнес процессе.
* Управление – входит в работу сверху и показывает материальные и информационные потоки, которые не преобразуются в процессе, но нужны для его выполнения.
* Механизм – входит в работу снизу и показывает людей, технические средства, информационные системы и т.п., при помощи которых процесс реализуется.
* Результаты

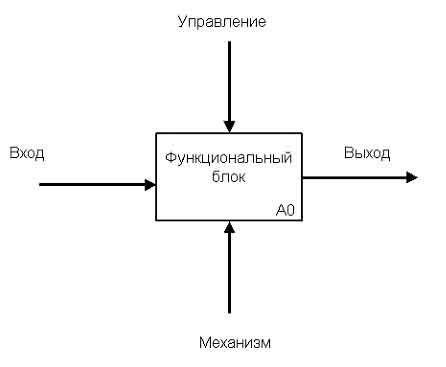


Рисунок 3.1.1 – Назначение стрелок в методологии IDEF0

## 3.2 IDEF0-диаграмма для разрабатываемого продукта

Начнем с построения контекстной IDEF0-диаграммы. Основной функцией разрабатываемого приложения является обслуживание пользователя, посредством обработки действий, совершенных им. Поэтому, определим единственную работу контекстной диаграммы как «Обслуживание пользователя приложения». Далее определим входные и выходные данные, а так лее механизмы и управление.

Для обслуживания пользователя, необходимо вывести пользователю необходимые данные, предоставить возможность сформировать запросы, открыть доступ к базе данных и обработать его запрос.

В качестве входных данных будет использоваться «Исодная база данных» и «Запросы пользователя». Последствием обработки запросов пользователя является изменение базы данных, поэтому выходными данными будет «измененная база данных». Процесс будет выполняться автоматизированными средствами приложения, используя методы сохранения и обработки информации.

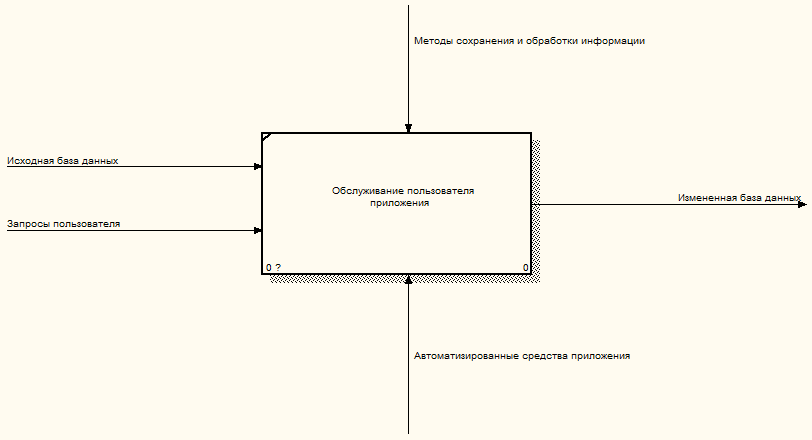


Рисунок 3.2.1 – Контекстная IDEF0-диаграмма

Для уточнения всех процессов произведем декомпозицию контекстной диаграммы.

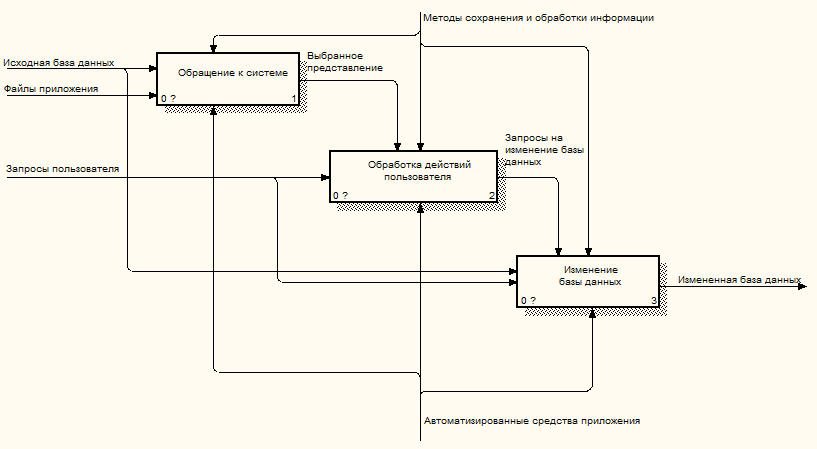


Рисунок 3.2.2 – Декомпозиция контекстной диаграммы

Имеем следующую последовательность обслуживания пользователя приложения:

1. Обращение к системе
2. Обработка запроса пользователя
3. Изменение базы данных

Последовательно декомпозируем все блоки полученной диаграммы.

### 3.2.1 Обращение к системе

Когда происходит запуск приложения, выполняется обращение к системе.

Сначала, анализируются файлы настроек, а в частности файл AndroidManifest.xml. В нем описываются основные компоненты приложения: службы, деятельности, контент-провайдеры, объявляется имя Java-пакета приложения, объявляются разрешения, перечисляются сторонние библиотеки. Также указывается минимальный уровень API Android, необходимый для работы приложения.

Далее, на основе обработанных настроек, выбирается необходимое Activity - класс, который представляет отдельный экран приложения или его визуальный интерфейс.

Для выбранного ранее Activity, загружаются данные из базы данных, выводятся в необходимые компоненты и представляются пользователю.

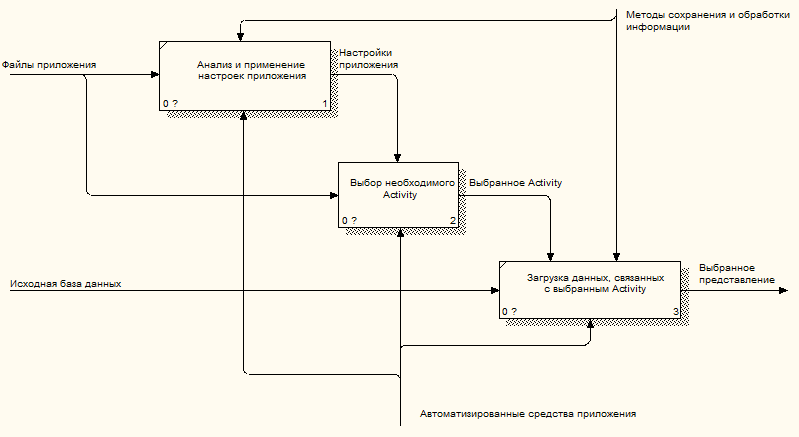


Рисунок 3.2.3 – Декомпозиция работы «Обращение к системе»

Подробнее опишем процесс выбора Activity.

