МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра информационных технологий в атомной энергетике

ОТЧЕТ ПО ПРОЕКТУ

по дисциплине «Разработка приложений на языке Котлин»

Студент группы ИКБО-50-23 Павлов Н.С.

_____

(подпись студента)

Руководитель проектной работы Золотухин С.А.

_____

(подпись руководителя)

Работа представлена «____»_____ 2025 г.

Допущен к работе «___» _____ 2025 г.

Москва 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современная жизнь требует от человека, особенно от студента, грамотного управления личными финансами. Проект "Finlytics: Офлайн-менеджер финансов" на языке Kotlin направлен на разработку десктопного приложения, которое предоставит пользователю удобный инструмент для учета доходов и расходов, категоризации трат и визуализации финансовой статистики без необходимости подключения к интернету. Основной целью является создание простого, интуитивно понятного и производительного приложения, которое помогает пользователям контролировать свой бюджет.

Краткое описание проекта:

Проект представляет собой десктопное приложение "Finlytics" для управления персональными финансами. Ключевые функции: внесение операций "доход" и "расход" с возможностью их категоризации, фильтрация данных по различным временным промежуткам, а также наглядное представление финансовой статистики в виде диаграмм и списков.

Проблематика:

Актуальность проекта обусловлена необходимостью простого и доступного инструмента для планирования и контроля личного бюджета. Многие существующие решения либо требуют постоянного подключения к интернету, либо обладают избыточной сложностью, либо неудобны в повседневном использовании. Студенты, как социальная группа с часто ограниченным бюджетом, особенно нуждаются в таком инструменте для формирования навыков финансовой грамотности.

Для достижения цели необходимо выполнить следующие задачи:

- Задача 1: Разработать архитектуру данных и пользовательский интерфейс для удобного внесения и редактирования финансовых операций.

- Задача 2: реализовать механизм категоризации доходов и расходов с возможностью добавления пользовательских категорий.
- Задача 3: внедрить систему фильтрации и анализа данных по временным промежуткам (день, неделя, месяц, год, произвольный интервал) с визуализацией результатов в виде диаграмм и отчетов.

Состав команды и распределение ролей:

- Враженко Д.О. – Team Lead, менеджер проекта, разработчик. Ответственный за постановку задач, общую архитектуру проекта, разработку ключевых модулей и составление отчетности.
- Хохряков А.Ю. – Разработчик. Ответственный за реализацию модуля работы с данными и бизнес-логики.
- Павлов Н.С. – Разработчик, UI/UX дизайнер. Ответственный за проектирование и реализацию пользовательского интерфейса, включая визуализацию данных (диаграммы).

1. ПЛАНИРОВАНИЕ И ПОДГОТОВКА

1.1 Техническое задание

1.1.1 Общие сведения

Наименование проекта: "Finlytics".

Разработчики: Враженко Д.О., Павлов Н.С., Хохряков А.Ю.

Язык разработки: Kotlin.

Целевая платформа: Десктоп.

Тип приложения: Офлайн-приложение.

1.1.2 Цели и назначение автоматизированной системы

Цель: Создать удобный и наглядный инструмент "Finlytics" для ведения учета личных финансов, не требующий подключения к интернету.

Назначение: Автоматизация процесса учета доходов и расходов, предоставление пользователю аналитики о своей финансовой деятельности для принятия обоснованных решений. Система предназначена для широкого круга пользователей, но особенно актуальна для студентов.

1.1.3 Характеристика объектов автоматизации

Объектом автоматизации является процесс персонального финансового учета, который включает в себя:

- Регистрацию финансовых операций (доходы и расходы).
- Категоризацию операций.
- Фильтрацию и поиск операций по дате и категориям.
- Анализ и визуализацию финансовых данных.

1.1.4 Требования к системе

Функциональные требования:

- Внесение, редактирование и удаление операций "доход" и "расход".
- Присвоение операциям категорий из предустановленного списка.
- Создание пользовательских категорий.
- Фильтрация операций по временным промежуткам: день, неделя, месяц, год, произвольный интервал.
- Отображение сводки на главном экране: диаграмма распределения расходов по категориям, общие суммы доходов, расходов и остаток.
- Просмотр статистики в виде списка.

Нефункциональные требования:

- Производительность: Быстрый отклик интерфейса.
- Надежность: Сохранность данных при некорректном завершении работы.
- Удобство использования: Интуитивно понятный интерфейс.
- Офлайн-работа: Все функции доступны без подключения к интернету.

1.1.5 Состав и содержание работ

Проектирование пользовательского интерфейса (UI/UX).

Разработка объектной модели данных.

Реализация модуля для хранения данных.

Реализация бизнес-логики (добавление, удаление, фильтрация операций, расчет статистики).

Создание модуля визуализации данных (диаграммы).

Интеграция всех модулей, тестирование и отладка.

1.1.6 Порядок разработки и приемки

Разработка ведется итерационно. Каждая итерация включает в себя реализацию одного из пунктов состава работ, его тестирование и

интеграцию. Приемка этапов осуществляется лидером проекта (Враженко Д.О.).

1.1.7 Требования к подготовке объекта к вводу системы в действие

Приложение будет распространяться в виде исполняемого JAR-файла, для работы которого требуется JRE версии 8 или выше. В качестве альтернативы рассматривается создание нативного исполняемого файла с помощью Kotlin/Native для упрощения развертывания.

1.1.8 Требования к документированию

Исходный код должен быть документирован в соответствии с общепринятыми стандартами для Kotlin.

Должна быть предоставлена краткая инструкция пользователя внутри приложения (например, в виде ToolTip'ов).

1.1.9 Источники разработки

Официальная документация по языку Kotlin.

Анализ существующих аналоговых приложений.

1.2 Диаграмма вариантов использования

1.2.1 Обоснование сценариев использования

Пользователь добавляет новую операцию (доход/расход).

Пользователь редактирует существующую операцию.

Пользователь удаляет операцию.

Пользователь просматривает историю операций.

Пользователь фильтрует операции по дате и категории.

Пользователь просматривает финансовую сводку.

Пользователь управляет категориями.

1.2.2 Схематическое представление (Use Case Diagram)

Диаграмма включает одного основного актора – Пользователь. Сценарии использования: "Добавить операцию", "Редактировать операцию", "Удалить операцию", "Просмотреть историю операций", "Отфильтровать операции", "Просмотреть финансовую сводку", "Добавить категорию".

Результат создания диаграммы прецедентов представлен на Рисунке 1.

Рисунок 1 — Диаграмма прецедентов разрабатываемого проекта

1.3 Создание репозитория

Для обеспечения эффективной совместной работы и контроля версий в процессе разработки проекта "Finlytics" был создан публичный репозиторий на платформе GitHub. Всем членам команды (Враженко Д.О., Павлов Н.С., Хохряков А.Ю.) предоставлены права на чтение и запись в репозиторий для обеспечения беспрепятственной совместной работы.

Результат создания репозитория разрабатываемого проекта представлен на Рисунке 2.

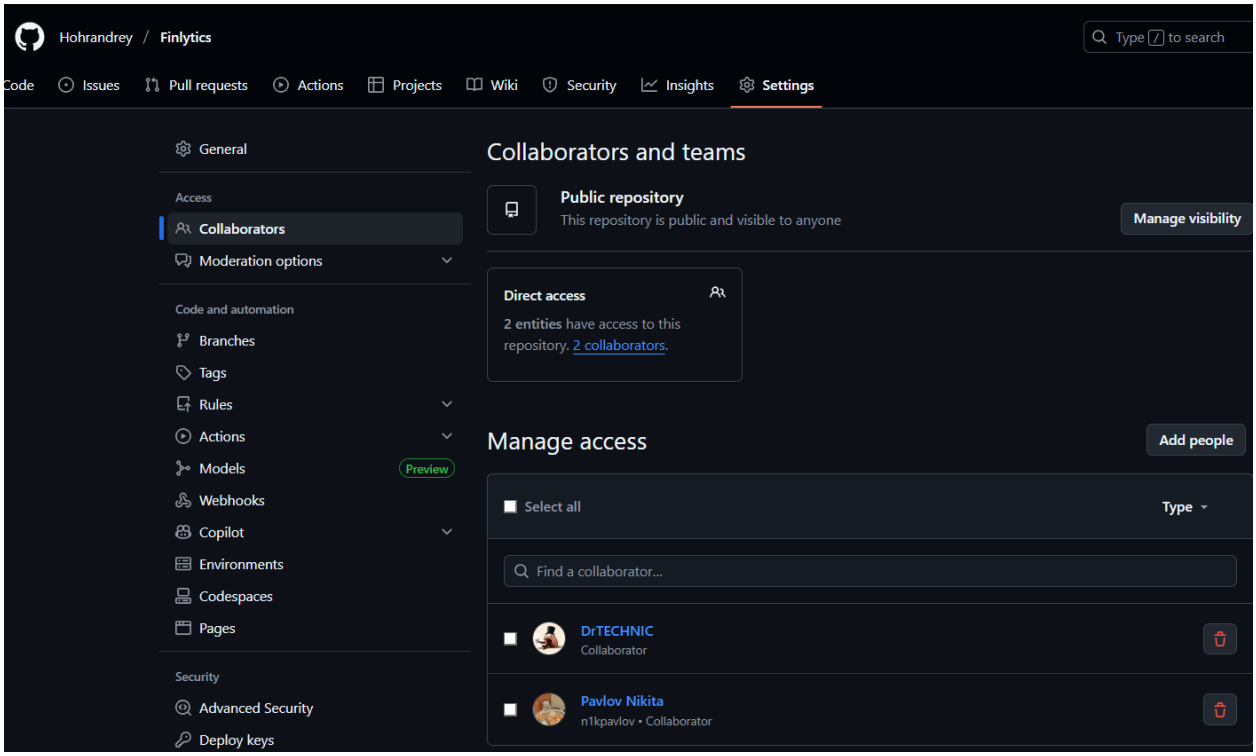

Рисунок 2 — Результат создания репозитория проекта

Ссылка на репозиторий: https://github.com/Hohrandrey/Finlytics

2. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА

2.1 Наполнение экранов

Проектирование пользовательского интерфейса приложения "Finlytics" было выполнено в Figma с фокусом на создание интуитивно понятного, эстетичного и функционального пространства для управления личными финансами.

Были разработаны и визуализированы следующие ключевые экраны:

- Главный экран ("Обзор"): Представляет собой сводку финансового состояния. На нем отображается круговая диаграмма распределения и статистика расходов/доходов по категориям, ключевые показатели (общие доходы, расходы и баланс) за выбранный период, а также инструменты для фильтрации данных по времени.
- Экран "История": Содержит полный список всех финансовых операций в виде списка. Предусмотрена возможность фильтрации по временному промежутку и типу операции.
- Экран "Настройки": Позволяет пользователю управлять системой категорий — просматривать, добавлять, редактировать и удалять категории как для доходов, так и для расходов.

Для интерактивного управления данными были спроектированы всплывающие окна (pop-up):

- Добавление/редактирование финансовой операции.
- Добавление/редактирование категории.
- Подтверждение удаления операции или категории.

2.2 Обоснование дизайн-решений

1. **Выбор темной темы (Основные цвета: `#1E1E1E`, `#2C2C2C`, `#444444`, `#F4F4F4`):**

Темная тема соответствует современным тенденциям в дизайне программного обеспечения и снижает нагрузку на глаза, особенно при длительной работе с приложением, что актуально для десктопного ПО.

Глубокий фон (`#1E1E1E`) позволяет контенту и данным (таким как диаграммы и цифры) выступать на первый план. Цвета `#2C2C2C` и `#444444` используются для разделения областей (карточек, панелей) и создания глубины интерфейса, не перегружая его.

Высококонтрастный текст (`#F4F4F4`) обеспечивает отличную читаемость на темном фоне.

2. Акцентный цвет (`#2176FF`):

Этот насыщенный синий цвет был выбран за его позитивные ассоциации с надежностью, спокойствием и профессионализмом, что важно для финансового приложения. Он используется для интерактивных элементов: кнопок ("Добавить операцию", "Сохранить"), активных полей ввода и выделения выбранного периода, что обеспечивает интуитивно понятную навигацию.

3. Цвета статусов (`#4ADE80` для доходов, `#EF4444` для расходов):

Зеленый (`#4ADE80`): Универсально ассоциируется с позитивом, успехом и ростом. Он интуитивно понятен для отображения доходов и положительного баланса.

Красный (`#EF4444`): Традиционный цвет для предупреждений и отрицательных значений. Он мгновенно сообщает пользователю о расходах и отрицательной динамике.

4. Вспомогательные цвета (для диаграмм и статистики):

Для круговой диаграммы и других элементов визуализации используется палитра контрастных, но гармонирующих с общей темой цветов. Это обеспечивает четкое различие между категориями расходов, делая диаграмму не только информативной, но и визуально привлекательной.

2.3 Макеты экранов и прототип

Результат создания макетов экранов и pop-up представлен на Рисунках 3 – 12. Для демонстрации интерактивности и пользовательского потока был создан кликабельный прототип в Figma.

