**ЗМІСТ**

[1 ПОСТАНОВКА ЗАДАЧІ 5](#_Toc514610116)

[1.1 Функціональні вимоги до програмного продукту 5](#_Toc514610117)

[1.2 Нефункціональні вимоги до програмного продукту 6](#_Toc514610118)

[1.3 Огляд та аналіз аналогів розроблюваного продукту 7](#_Toc514610119)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ВИКОРИСТОВУВАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 11](#_Toc514610120)

[2.1 Цільова платформа 11](#_Toc514610121)

[2.2 Язык программирования 13](#_Toc514610122)

[2.3 Система управления базами данных 14](#_Toc514610123)

[3 МОДЕЛИРОВАНИЕ ПРОЦЕССОВ 16](#_Toc514610124)

[3.1 Методология IDEF0 17](#_Toc514610125)

[3.2 IDEF0-диаграмма для разрабатываемого продукта 18](#_Toc514610126)

[3.2.1 Обращение к системе 20](#_Toc514610127)

[3.2.2 Обработка действий пользователя 22](#_Toc514610128)

[3.2.3 Изменение базы данных 23](#_Toc514610129)

[4 ОПИСАНИЕ АЛГОРИТМОВ 25](#_Toc514610130)

[4.1 Основной цикл программы 25](#_Toc514610131)

[4.2 Создание счетов 26](#_Toc514610132)

[4.3 Создание категорий 27](#_Toc514610133)

[4.4 Создание платежа 28](#_Toc514610134)

[5 СТРУКТУРА ДАННЫХ И РЕСУРСОВ ПРОГРАММЫ 30](#_Toc514610135)

[5.1 Класс MainActivity 30](#_Toc514610136)

[5.2 Класс UsingDataBaseListFragment 31](#_Toc514610137)

[5.3 Класс FragmentTransactions 31](#_Toc514610138)

[5.4 Класс UsingDataBaseActivity 33](#_Toc514610139)

[5.5 Класс SingleEntityActivity 34](#_Toc514610140)

[5.6 Класс AccountsActivity 34](#_Toc514610141)

[5.7 Класс SingleAccountActivity 35](#_Toc514610142)

[5.8 Класс CategoriesActivity 36](#_Toc514610143)

[5.9 Класс FragmentCategories 36](#_Toc514610144)

[5.10 Класс SingleCategoryActivity 37](#_Toc514610145)

[5.11 Класс SingleTransactionActivity 38](#_Toc514610146)

[5.12 Класс SQLiteHandler 39](#_Toc514610147)

[5.13 Описание базы данных 40](#_Toc514610148)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 42](#_Toc514610149)

[6.1 Начало роботы с приложением 42](#_Toc514610150)

[6.2 Окно «Журнал» 42](#_Toc514610151)

[6.3 Окно «Платеж» 45](#_Toc514610152)

[6.4 Окна «Счета» и «Счет» 46](#_Toc514610153)

[6.5 Окно «Категории» 48](#_Toc514610154)

[6.6 Окно «Категория» 49](#_Toc514610155)

[6.7 Окно «Круговая диаграмма» и «Столбчатая диаграмма» 50](#_Toc514610156)

[7 ЗАКЛЮЧЕНИЕ 52](#_Toc514610157)

[СПИСОК ЛИТЕРАТУРЫ 53](#_Toc514610158)

ВСТУП

Проблема управління особистим бюджетом є актуальною для кожної людини. Ми постійно здійснюємо різні покупки, беремо гроші в борг, зберігаємо заощадження. І наш бюджет стрімко падає, якщо ми неефективно витрачаємо наявні фінанси. Грамотно розпоряджається своїм бюджетом людина постійно стежить за тим, скільки грошей у нього є, розраховує, скільки грошей йому потрібно витратити, і приймає рішення, де він може заощадити і від чого він може взагалі відмовитися. Ми могли б багаторазово спростити дані процеси, розробивши автоматизовану систему управління особистим бюджетом. Саме для цієї мети і був придуманий проект «Домашня бухгалтерія», про який далі піде мова.

Мета «Домашньої бухгалтерії» - автоматизувати наші повсякденні фінансові розрахунки.

Майже кожен день ми витрачаємо гроші на різні покупки та послуги. І для найбільш ефективного розподілення бюджету необхідно контролювати будь-які транзакції, щоб потім проаналізувати їх і зробити певні висновки. Ми завжди витрачаємо гроші та отримуємо їх у різні джерела, наприклад готівка або банківська карта. І дуже важко контролювати бюджет на кожному з джерел, а тим паче й причини витрат/надходжень на кожне з них.

Розроблювана система дозволить нам вести щоденник витрат і надходжень, розраховувати, як вони вплинуть на загальний бюджет, а також аналізувати, коли і через що було витрачено / отримано найбільшу кількість грошей. Також, буде можливість групувати витрати / надходження, тобто транзакції, за датою їх здійснення, за їх причино. Це дозволить ефективно аналізувати сезонність транзакції, зробити висновки щодо того, на що витратилось більше, ніж планувалось та скорегувати майбутнє користування власним бюджетом. Система дозволить групувати транзакції за різними джерелами для більш точного аналізу звідки витрачається, або надходить найбільше грошей. Основним методом аналізу буде перегляд усієї статистики за допомогою зручних діаграм та графіків, перегляд транзакцій з різними фільтрами та різною датою створення.

Система повинна бути ефективною, в плані обробки даних, виконання програмних функцій; повинна бути стійкою, тобто виконувати всі функції, які будуть розроблені, незалежно від зовнішніх факторів або ж реагувати на зміни шляхом виклику системної помилки (можливо, показувати на якому місці стався збій). Повинні бути створені діалогові вікна для інтерактивного режиму роботи з користувачем. Виконання вимог ергономіки інтерфейсу, створення комфортних умов роботи.

Ціль даної роботи є проектування та розробка мобільного додатка для ведення домашньої бухгалтерії. Користувачеві буде надаватися можливість внесення витрат / доходів, ведення обліку їх категорій, створення власних рахунків і робота з ними. Також, можливий перегляд історії для конкретного проміжку часу і перегляд статистики у вигляді кругової і стовпчатої ​​діаграм.

У пояснювальній записці будуть описані основні етапи проектування та розробки програмного забезпечення, основні алгоритми його роботи, деталі реалізації, та, у кінцевому варіанті – керівництво користувача.

# ПОСТАНОВКА ЗАДАЧІ

## Функціональні вимоги до програмного продукту

Функціональні вимоги визначають основний фронт робіт розробника, регламентують поведінку додатки в тій чи іншій ситуації. Вони ставлять завдання, які повинна виконувати система.

Необхідно створити програмне забезпечення для ведення домашньої бухгалтерії, що надає достатній функціонал для аналізу витрат / доходів і отримання наочної статистики.