Определение стартового Activity находится в файле AndroidManifest.xml. При его анализе необходимо найти Activity, которое содержит атрибуты, указывающие на то, что он стартовый. Далее в пакете приложения находится класс, который ему соответствует. Далее управление передается выбранному классу.

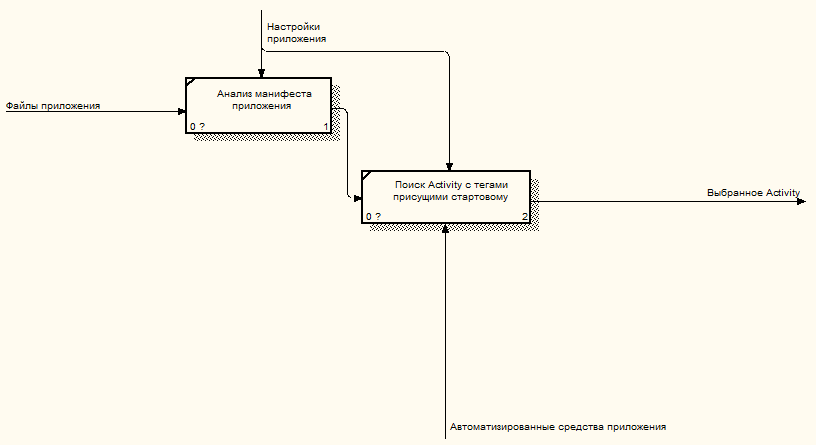


Рисунок 3.2.4 – Декомпозиция работы «Выбор необходимого Activity»

### 3.2.2 Обработка действий пользователя

После того, как пользователю представилась необходимая информация, он получает возможность манипулировать этими данными. Необходимо корректно обрабатывать любые его действия.

Сначала, система ожидает введения пользователем некоторых данных. После того, как он вводит данные, и дает команду на их обработку (например, кнопка «сохранить»), система получает данные.

Далее, происходит обработка данных, а сначала – проверка их корректности. Система производит проверку данных и в случае некорректного ввода информирует об этом пользователя. Если система получает корректные данные – происходит их обработка, и формируются запросы на изменение базы данных.

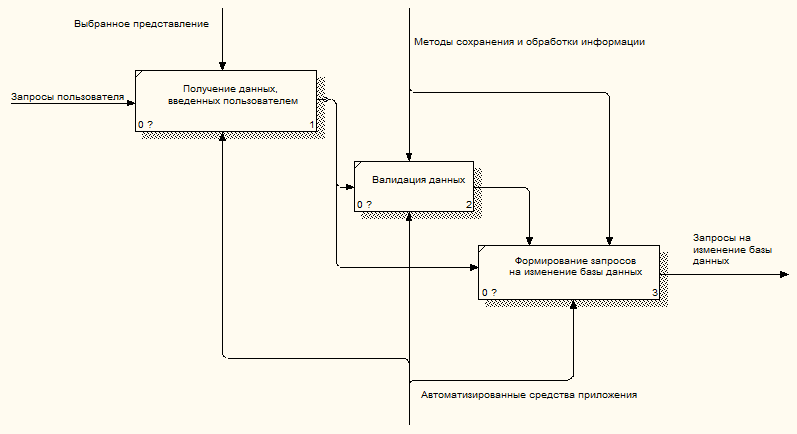


Рисунок 3.2.5 – Декомпозиция работы «Обработка действий пользователя»

### 3.2.3 Изменение базы данных

После действий пользователя, были сформированы запросы на изменение базы данных. Запросы необходимо обработать и внести соответствующие изменения в базу данных.

Сначала, в зависимости от того на каком из Activity производились действия пользователя, выбирается необходимая таблица. Далее, необходимо сформировать пул данных. После формирования пула, необходимо внести его в базу данных.

После выполнения данных операций, пользователю снова представляется Activity с возможностью повторного манипулирования данными.

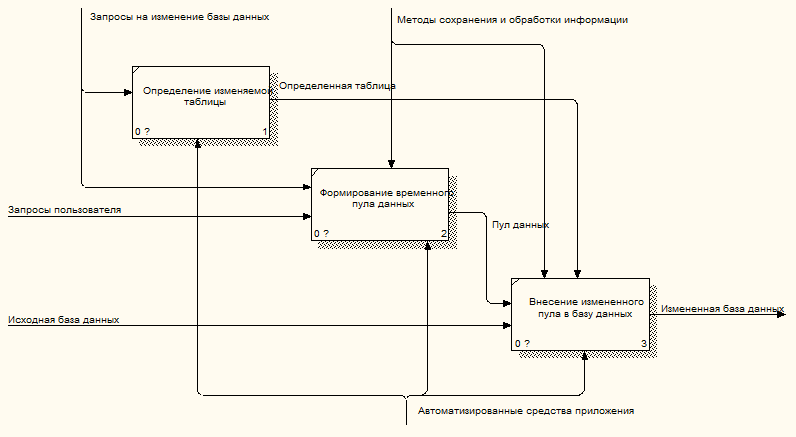


Рисунок 3.2.6 – Декомпозиция работы «Изменение базы данных»

# 4 СТРУКТУРА ДАННЫХ И РЕСУРСОВ ПРОГРАММЫ

В этом разделе приводятся результаты разработки структур и ресурсов данных, с какими работает программа и их характеристику.

Для классов будут описаны:

* Описание и предназначение класса
* Описание и предназначение переменных класса
* Описание и предназначение методов класса
* Схема родственных классов, если таковая присутствует

Также, приводится детальное описание базы данных.

## 4.1 Класс MainActivity

В данном классе описана логика стартового окна приложения. Реализуется логика бокового меню, а также логика просмотра журнала всех платежей.

Базовый класс: AppCompatActivity

Реализуемые интерфейсы: NavigationView.OnNavigationItemSelectedListener

Поля класса:

* DateSelector dateSelector – объект, генерирующий различные отрезки времени: день, неделя, месяц, год и показывающий их пользователю

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации всех фрагментов и загрузчиков
* void onBackPressed() – обработка нажатия кнопки «назад» на устройстве
* boolean onNavigationItemSelected(MenuItem item) – обработка нажатия элементов в боковом меню. Вызывает новое Activity.
* DateSelector getDateSelector() – геттер поля dateSelector, предназначенный для фрагментов, вызываемых данным Activity

Вложенные классы:

* SectionsPagerAdapter extends FragmentStatePagerAdapter – отвечает за постраничный просмотр журнала. Предоставляет возможность просмотра журнала по категориям: «Все», «Траты», «Пополнения». Использует фрагмент FragmentTransactions.