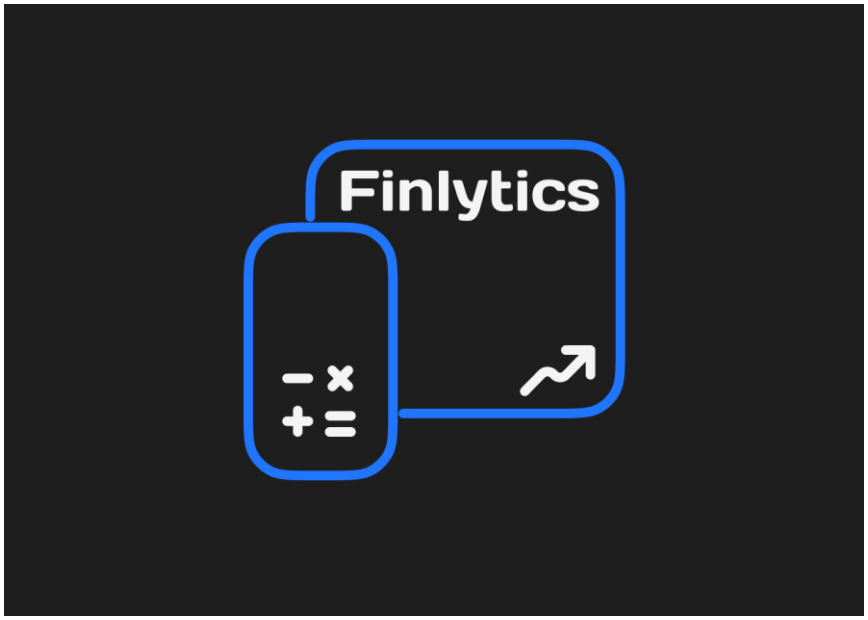

Рисунок 3 — Экран загрузки

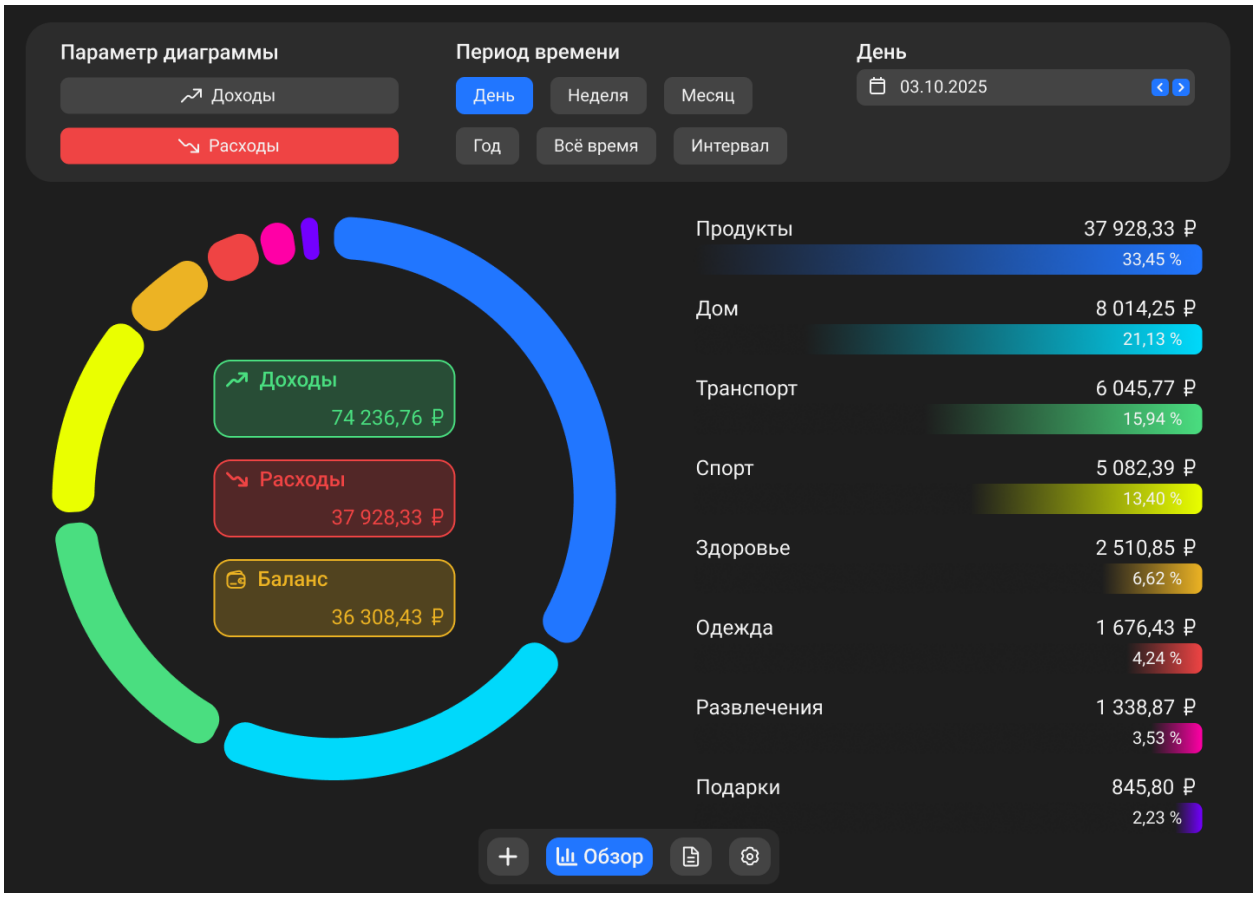

Рисунок 4 — Экран домашней страницы

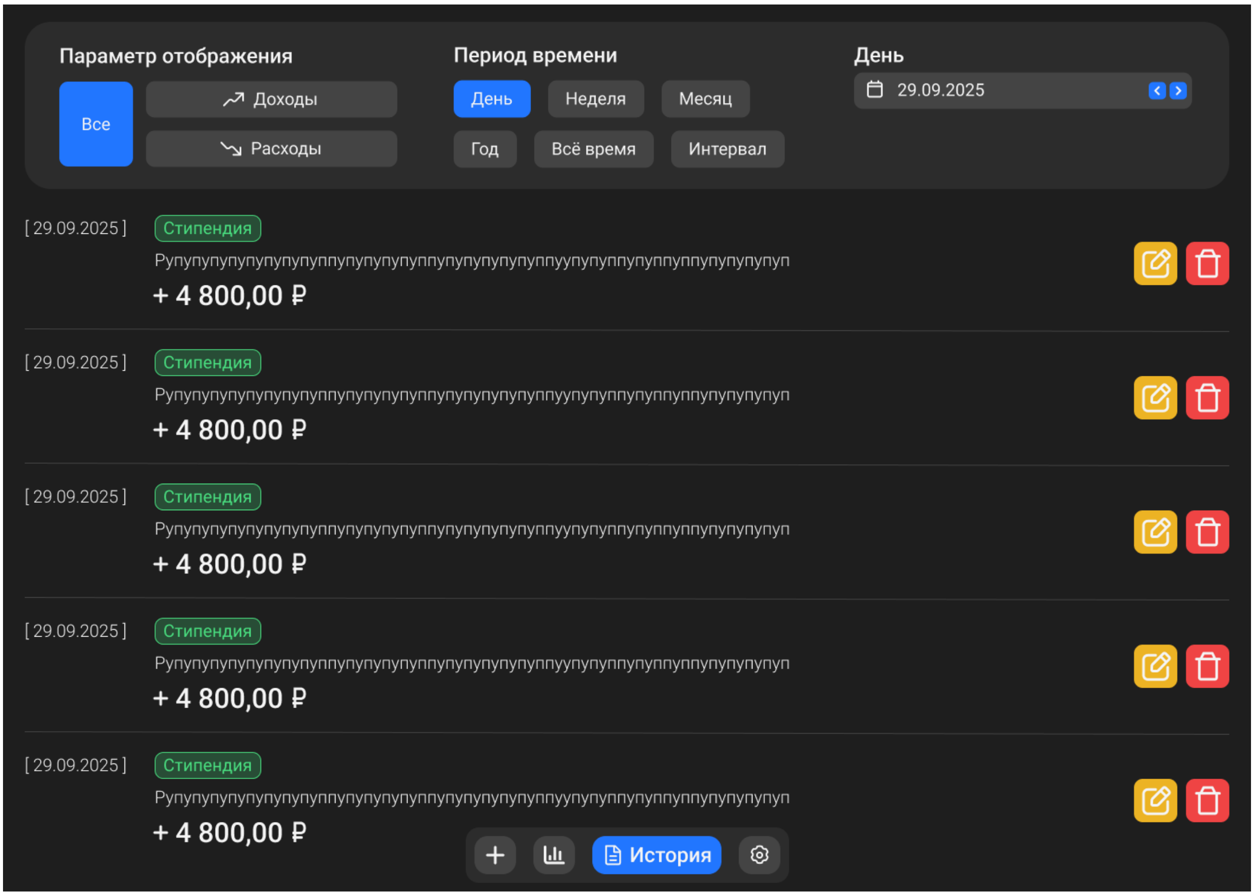

Рисунок 5 — Экран истории

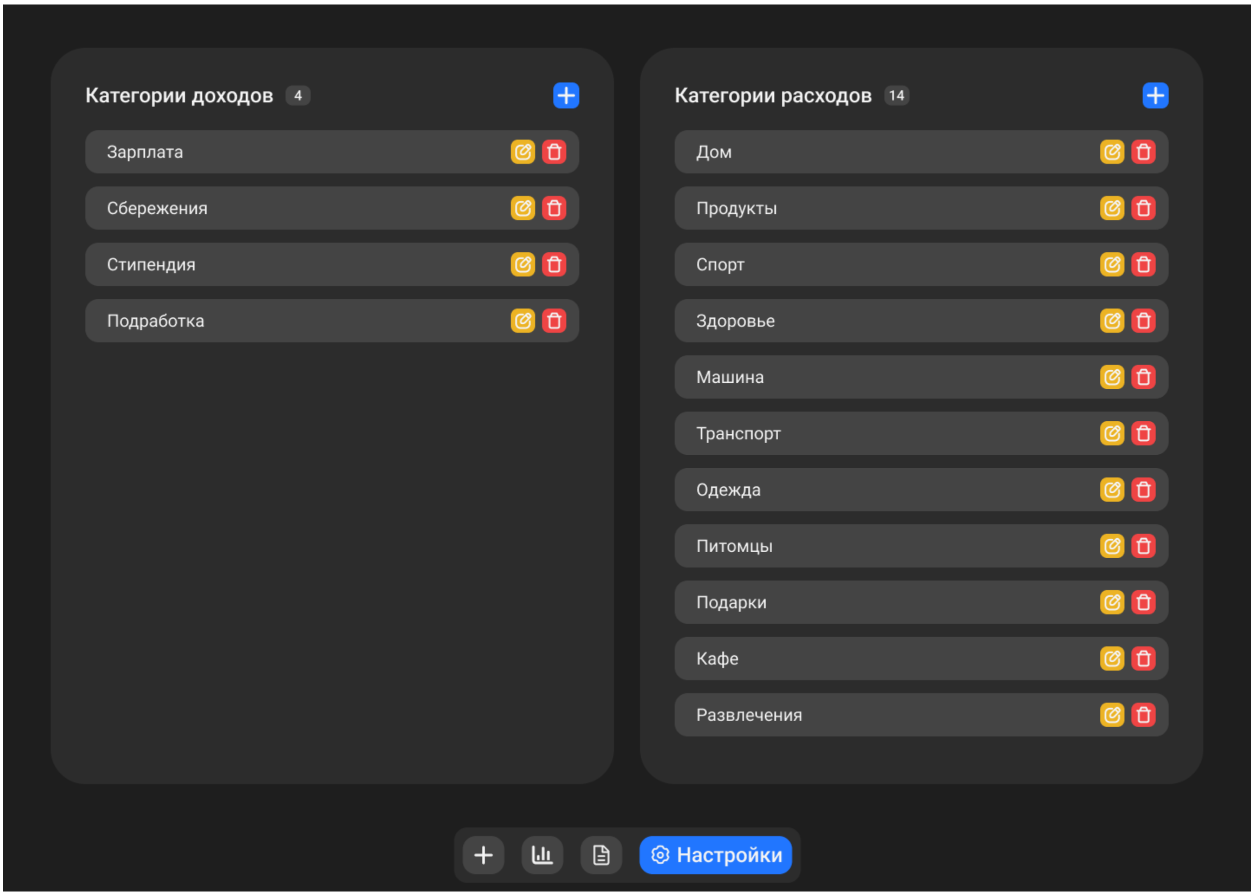

Рисунок 6 — Экран настроек

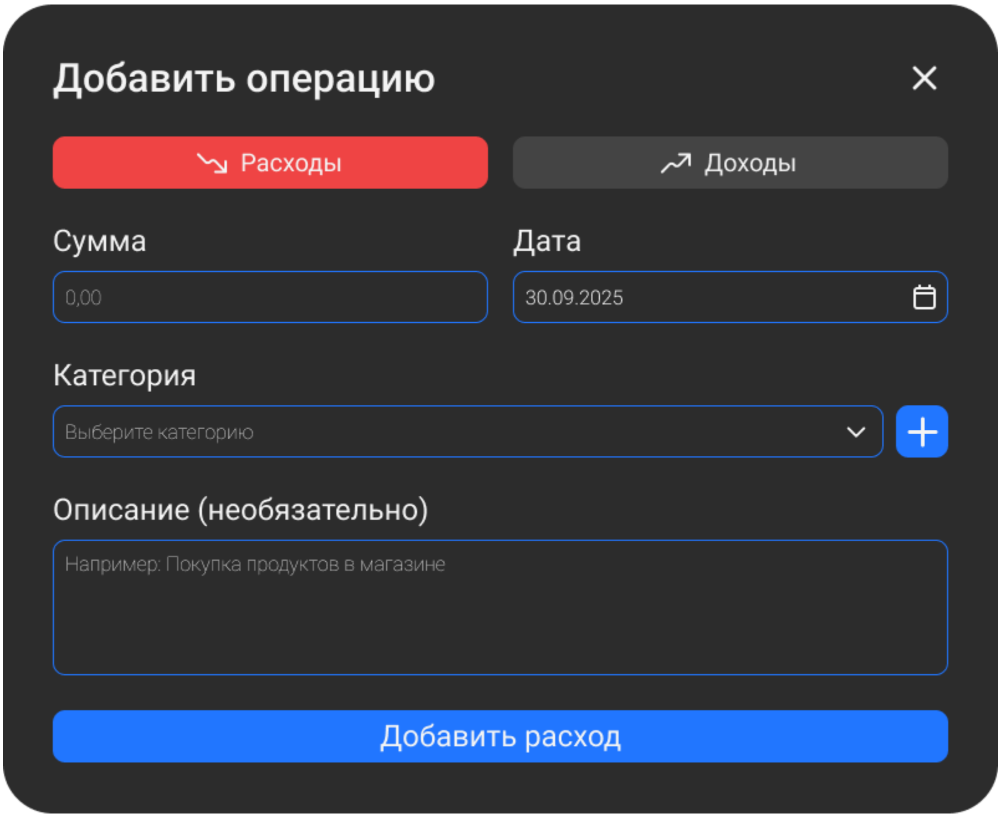

Рисунок 7 — Pop-up добавления операции

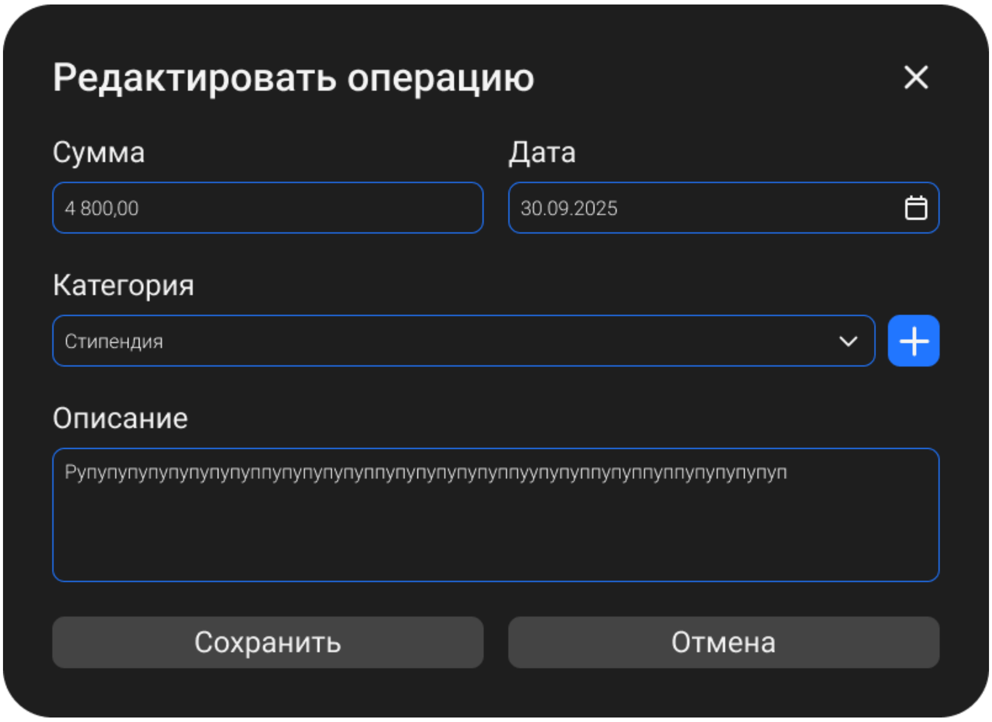

Рисунок 8 — Pop-up редактирования операции

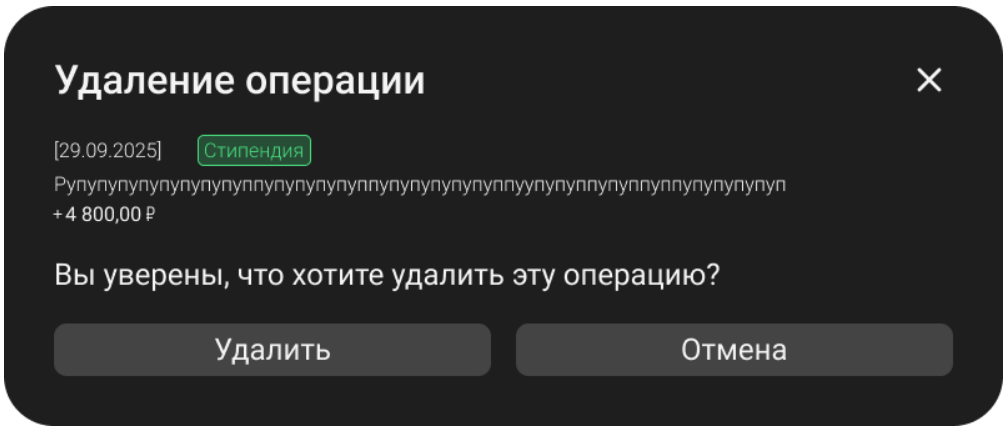

Рисунок 9 — Pop-up удаления операции

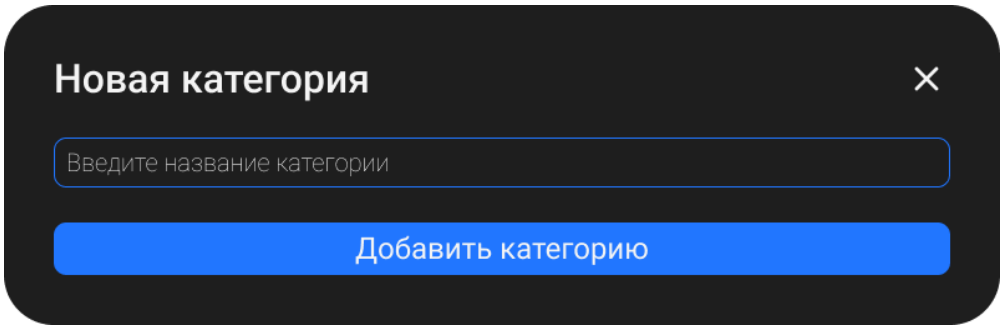

Рисунок 10 — Pop-up добавления категории

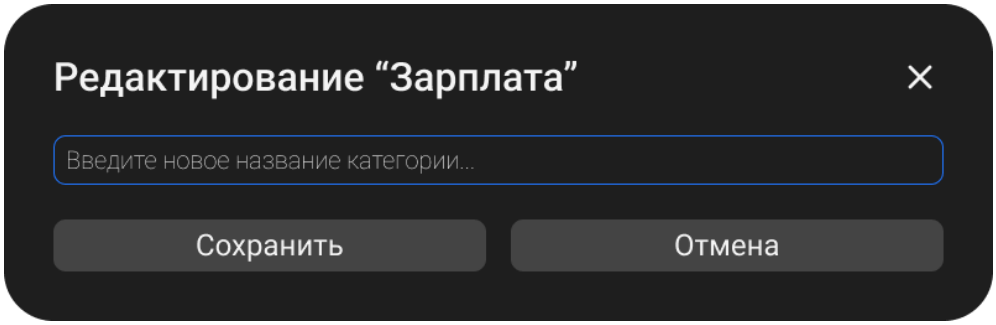

Рисунок 11 — Pop-up редактирования категории

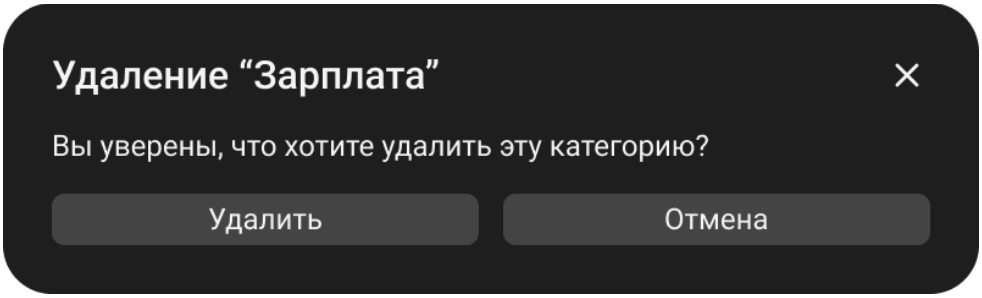

Рисунок 12 — Pop-up удаления категории

2.4 Gradle и README файлы

```
plugins {
    kotlin("jvm") version "1.9.24"
    id("org.jetbrains.compose") version "1.6.11"
}

group = "com.finlytics"
version = "1.0"

repositories {
    google()
    mavenCentral()
    maven( url = "https://maven.pkg.jetbrains.space/public/p/compose/dev")
}

kotlin {
    jvmToolchain( jdkVersion = 21)
}

tasks.withType<org.jetbrains.kotlin.gradle.tasks.KotlinCompile>().configureEach {
    kotlinOptions.jvmTarget = "21"
    kotlinOptions.freeCompilerArgs += listOf("-Xopt-in=kotlin.RequiresOptIn")
}

tasks.withType<JavaCompile> {
    options.encoding = "UTF-8"
    options.compilerArgs.addAll( elements = listOf("-Xlint:unchecked", "-Xlint:deprecation"))
}

tasks.withType<Test> {
    systemProperty( name = "file.encoding", value = "UTF-8")
}

tasks.withType<JavaExec> {
    systemProperty( name = "file.encoding", value = "UTF-8")
}
```

Рисунок 13 — файл build.gradle (часть 1)

```
dependencies {
    implementation(compose.desktop.currentOs)
    implementation("org.xerial:sqlite-jdbc:3.46.1.0")
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-swing:1.8.1")
}

compose.desktop {
    application {
        mainClass = "main.MainKt"

        nativeDistributions {
            targetFormats(
                ...formats = org.jetbrains.compose.desktop.application.dsl.TargetFormat.Dmg,
                org.jetbrains.compose.desktop.application.dsl.TargetFormat.Msi,
                org.jetbrains.compose.desktop.application.dsl.TargetFormat.Deb
            )
            packageName = "Finlytics"
            packageVersion = "1.0.0"

            windows {
                menu = true
                console = true
            }
        }
    }
}
```

Рисунок 14 — файл build.gradle (часть 2)

README

Finlytics

📋 Общие сведения

Параметр	Значение
Наименование проекта	Finlytics
Разработчики	Враженко Д.О., Павлов Н.С., Хохряков А.Ю.
Язык разработки	Kotlin
Целевая платформа	Десктоп
Тип приложения	Офлайн-приложение

🎯 Цели и назначение

> Создать удобный и наглядный инструмент для ведения учета личных финансов, не требующий подключения к интернету.

Назначение: Автоматизация процесса учета доходов и расходов, предоставление пользователю аналитики о своей финансовой деятельности для принятия обоснованных решений.

⚙️ Характеристика объектов автоматизации

Рисунок 15 — README файл

3. РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Архитектура приложения