З точки зору функціональних характеристик виділяються нижче перераховані вимоги:

Робота з платежами (транзакціями):

* Вибір дати і часу
* Введення суми платежу
* Вибір категорії, для якої здійснюється платіж
* Вибір рахунку, для якого здійснюється платіж
* Створення нотатки
* Збереження нового платежу
* Редагування і видалення існуючих платежів
* Внесення платежу за допомогою аналізу входять смс на основі шаблонів

Робота с категоріями:

* Введення назви категорії
* Вибір типу категорії: витрата, дохід
* Вибір логотипу категорії
* Збереження нової категорії
* Редагування і видалення існуючих категорій

Работа із рахунками:

* Введення назви рахунку
* Введення початкового балансу рахунку
* Збереження нового рахунку
* Редагування і видалення існуючого рахунку

Перегляд статистики у вигляді діаграм:

* Кругова діаграма
* Стовпчаста діаграма

Основний діалог програми:

* Перегляд журналу платежів окремо для витрат, поповнень
* Перегляд журналу для різних часових діапазонів: день, тиждень, місяць, рік
* Перегляд діаграм для різних часових діапазонів
* Навігаційне меню

## Нефункціональні вимоги до програмного продукту

Нефункціональні вимоги, визначають вимоги до системи в цілому, а не в окремих випадках використання, описують властивості і характеристики, які повинна демонструвати система, а також обмеження, які повинні бути дотримані в окремих випадках.

Виділимо нефункціональні вимоги для наступних груп: Зовнішні інтерфейси, атрибути якості, обмеження.

Зовнішні інтерфейси:

- Інтерфейс програми повинен мати привабливий дизайн

- Інтерфейс повинен бути інтуїтивно зрозумілим

- Інтерфейс повинен бути естетичним

- Інтерфейс програми повинен відповідати основним правилам юзабіліті

Дизайн додатків повинен бути в стилі Material designАтрибуты якості:

* Надійність - додаток повинен вірно реагувати на будь-які данні, введені користувачем, що не зависати під час роботи. Додаток має бути захищеним від помилок і гарантувати цілісність даних. Продуктивність - додаток повинен виконувати всі передбачені операції без довгих очікувань і зависань. Виконання кожної операції має проходити в найкоротший термін.
* Експлуатаційна придатність - додаток повинен виконувати всі необхідні вимоги і бути працездатним незалежно від часу його використання Обмеження:
* Додаток виповнюється на мобільних пристроях під управлінням операційної системи Android
* Мінімальним API є 16

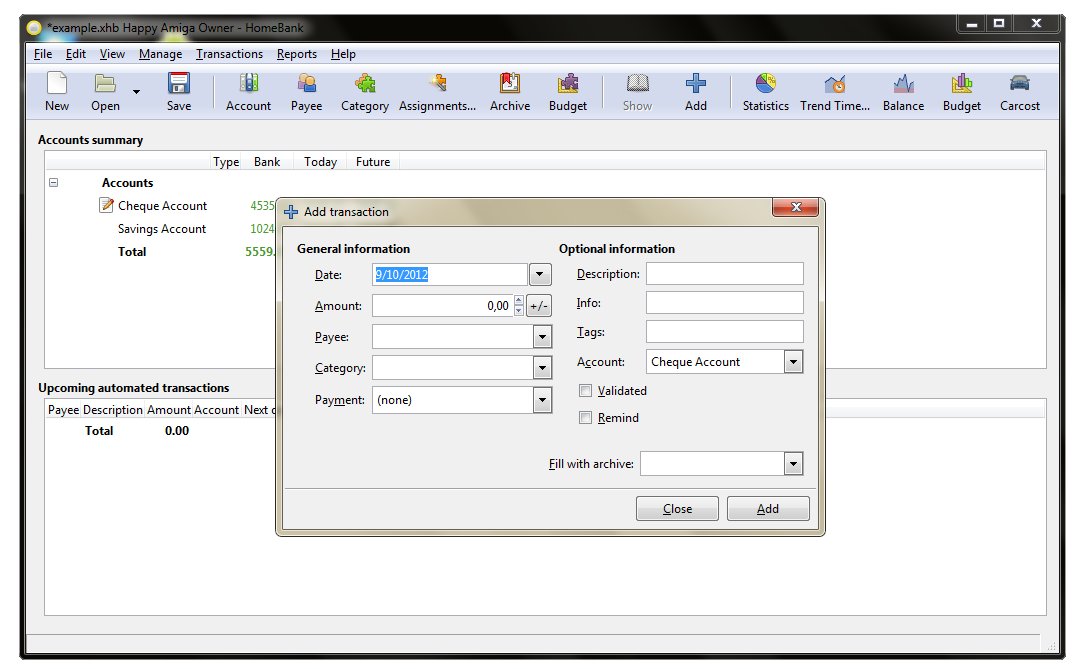
## Огляд та аналіз аналогів розроблюваного продукту

Розглянемо деякі раніше створені програмні продукти.

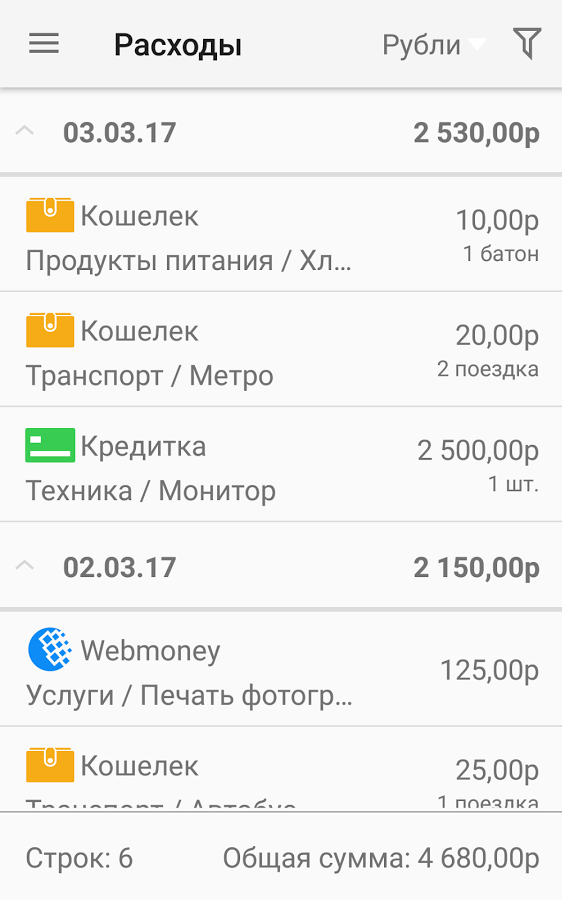
HomeBank - загальнодоступна програма для ведення домашньої бухгалтерії. Надається можливість контролювати свої доходи і витрати, аналізувати бюджет. Програма дозволяє впорядковувати платежі за категоріями, планувати транзакції для автоматичного вносити в базу даних (наприклад, щоденні рахунки). У програмі також є функція одночасного редагування кількох полів.

За допомогою HomeBank можна аналізувати фінансовий стан і складати звіти. Програма надає можливість генерувати звіти по поточному стану бюджету. Також, можливо налаштувати фільтрацію транзакцій і параметрів звіту. Присутній функціонал візуалізації даних у вигляді графіків і діаграм.

Можна зробити висновок, що програма HomeBank добре справляється з поставленими завданнями. Але, також, в ній присутні мінуси: програма поширюється виключно на комп'ютери, що не дає можливості оперативно занести платежі. Також в програмі HomeBank занадто захаращений інтерфейс і присутній надмірний функціонал.