## 4.2 Класс UsingDataBaseListFragment

Данный класс является абстрактным и предоставляет возможность вывода списка элементов.

Базовый класс: ListFragment

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* SQLiteHandler handler – объект для управления созданием базы данных и управлением версиями
* SQLiteDatabase db – объект, который имеет методы для создания, удаления, выполнения команд SQL и выполнения других общих задач управления базами данных
* Cursor cursor – объект, обеспечивающий произвольный доступ на чтение и запись к набору результатов, возвращаемому запросом базы данных

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации полей класса. Также открывает доступ к базе данных
* void onDestroy() – закрывает используемые ресурсы

## 4.3 Класс FragmentTransactions

В данном классе описан Fragment, реализующий вывод платежей. Тут генерируется контекстное меню при долгом нажатии на элемент списка. Для манипуляции платежами вызывает SingleTransactionActivity.

Базовый класс: UsingDataBaseListFragment

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* String query – запрос к базе данных для получения платежей
* SimpleCursorAdapter transactionsCursorAdapter – адаптер для вывода платежей
* DateSelector dateSelector – объект, генерирующий различные отрезки времени: день, неделя, месяц, год и показывающий их пользователю

Методы класса:

* static FragmentTransactions FragmentTransactionsFactory(String query) – фабричный метод для генерации объектов класса, инициализируя поле query переданным значением
* void onActivityCreated(@Nullable Bundle savedInstanceState) – инициализация полей и обработчиков событий
* void onResume() – обработка возвращения пользователя к Fragment
* void onCreateOptionsMenu(Menu menu, MenuInflater inflater) – генерация меню
* boolean onOptionsItemSelected(MenuItem item) – обработка нажатия на элементе меню
* void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) – генерация контекстного меню
* boolean onContextItemSelected(MenuItem item) – обработка нажатия на элементе в контекстном меню
* setQuery(String query) – инициализация поля query переданным значением
* String getBaseQuery() – метод для получения запроса к базе данных, возвращающего все платежи
* void initializeAdapter() – инициализация адаптера для вывода элементов списка
* void requeryCursor() – перезагрузка данных для курсора
* String initializeConditions(String query) – применение фильтра отрезка времени для получения платежей
* decreaseDate(View view) – уменьшить на единицу отрезок времени
* increaseDate(View view) – увеличить на единицу отрезок времени
* updateCursors() – обновить курсоры всех фрагментов, хранящихся во FragmentManager

## 4.4 Класс UsingDataBaseActivity

Данный класс является абстрактным и предоставляет возможность Activity работать с базами данных. Производит открытие и закрытие используемых ресурсов.

Базовый класс: AppCompatActivity

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* SQLiteHandler handler – объект для управления созданием базы данных и управлением версиями
* SQLiteDatabase db – объект, который имеет методы для создания, удаления, выполнения команд SQL и выполнения других общих задач управления базами данных
* Cursor cursor – объект, обеспечивающий произвольный доступ на чтение и запись к набору результатов, возвращаемому запросом базы данных

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации полей класса. Также открывает доступ к базе данных
* void onDestroy() – закрывает используемые ресурсы

## 4.5 Класс SingleEntityActivity

Данный класс является абстрактным и предоставляет возможность Activity работать с конкретными сущностями: счёт, платеж, категория.

Базовый класс: UsingDataBaseActivity

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* long entityId – Id записи элемента в базе данных

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации всех фрагментов и загрузчиков. Также получает entityId из переданных значений
* boolean onCreateOptionsMenu – генерирует меню Activity
* abstract boolean onOptionsItemSelected(MenuItem item) – обработка нажатия элемента в меню

## 4.6 Класс AccountsActivity

В данном классе описано Activity, предоставляющий интерфейс доступа к счетам. Тут выводится список счетов, генерируется контекстное меню при долгом нажатии на элемент. Для манипуляции счетами вызывает SingleAccountActivity.

Базовый класс: UsingDataBaseActivity

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* ListView accountsListView – список для отображения счетов
* SimpleCursorAdapter accountsCursorAdapter – адаптер для отображения элементов списка

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации всех фрагментов и загрузчиков
* void onResume() – выполняет загрузку данных, когда пользователь возвращается к Activity
* void onCreateContextMenu(ContextMenu menu, View v, ContextMenu. ContextMenuInfo menuInfo) – генерирует контекстное меню при долгом нажатии на элементе списка
* boolean onContextItemSelected(MenuItem item) – обработка нажатия элемента в конекстном меню
* void initializeAdapter() – инициализация адаптера для вывода элементов списка
* void requeryCursor() – перезагрузка данных для курсора

## 4.7 Класс SingleAccountActivity

В данном классе описано взаимодействие с конкретным счётом пользователя: создание, изменение и удаление.

Базовый класс: SingleEntityActivity

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* TextView titleTextView, startBalanceTextView – поля редактирования названия счёта и начального баланса соответственно

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации всех фрагментов и загрузчиков
* boolean onOptionsItemSelected(MenuItem item) – обработка нажатия элемента в меню
* void initializeViews() – инициализация элементов представления
* void saveAccount(View view) – валидация данных и сохранение счёта

## 4.8 Класс CategoriesActivity

В данном классе описано Activity, предоставляющий интерфейс доступа к категориям. Реализуется логика отображения категорий и разделения категорий на группы.

Базовый класс: AppCompatActivity

Реализуемые интерфейсы: Отсутствуют

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации всех фрагментов и загрузчиков

Вложенные классы:

* SectionsPagerAdapter extends FragmentStatePagerAdapter – отвечает за постраничный просмотр категорий. Предоставляет возможность просмотра категорий по группам: «Траты», «Пополнения». Использует класс FragmentCategories

## 4.9 Класс FragmentCategories

В данном классе описан Fragment, реализующий вывод категорий. Тут генерируется контекстное меню при долгом нажатии на элемент списка. Для манипуляции категориями вызывает SingleCategoryActivity.

Базовый класс: UsingDataBaseListFragment

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* String query – запрос к базе данных для получения платежей
* SimpleCursorAdapter categoriesCursorAdapter– адаптер для вывода категорий

Методы класса:

* static FragmentCategories FragmentCategoriesFactory (String query) – фабричный метод для генерации объектов класса, инициализируя поле query переданным значением
* void onActivityCreated(@Nullable Bundle savedInstanceState) – инициализация полей и обработчиков событий
* void onResume() – обработка возвращения пользователя к Fragment
* void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) – генерация контекстного меню
* boolean onContextItemSelected(MenuItem item) – обработка нажатия на элементе в контекстном меню
* setQuery(String query) – инициализация поля query переданным значением
* String getBaseQuery() – метод для получения запроса к базе данных, возвращающего все платежи
* void initializeAdapter() – инициализация адаптера для вывода элементов списка
* void requeryCursor() – перезагрузка данных для курсора

## 4.10 Класс SingleCategoryActivity

В данном классе описано взаимодействие с конкретной категорией платежа пользователя: создание, изменение и удаление.