MVVM (Model-View-ViewModel) выбран, потому что:

- Идеально подходит для приложений с графическим интерфейсом
- Четкое разделение логики и интерфейса
- Упрощает тестирование компонентов
- Автоматическое обновление UI при изменении данных
- Эффективная работа с привязками данных
- Легкая поддержка и расширение функционала
- Подходит для локальных приложений с БД

Схематичное изображение архитектуры разрабатываемого проекта представлено на Рисунке 15.

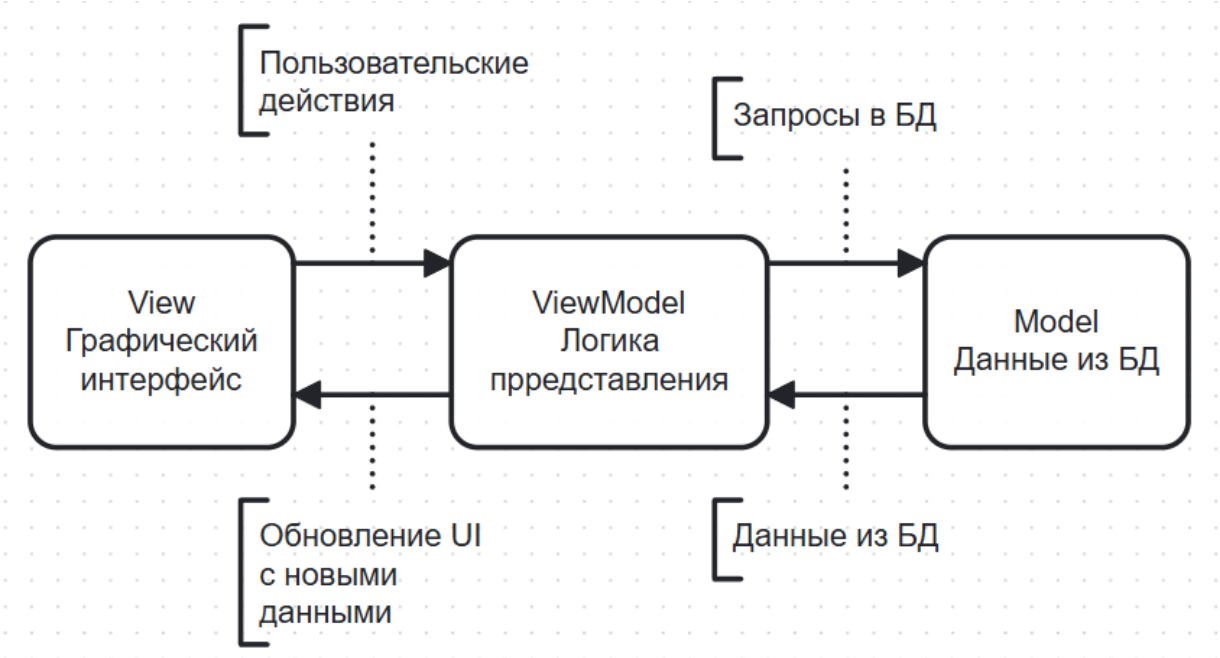

Рисунок 16 — Результат создания схемы архитектуры проекта

3.2 Реализация ключевых компонентов

3.2.1 Frontend

Для реализации клиентской части проекта "Finlytics" использовался фреймворк Compose for Desktop, который позволяет создавать современные кроссплатформенные десктопные приложения на Kotlin с декларативным подходом к построению пользовательского интерфейса.

Основные реализованные компоненты и их функции:

1. Навигация и структура приложения:

- Main.kt – точка входа в приложение, инициализирующая окно и настраивающая окружение Compose Desktop.
- App.kt – главный компонент приложения, устанавливающий тему и стили Material Design;
- AppNavigation.kt – компонент-маршрутизатор, управляющий отображением текущего экрана на основе состояния ViewModel;
- NavigationBar.kt – навигационная панель с иконками для переключения между основными экранами приложения.

2. Основные экраны приложения:

- OverviewScreen.kt – отображает финансовую сводку, круговую диаграмму распределения расходов по категориям, ключевые показатели (баланс, доходы, расходы) и предоставляет фильтры по временным периодам;
- HistoryScreen.kt – показывает полный список финансовых операций с расширенными возможностями фильтрации по типу операций и временным периодам, поддерживает редактирование и удаление операций;
- SettingsScreen.kt – предоставляет интерфейс для управления категориями доходов и расходов: просмотр, добавление и удаление пользовательских категорий.

3. Компоненты пользовательского интерфейса:

- PieChart.kt – кастомный компонент для визуализации распределения расходов по категориям с использованием Canvas API Compose;
- OperationDialog.kt – диалоговое окно для добавления и редактирования финансовых операций с валидацией вводимых данных;
- AddCategoryDialog.kt – диалоговое окно для создания новых категорий доходов и расходов;

4. Модель состояния и управление данными:

- FinanceState.kt – data-класс, представляющий иммутабельное состояние приложения для реактивного обновления UI;
- FinanceViewModel.kt – ViewModel, реализующий бизнес-логику приложения и управляющий состоянием через Flow.

5. Ресурсы и визуальные элементы:

- AppColors.kt – объект, содержащий цветовую палитру приложения для поддержки единого визуального стиля;
- FinlyticsIconPack – полная система векторных иконок приложения, состоящая из:
  - __FinlyticsIconPack.kt – основной объект-контейнер, предоставляющий доступ ко всем иконкам приложения;
  - 15 файлов векторных иконок (Add.kt, Close.kt, Date.kt, Delete.kt, Edit.kt, Expenses.kt, History.kt, Income.kt, Left.kt, Minus.kt, Plus.kt, Right.kt, Settings.kt, Statistic.kt, Wallet.kt) – набор кастомных векторных изображений для всех элементов интерфейса.