* Малюнок 1.3.1 - Додаток HomeBank
* Далі, розглянемо додаток «Keepsoft домашня бухгалтерія»
* Цей додаток містить у собі наступний функціонал:
* Облік витрат / доходів
* Можливість ведення рахунків
* Можливість розподілу витрат / доходів за категоріями
* Облік боргів
* Складання бюджету витрат і доходів
* Планування бюджету
* параметризованих пошук
* Можливість застосування фільтрів
* Погашення кредитів і боргів по частинах

Головним мінусом даної програми є те, що безкоштовного функціоналу мало для того, щоб зручно користуватися програмою. Також відсутня функція візуального представлення даних: у вигляді діаграм.

Малюнок 1.3.2 - Додаток «Keepsoft домашня бухгалтерія»

Наступна розглянута програма - Finance PM. Дана програма реалізує наступний функціонал:

Можливість управління гаманцями

* Можливість додавання таких операцій як доходи, витрати і переклади
* Можливість редагувати категорії
* Можливість роботи з декількома валютами
* Можливість регулярних платежів
* Можливість синхронізації даних
* Можливість створення шаблонних операцій

В цілому, як і раніше розглядається додаток «Keepsoft домашня бухгалтерія», даний варіант справляється з поставленими для нього завданнями, але має аналогічний мінус - необхідність придбання дорогої ліцензії.



Малюнок 1.3.3 - Додаток «Finance PM»

# ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ВИКОРИСТОВУВАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

## Цільова платформа

Мета даної роботи - створити додаток з ведення домашньої бухгалтерії для мобільного пристрою. Тому, необхідно вибрати операційну систему, на якій воно буде виконуватися.

Серед головних кандидатів можна виділити Android і iOS. Розглянемо дані операційні системи окремо.

Android - операційна система, розроблена для мобільних пристроїв, електронних книг, смарт-годин, телевізорів та інших пристроїв. Операційна система Android має ядро Linux і має власну реалізацію Java Virtual Machine - Dalvik.

Компанія Google надає можливість писати власні програми на мові програмування Java, використовуючи бібліотеки Android Native Development Kit. Також, надається можливість імпортувати бібліотеки і компоненти додатків мови C та інших.

* Серед переваг Android можна виділити:
* Android має найкращі показники в виряджаючи функцій, як інтернет-серфінг, взаємодія з додатками Google,
* Android є платформою з відкритим вихідним кодом, що надає широке коло можливостей
* Android надає встановлювати програми без інтернету, а також підтримує налагодження додатків по USB, що дозволяє писати і тестувати власні програми на своєму Android-пристрої

- Android доступний для різних апаратних платформ, таких як ARM, MIPS, x86.

* Недоліками Android є:

- Android має ряд вразливостей в безпеці

У зв'язку з тим, що можна встановлювати додатки з «недовірених джерел», розвивається піратство

- Android має надмірну фрагментацію, що тягне за собою чималі проблеми для розробників

Далі, розглянемо операційну систему IOS.

ІOS - Операційна система, що випускається компанією Apple. Призначена для смартфонів, планшетів і програвачів

Для створення програмного забезпечення під ІOS використовується мова програмування Objective-C або Swift і середовище розробки Xcode. Також, можна розробляти на C і C ++.

Серед перевг ІOS можна виділити

Інтерфейс, зрозумілий на інтуїтивному рівні

* Висока, в порівнянні з іншими ОС, швидкість роботи
* Висока практичність додатків
* Відсутність вразливостей безпеки

Багатозадачність (мале споживання ресурсов згорнути Додатками) .Серед недоліків iOS можна виділити:

* Відсутня можливість прямого доступу до файлової системи

Відсутність можливості писати програми без платного софта

Відсутність по-своєму налаштувати пристрій.

Обмеження на встановлювані додатки

На даний час близько 60% всіх мобільних пристроїв знаходяться під управлінням операційної системи Android, коли під керуванням ІOS - менше 20%.

Проаналізувавши операційні системи Android і iOS, прийнято рішення, що цільова платформа для розробляється буде Android.

Дане рішення було прийнято за далі описаним причин. По-перше, Android - відкрита платформа, в зв'язку з чим, на ній можна реалізувати більше функцій. По-друге, Android надає широкий, у порівнянні з iOS, вибір API і різних бібліотек. По-третє, Android має більш широку аудиторію, ніж інші операційні системи.

## 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 является объектно-ориентированным языком программирования и имеет строгую типизацию.

Перечислим основные особенности языка.

Программы, написанные на данном языке, транслируются в специальный байт-код, который может быть распознан на любом устройстве, на котором установлена виртуальная машина Java.

Программы на Java, во время исполнения, полностью контролируются виртуальной машиной, которая поддерживает систему безопасности. Любые действия, которые превышают полномочия программы вызовут прерывание.

Использование виртуальной машины приводит к некоторому снижению производительности. Поэтому, был принят ряд усовершенствований:

* Байт-код транслируется в машинный код непосредственно во время работы программы, позволяя сохранять версии классов в машинном коде
* В стандартных библиотеках используется код, ориентированный на платформу
* Используются аппаратные средства, ускоряющие обработку байт-кода

Среди основных возможностей Java можно выделить:

* Автоматизированная работа с памятью
* Система обработки исключений
* Обширная стандартная библиотека классов
* Возможность создания сетевых приложений
* Возможность создания многопоточных приложений
* Система обобщений.
* Возможность использования лямбда выражений и замыканий

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.

## 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 является встраиваемой реляционной системой управления базами данных.

SQLite не использует парадигму клиент-сервер, поэтому используемый движок является не отдельно работающим процессом, с которым взаимодействует приложение, а предоставляет библиотеку, с которой возможно компоновать программу. В таком случае движок становится составной частью программы. Для обмена данными используются вызовы функций библиотеки SQLite.

Используя такой подход, увеличивается производительность, уменьшаются расходы ресурсов, а также упрощается процесс написания программы.

SQLite хранит базу данных в единственном файле на том компьютере, на котором исполняется программа. Несмотря на такой поход, данные из базы данных можно читать из разных процессов и потоков.

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

# МОДЕЛИРОВАНИЕ ПРОЦЕССОВ

Для улучшения качества разработки программного обеспечения, необходимо провести тщательное моделирование процессов. До того, как писать код непосредственно, нужно описать процесс через различные элементы: действия, данные, события. Моделирование процессов описывает логическую взаимосвязь элементов, присущих процессу.

Благодаря моделированию процессов, можно понять работу в целом и провести её анализ. Для больших проектов стоит выполнять подробное и многогранное моделирование.

Среди основных целей моделирования можно выделить:

* Описать процессы. Моделируя процессы, можно проследить, что выполняется в процессах от их начала до завершения. Это позволяет получить общий взгляд на процессы и помогает увеличить их эффективность
* Нормировать процессы. Моделирование процессов задает правила выполнения процессов, т.е. то, каким образом они должны быть выполнены.
* Установление взаимосвязи процессов. Во время моделирования устанавливается связь между процессами и требованиями к ним.

В данной работе будет рассмотрено функциональное моделирование – вид моделирования, подразумевающий описание процессов в виде взаимосвязанных, четко структурированных функций. В этом виде моделирования реальную последовательность функций соблюдать не обязательно.

Функциональное моделирование будет выполнено с помощью методологии IDEF0.