Базовый класс: SingleEntityActivity

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* TextView titleTextView– поле редактирования названия категории
* RadioButton incomeCategoryRadioButton, outgoCategoryRadioButton – радио-кнопки выбора типа категории (трата/пополнение)
* Spinner logoSpinner – выпадающий список для выбора логотипа категории

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации всех фрагментов и загрузчиков
* boolean onOptionsItemSelected(MenuItem item) – обработка нажатия элемента в меню
* void initializeViews() – инициализация элементов представления
* void initializeSpinners() – инициализация выпадающего списка
* void saveCategory (View view) – валидация данных и сохранение категории

## 4.10 Класс SingleTransactionActivity

В данном классе описано взаимодействие с конкретным платежом пользователя: создание, изменение и удаление.

Базовый класс: SingleEntityActivity

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* TextView currentDateTextView, currentTimeTextView – поля вывода даты и времени соответственно
* Spinner categoriesSpinner, accountsSpinner – выпадающие списки категорий и счетов
* EditText transactionAmountEditText, noteEditTextEditText – поля ввода суммы платежа и заметки соответственно
* Button signButton – кнопка управление знаком суммы платежа

Методы класса:

* void onCreate(Bundle savedInstanceState) – выполнение инициализации всех фрагментов и загрузчиков
* void onResume() – обработка возвращения пользователя к Activity
* boolean onOptionsItemSelected(MenuItem item) – обработка нажатия элемента в меню
* void initializeViews() – инициализация элементов представления
* void initializeSpinners() – инициализация выпадающих списков
* void initializeListeners() – инициализация обработчиков событий
* void setInitialDateTime() – установка текущего времени и его отображение
* void setDate(View view) – сохранение даты, выбранной пользователем
* void setTime(View view) – сохранение времени, выбранного пользователем
* void saveTransactionContinue(View view) – обработчик нажатия кнопки «Сохранить и новый»
* void saveTransactionClose(View view) – обработчик нажатия кнопки «Сохранить»
* boolean executeSaving() – сохранение данных
* boolean executeDataBaseSaving() – сохранение данных в базу данных
* boolean validateData() – валидация данных
* void addNewCategory (View view) – обработка нажатия кнопки создания новой категории
* void addNewAccount(View view) – обработка нажатия кнопки создания нового счёта

## 4.11 Класс SQLiteHandler

Класс, реализующий управление созданием базы данных и управлением её версиями.

Базовый класс: SQLiteOpenHelper

Реализуемые интерфейсы: Отсутствуют

Поля класса:

* static final int DATABASE\_VERSION – версия базы данных
* static final String DATABASE\_NAME – название базы данных
* Context context – контекст, в котором используются объекты данного класса

Методы класса

* SQLiteHandler (Context context) – конструктор класса. Производится инициализация переменных
* void onCreate(SQLiteDatabase db) – первичное создание базы данных
* void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) – обновление базы данных
* void onConfigure(SQLiteDatabase db) – конфигурация подключения к базе данных
* createCategories(SQLiteDatabase db) – первичное заполнение базы данных категориями
* createAccounts(SQLiteDatabase db) – первичное заполнение базы данных счетами

## 4.12 Описание базы данных

Схема базы данных, используемой в разрабатываемом приложении, имеет следующий вид:

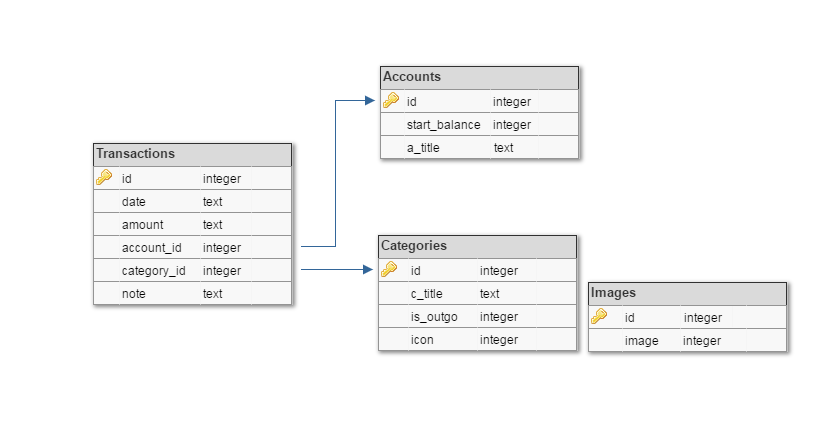


Рисунок 4.12.1 – Схема базы данных

Таблица Transactions – каждая запись таблицы представляет собой платеж в приложении

* date – дата платежа
* amount – сумма платежа
* account\_id – id счёта, для которого производится платеж
* category\_id – id категории, для которой производится платеж
* note – заметка

Таблица Accounts – каждая запись таблицы представляет собой счёт в приложении

* start\_balance – баланс счета на момент его создания
* a\_title – название счета

Таблица Categories – каждая запись таблицы представляет собой категорию в приложении

* с\_title – название счета
* is\_outgo – поле, определяющее трата это или пополнение
* icon – иконка категории

Таблица Images – каждая запись таблицы представляет собой изображение в приложении

* image – название изображения из файлов ресурсов

Каждая таблица содержит поле id, которое уникально идентифицирует каждую запись.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Начало роботы с приложением

Используя данное приложение, пользователю предоставляется возможность выполнять следующие действия:

* Проводить операции над платежами
* Добавлять/изменять/удалять категории
* Добавлять/изменять/удалять счета
* Просмотр статистики в виде диаграмм
* Просмотр журнала за определенный период времени
* Просмотр журнала отдельно для расходов и доходов
* Просмотр категорий отдельно для расходов и доходов
* Просмотр счетов

## 5.2 Окно «Журнал»

При первом запуске приложения пользователь увидит следующее окно:

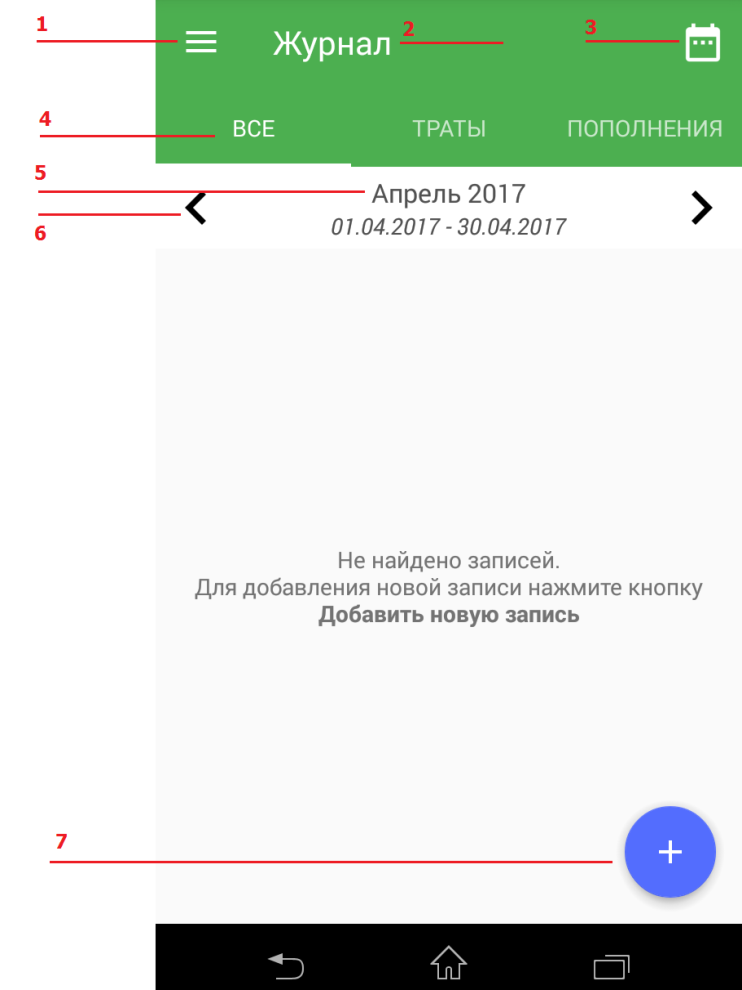


Рисунок 5.1 – Окно приложения «Журнал»

Опишем подробно каждый из замаркированных элементов.

1. Кнопка открытия меню приложения. При её нажатии будет открыто боковое меню. Также, его можно открыть с помощью слайда с края экрана слева направо. Детальнее меню будет описано в одном из следующих подразделов
2. Заголовок окна
3. Меню выбора шага периода для отображения журнала. Доступны опции: день, неделя, месяц, год. После изменения шага временного периода, данные будут обновлены автоматически и будут отображены для периода, содержащего текущую дату
4. Заголовок постраничного просмотра. Журнал можно просматривать для всех платежей, а также отдельно для трат и пополнений. Сменить страницу просмотра можно либо нажав на один из заголовков или слайдом по экрану в одну из сторон
5. Период, для которого отображается журнал. Значения показаны в двух форматах: дд.мм.гггг – дд.мм.гггг и в удобном удобно читаемом формате
6. Кнопка перехода к предыдущему периоду времени (аналогичная и для перехода к следующему периоду времени). Нажав на неё, будет обновлен период времени и соответственно обновлены отображаемые данные. Например, если выбрана опция «Месяц», то после нажатия данной кнопки, будут отображены данные для «Март 2017»
7. Кнопка добавления нового платежа. Нажав на неё, пользователь перейдет к новому окну

После добавления некоторых записей, окно журнал примет следующий вид (процесс добавления новых записей будет рассмотрен далее):

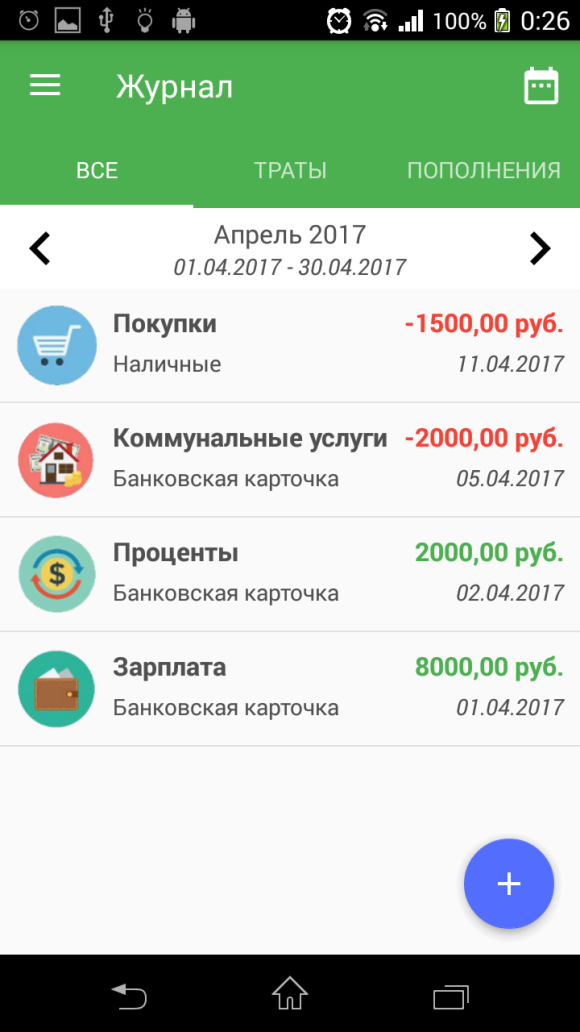


Рисунок 5.2 – Окно приложения «Журнал». Группа «Все»

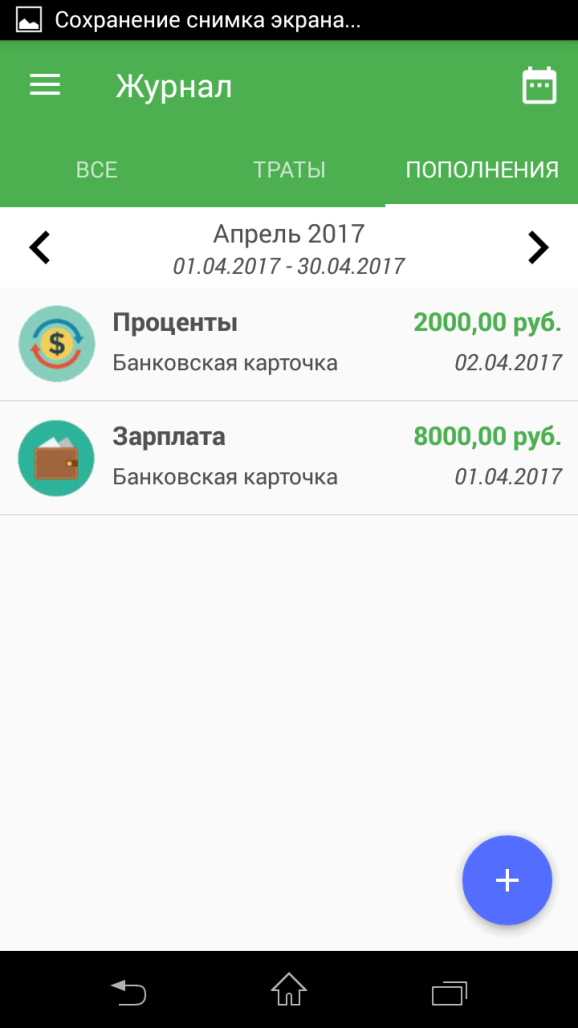


Рисунок 5.3 – Окно приложения «Журнал». Группа «Пополнения»

Каждая запись в журнале имеет стандартную форму. Рассмотрим её подробнее.