Листинги компонентов интерфейса:

Структура и навигация:

Листинг 1 – Main.kt – Точка входа в приложение

```
package main
```

```
import androidx.compose.desktop.ui.tooling.preview.Preview
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.MaterialTheme
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.rememberWindowState
import androidx.compose.ui.window.application
import repository.FinanceRepository
import ui.App
import ui.components.AddCategoryDialog
import ui.components.OperationDialog
import utils.LoggingConfig
import viewmodel.FinanceViewModel

/**
 * Предварительный просмотр приложения для Compose Preview.
 * Используется при разработке для предварительного просмотра UI.
 */
@Composable
@Preview
fun AppPreview() {
    val repository = remember { FinanceRepository() }
    val viewModel = remember { FinanceViewModel(repository) }

    MaterialTheme {
        Column(modifier = Modifier.fillMaxSize()) {
            App(viewModel)

            if (viewModel.showOperationDialog) {
                OperationDialog(viewModel)
            }

            if (viewModel.showAddCategoryDialog) {
                AddCategoryDialog(viewModel)
            }
        }
    }
}

/**
 * Точка входа в приложение.
 * Инициализирует окно приложения и настраивает окружение.
 */
fun main() = application {
    // Логирование для корректной работы с UTF-8
    LoggingConfig.setupLogging()

    val windowState = rememberWindowState(width = 1440.dp, height = 1024.dp)

    Window(
        onCloseRequest = ::exitApplication,
        title = "Finlytics - Менеджер личных финансов",
        state = windowState,
        undecorated = false,
        resizable = true
    ) {
        // Минимальный размер окна для удобства использования
        window.minimumSize = java.awt.Dimension(800, 600)
        AppPreview()
    }
}
```

Листинг 2 – App.kt – Главный компонент приложения

```
package ui

import androidx.compose.material.MaterialTheme
import androidx.compose.runtime.Composable
import viewmodel.FinanceViewModel

/**
 * Главный компонент приложения, устанавливающий тему и стили.
 *
 * @param viewModel ViewModel для управления состоянием приложения
 */
@Composable
fun App(viewModel: FinanceViewModel) {
    MaterialTheme {
        AppNavigation(viewModel)
    }
}
```

Листинг 3 – AppNavigation.kt – Компонент навигации

```
package ui

import androidx.compose.runtime.Composable
import ui.screens.HistoryScreen
import ui.screens.OverviewScreen
import ui.screens.SettingsScreen
import viewmodel.FinanceViewModel

/**
 * Компонент навигации между экранами приложения.
 * В зависимости от текущего экрана в ViewModel отображает соответствующий
экран.
 *
 * @param viewModel ViewModel с информацией о текущем экране и данными
 */
@Composable
fun AppNavigation(viewModel: FinanceViewModel) {
    when (viewModel.currentScreen) {
        "Overview" -> OverviewScreen(viewModel)
        "History" -> HistoryScreen(viewModel)
        "Settings" -> SettingsScreen(viewModel)
    }
}
```

Листинг 4 – NavigationBar.kt – Навигационная панель

```
package ui.components

import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Icon
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.interaction.MutableInteractionSource
import ui.theme.AppColors
import ui.theme.icons.FinlyticsIconPack
import ui.theme.icons.finlyticsiconpack.*
import viewmodel.FinanceViewModel

/**
 * Навигационная панель приложения.
 * Отображает кнопки для переключения между основными экранами приложения.
 *
 * @param viewModel ViewModel для управления навигацией и состоянием
 */
@Composable
fun NavigationBar(viewModel: FinanceViewModel) {
    val currentScreen = viewModel.currentScreen

    Row(
        modifier = Modifier
            .wrapContentSize()
            .background(AppColors.DarkGreyColor, RoundedCornerShape(15.dp))
            .padding(10.dp),
        horizontalArrangement = Arrangement.Center,
        verticalAlignment = Alignment.CenterVertically
    ) {
        NavigationButton(
            text = "Добавить",
            isSelected = currentScreen == "AddNew",
            onClick = { viewModel.showAddOperation() },
            Icon = FinlyticsIconPack.Add
        )

        Spacer(Modifier.width(20.dp))

        NavigationButton(
            text = "Обзор",
            isSelected = currentScreen == "Overview",
            onClick = { viewModel.navigateTo("Overview") },
            Icon = FinlyticsIconPack.Statistic
        )

        Spacer(Modifier.width(20.dp))

        NavigationButton(
            text = "История",
            isSelected = currentScreen == "History",
            onClick = { viewModel.navigateTo("History") },
            Icon = FinlyticsIconPack.History
        )

        Spacer(Modifier.width(20.dp))

        NavigationButton(
            text = "Настройки",
            isSelected = currentScreen == "Settings",
            onClick = { viewModel.navigateTo("Settings") },
            Icon = FinlyticsIconPack.Settings
        )
    }
```

```
}

/**
 * Компонент кнопки навигации.
 * Отображает иконку и, при активном состоянии, текст кнопки.
 *
 * @param text Текст кнопки
 * @param isSelected Флаг активности кнопки
 * @param onClick Обработчик нажатия на кнопку
 * @param Icon Векторное изображение иконки
 */
@Composable
fun NavigationButton(
    text: String,
    isSelected: Boolean,
    onClick: () -> Unit,
    Icon: ImageVector
) {
    val backgroundColor = if (isSelected) AppColors.BlueColor else
AppColors.LightGreyColor
    val textColor = AppColors.LightColor
    val iconColor = AppColors.LightColor
    val interactionSource = remember { MutableInteractionSource() }

    Box(
        modifier = Modifier
            .wrapContentWidth()
            .background(backgroundColor, RoundedCornerShape(15.dp))
            .padding(horizontal = 10.dp, vertical = 8.dp)
            .clickable(
                interactionSource = interactionSource,
                indication = null,
                onClick = onClick
            ),
        contentAlignment = Alignment.Center
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically
        ) {
            Icon(
                imageVector = Icon,
                contentDescription = text,
                modifier = Modifier.size(28.dp),
                tint = iconColor
            )

            if (isSelected) {
                Spacer(Modifier.width(5.dp))
                Text(
                    text = text,
                    style = TextStyle(
                        fontSize = 24.sp,
                        fontWeight = FontWeight.Medium,
                        color = textColor,
                        letterSpacing = 0.sp
                    ),
                    maxLines = 1
                )
            }
        }
    }
}
```

Основные экраны:

Листинг 5 – OverviewScreen.kt – Экран финансовой сводки

```
package ui.screens

import androidx.compose.foundation.*
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import java.time.LocalDate
import java.time.format.DateTimeFormatter
import java.time.temporal.TemporalAdjusters
import kotlin.math.min
import ui.components.NavigationBar
import ui.components.PieChart
import ui.theme.AppColors
import ui.theme.icons.FinlyticsIconPack
import ui.theme.icons.finlyticsiconpack.*
import viewmodel.FinanceViewModel

/**
 * Экран обзора финансовой статистики.
 * Отображает круговую диаграмму распределения расходов/доходов,
 * финансовую сводку (баланс, доходы, расходы) и список категорий.
 *
 * @param viewModel ViewModel для управления финансовыми данными
 */
@Composable
fun OverviewScreen(viewModel: FinanceViewModel) {
    val state by viewModel.state.collectAsState()

    // Состояния для фильтров
    var selectedFilter by remember { mutableStateOf("Расходы") }
    var selectedPeriod by remember { mutableStateOf("Всё время") }
    var selectedDate by remember { mutableStateOf(LocalDate.now()) }

    // Форматирование даты для отображения
    val dateFormatter = DateTimeFormatter.ofPattern("dd.MM.yyyy")
    val monthYearFormatter = DateTimeFormatter.ofPattern("MM.yyyy")
    val yearFormatter = DateTimeFormatter.ofPattern("yyyy")

    // Функция для фильтрации операций по периоду (все операции за период)
    val allOperationsForPeriod = remember(state.operations, selectedPeriod,
selectedDate) {
        filterOperationsByPeriod(
            operations = state.operations,
            period = selectedPeriod,
            selectedDate = selectedDate
        )
```

```
    }

    // Функция для фильтрации операций по типу и периоду (для диаграммы)
    val filteredOperationsByType = remember(state.operations,
selectedFilter, selectedPeriod, selectedDate) {
        filterOperationsByType(
            operations = state.operations,
            filterType = selectedFilter,
            period = selectedPeriod,
            selectedDate = selectedDate
        )
    }

    // Вычисляем статистику на основе ВСЕХ операций за период (вне
зависимости от фильтра)
    val totalIncome = allOperationsForPeriod.filter { it.type == "Доход"
}.sumOf { it.amount }
    val totalExpenses = allOperationsForPeriod.filter { it.type == "Расход"
}.sumOf { it.amount }
    val balance = totalIncome - totalExpenses

    // Группируем операции по категориям в зависимости от выбранного фильтра
(только для диаграммы)
    val operationsByCategory = remember(selectedFilter,
filteredOperationsByType) {
        when (selectedFilter) {
            "Доходы" -> {
                filteredOperationsByType
                    .filter { it.type == "Доход" }
                    .groupBy { it.category }
                    .mapValues { entry -> entry.value.sumOf { op ->
op.amount } }
            }
            "Расходы" -> {
                filteredOperationsByType
                    .filter { it.type == "Расход" }
                    .groupBy { it.category }
                    .mapValues { entry -> entry.value.sumOf { op ->
op.amount } }
            }
            else -> emptyMap()
        }
    }

    // Подготавливаем список категорий для отображения с процентами
    val categoryList = operationsByCategory.entries.sortedByDescending {
it.value }

    // Функция для получения отображаемой даты в зависимости от периода
    val displayDateText = remember(selectedPeriod, selectedDate) {
        when (selectedPeriod) {
            "День" -> selectedDate.format(dateFormatter)
            "Неделя" -> {
                val weekStart =
selectedDate.with(TemporalAdjusters.previousOrSame(java.time.DayOfWeek.MONDA
Y))
                val weekEnd = weekStart.plusDays(6)
                "${weekStart.format(dateFormatter)} -
${weekEnd.format(dateFormatter)}"
            }
            "Месяц" -> selectedDate.format(monthYearFormatter)
            "Год" -> selectedDate.format(yearFormatter)
            else -> ""
        }
```

```
    }

    // Отладочная информация о данных
    LaunchedEffect(allOperationsForPeriod, filteredOperationsByType,
operationsByCategory) {
        println("\n=== OVERVIEW SCREEN ===")
        println("Фильтр: $selectedFilter, Период: $selectedPeriod")
        println("Всего операций: ${state.operations.size}")
        println("Операций за период (все): ${allOperationsForPeriod.size}")
        println("Операций за период (${selectedFilter}):
${filteredOperationsByType.size}")
        println("Текущий баланс: ${balance} руб.")
        println("Общие доходы: ${totalIncome} руб.")
        println("Общие расходы: ${totalExpenses} руб.")
        println("Категорий для диаграммы (${selectedFilter}):
${operationsByCategory.size}")
        println("Данные для диаграммы: $operationsByCategory")
        println("========================\n")
    }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(AppColors.DarkColor)
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(horizontal = 25.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            // Секция настроек
            Box(
                modifier = Modifier
                    .padding(top = 20.dp)
                    .height(193.dp)
                    .background(AppColors.DarkGreyColor,
RoundedCornerShape(30.dp))
            ) {
                Row(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(start = 40.dp, end = 40.dp, top = 15.dp),
                    horizontalArrangement = Arrangement.SpaceBetween,
                    verticalAlignment = Alignment.Top
                ) {
                    // Фильтр по типу
                    Column(
                        modifier = Modifier.width(390.dp),
                        verticalArrangement = Arrangement.spacedBy(15.dp)
                    ) {
                        Text(
                            "Параметр отображения",
                            fontSize = 24.sp,
                            fontWeight = FontWeight.Medium,
                            color = AppColors.LightColor
                        )

                        // Кнопка "Доходы"
                        FilterButton(
                            text = "Доходы",
                            isSelected = selectedFilter == "Доходы",
                            icon = true,
                            modifier = Modifier.width(390.dp).height(42.dp),
```

```
                            onClick = { selectedFilter = "Доходы" }
                        )

                        // Кнопка "Расходы"
                        FilterButton(
                            text = "Расходы",
                            isSelected = selectedFilter == "Расходы",
                            icon = true,
                            isIncome = false,
                            modifier = Modifier.width(390.dp).height(42.dp),
                            onClick = { selectedFilter = "Расходы" }
                        )
                    }

                    // Период времени
                    Column(
                        verticalArrangement = Arrangement.spacedBy(15.dp)
                    ) {
                        Text(
                            "Период времени",
                            fontSize = 24.sp,
                            fontWeight = FontWeight.Medium,
                            color = AppColors.LightColor
                        )

                        // Кнопки периодов
                        Row(
                            modifier = Modifier.width(350.dp),
                            horizontalArrangement =
Arrangement.spacedBy(20.dp),
                            verticalAlignment = Alignment.CenterVertically
                        ) {
                            PeriodButton(
                                text = "День",
                                isSelected = selectedPeriod == "День",
                                onClick = {
                                    selectedPeriod = "День"
                                    selectedDate = LocalDate.now()
                                }
                            )
                            PeriodButton(
                                text = "Неделя",
                                isSelected = selectedPeriod == "Неделя",
                                onClick = {
                                    selectedPeriod = "Неделя"
                                    selectedDate = LocalDate.now()
                                }
                            )
                            PeriodButton(
                                text = "Месяц",
                                isSelected = selectedPeriod == "Месяц",
                                onClick = {
                                    selectedPeriod = "Месяц"
                                    selectedDate = LocalDate.now()
                                }
                            )
                        }

                        Row(
                            modifier = Modifier.width(350.dp),
                            horizontalArrangement =
Arrangement.spacedBy(20.dp),
                            verticalAlignment = Alignment.CenterVertically
                        ) {
```

```
                            PeriodButton(
                                text = "Год",
                                isSelected = selectedPeriod == "Год",
                                onClick = {
                                    selectedPeriod = "Год"
                                    selectedDate = LocalDate.now()
                                }
                            )
                            PeriodButton(
                                text = "Всё время",
                                isSelected = selectedPeriod == "Всё время",
                                onClick = {
                                    selectedPeriod = "Всё время"
                                }
                            )
                        }
                    }

                    // Настройки даты
                    if (selectedPeriod != "Всё время") {
                        Column(
                            verticalArrangement =
Arrangement.spacedBy(15.dp)
                        ) {
                            Text(
                                selectedPeriod,
                                fontSize = 24.sp,
                                fontWeight = FontWeight.Medium,
                                color = AppColors.LightColor
                            )

                            // Поле ввода даты
                            Box(
                                modifier = Modifier
                                    .width(390.dp)
                                    .height(42.dp)
                                    .background(AppColors.LightGreyColor,
RoundedCornerShape(10.dp))
                            ) {
                                Row(
                                    modifier = Modifier.fillMaxSize(),
                                    verticalAlignment =
Alignment.CenterVertically
                                ) {
                                    Spacer(Modifier.width(16.dp))

                                    Icon(
                                        imageVector =
FinlyticsIconPack.Date,
                                        contentDescription = "date",
                                        modifier = Modifier.size(18.dp),
                                        tint = AppColors.LightColor
                                    )

                                    Spacer(Modifier.width(18.dp))

                                    Text(
                                        displayDateText,
                                        fontSize = 20.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.LightColor
                                    )

                                    Spacer(Modifier.weight(1f))
```

```
                                    // Кнопки навигации
                                    Row(
                                        modifier = Modifier.padding(end =
6.dp),
                                        horizontalArrangement =
Arrangement.spacedBy(4.dp),
                                        verticalAlignment =
Alignment.CenterVertically
                                    ) {
                                        IconButton(
                                            onClick = {
                                                // Предыдущий период
                                                selectedDate = when
(selectedPeriod) {
                                                    "День" ->
selectedDate.minusDays(1)
                                                    "Неделя" ->
selectedDate.minusWeeks(1)
                                                    "Месяц" ->
selectedDate.minusMonths(1)
                                                    "Год" ->
selectedDate.minusYears(1)
                                                    else -> selectedDate
                                                }
                                            },
                                            modifier = Modifier
.background(AppColors.BlueColor, RoundedCornerShape(5.dp))
                                                .size(20.dp)
                                        ) {
                                            Icon(
                                                imageVector =
FinlyticsIconPack.Left,
                                                contentDescription =
"previous_period",
                                                modifier =
Modifier.size(20.dp),
                                                tint = AppColors.LightColor
                                            )
                                        }

                                        IconButton(
                                            onClick = {
                                                // Следующий период
                                                selectedDate = when
(selectedPeriod) {
                                                    "День" ->
selectedDate.plusDays(1)
                                                    "Неделя" ->
selectedDate.plusWeeks(1)
                                                    "Месяц" ->
selectedDate.plusMonths(1)
                                                    "Год" ->
selectedDate.plusYears(1)
                                                    else -> selectedDate
                                                }
                                            },
                                            modifier = Modifier
.background(AppColors.BlueColor, RoundedCornerShape(5.dp))
                                                .size(20.dp)
                                        ) {
                                            Icon(
```

```
                                                imageVector =
FinlyticsIconPack.Right,
                                                contentDescription =
"next_period",
                                                modifier =
Modifier.size(20.dp),
                                                tint = AppColors.LightColor
                                            )
                                        }
                                    }
                                }
                            }
                        } else {
                            // Пустой колонка для выравнивания, когда поле даты
скрыто
                            Spacer(modifier = Modifier.width(390.dp))
                        }
                    }
                }

                Spacer(Modifier.height(40.dp))

                // Основной контент
                Row(
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(bottom = 100.dp),
                    horizontalArrangement = Arrangement.spacedBy(90.dp)
                ) {
                    // Диаграмма и статистика (левая часть)
                    Box(
                        modifier = Modifier.size(650.dp),
                        contentAlignment = Alignment.TopStart
                    ) {
                        // Диаграмма для выбранного фильтра
                        Box(
                            modifier = Modifier
                                .size(650.dp)
                                .align(Alignment.TopCenter)
                        ) {
                            if (operationsByCategory.isNotEmpty()) {
                                PieChart(
                                    operationsByCategory,
                                    modifier = Modifier
                                        .fillMaxSize()
                                        .padding(32.dp),
                                    colors = if (selectedFilter == "Доходы") {
                                        listOf(
                                            AppColors.IncomeColor1,
                                            AppColors.IncomeColor2,
                                            AppColors.IncomeColor3,
                                            AppColors.IncomeColor4,
                                            AppColors.IncomeColor5,
                                            AppColors.IncomeColor6,
                                            AppColors.IncomeColor7,
                                            AppColors.IncomeColor8
                                        )
                                    } else {
                                        listOf(
                                            AppColors.ExpensesColor1,
                                            AppColors.ExpensesColor2,
                                            AppColors.ExpensesColor3,
                                            AppColors.ExpensesColor4,
```

```
                                        AppColors.ExpensesColor5,
                                        AppColors.ExpensesColor6,
                                        AppColors.ExpensesColor7,
                                        AppColors.ExpensesColor8
                                    )
                                }
                            )
                        } else {
                            Spacer(modifier = Modifier.width(650.dp))
                        }
                    }

                    // Блок статистики
                    Column(
                        modifier = Modifier
                            .width(280.dp)
                            .align(Alignment.Center),
                        verticalArrangement = Arrangement.spacedBy(25.dp)
                    ) {
                        // Доходы за период
                        Surface(
                            color = AppColors.GreenColor.copy(alpha =
0.25f),
                            shape = RoundedCornerShape(20.dp),
                            border = BorderStroke(2.dp,
AppColors.GreenColor)
                        ) {
                            Column(
                                modifier = Modifier
                                    .height(90.dp)
                                    .fillMaxWidth()
                                    .padding(horizontal = 16.dp),
                                verticalArrangement = Arrangement.Center
                            ) {
                                Row(
                                    verticalAlignment =
Alignment.CenterVertically,
                                    horizontalArrangement =
Arrangement.spacedBy(15.dp)
                                ) {
                                    // Иконка доходов
                                    Icon(
                                        imageVector =
FinlyticsIconPack.Income,
                                        contentDescription = "income",
                                        modifier = Modifier.size(22.dp),
                                        tint = AppColors.GreenColor
                                    )
                                    Text(
                                        "Доходы",
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Medium,
                                        color = AppColors.GreenColor
                                    )
                                }
                                Spacer(Modifier.height(8.dp))
                                Row(
                                    verticalAlignment =
Alignment.CenterVertically,
                                    horizontalArrangement =
Arrangement.spacedBy(10.dp)
                                ) {
                                    Spacer(Modifier.weight(1f))
                                    Text(
```

```
                                        String.format("%.2f", totalIncome),
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.GreenColor
                                    )
                                    // Иконка валюты
                                    Text(
                                        text = "₽",
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.GreenColor
                                    )
                                }
                            }
                        }

                        // Расходы за период
                        Surface(
                            color = AppColors.RedColor.copy(alpha = 0.25f),
                            shape = RoundedCornerShape(20.dp),
                            border = BorderStroke(2.dp, AppColors.RedColor)
                        ) {
                            Column(
                                modifier = Modifier
                                    .height(90.dp)
                                    .fillMaxWidth()
                                    .padding(horizontal = 16.dp),
                                verticalArrangement = Arrangement.Center
                            ) {
                                Row(
                                    verticalAlignment =
Alignment.CenterVertically,
                                    horizontalArrangement =
Arrangement.spacedBy(15.dp)
                                ) {
                                    // Иконка расходов
                                    Icon(
                                        imageVector =
FinlyticsIconPack.Expenses,
                                        contentDescription = "expenses",
                                        modifier = Modifier.size(22.dp),
                                        tint = AppColors.RedColor
                                    )
                                    Text(
                                        "Расходы",
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Medium,
                                        color = AppColors.RedColor
                                    )
                                }
                                Spacer(Modifier.height(8.dp))
                                Row(
                                    verticalAlignment =
Alignment.CenterVertically,
                                    horizontalArrangement =
Arrangement.spacedBy(10.dp)
                                ) {
                                    Spacer(Modifier.weight(1f))
                                    Text(
                                        String.format("%.2f",
totalExpenses),
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.RedColor
```

```
                                    )
                                    // Иконка валюты
                                    Text(
                                        text = "₽",
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.RedColor
                                    )
                                }
                            }
                        }

                        // Баланс за период
                        Surface(
                            color = AppColors.YellowColor.copy(alpha =
0.25f),
                            shape = RoundedCornerShape(20.dp),
                            border = BorderStroke(2.dp,
AppColors.YellowColor)
                        ) {
                            Column(
                                modifier = Modifier
                                    .height(90.dp)
                                    .fillMaxWidth()
                                    .padding(horizontal = 16.dp),
                                verticalArrangement = Arrangement.Center
                            ) {
                                Row(
                                    verticalAlignment =
Alignment.CenterVertically,
                                    horizontalArrangement =
Arrangement.spacedBy(15.dp)
                                ) {
                                    // Иконка кошелька
                                    Icon(
                                        imageVector =
FinlyticsIconPack.Wallet,
                                        contentDescription = "wallet",
                                        modifier = Modifier.size(20.dp),
                                        tint = AppColors.YellowColor
                                    )
                                    Text(
                                        "Баланс",
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Medium,
                                        color = AppColors.YellowColor
                                    )
                                }
                                Spacer(Modifier.height(8.dp))
                                Row(
                                    verticalAlignment =
Alignment.CenterVertically,
                                    horizontalArrangement =
Arrangement.spacedBy(10.dp)
                                ) {
                                    Spacer(Modifier.weight(1f))
                                    Text(
                                        String.format("%.2f", balance),
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.YellowColor
                                    )
                                    // Иконка валюты
                                    Text(
```

```
                                        text = "₽",
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.YellowColor
                                    )
                                }
                            }
                        }
                    }

                    // Список категорий (правая часть)
                    Column(
                        modifier = Modifier
                            .weight(1f)
                            .fillMaxHeight()
                            .verticalScroll(rememberScrollState()),
                        verticalArrangement = if (categoryList.isEmpty()) {
                            Arrangement.Center
                        } else {
                            Arrangement.spacedBy(25.dp)
                        }
                    ) {
                        if (categoryList.isEmpty()) {
                            // Сообщение об отсутствии данных для выбранного
фильтра
                            Text(
                                if (filteredOperationsByType.isEmpty()) "Нет
операций за выбранный период" else "Нет ${selectedFilter.lowercase()} за
выбранный период",
                                fontSize = 20.sp,
                                color = AppColors.LightColor.copy(alpha = 0.5f),
                                modifier =
Modifier.align(Alignment.CenterHorizontally)
                            )
                        } else {
                            val totalForCategoryType = if (selectedFilter ==
"Доходы") {
                                filteredOperationsByType
                                    .filter { it.type == "Доход" }
                                    .sumOf { it.amount }
                            } else {
                                filteredOperationsByType
                                    .filter { it.type == "Расход" }
                                    .sumOf { it.amount }
                            }

                            val gradients = if (selectedFilter == "Доходы") {
                                listOf(

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor1)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor2)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor3)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor4)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor5)),
```

```
Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor6)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor7)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.IncomeColor8))
                            )
                        } else {
                            listOf(

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor1)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor2)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor3)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor4)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor5)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor6)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor7)),

Brush.horizontalGradient(listOf(AppColors.DarkColor,
AppColors.ExpensesColor8))
                            )
                        }

                        categoryList.forEachIndexed { index, (category,
amount) ->
                            val percentage = if (totalForCategoryType > 0) {
                                (amount / totalForCategoryType) * 100
                            } else {
                                0.0
                            }

                            Column(
                                modifier = Modifier
                                    .fillMaxWidth()
                                    .height(67.dp)
                            ) {
                                Row(
                                    verticalAlignment =
Alignment.CenterVertically
                                ) {
                                    // Название категории
                                    Text(
                                        category,
                                        fontSize = 24.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.LightColor
                                    )
```

```
                                    Spacer(Modifier.weight(1f))

                                    // Сумма
                                    Row(
                                        horizontalArrangement =
Arrangement.spacedBy(4.dp)
                                    ) {
                                        Text(
                                            String.format("%.2f", amount),
                                            fontSize = 24.sp,
                                            fontWeight = FontWeight.Normal,
                                            color = AppColors.LightColor
                                        )
                                        // Иконка валюты
                                        Text(
                                            text = "₽",
                                            fontSize = 24.sp,
                                            fontWeight = FontWeight.Normal,
                                            color = AppColors.LightColor
                                        )
                                    }
                                }

                                Spacer(Modifier.height(8.dp))

                                Box(modifier = Modifier.height(35.dp)) {
                                    // Прогресс-бар
                                    val normalizedPercentage =
min(percentage, 100.0)
                                    val progressWidth = normalizedPercentage
/ 100.0


                                    Box(
                                        modifier = Modifier
                                            .fillMaxWidth()
                                            .height(35.dp)
                                            .background(
                                                AppColors.DarkColor,
                                                shape =
RoundedCornerShape(9.dp)
                                            )
                                    ) {
                                        // Градиентная часть прогресс-бара
                                        Box(
                                            modifier = Modifier
                                                .align(Alignment.CenterEnd)
                                                .fillMaxWidth(fraction =
progressWidth.toFloat())
                                                .height(35.dp)
                                                .background(

gradients.getOrElse(index) { gradients.last() },
                                                    shape =
RoundedCornerShape(9.dp)
                                                )
                                        )
                                    }

                                    // Процент
                                    Text(
                                        String.format("%.2f", percentage) +
" %",
                                        fontSize = 20.sp,
                                        fontWeight = FontWeight.Normal,
```

```
                                        color = AppColors.LightColor,
                                        modifier = Modifier
                                            .align(Alignment.CenterEnd)
                                            .padding(end = 8.dp)
                                    )
                                }
                            }
                        }
                    }
                }
            }

            // Навигационная панель
            Box(
                modifier = Modifier
                    .padding(bottom = 10.dp)
                    .align(Alignment.BottomCenter)
            ) {
                NavigationBar(viewModel)
            }
        }
    }

    /**
     * Фильтрует операции по временному периоду.
     */
    private fun filterOperationsByPeriod(
        operations: List<models.Operation>,
        period: String,
        selectedDate: LocalDate
    ): List<models.Operation> {
        return when (period) {
            "День" -> {
                operations.filter { it.date == selectedDate }
            }
            "Неделя" -> {
                val weekStart =
    selectedDate.with(TemporalAdjusters.previousOrSame(java.time.DayOfWeek.MONDA
    Y))
                val weekEnd = weekStart.plusDays(6)
                operations.filter {
                    val opDate = it.date
                    opDate in weekStart..weekEnd
                }
            }
            "Месяц" -> {
                val monthStart = selectedDate.withDayOfMonth(1)
                val monthEnd =
    selectedDate.with(TemporalAdjusters.lastDayOfMonth())
                operations.filter {
                    val opDate = it.date
                    opDate in monthStart..monthEnd
                }
            }
            "Год" -> {
                val yearStart = selectedDate.withDayOfYear(1)
                val yearEnd =
    selectedDate.with(TemporalAdjusters.lastDayOfYear())
                operations.filter {
                    val opDate = it.date
                    opDate in yearStart..yearEnd
                }
            }
```

```
        else -> operations // "Всё время"
    }.sortedByDescending { it.date }
}

/**
 * Фильтрует операции по типу и временному периоду.
 */
private fun filterOperationsByType(
    operations: List<models.Operation>,
    filterType: String,
    period: String,
    selectedDate: LocalDate
): List<models.Operation> {
    // Фильтрация по типу операции
    val filteredByType = when (filterType) {
        "Доходы" -> operations.filter { it.type == "Доход" }
        "Расходы" -> operations.filter { it.type == "Расход" }
        else -> operations
    }

    // Фильтрация по периоду времени
    return when (period) {
        "День" -> {
            filteredByType.filter { it.date == selectedDate }
        }
        "Неделя" -> {
            val weekStart =
selectedDate.with(TemporalAdjusters.previousOrSame(java.time.DayOfWeek.MONDA
Y))
            val weekEnd = weekStart.plusDays(6)
            filteredByType.filter {
                val opDate = it.date
                opDate in weekStart..weekEnd
            }
        }
        "Месяц" -> {
            val monthStart = selectedDate.withDayOfMonth(1)
            val monthEnd =
selectedDate.with(TemporalAdjusters.lastDayOfMonth())
            filteredByType.filter {
                val opDate = it.date
                opDate in monthStart..monthEnd
            }
        }
        "Год" -> {
            val yearStart = selectedDate.withDayOfYear(1)
            val yearEnd =
selectedDate.with(TemporalAdjusters.lastDayOfYear())
            filteredByType.filter {
                val opDate = it.date
                opDate in yearStart..yearEnd
            }
        }
        else -> filteredByType // "Всё время"
    }.sortedByDescending { it.date }
}

@Composable
private fun FilterButton(
    text: String,
    isSelected: Boolean,
    modifier: Modifier = Modifier,
    icon: Boolean = false,
    isAll: Boolean = false,
```

```
    isIncome: Boolean = true,
    onClick: () -> Unit
) {
    Button(
        onClick = onClick,
        modifier = modifier,
        shape = RoundedCornerShape(10.dp),
        colors = ButtonDefaults.buttonColors(
            backgroundColor = if (!isSelected) AppColors.LightGreyColor
            else if (isAll) AppColors.BlueColor
            else if (isIncome) AppColors.GreenColor
            else AppColors.RedColor
        ),
        elevation = null
    ) {
        if (icon) {
            Row(
                horizontalArrangement = Arrangement.Center,
                verticalAlignment = Alignment.CenterVertically
            ) {
                if (isIncome) {
                    Icon(
                        imageVector = FinlyticsIconPack.Income,
                        contentDescription = "income",
                        modifier = Modifier.size(22.dp),
                        tint = AppColors.LightColor
                    )
                } else {
                    Icon(
                        imageVector = FinlyticsIconPack.Expenses,
                        contentDescription = "expenses",
                        modifier = Modifier.size(22.dp),
                        tint = AppColors.LightColor
                    )
                }
                Spacer(Modifier.width(12.dp))
                Text(
                    text,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Normal,
                    color = AppColors.LightColor
                )
            }
        } else {
            Text(
                text,
                fontSize = 20.sp,
                fontWeight = FontWeight.Normal,
                color = AppColors.LightColor
            )
        }
    }
}

@Composable
private fun PeriodButton(
    text: String,
    isSelected: Boolean,
    onClick: () -> Unit
) {
    Button(
        onClick = onClick,
        modifier = Modifier.defaultMinSize(minWidth = 70.dp),
        shape = RoundedCornerShape(10.dp),
```

```
        colors = ButtonDefaults.buttonColors(
            backgroundColor = if (isSelected) AppColors.BlueColor else
AppColors.LightGreyColor
        ),
        elevation = null
    ) {
        Text(
            text,
            fontSize = 20.sp,
            fontWeight = FontWeight.Normal,
            color = AppColors.LightColor
        )
    }
}
```