## Методология IDEF0

Методология IDEF0 подразумевает построение иерархической системы диаграмм - единичных описаний функций в системе. Сначала проводится наиболее общее описание системы, её внешнее взаимодействие (контекстная диаграмма), после чего проводится декомпозиция. Во время декомпозиции система разбивается на другие системы (подсистемы) и каждая подсистема описывается отдельно. Затем, для достижения необходимой степени подробности, каждая подсистема декомпозируется.

Каждая IDEF0-диаграмма содержит блоки и дуги. Блоки изображают функции моделируемой системы. Дуги обозначают взаимосвязь и взаимодействие между блоками.

Функциональные блоки (изображаются прямоугольниками) означают функции, процессы или задачи, выполняемые в системе. Имя блока должно выражаться существительным, обозначающим действие.

Блоки в IDEF0 размещаются по принципу доминирования: самый доминирующий блок располагается сверху, менее доминирующие – ниже. Под доминированием понимается степень влияния, которое один блок оказывает на другой.

Взаимодействие блоков 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 потоки, которые преобразуются в 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, при помощи которых процесс реализуется
* Результат – выходит из работы справа и показывает данные, материальные и информационные потоки, возвращаемые блоком

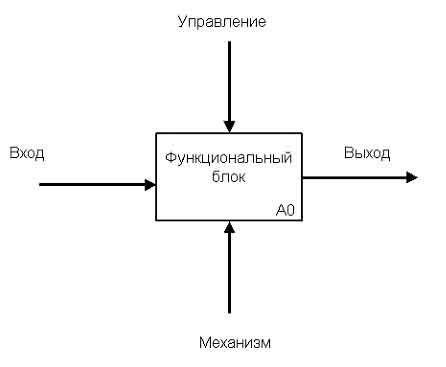


Рисунок 3.1.1 – Назначение стрелок в методологии IDEF0

## 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 «Исходная 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.

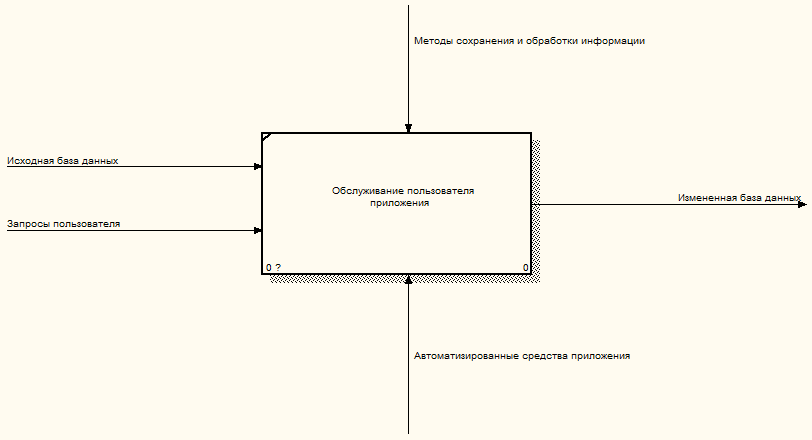


Рисунок 3.2.1 – Контекстная IDEF0-диаграмма

Для уточнения всех процессов произведем декомпозицию контекстной диаграммы.

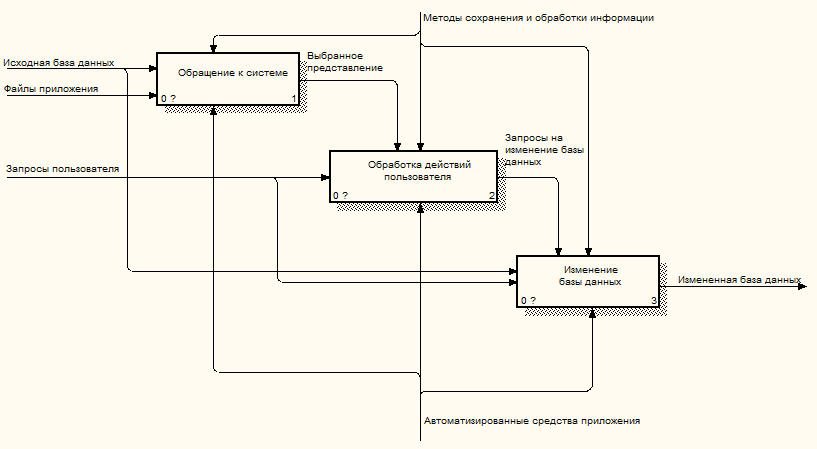


Рисунок 3.2.2 – Декомпозиция контекстной диаграммы

Имеем следующую последовательность обслуживания пользователя приложения:

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

Последовательно декомпозируем все блоки полученной диаграммы.

### 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.

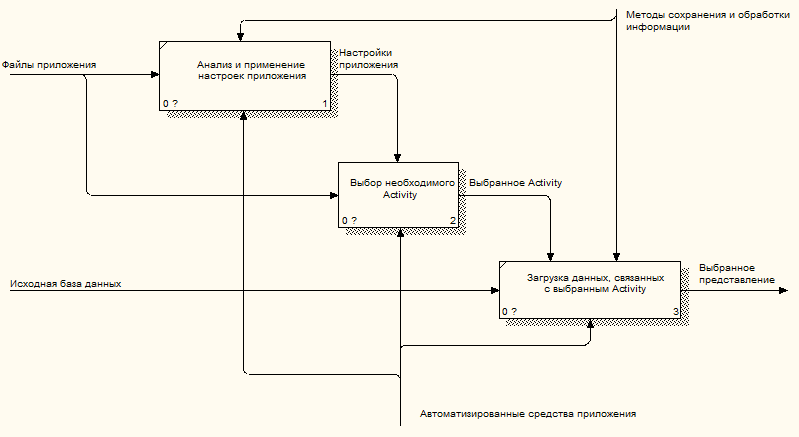


Рисунок 3.2.3 – Декомпозиция работы «Обращение к системе»

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.

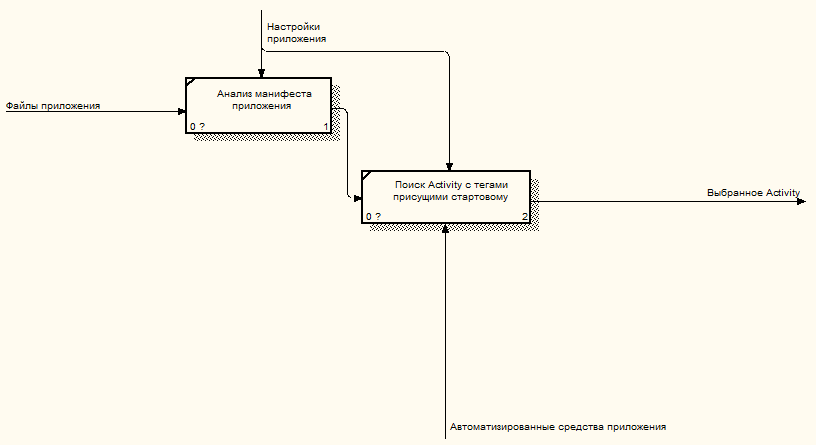


Рисунок 3.2.4 – Декомпозиция работы «Выбор необходимого Activity»