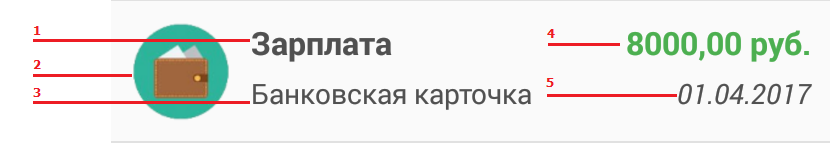


Рисунок 5.4 – Элемент списка журнала

1. Название категории, для которой совершался платеж
2. Логотип категории
3. Название счёта, для которого совершался платеж
4. Сумма платежа. В случае пополнения цвет будет зеленым, в обратном случае – красным
5. Дата совершения платежа

## 5.3 Окно «Платеж»

Подробно рассмотрим каждый элемент окна «Платеж»

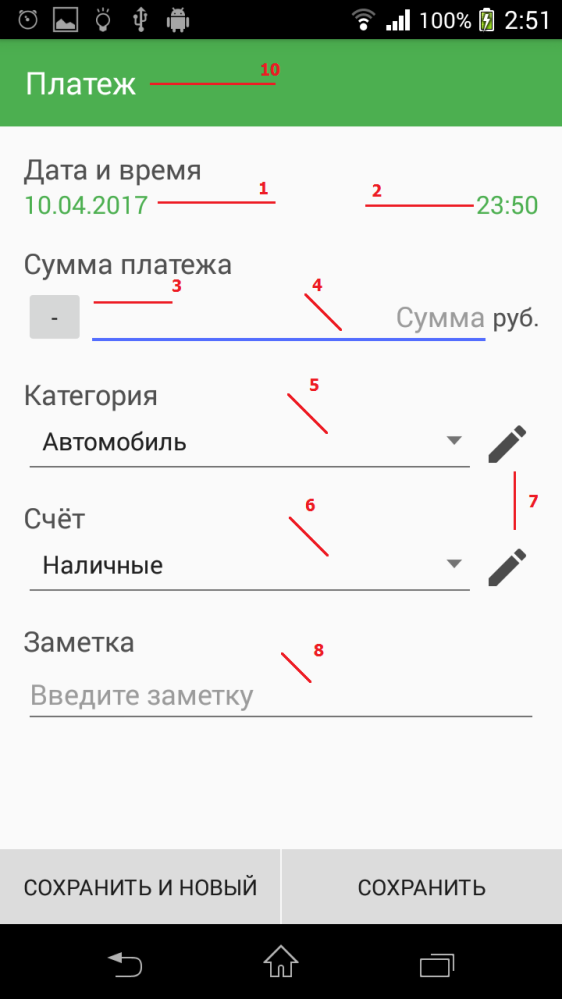


Рисунок 5.5 – Окно приложения «Платеж»

1. Текстовое поле с указание даты платежа. При нажатии на каждое – появится диалоговое окно с возможностью изменить данный параметр.
2. Текстовое поле с указанием времени платежа. При нажатии результат аналогичный дате.
3. Поле для выбора знака суммы (отрицательная или положительная)
4. Поле для ввода суммы платежа
5. Выпадающий список для выбора категории платежа
6. Выпадающий список для выбора счёта, для которого производится платеж
7. Кнопки для добавления новой категории и счета
8. Поле для ввода заметки о платеже
9. Кнопка «Сохранить» сохраняет введенные данные и возвращает пользователя к просмотру платежей, а кнопка «Сохранить и новый» - сохраняет платеж и предоставляет возможность ввести новый
10. Заголовок окна

## 5.4 Окна «Счета» и «Счет»

Пользователю предоставляется возможность просматривать счета. Окно просмотра счетов выглядит следующим образом:

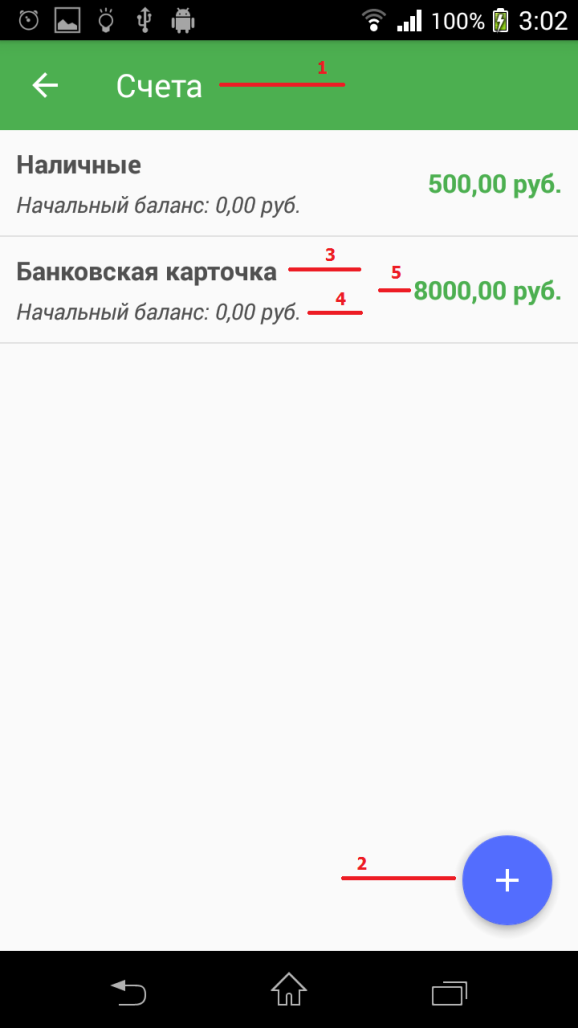


Рисунок 5.6 – Окно приложения «Счета»