Листинг 6 – HistoryScreen.kt – Экран истории операций

```
package ui.screens

import androidx.compose.foundation.*
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.runtime.*
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import java.time.LocalDate
import java.time.format.DateTimeFormatter
import java.time.temporal.TemporalAdjusters
import models.Operation
import ui.components.NavigationBar
import ui.theme.AppColors
import ui.theme.icons.FinlyticsIconPack
import ui.theme.icons.finlyticsiconpack.*
import viewmodel.FinanceViewModel

/**
 * Экран истории операций.
 * Отображает список финансовых операций с возможностью фильтрации по типу и
периоду.
 * Предоставляет функции редактирования и удаления операций.
 *
 * @param viewModel ViewModel для управления данными операций
 */
@Composable
fun HistoryScreen(viewModel: FinanceViewModel) {
    val state by viewModel.state.collectAsState()

    // Состояния для фильтров
    var selectedFilter by remember { mutableStateOf("Все") }
    var selectedPeriod by remember { mutableStateOf("Всё время") }
    var selectedDate by remember { mutableStateOf(LocalDate.now()) }
```

```
    var operationToDelete by remember { mutableStateOf<Operation?>(null) }

    // Форматирование даты для отображения
    val dateFormatter = DateTimeFormatter.ofPattern("dd.MM.yyyy")
    val monthYearFormatter = DateTimeFormatter.ofPattern("MM.yyyy")
    val yearFormatter = DateTimeFormatter.ofPattern("yyyy")

    // Функция для фильтрации операций
    val filteredOperations = remember(state.operations, selectedFilter,
selectedPeriod, selectedDate) {
        filterOperations(
            operations = state.operations,
            filterType = selectedFilter,
            period = selectedPeriod,
            selectedDate = selectedDate
        )
    }

    // Отладочная информация об операциях
    LaunchedEffect(state.operations, filteredOperations) {
        println("\n=== HISTORY SCREEN ===")
        println("Всего операций в базе: ${state.operations.size}")
        println("Отфильтровано операций: ${filteredOperations.size}")
        println("Фильтр: $selectedFilter, Период: $selectedPeriod")
        println("Операций доходов: ${filteredOperations.count { it.type ==
"Доход" }}")
        println("Операций расходов: ${filteredOperations.count { it.type ==
"Расход" }}")
        if (filteredOperations.isNotEmpty()) {
            println("Последние 3 отфильтрованные операции:")
            filteredOperations.take(3).forEach { op ->
                println("  - ${op.type}: ${op.category} - ${op.amount} руб.
(${op.date})")
            }
        }
        println("======================\n")
    }

    // Функция для получения отображаемой даты в зависимости от периода
    val displayDateText = remember(selectedPeriod, selectedDate) {
        when (selectedPeriod) {
            "День" -> selectedDate.format(dateFormatter)
            "Неделя" -> {
                val weekStart =
selectedDate.with(TemporalAdjusters.previousOrSame(java.time.DayOfWeek.MONDA
Y))
                val weekEnd = weekStart.plusDays(6)
                "${weekStart.format(dateFormatter)} -
${weekEnd.format(dateFormatter)}"
            }
            "Месяц" -> selectedDate.format(monthYearFormatter)
            "Год" -> selectedDate.format(yearFormatter)
            else -> ""
        }
    }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(AppColors.DarkColor)
    ) {
        // Контент страницы
        Box(
            modifier = Modifier
```

```
                .fillMaxWidth()
                .padding(horizontal = 25.dp)
        ) {
            // Секция настроек
            Box(
                modifier = Modifier
                    .padding(top = 20.dp)
                    .height(193.dp)
                    .background(AppColors.DarkGreyColor,
RoundedCornerShape(30.dp))
            ) {
                Row(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(start = 40.dp, end = 40.dp, top = 15.dp),
                    horizontalArrangement = Arrangement.SpaceBetween,
                    verticalAlignment = Alignment.Top
                ) {
                    // Фильтр по типу
                    Column(
                        modifier = Modifier.width(390.dp),
                        verticalArrangement = Arrangement.spacedBy(15.dp)
                    ) {
                        Text(
                            "Параметр отображения",
                            fontSize = 24.sp,
                            fontWeight = FontWeight.Medium,
                            color = AppColors.LightColor
                        )

                        Row(
                            horizontalArrangement =
Arrangement.spacedBy(15.dp),
                            verticalAlignment = Alignment.CenterVertically
                        ) {
                            // Кнопка "Все"
                            FilterButton(
                                text = "Все",
                                isSelected = selectedFilter == "Все",
                                isAll = true,
                                modifier =
Modifier.width(85.dp).height(98.dp),
                                onClick = { selectedFilter = "Все" }
                            )

                            Column(
                                verticalArrangement =
Arrangement.spacedBy(15.dp)
                            ) {
                                // Кнопка "Доходы"
                                FilterButton(
                                    text = "Доходы",
                                    isSelected = selectedFilter == "Доходы",
                                    icon = true,
                                    modifier =
Modifier.width(290.dp).height(42.dp),
                                    onClick = { selectedFilter = "Доходы" }
                                )

                                // Кнопка "Расходы"
                                FilterButton(
                                    text = "Расходы",
                                    isSelected = selectedFilter ==
"Расходы",
```

```
                                    icon = true,
                                    isIncome = false,
                                    modifier =
Modifier.width(290.dp).height(42.dp),
                                    onClick = { selectedFilter = "Расходы" }
                                )
                            }
                        }
                    }

                    // Период времени
                    Column(
                        verticalArrangement = Arrangement.spacedBy(15.dp)
                    ) {
                        Text(
                            "Период времени",
                            fontSize = 24.sp,
                            fontWeight = FontWeight.Medium,
                            color = AppColors.LightColor
                        )

                        // Кнопки периодов
                        Row(
                            modifier = Modifier.width(350.dp),
                            horizontalArrangement =
Arrangement.spacedBy(20.dp),
                            verticalAlignment = Alignment.CenterVertically
                        ) {
                            PeriodButton(
                                text = "День",
                                isSelected = selectedPeriod == "День",
                                onClick = {
                                    selectedPeriod = "День"
                                    selectedDate = LocalDate.now()
                                }
                            )
                            PeriodButton(
                                text = "Неделя",
                                isSelected = selectedPeriod == "Неделя",
                                onClick = {
                                    selectedPeriod = "Неделя"
                                    selectedDate = LocalDate.now()
                                }
                            )
                            PeriodButton(
                                text = "Месяц",
                                isSelected = selectedPeriod == "Месяц",
                                onClick = {
                                    selectedPeriod = "Месяц"
                                    selectedDate = LocalDate.now()
                                }
                            )
                        }

                        Row(
                            modifier = Modifier.width(350.dp),
                            horizontalArrangement =
Arrangement.spacedBy(20.dp),
                            verticalAlignment = Alignment.CenterVertically
                        ) {
                            PeriodButton(
                                text = "Год",
                                isSelected = selectedPeriod == "Год",
                                onClick = {
```

```
                                    selectedPeriod = "Год"
                                    selectedDate = LocalDate.now()
                                }
                            )
                            PeriodButton(
                                text = "Всё время",
                                isSelected = selectedPeriod == "Всё время",
                                onClick = {
                                    selectedPeriod = "Всё время"
                                }
                            )
                        }
                    }

                    // Настройки даты
                    if (selectedPeriod != "Всё время") {
                        Column(
                            verticalArrangement =
Arrangement.spacedBy(15.dp)
                        ) {
                            Text(
                                selectedPeriod,
                                fontSize = 24.sp,
                                fontWeight = FontWeight.Medium,
                                color = AppColors.LightColor
                            )

                            // Поле ввода даты
                            Box(
                                modifier = Modifier
                                    .width(390.dp)
                                    .height(42.dp)
                                    .background(AppColors.LightGreyColor,
RoundedCornerShape(10.dp))
                            ) {
                                Row(
                                    modifier = Modifier.fillMaxSize(),
                                    verticalAlignment =
Alignment.CenterVertically
                                ) {
                                    Spacer(Modifier.width(16.dp))

                                    Icon(
                                        imageVector =
FinlyticsIconPack.Date,

                                        contentDescription = "date",
                                        modifier = Modifier.size(18.dp),
                                        tint = AppColors.LightColor
                                    )

                                    Spacer(Modifier.width(18.dp))

                                    Text(
                                        displayDateText,
                                        fontSize = 20.sp,
                                        fontWeight = FontWeight.Normal,
                                        color = AppColors.LightColor
                                    )

                                    Spacer(Modifier.weight(1f))

                                    // Кнопки навигации
                                    Row(
                                        modifier = Modifier.padding(end =
```

```
6.dp),
                                        horizontalArrangement =
Arrangement.spacedBy(4.dp),
                                        verticalAlignment =
Alignment.CenterVertically
                                    ) {
                                        IconButton(
                                            onClick = {
                                                selectedDate = when
(selectedPeriod) {
                                                    "День" ->
selectedDate.minusDays(1)
                                                    "Неделя" ->
selectedDate.minusWeeks(1)
                                                    "Месяц" ->
selectedDate.minusMonths(1)
                                                    "Год" ->
selectedDate.minusYears(1)
                                                    else -> selectedDate
                                                }
                                            },
                                            modifier = Modifier
.background(AppColors.BlueColor, RoundedCornerShape(5.dp))
                                                .size(20.dp)
                                        ) {
                                            Icon(
                                                imageVector =
FinlyticsIconPack.Left,
                                                contentDescription =
"previous_period",
                                                modifier =
Modifier.size(20.dp),
                                                tint = AppColors.LightColor
                                            )
                                        }

                                        IconButton(
                                            onClick = {
                                                selectedDate = when
(selectedPeriod) {
                                                    "День" ->
selectedDate.plusDays(1)
                                                    "Неделя" ->
selectedDate.plusWeeks(1)
                                                    "Месяц" ->
selectedDate.plusMonths(1)
                                                    "Год" ->
selectedDate.plusYears(1)
                                                    else -> selectedDate
                                                }
                                            },
                                            modifier = Modifier
.background(AppColors.BlueColor, RoundedCornerShape(5.dp))
                                                .size(20.dp)
                                        ) {
                                            Icon(
                                                imageVector =
FinlyticsIconPack.Right,
                                                contentDescription =
"next_period",
                                                modifier =
Modifier.size(20.dp),
```

```
                                                tint = AppColors.LightColor
                                            )
                                        }
                                    }
                                }
                            }
                        } else {
                            // Пустой колонка для выравнивания, когда поле даты
скрыто
                            Spacer(modifier = Modifier.width(390.dp))
                        }
                    }
                }

                // Основной контент
                Box(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(top = 243.dp)
                ) {
                    if (filteredOperations.isEmpty()) {
                        // Сообщение, если операций нет
                        Box(
                            modifier = Modifier.fillMaxSize(),
                            contentAlignment = Alignment.Center
                        ) {
                            Text(
                                "Нет операций за выбранный период",
                                fontSize = 24.sp,
                                fontWeight = FontWeight.Medium,
                                color = AppColors.LightGreyColor
                            )
                        }
                    } else {
                        // Список операций
                        LazyColumn(
                            modifier = Modifier.fillMaxSize(),
                            verticalArrangement = Arrangement.spacedBy(20.dp)
                        ) {
                            items(filteredOperations) { operation ->
                                TransactionItem(
                                    operation = operation,
                                    onEditClick = {
viewModel.showEditOperation(operation) },
                                    onDeleteClick = { operationToDelete =
operation }
                                )

                                // Разделитель (кроме последнего элемента)
                                if (operation != filteredOperations.last()) {
                                    Divider(
                                        modifier = Modifier
                                            .fillMaxWidth()
                                            .height(1.dp),
                                        color = AppColors.LightGreyColor
                                    )
                                }
                            }
                        }
                    }
                }
            }
        }
```

```
        // Навигационная панель
        Box(
            modifier = Modifier
                .padding(bottom = 10.dp)
                .align(Alignment.BottomCenter)
        ) {
            NavigationBar(viewModel)
        }
    }

    // Диалог подтверждения удаления
    if (operationToDelete != null) {
        AlertDialog(
            onDismissRequest = { operationToDelete = null },
            title = { Text("Подтверждение удаления") },
            text = {
                Text("Вы уверены, что хотите удалить операцию?\n" +
                        "${operationToDelete!!.category}: " +
                        "${operationToDelete?.name ?: ""}\n" +
                        "${String.format("%.2f",
operationToDelete!!.amount)} ₽")
            },
            confirmButton = {
                Button(
                    onClick = {
                        viewModel.deleteOperation(operationToDelete!!)
                        operationToDelete = null
                    },
                    colors = ButtonDefaults.buttonColors(backgroundColor =
AppColors.RedColor)
                ) {
                    Text("Удалить", color = AppColors.LightColor)
                }
            },
            dismissButton = {
                TextButton(
                    onClick = { operationToDelete = null }
                ) {
                    Text("Отмена", color = AppColors.LightColor)
                }
            }
        )
    }
}

/**
 * Фильтрует список операций по заданным параметрам.
 *
 * @param operations Исходный список операций
 * @param filterType Тип фильтра ("Все", "Доходы", "Расходы")
 * @param period Временной период ("День", "Неделя", "Месяц", "Год", "Всё
время")
 * @param selectedDate Базовая дата для фильтрации
 * @return Отфильтрованный список операций
 */
private fun filterOperations(
    operations: List<Operation>,
    filterType: String,
    period: String,
    selectedDate: LocalDate
): List<Operation> {
    // Фильтрация по типу операции
    val filteredByType = when (filterType) {
        "Доходы" -> operations.filter { it.type == "Доход" }
```

```
        "Расходы" -> operations.filter { it.type == "Расход" }
        else -> operations // "Все"
    }

    // Фильтрация по периоду времени
    return when (period) {
        "День" -> {
            filteredByType.filter { it.date == selectedDate }
        }
        "Неделя" -> {
            val weekStart =
selectedDate.with(TemporalAdjusters.previousOrSame(java.time.DayOfWeek.MONDA
Y))
            val weekEnd = weekStart.plusDays(6)
            filteredByType.filter {
                val opDate = it.date
                opDate in weekStart..weekEnd
            }
        }
        "Месяц" -> {
            val monthStart = selectedDate.withDayOfMonth(1)
            val monthEnd =
selectedDate.with(TemporalAdjusters.lastDayOfMonth())
            filteredByType.filter {
                val opDate = it.date
                opDate in monthStart..monthEnd
            }
        }
        "Год" -> {
            val yearStart = selectedDate.withDayOfYear(1)
            val yearEnd =
selectedDate.with(TemporalAdjusters.lastDayOfYear())
            filteredByType.filter {
                val opDate = it.date
                opDate in yearStart..yearEnd
            }
        }
        else -> filteredByType
    }.sortedByDescending { it.date }
}

/**
 * Компонент кнопки фильтра по типу операции.
 */
@Composable
private fun FilterButton(
    text: String,
    isSelected: Boolean,
    modifier: Modifier = Modifier,
    icon: Boolean = false,
    isAll: Boolean = false,
    isIncome: Boolean = true,
    onClick: () -> Unit
) {
    Button(
        onClick = onClick,
        modifier = modifier,
        shape = RoundedCornerShape(10.dp),
        colors = ButtonDefaults.buttonColors(
            backgroundColor = if (!isSelected) AppColors.LightGreyColor
            else if (isAll) AppColors.BlueColor
            else if (isIncome) AppColors.GreenColor
            else AppColors.RedColor
        ),
```

```
        elevation = null
    ) {
        if (icon) {
            Row(
                horizontalArrangement = Arrangement.Center,
                verticalAlignment = Alignment.CenterVertically
            ) {
                if (isIncome) {
                    Icon(
                        imageVector = FinlyticsIconPack.Income,
                        contentDescription = "income",
                        modifier = Modifier.size(22.dp),
                        tint = AppColors.LightColor
                    )
                } else {
                    Icon(
                        imageVector = FinlyticsIconPack.Expenses,
                        contentDescription = "expenses",
                        modifier = Modifier.size(22.dp),
                        tint = AppColors.LightColor
                    )
                }
                Spacer(Modifier.width(12.dp))
                Text(
                    text,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Normal,
                    color = AppColors.LightColor
                )
            }
        } else {
            Text(
                text,
                fontSize = 20.sp,
                fontWeight = FontWeight.Normal,
                color = AppColors.LightColor
            )
        }
    }
}

/**
 * Компонент кнопки выбора временного периода.
 */
@Composable
private fun PeriodButton(
    text: String,
    isSelected: Boolean,
    onClick: () -> Unit
) {
    Button(
        onClick = onClick,
        modifier = Modifier.defaultMinSize(minWidth = 70.dp),
        shape = RoundedCornerShape(10.dp),
        colors = ButtonDefaults.buttonColors(
            backgroundColor = if (isSelected) AppColors.BlueColor else
AppColors.LightGreyColor
        ),
        elevation = null
    ) {
        Text(
            text,
            fontSize = 20.sp,
            fontWeight = FontWeight.Normal,
```

```
            color = AppColors.LightColor
        )
    }
}

/**
 * Компонент элемента списка операций.
 * Отображает информацию об операции и кнопки действий.
 */
@Composable
private fun TransactionItem(
    operation: Operation,
    onEditClick: () -> Unit,
    onDeleteClick: () -> Unit
) {
    val formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy")
    val dateText = operation.date.format(formatter)

    Box(
        modifier = Modifier
            .fillMaxWidth()
            .wrapContentHeight()
            .padding(bottom = 20.dp)
    ) {
        // Дата слева
        Text(
            text = "[ $dateText ]",
            fontSize = 20.sp,
            fontWeight = FontWeight.Light,
            color = AppColors.LightColor,
            modifier = Modifier
                .align(Alignment.TopStart)
                .offset(x = 0.dp, y = 4.dp)
        )

        // Контент операции
        Column(
            modifier = Modifier
                .width(1120.dp)
                .align(Alignment.CenterStart)
                .offset(x = 150.dp)
                .wrapContentHeight(),
            verticalArrangement = Arrangement.spacedBy(10.dp)
        ) {
            // Категория с цветным фоном
            Box(
                modifier = Modifier
                    .wrapContentWidth()
                    .height(32.dp)
                    .border(
                        width = 1.dp,
                        color = if (operation.type == "Доход")
AppColors.GreenColor else AppColors.RedColor,
                        shape = RoundedCornerShape(10.dp)
                    )
                    .background(
                        color = if (operation.type == "Доход")
                            AppColors.GreenColor.copy(alpha = 0.25f)
                        else
                            AppColors.RedColor.copy(alpha = 0.25f),
                        shape = RoundedCornerShape(10.dp)
                    )
                    .padding(horizontal = 10.dp, vertical = 4.dp),
                contentAlignment = Alignment.Center
```

```
            ) {
                Text(
                    text = operation.category,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Normal,
                    color = if (operation.type == "Доход")
AppColors.GreenColor else AppColors.RedColor
                )
            }

            // Описание
            Text(
                text = if (operation.name == "" || operation.name == null)
"Без описания" else operation.name,
                fontSize = 20.sp,
                fontWeight = FontWeight.Light,
                color = AppColors.LightColor
            )

            // Сумма
            Row(
                horizontalArrangement = Arrangement.spacedBy(10.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                // Иконка плюса/минуса
                Box(
                    modifier = Modifier.size(14.dp),
                    contentAlignment = Alignment.Center
                ) {
                    if (operation.type == "Доход") {
                        Icon(
                            imageVector = FinlyticsIconPack.Plus,
                            contentDescription = "plus",
                            modifier = Modifier.size(14.dp),
                            tint = AppColors.LightColor
                        )
                    } else {
                        Icon(
                            imageVector = FinlyticsIconPack.Minus,
                            contentDescription = "minus",
                            modifier = Modifier.size(14.dp),
                            tint = AppColors.LightColor
                        )
                    }
                }

                Text(
                    text = String.format("%.2f",
operation.amount).replace('.', ','),
                    fontSize = 32.sp,
                    fontWeight = FontWeight.Medium,
                    color = AppColors.LightColor
                )

                // Символ рубля
                Text(
                    text = "₽",
                    fontSize = 32.sp,
                    fontWeight = FontWeight.Medium,
                    color = AppColors.LightColor
                )
            }
        }
```

```
        // Кнопки действий справа
        Row(
            modifier = Modifier.align(Alignment.CenterEnd),
            horizontalArrangement = Arrangement.spacedBy(10.dp)
        ) {
            // Кнопка редактирования
            IconButton(
                onClick = onEditClick,
                modifier = Modifier.size(50.dp)
            ) {
                Box(
                    modifier = Modifier
                        .fillMaxSize()
                        .background(AppColors.YellowColor,
RoundedCornerShape(10.dp)),
                    contentAlignment = Alignment.Center
                ) {
                    Icon(
                        imageVector = FinlyticsIconPack.Edit,
                        contentDescription = "edit",
                        modifier = Modifier.size(50.dp),
                        tint = AppColors.LightColor
                    )
                }
            }

            // Кнопка удаления
            IconButton(
                onClick = onDeleteClick,
                modifier = Modifier.size(50.dp)
            ) {
                Box(
                    modifier = Modifier
                        .fillMaxSize()
                        .background(AppColors.RedColor,
RoundedCornerShape(10.dp)),
                    contentAlignment = Alignment.Center
                ) {
                    Icon(
                        imageVector = FinlyticsIconPack.Delete,
                        contentDescription = "delete",
                        modifier = Modifier.size(50.dp),
                        tint = AppColors.LightColor
                    )
                }
            }
        }
    }
}
```