### 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.

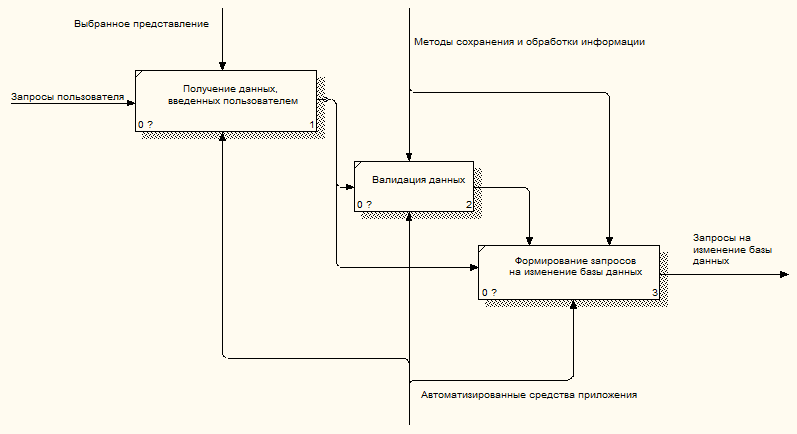


Рисунок 3.2.5 – Декомпозиция работы «Обработка действий пользователя»

### 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.

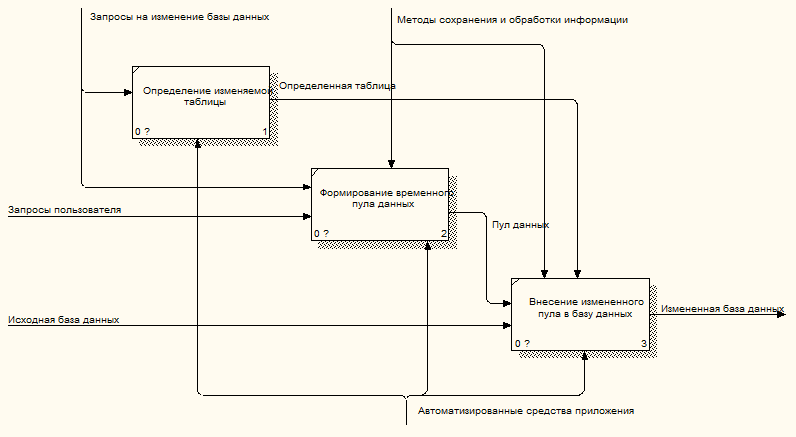


Рисунок 3.2.6 – Декомпозиция работы «Изменение базы данных»

# ОПИСАНИЕ АЛГОРИТМОВ

Опишем основные алгоритмы, которые используются в разрабатываемом приложении. Проведем графическую визуализацию с помощью блок-схем.

## Основной цикл программы

Запуская приложение, пользователь получает просмотреть данные про платежи за определенный период времени, а также отдельно для категорий трат и пополнений. Также, предоставляется возможность просмотреть существующие счета и категории.

Далее, пользователь может принять решение о создании счета, или категории. При наличии счета и категорий, предоставляется возможность создать новый платеж. После создания нового платежа, происходит просмотр диаграмм платежей.

После этого, основной цикл программы либо заканчивается, либо циклически повторяется, начиная с первого этапа.

Блок-схему основного цикла программы можно просмотреть на рисунке 4.1.1.

Основные процессы детальнее будут описаны в следующих подразделах.



Рисунок 4.1.1 – Блок-схема основного цикла программы

## Создание счетов

Нажимая соответствующую кнопку в окне счетов, пользователь переходит к окну создания счета.

Сначала, пользователем вводится название счета, а далее – начальная сумма на счету. При нажатии кнопки для сохранения, происходит проверка введенных данных. При успешной проверке – данные сохраняются в базу данных, иначе – выводится сообщение об ошибке, и требования к параметру, который введен неверно.

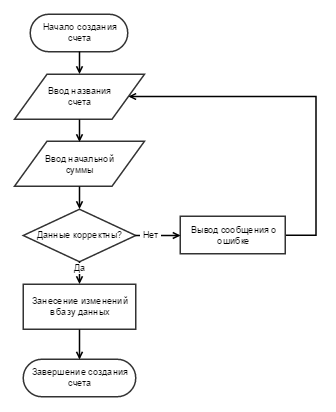


Рисунок 4.2.1 – Блок-схема алгоритма создания счета

## Создание категорий

Нажимая соответствующую кнопку в окне категорий, пользователь переходит к окну создания категории.

Для создания новой категории, пользователь вводит название новой категории, выбирает тип категории (трата или пополнение), а также выбирает иконку для новой категории.

Прежде чем сохранить новую категорию, проходит проверка введенных данных, и, в случае нахождения ошибки – пользователю выводится соответствующее сообщение.

Блок-схема алгоритма создания категории изображена на рисунке 4.3.1.

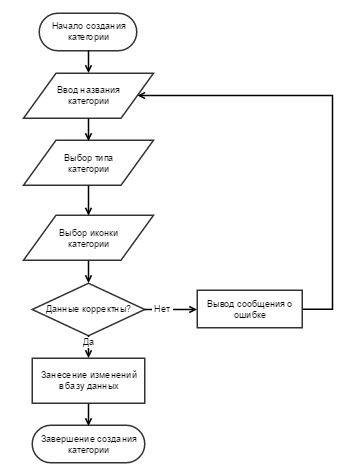


Рисунок 4.3.1 – Блок-схема алгоритма создания категории

## Создание платежа

Последовательность шагов для создания нового платежа, можно изобразить с помощью блок-схемы на рисунке 4.4.1.

Для создания платежа, пользователь вводит дату и время совершения платежа. Далее, происходит ввод суммы платежа, а также выбор категории платежа и счёта для которого данная операция производится.

Перед сохранением платежа в базе, данные введенные пользователем подлежат проверке. При нахождении неверных данных выводится сообщение об ошибке, и требования к параметру, который введен неверно.

Также, если пользователь выберет опцию сохранить текущий платеж и создать новый, то алгоритм повторится с самого начала, иначе – завершить алгоритм.

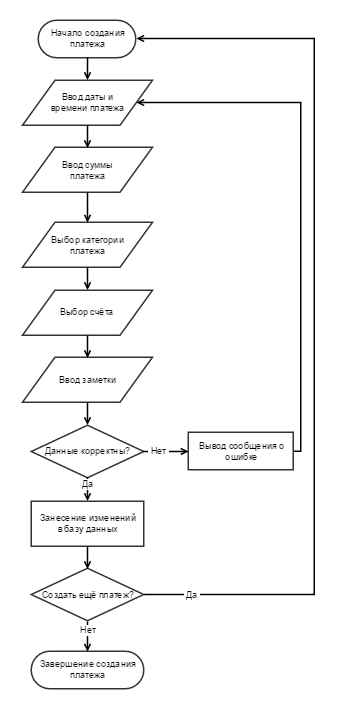


Рисунок 4.4.1 – Блок-схема алгоритма создания платежа

# 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.

## 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.

## 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

## 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

## 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

## 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

## 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

## 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

## 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

## 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

## 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

## 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

## 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

## Описание базы данных

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

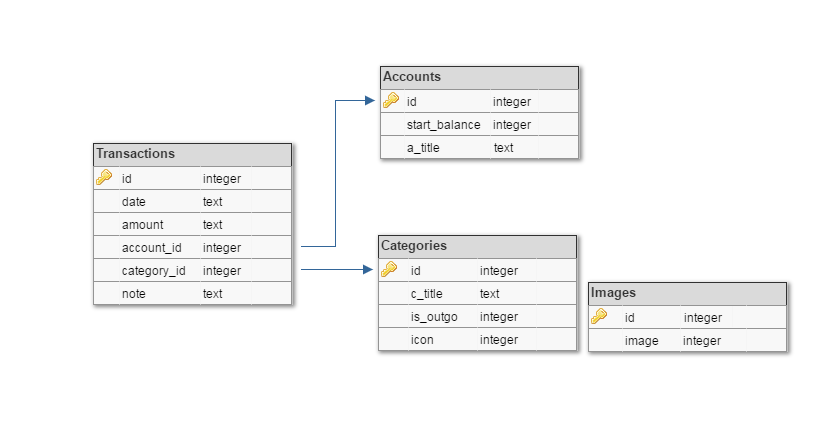


Рисунок 5.13.1 – Схема базы данных

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.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 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