1. Заголовок окна
2. Кнопка для добавления нового счета

Опишем элементы списка счетов:

1. Название счета
2. Начальный баланс счета
3. Текущий баланс счета

Окно «Счет» имеет достаточно тривиальный вид: имеет поля ввода названия платежа, его начального баланса и кнопку сохранения. Данное окно имеет следующий вид:

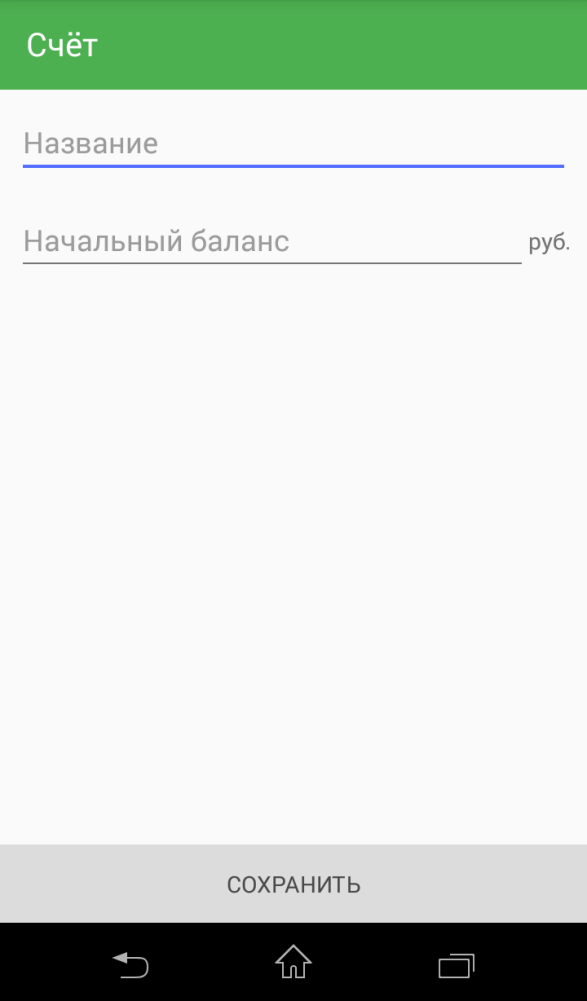


Рисунок 5.7 – Окно приложения «Счет»

## 5.5 Окно «Категории»

Детально опишем каждый элемент окна категории:

1. Заголовок окна
2. Заголовок постраничного просмотра. Категории можно просматривать для категорий трат и пополнений отдельно. Сменить страницу просмотра можно либо нажав на один из заголовков или слайдом по экрану в одну из сторон.
3. Кнопка для добавления новой категории

Опишем элементы списка категорий:

1. Логотип категории
2. Название категории

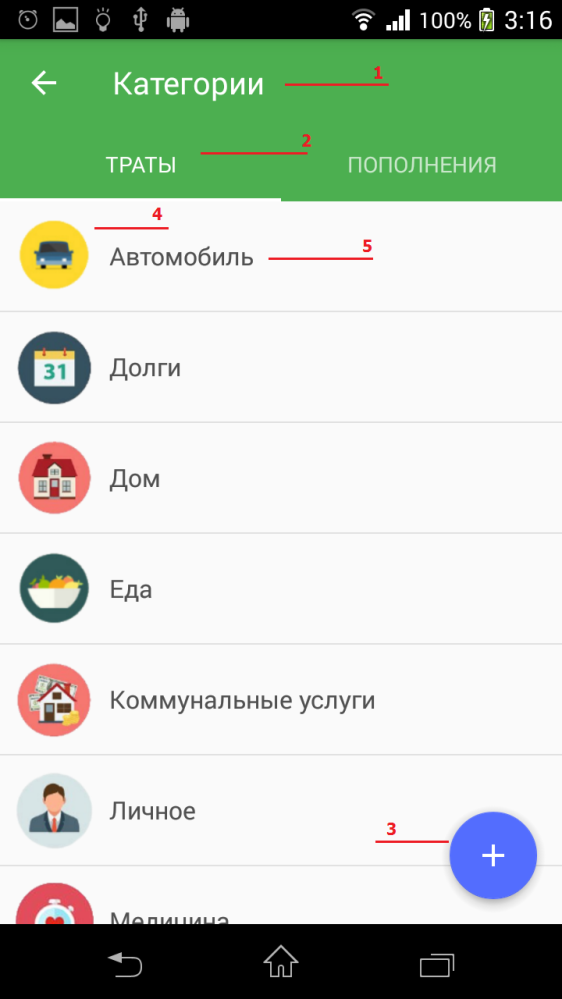


Рисунок 5.8 – Окно приложения «Категории»

## 5.6 Окно «Категория»

Окно категории имеет следующие компоненты:

1. Заголовок окна
2. Выпадающий список для выбора логотипа категории
3. Поле для ввода названия категории
4. Радио-кнопки для выбора типа категории (трата/пополнение)
5. Кнопка для сохранения категории

Скриншот данного окна можно увидеть на рисунке 5.9.