Листинг 7 – SettingsScreen.kt – Экран настроек категорий

```
package ui.screens

import androidx.compose.foundation.*
import androidx.compose.foundation.interaction.MutableInteractionSource
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.runtime.*
```

```
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import ui.components.NavigationBar
import ui.theme.AppColors
import ui.theme.icons.FinlyticsIconPack
import ui.theme.icons.finlyticsiconpack.*
import viewmodel.FinanceViewModel

/**
 * Экран настроек приложения.
 * Предоставляет управление категориями доходов и расходов:
 * просмотр, добавление и удаление категорий.
 *
 * @param viewModel ViewModel для управления категориями
 */
@Composable
fun SettingsScreen(viewModel: FinanceViewModel) {
    val state by viewModel.state.collectAsState()

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(AppColors.DarkColor)
    ) {
        // Основной контент
        Box(
            modifier = Modifier
                .fillMaxSize()
                .padding(top = 50.dp, bottom = 124.dp)
        ) {
            Row(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(horizontal = 55.dp)
                    .align(Alignment.Center),
                horizontalArrangement = Arrangement.spacedBy(30.dp)
            ) {
                // Левая панель - категории доходов
                CategoryPanel(
                    title = "Категории доходов",
                    count = state.incomeCategories.size,
                    categories = state.incomeCategories,
                    onAddClick = { viewModel.showAddCategory(true) },
                    onDeleteClick = { category ->
viewModel.deleteCategory(category, true) },
                    modifier = Modifier.weight(1f)
                )

                // Правая панель - категории расходов
                CategoryPanel(
                    title = "Категории расходов",
                    count = state.expenseCategories.size,
                    categories = state.expenseCategories,
                    onAddClick = { viewModel.showAddCategory(false) },
                    onDeleteClick = { category ->
viewModel.deleteCategory(category, false) },
                    modifier = Modifier.weight(1f)
                )
            }
        }
```

```
        // Навигационная панель
        Box(
            modifier = Modifier
                .padding(bottom = 10.dp)
                .align(Alignment.BottomCenter)
        ) {
            NavigationBar(viewModel)
        }
    }
}

/**
 * Панель управления категориями (доходов или расходов).
 */
@Composable
fun CategoryPanel(
    title: String,
    count: Int,
    categories: List<String>,
    onAddClick: () -> Unit,
    onDeleteClick: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    Box(
        modifier = modifier
            .fillMaxHeight()
            .clip(RoundedCornerShape(40.dp))
            .background(AppColors.DarkGreyColor)
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(40.dp)
        ) {
            // Заголовок панели
            PanelHeader(
                title = title,
                count = count,
                onAddClick = onAddClick
            )

            Spacer(modifier = Modifier.height(25.dp))

            // Список категорий
            if (categories.isEmpty()) {
                Box(
                    modifier = Modifier.fillMaxSize(),
                    contentAlignment = Alignment.Center
                ) {
                    Text(
                        text = "Нет категорий",
                        color = AppColors.LightColor.copy(alpha = 0.5f),
                        fontSize = 16.sp
                    )
                }
            } else {
                CategoryList(
                    categories = categories,
                    onDeleteClick = onDeleteClick
                )
            }
        }
    }
```

```
}

/**
 * Заголовок панели категорий с кнопкой добавления.
 */
@Composable
fun PanelHeader(
    title: String,
    count: Int,
    onAddClick: () -> Unit
) {
    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.SpaceBetween,
        verticalAlignment = Alignment.CenterVertically
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(13.dp)
        ) {
            Text(
                text = title,
                color = AppColors.LightColor,
                fontSize = 24.sp,
                fontWeight = FontWeight.Medium,
                letterSpacing = 0.sp
            )

            // Счетчик категорий
            Box(
                modifier = Modifier
                    .clip(RoundedCornerShape(8.dp))
                    .background(AppColors.LightGreyColor)
                    .padding(horizontal = 10.dp, vertical = 2.dp)
            ) {
                Text(
                    text = count.toString(),
                    color = AppColors.LightColor,
                    fontSize = 16.sp,
                    fontWeight = FontWeight.Medium,
                    letterSpacing = 0.sp
                )
            }
        }

        // Кнопка добавления
        AddButton(onClick = onAddClick)
    }
}

/**
 * Кнопка добавления новой категории.
 */
@Composable
fun AddButton(onClick: () -> Unit) {
    Box(
        modifier = Modifier
            .size(30.dp)
            .clip(RoundedCornerShape(8.dp))
            .background(AppColors.BlueColor)
            .clickable(
                interactionSource = remember { MutableInteractionSource() },
                indication = null,
                onClick = onClick
```

```
            ),
        contentAlignment = Alignment.Center
    ) {
        Icon(
            imageVector = FinlyticsIconPack.Add,
            contentDescription = "add",
            modifier = Modifier.size(28.dp),
            tint = AppColors.LightColor
        )
    }
}

/**
 * Прокручиваемый список категорий.
 */
@Composable
fun CategoryList(
    categories: List<String>,
    onDeleteClick: (String) -> Unit
) {
    val lazyListState = rememberLazyListState()

    LazyColumn(
        state = lazyListState,
        modifier = Modifier.fillMaxHeight(),
        verticalArrangement = Arrangement.spacedBy(15.dp)
    ) {
        items(categories) { category ->
            CategoryItem(
                category = category,
                onDeleteClick = { onDeleteClick(category) }
            )
        }
    }
}

/**
 * Элемент списка категорий с кнопками действий.
 */
@Composable
fun CategoryItem(
    category: String,
    onDeleteClick: () -> Unit
) {
    Box(
        modifier = Modifier
            .fillMaxWidth()
            .height(50.dp)
    ) {
        // Фон элемента
        Box(
            modifier = Modifier
                .fillMaxSize()
                .clip(RoundedCornerShape(15.dp))
                .background(AppColors.LightGreyColor)
        )

        // Название категории
        Text(
            text = category,
            color = AppColors.LightColor,
            fontSize = 20.sp,
            fontWeight = FontWeight.Normal,
            letterSpacing = 0.sp,
```

```
            modifier = Modifier
                .align(Alignment.CenterStart)
                .padding(start = 25.dp)
        )

        // Кнопки действий
        Row(
            modifier = Modifier
                .align(Alignment.CenterEnd)
                .padding(end = 25.dp),
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            // Кнопка редактирования (заглушка)
            EditButton(onClick = { /* TODO: Реализовать редактирование */ })

            // Кнопка удаления
            DeleteButton(onClick = onDeleteClick)
        }
    }
}

/**
 * Кнопка редактирования категории (заглушка).
 */
@Composable
fun EditButton(onClick: () -> Unit) {
    Box(
        modifier = Modifier
            .size(28.dp)
            .clip(RoundedCornerShape(8.dp))
            .background(AppColors.YellowColor)
            .clickable(
                interactionSource = remember { MutableInteractionSource() },
                indication = null,
                onClick = onClick
            ),
        contentAlignment = Alignment.Center
    ) {
        Icon(
            imageVector = FinlyticsIconPack.Edit,
            contentDescription = "edit",
            modifier = Modifier.size(28.dp),
            tint = AppColors.LightColor
        )
    }
}

/**
 * Кнопка удаления категории.
 */
@Composable
fun DeleteButton(onClick: () -> Unit) {
    Box(
        modifier = Modifier
            .size(28.dp)
            .clip(RoundedCornerShape(8.dp))
            .background(AppColors.RedColor)
            .clickable(
                interactionSource = remember { MutableInteractionSource() },
                indication = null,
                onClick = onClick
            ),
        contentAlignment = Alignment.Center
    ) {
```

```
        Icon(
            imageVector = FinlyticsIconPack.Delete,
            contentDescription = "delete",
            modifier = Modifier.size(28.dp),
            tint = AppColors.LightColor
        )
    }
}
```