## Окно «Журнал»

При первом запуске приложения пользователь увидит следующее окно:

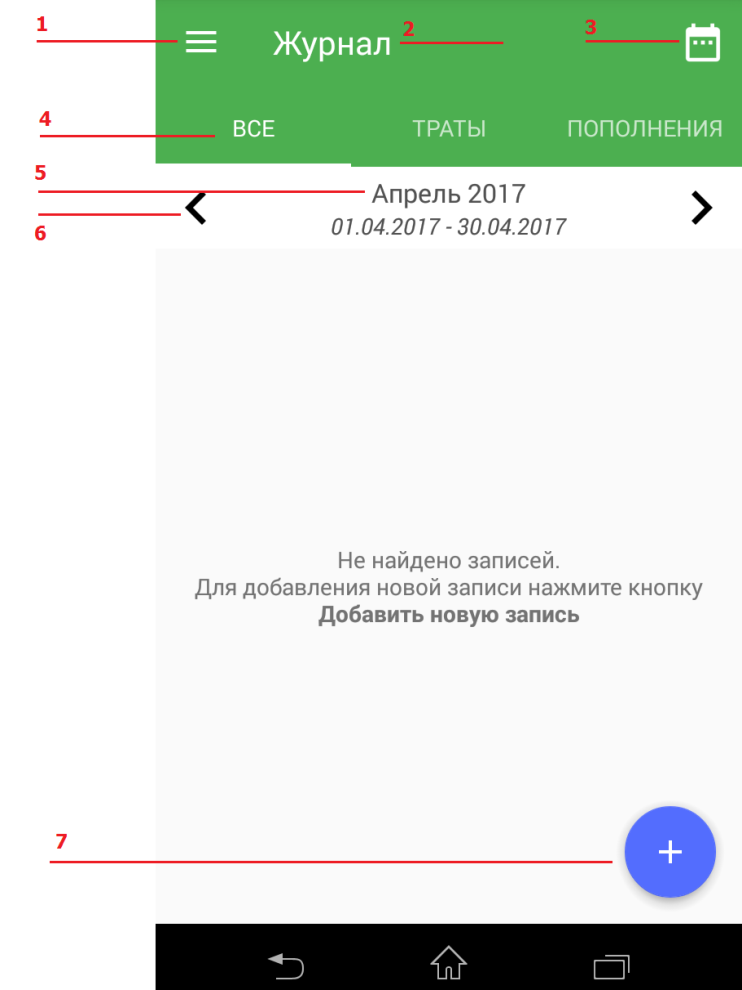


Рисунок 6.2.1 – Окно приложения «Журнал»

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):

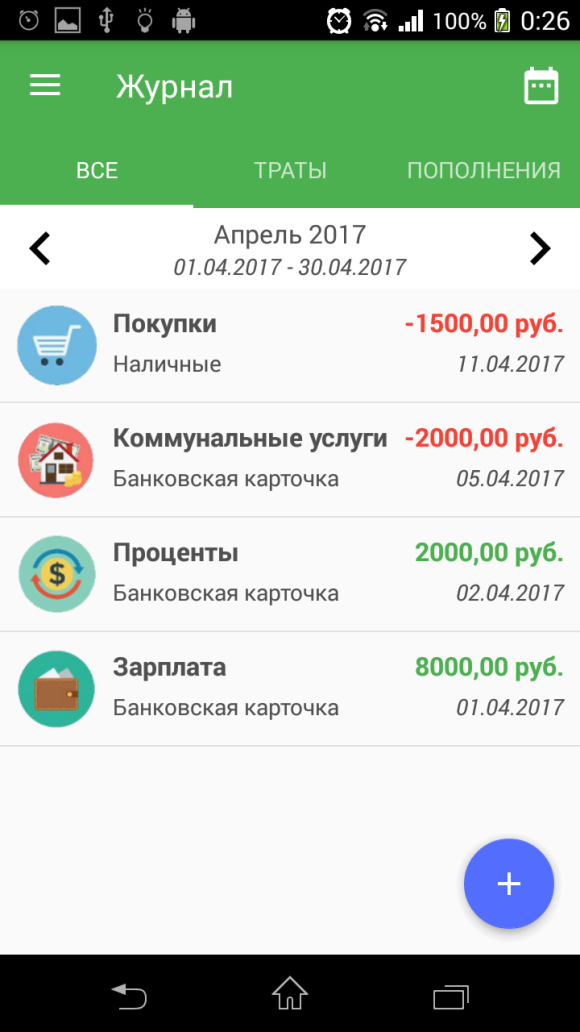


Рисунок 6.2.2 – Окно приложения «Журнал». Группа «Все»

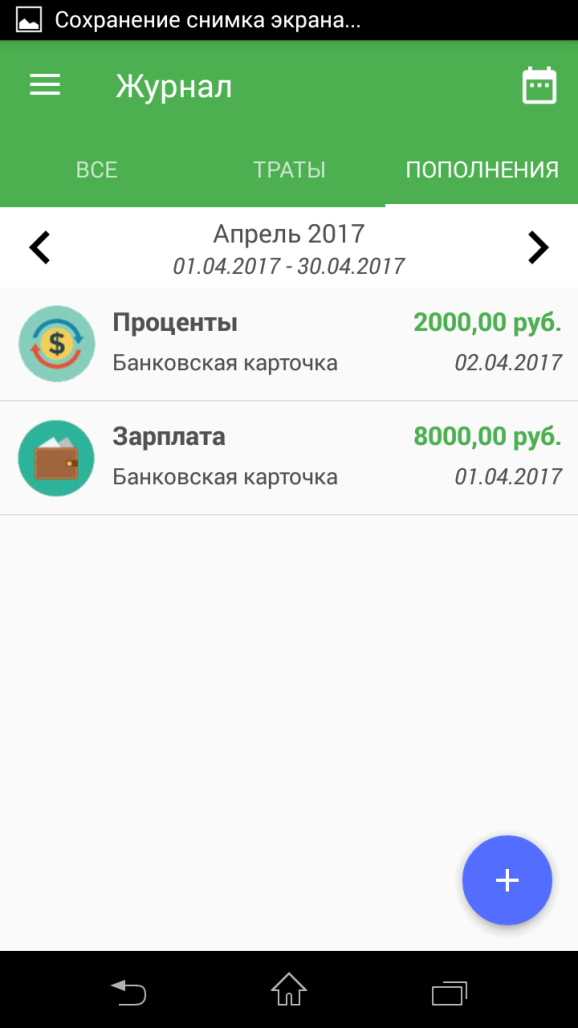


Рисунок 6.2.3 – Окно приложения «Журнал». Группа «Пополнения»