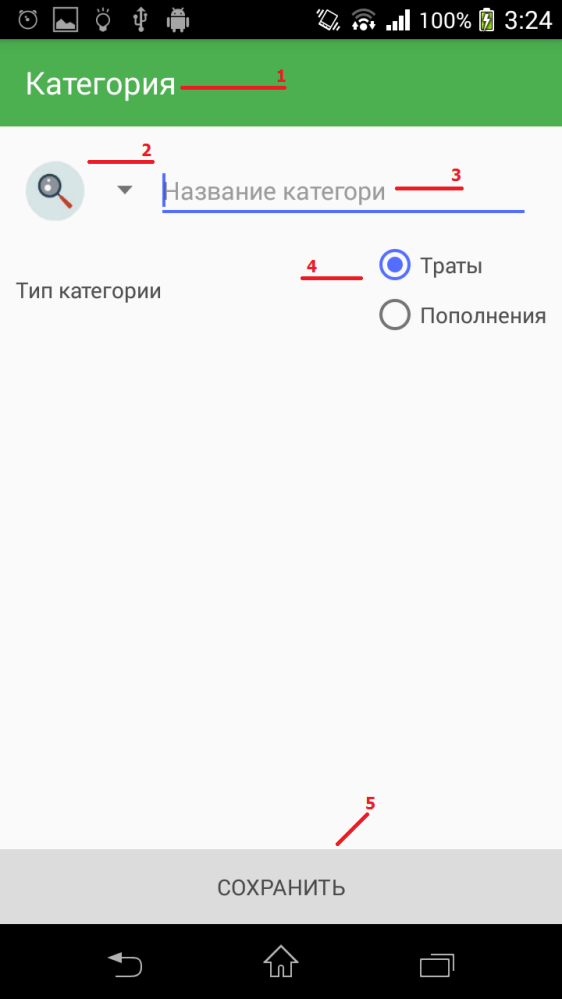
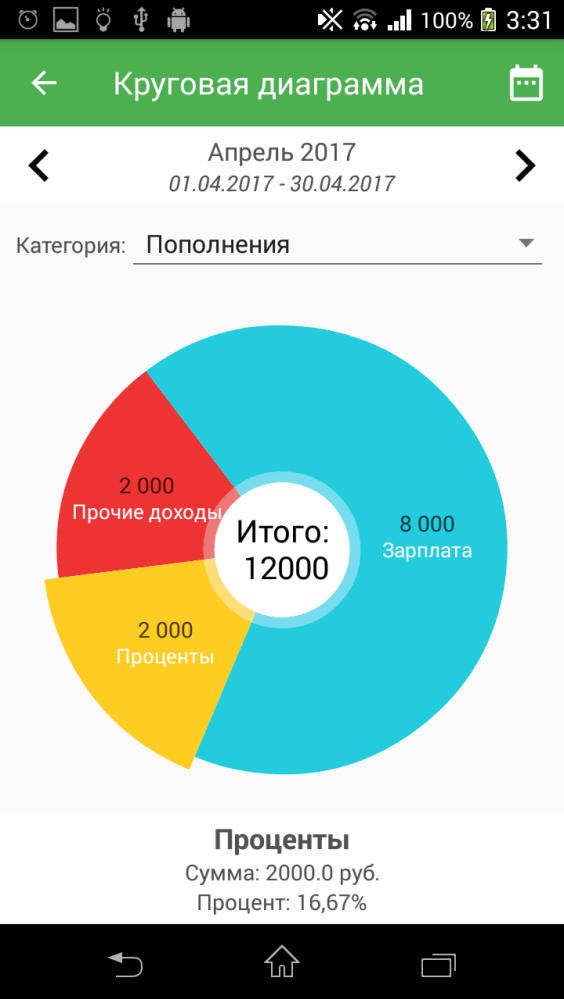
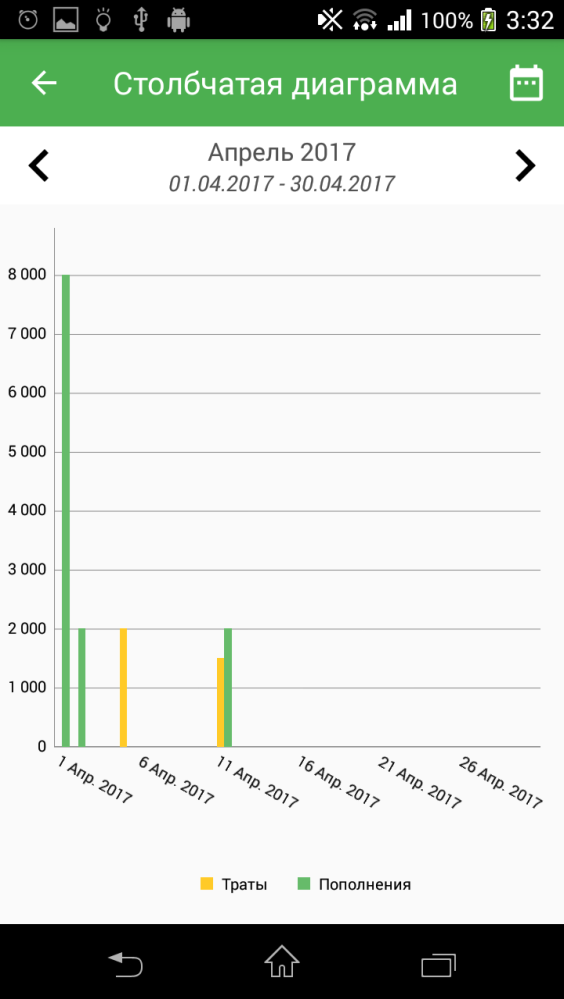


Рисунок 5.9 – Окно приложения «Категория»

## 5.7 Окно «Круговая диаграмма» и «Столбчатая диаграмма»

Данные окна несут в себе информативный характер, поэтому главным их элементом является сам график. Также тут доступны функции для выбора периода времени, аналогично окну «Журнал». Их вид можно увидеть на рисунках 5.10, 5.11

Рисунки 5.10, 5.11 – Окна приложения «Круговая диаграмма» и «Столбчатая диаграмма» соответственно

# 6 ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было разработано мобильное приложение, используя которое пользователь может вести собственную домашнюю бухгалтерию. Для этого были реализованы основные функции, такие как: работа с платежами (их создание, изменение, удаление), работа со счетами, работа с категориями.

Каждому ключевому процессу было выделено отдельное окно, что позволяет пользователю сосредоточиться на выполнении конкретного процесса, не отвлекаясь на прочие детали.

Немаловажную в разработанном приложении роль, играет возможность просмотра диаграмм трат и платежей. Представлены два вида диаграмм: круговая и столбчатая. Плюсом просмотра статистики в таком виде является то, что человеку удобнее воспринять визуальное представление данных, а не сырые цифры.

Для реализации данного приложения использовался язык программирования Java и система управления базами данных SQLite. Разметка окон приложения делалась с помощью расширяемого языка разметки XML. Использовалась интегрированная среда разработки Android Studio.

# СПИСОК ЛИТЕРАТУРЫ

1. Эккель Б. Философия Java [Текст] / Б. Эккель. – СПб: Питер, 2009. – 640 с.
2. Шилдт Г. Java 8. Полное руководство [Текст] / Г. Шилдт. – Москва: Издательский дом «Вильямс», 2016. – 1376 с.
3. Дейтел П. Android для разработчиков [Текст] / П. Дейтел, Х.Дейтел. – СПб: Питер, 2016. – 512 с.
4. Харди Б. Программирование под Android [Текст] /Б. Харди, Б. Филлипс – СПб: Питер, 2014. – 592 с.
5. Черемных С.В Моделирование и анализ систем [Текст] / С.В. Черемных – Москва: Финансы и статистика, 2006. – 192 с.
6. Сайт Android [Электронный ресурс]. – Режим доступа: https://www.android.com/
7. Сайт SQLite [Электронный ресурс]. – Режим доступа: <http://www.sqlite.org>
8. Сайт Android Developers [Электронный ресурс]. – Режим доступа: <https://developer.android.com>