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

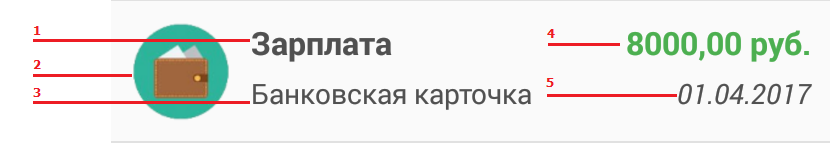


Рисунок 6.2.4 – Элемент списка журнала

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

## Окно «Платеж»

Подробно рассмотрим каждый элемент окна «Платеж»

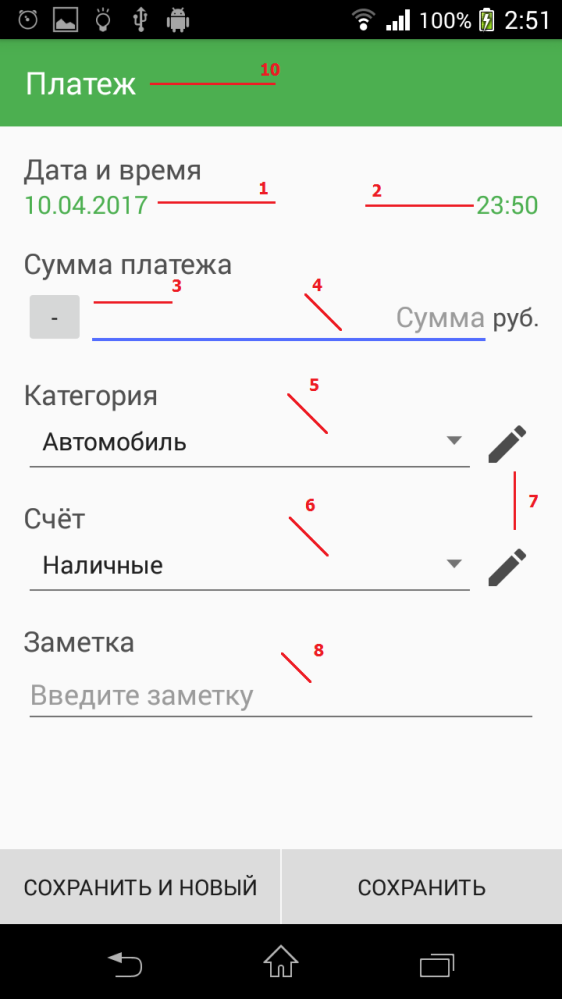


Рисунок 6.2.5 – Окно приложения «Платеж»

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

## Окна «Счета» и «Счет»

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

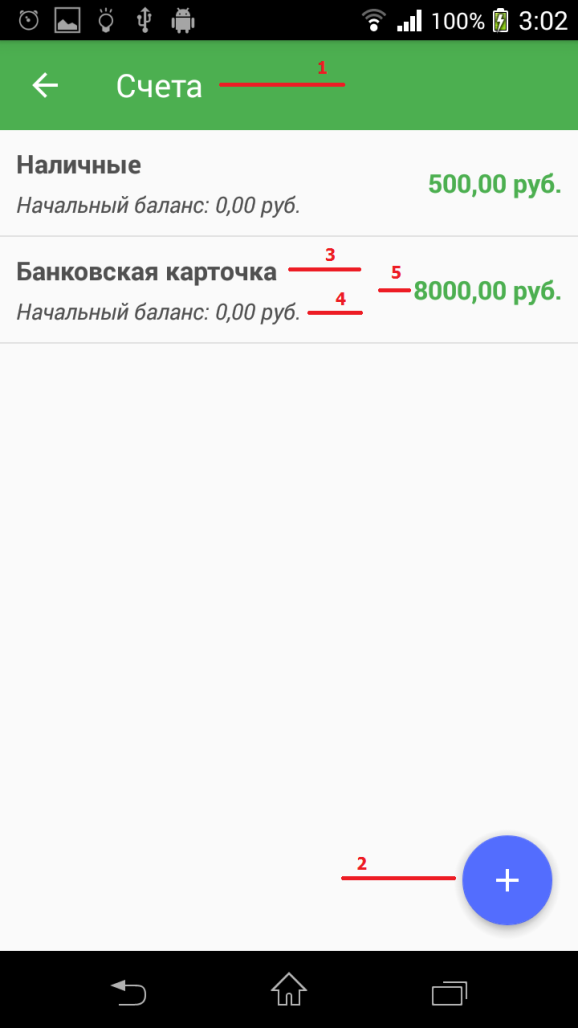


Рисунок 6.2.6 – Окно приложения «Счета»

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:

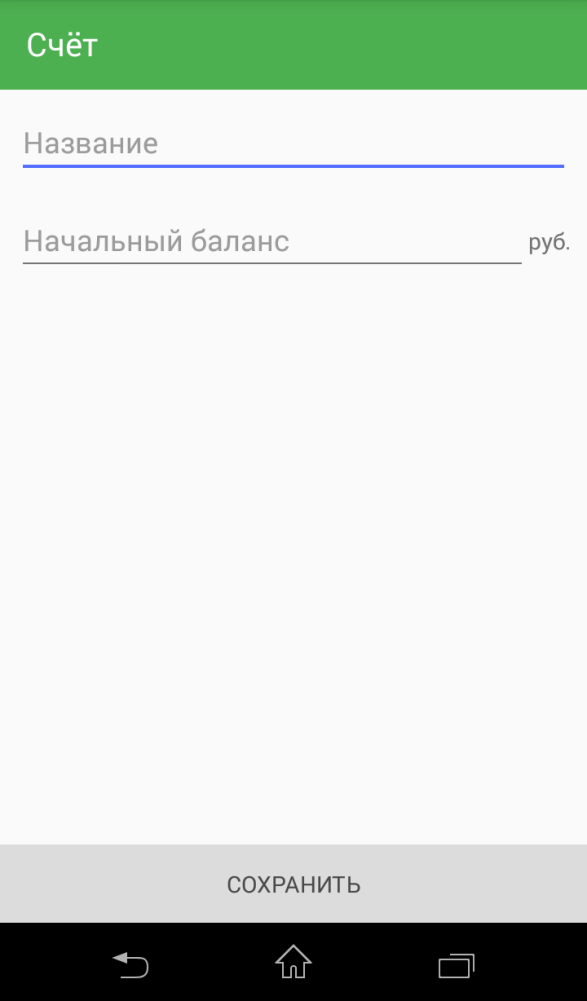


Рисунок 6.2.7 – Окно приложения «Счет»

## 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

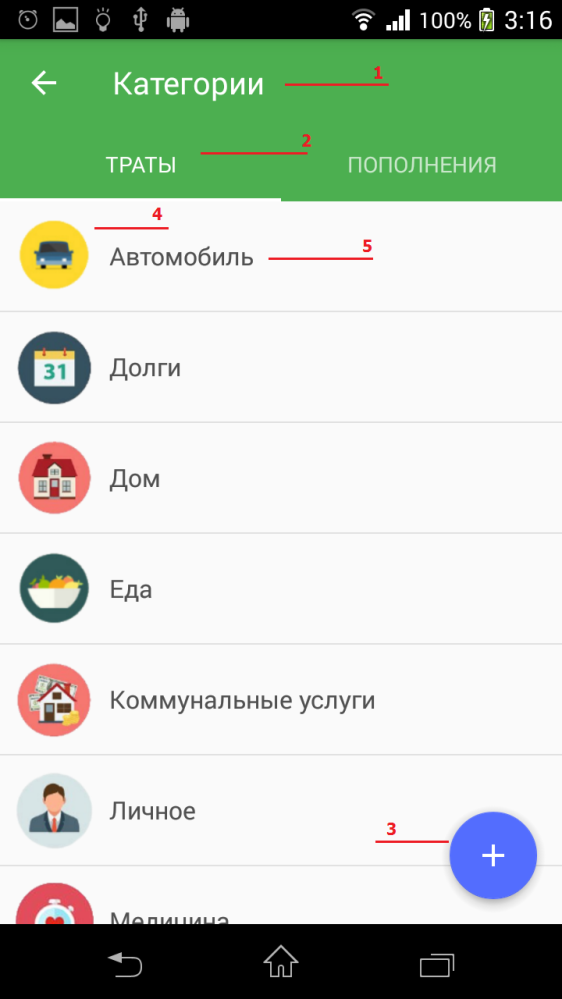


Рисунок 6.2.8 – Окно приложения «Категории»

## 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.

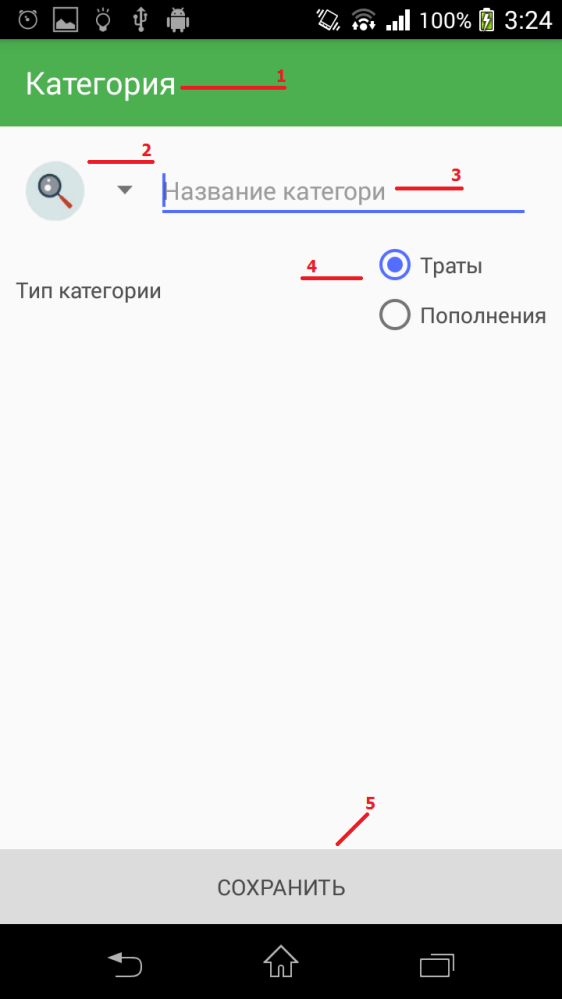
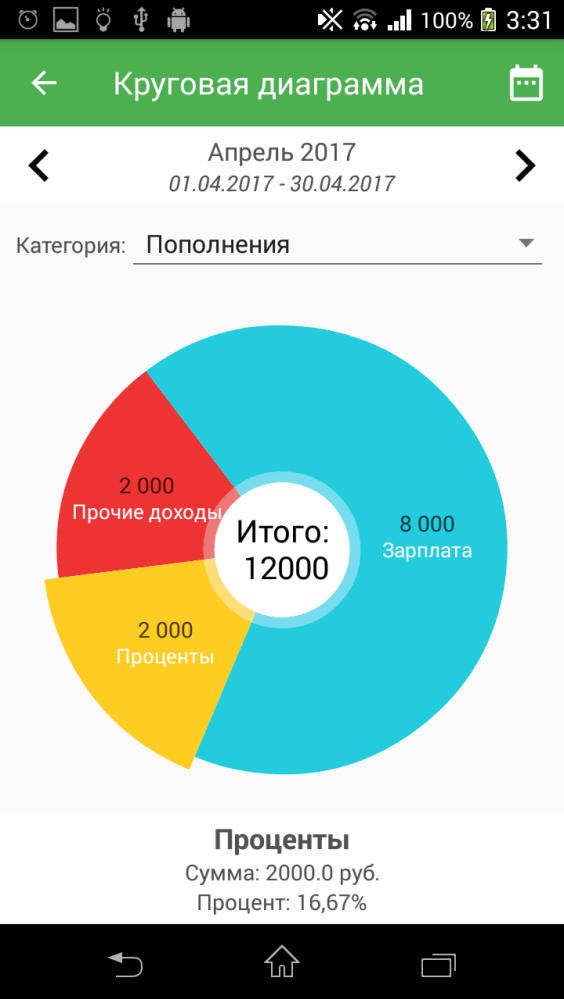
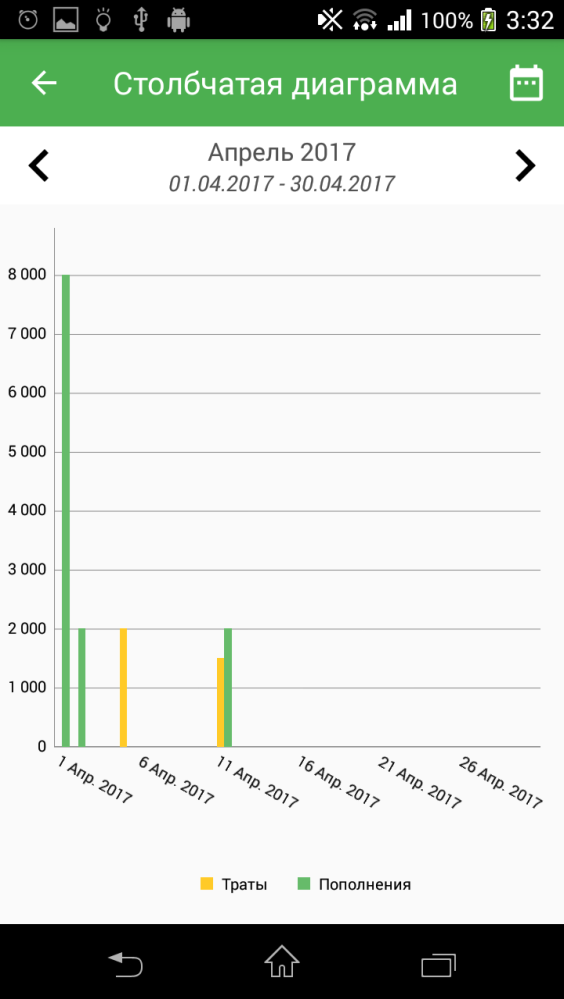


Рисунок 6.2.9 – Окно приложения «Категория»

## 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 6.2.10, 6.2.11 – Окна приложения «Круговая диаграмма» и «Столбчатая диаграмма» соответственно

# 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 производилась 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>