Компоненты интерфейса:

Листинг 8 – PieChart.kt – Компонент круговой диаграммы

```
package ui.components

import androidx.compose.foundation.Canvas
import androidx.compose.foundation.layout.*
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.geometry.Size
import androidx.compose.ui.graphics.*
import androidx.compose.ui.graphics.drawscope.Stroke
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlin.math.*
import ui.theme.AppColors

/**
 * Компонент круговой диаграммы для визуализации распределения расходов по категориям.
 * Использует Canvas для отрисовки сегментов диаграммы.
 *
 * @param data Карта данных: ключ - название категории, значение - сумма расходов
 * @param modifier Модификатор для настройки размера и позиционирования
 * @param colors Список цветов для сегментов диаграммы
 */
@Composable
fun PieChart(
    data: Map<String, Double>,
    modifier: Modifier = Modifier,
    colors: List<Color> = listOf(
        AppColors.IncomeColor1,
        AppColors.IncomeColor2,
        AppColors.IncomeColor3,
        AppColors.IncomeColor4,
        AppColors.IncomeColor5,
        AppColors.IncomeColor6,
        AppColors.IncomeColor7,
        AppColors.IncomeColor8,
        AppColors.ExpensesColor1,
        AppColors.ExpensesColor2,
        AppColors.ExpensesColor3,
        AppColors.ExpensesColor4,
        AppColors.ExpensesColor5,
        AppColors.ExpensesColor6,
        AppColors.ExpensesColor7,
        AppColors.ExpensesColor8
    )
```

```
) {
    println("\n=== PIE CHART ===")
    println("Полученные данные: $data")
    println("Количество категорий: ${data.size}")
    println("Данные не пустые: ${data.isNotEmpty()}")

    // Фильтруем данные, оставляя только категории с положительными
значениями
    val filteredData = data.filterValues { it > 0 }
        .toList()
        .sortedByDescending { it.second }
    val total = filteredData.sumOf { it.second }

    println("Отфильтрованные данные: $filteredData")
    println("Общая сумма расходов: $total")

    if (filteredData.isNotEmpty()) {
        println("Детализация по категориям:")
        filteredData.forEach { (category, amount) ->
            val percentage = (amount / total * 100)
            println("  - $category: $amount руб.
(${"%.1f".format(percentage)}%)")
        }
    }
    println("==================\n")

    // Если данных нет, показываем сообщение
    if (total == 0.0 || filteredData.isEmpty()) {
        Box(
            modifier = modifier,
            contentAlignment = Alignment.Center
        ) {
            Column(
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.spacedBy(8.dp)
            ) {
                // Иконка пустой диаграммы (круг)
                Canvas(
                    modifier = Modifier.size(100.dp)
                ) {
                    drawCircle(
                        color = Color.LightGray.copy(alpha = 0.3f),
                        style = Stroke(width = 4f)
                    )
                }
                Text(
                    "Нет данных по расходам",
                    color = Color.LightGray.copy(alpha = 0.8f),
                    fontSize = 16.sp
                )
            }
        }
        return
    }

    Box(
        modifier = modifier,
        contentAlignment = Alignment.Center
    ) {
        // Основная круговая диаграмма
        Canvas(
            modifier = Modifier
                .fillMaxSize()
                .aspectRatio(1f)
```

```
        ) {
            val canvasSize = min(size.width, size.height)
            val radius = canvasSize / 2 - 20
            val centerOffset = Offset(size.width / 2, size.height / 2)

            // Вычисляем углы для каждого сегмента
            var startAngle = -90f

            filteredData.forEachIndexed { index, (_, amount) ->
                val sweepAngle = (amount / total * 360).toFloat()

                // Рисуем сегмент
                drawArc(
                    color = colors.getOrElse(index) { colors.get(8 - index)
},
                    startAngle = startAngle,
                    sweepAngle = sweepAngle,
                    useCenter = true,
                    topLeft = Offset(
                        centerOffset.x - radius,
                        centerOffset.y - radius
                    ),
                    size = Size(radius * 2, radius * 2)
                )

                startAngle += sweepAngle
            }

            val centerCircleRadius = radius * 0.9f

            drawCircle(
                color = AppColors.DarkColor,
                center = centerOffset,
                radius = centerCircleRadius
            )
        }
    }
}
```

Листинг 9 – OperationDialog.kt – Диалог операций

```
package ui.components

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import models.Operation
import viewmodel.FinanceViewModel
import java.time.LocalDate
import java.time.format.DateTimeParseException
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.runtime.LaunchedEffect
import ui.theme.AppColors

/**
 * Диалоговое окно для добавления или редактирования финансовой операции.
 * Поддерживает ввод всех параметров операции: тип, сумму, описание,
```

```
категорию и дату.
 *
 * @param viewModel ViewModel для управления операциями
 */
@Composable
fun OperationDialog(viewModel: FinanceViewModel) {
    val state by viewModel.state.collectAsState()

    // Отладочная информация о редактируемой операции
    LaunchedEffect(viewModel.editingOperation) {
        println("\n=== OPERATION DIALOG ===")
        println("Режим: ${if (viewModel.editingOperation == null)
"Добавление" else "Редактирование"}")
        println("Операция для редактирования:
${viewModel.editingOperation}")
        println("========================\n")
    }

    var type by remember { mutableStateOf(viewModel.editingOperation?.type
?: "Расход") }
    var amount by remember {
mutableStateOf(viewModel.editingOperation?.amount?.toString() ?: "") }
    var name by remember { mutableStateOf(viewModel.editingOperation?.name
?: "") }
    var category by remember {
mutableStateOf(viewModel.editingOperation?.category ?: "") }
    var dateText by remember {
mutableStateOf(viewModel.editingOperation?.date?.toString() ?:
LocalDate.now().toString()) }
    var dateError by remember { mutableStateOf("") }
    var amountError by remember { mutableStateOf("") }
    var categoryError by remember { mutableStateOf("") }

    val categories = if (type == "Доход") state.incomeCategories else
state.expenseCategories

    // Преобразуем текст в LocalDate
    val selectedDate = remember(dateText) {
        try {
            LocalDate.parse(dateText)
        } catch (e: DateTimeParseException) {
            null
        }
    }

    AlertDialog(
        onDismissRequest = {
            viewModel.hideOperationDialog()
            amountError = ""
            categoryError = ""
            dateError = ""
        },
        title = { Text(if (viewModel.editingOperation == null) "Новая
операция" else "Редактирование операции") },
        text = {
            Column {
                // Выбор типа операции (Доход/Расход)
                Row(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement = Arrangement.SpaceEvenly
                ) {
                    Row(
                        verticalAlignment = Alignment.CenterVertically,
                        modifier = Modifier.clickable {
```

```
                            type = "Доход"
                            amountError = ""
                            categoryError = ""
                        }
                    ) {
                        RadioButton(
                            selected = type == "Доход",
                            colors =
RadioButtonDefaults.colors(AppColors.GreenColor),
                            onClick = {
                                type = "Доход"
                                amountError = ""
                                categoryError = ""
                            }
                        )
                        Text("Доход")
                    }

                    Row(
                        verticalAlignment = Alignment.CenterVertically,
                        modifier = Modifier.clickable {
                            type = "Расход"
                            amountError = ""
                            categoryError = ""
                        }
                    ) {
                        RadioButton(
                            selected = type == "Расход",
                            colors =
RadioButtonDefaults.colors(AppColors.RedColor),
                            onClick = {
                                type = "Расход"
                                amountError = ""
                                categoryError = ""
                            }
                        )
                        Text("Расход")
                    }
                }

                Spacer(Modifier.height(16.dp))

                // Поле для ввода суммы
                OutlinedTextField(
                    value = amount,
                    onValueChange = {
                        amount = it
                        amountError = ""
                    },
                    label = { Text("Сумма (руб.)") },
                    modifier = Modifier.fillMaxWidth(),
                    isError = amountError.isNotEmpty()
                )

                if (amountError.isNotEmpty()) {
                    Text(
                        text = amountError,
                        color = AppColors.RedColor,
                        style = MaterialTheme.typography.caption,
                        modifier = Modifier.padding(top = 4.dp)
                    )
                }

                Spacer(Modifier.height(16.dp))
```

```
                // Поле для ввода даты с валидацией
                OutlinedTextField(
                    value = dateText,
                    onValueChange = {
                        dateText = it
                        dateError = ""

                        // Проверка формата даты в реальном времени
                        try {
                            LocalDate.parse(it)
                            dateError = ""
                        } catch (e: DateTimeParseException) {
                            if (it.isNotEmpty()) {
                                if (it.length > 5) {
                                    dateError = "Формат даты: ГГГГ-ММ-ДД
(например: ${LocalDate.now()})"
                                }
                            }
                        }
                    },
                    label = { Text("Дата (ГГГГ-ММ-ДД)") },
                    placeholder = { Text("Пример: ${LocalDate.now()}") },
                    modifier = Modifier.fillMaxWidth(),
                    isError = dateError.isNotEmpty()
                )

                if (dateError.isNotEmpty()) {
                    Text(
                        text = dateError,
                        color = AppColors.RedColor,
                        style = MaterialTheme.typography.caption,
                        modifier = Modifier.padding(top = 4.dp)
                    )
                }

                // Быстрые кнопки для выбора стандартных дат
                Row(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement = Arrangement.SpaceBetween
                ) {
                    listOf("Сегодня", "Вчера", "Завтра").forEach { label ->
                        Button(
                            onClick = {
                                val newDate = when (label) {
                                    "Сегодня" -> LocalDate.now()
                                    "Вчера" -> LocalDate.now().minusDays(1)
                                    "Завтра" -> LocalDate.now().plusDays(1)
                                    else -> LocalDate.now()
                                }
                                dateText = newDate.toString()
                                dateError = ""
                            },
                            modifier =
Modifier.weight(1f).padding(horizontal = 2.dp),
                            colors = if (selectedDate?.let {
                                    when (label) {
                                        "Сегодня" -> it == LocalDate.now()
                                        "Вчера" -> it ==
LocalDate.now().minusDays(1)
                                        "Завтра" -> it ==
LocalDate.now().plusDays(1)
                                        else -> false
                                    }
```

```
                                } == true) {
                                ButtonDefaults.buttonColors(backgroundColor
= AppColors.BlueColor)
                            } else {
                                ButtonDefaults.buttonColors(backgroundColor
= AppColors.LightGreyColor)
                            }
                        ) {
                            Text(
                                label,
                                color = if (selectedDate?.let {
                                        when (label) {
                                            "Сегодня" -> it ==
LocalDate.now()
                                            "Вчера" -> it ==
LocalDate.now().minusDays(1)
                                            "Завтра" -> it ==
LocalDate.now().plusDays(1)
                                            else -> false
                                        }
                                    } == true) {
                                    AppColors.LightColor
                                } else {
                                    AppColors.LightColor
                                }
                            )
                        }
                    }
                }

                Spacer(Modifier.height(8.dp))

                // Поле для ввода описания
                OutlinedTextField(
                    value = name,
                    onValueChange = {
                        name = it
                    },
                    label = { Text("Описание (необязательно)") },
                    modifier = Modifier.fillMaxWidth()
                )

                Spacer(Modifier.height(16.dp))

                // Выбор категории из списка
                if (categories.isNotEmpty()) {
                    Column {
                        Text("Категория:", style =
MaterialTheme.typography.caption)
                        Spacer(Modifier.height(4.dp))

                        // Прокручиваемый список категорий
                        LazyColumn(
                            modifier = Modifier.heightIn(max = 200.dp),
                            verticalArrangement = Arrangement.spacedBy(4.dp)
                        ) {
                            items(categories) { item ->
                                Button(
                                    onClick = {
                                        category = item
                                        categoryError = ""
                                    },
                                    modifier = Modifier.fillMaxWidth(),
                                    colors = if (category == item)
```

```
ButtonDefaults.buttonColors(backgroundColor = AppColors.BlueColor)
                                    else

ButtonDefaults.buttonColors(backgroundColor = AppColors.LightGreyColor)
                                ) {
                                    Text(
                                        item,
                                        color = if (category == item)
                                            AppColors.LightColor
                                        else
                                            AppColors.LightColor,
                                        maxLines = 1,
                                        overflow = TextOverflow.Ellipsis
                                    )
                                }
                            }
                        }

                        if (categoryError.isNotEmpty()) {
                            Text(
                                text = categoryError,
                                color = AppColors.RedColor,
                                style = MaterialTheme.typography.caption,
                                modifier = Modifier.padding(top = 4.dp)
                            )
                        }
                    }
                } else {
                    // Если категорий нет, предлагаем добавить новую
                    Column {
                        Text(
                            text = "Нет доступных категорий",
                            color = AppColors.RedColor,
                            style = MaterialTheme.typography.caption
                        )
                        Spacer(Modifier.height(8.dp))
                        Button(
                            onClick = {
                                viewModel.showAddCategory(type == "Доход")
                            },
                            modifier = Modifier.fillMaxWidth()
                        ) {
                            Text("Добавить категорию ${if (type == "Доход")
"доходов" else "расходов"}")
                        }
                    }
                }
            }
        },
        confirmButton = {
            Button(
                colors = ButtonDefaults.buttonColors(backgroundColor =
AppColors.BlueColor),
                onClick = {
                    // Валидация суммы
                    val sum = amount.toDoubleOrNull()
                    if (sum == null || sum <= 0) {
                        amountError = "Введите корректную сумму"
                        return@Button
                    }

                    // Валидация категории
                    if (category.isEmpty()) {
```

```
                        categoryError = "Выберите категорию"
                        return@Button
                    }

                    // Валидация даты
                    val parsedDate = try {
                        LocalDate.parse(dateText)
                    } catch (e: DateTimeParseException) {
                        dateError = "Неверный формат даты. Используйте ГГГГ-
ММ-ДД"
                        return@Button
                    }

                    // Дополнительная проверка: дата не должна быть слишком
далеко в будущем
                    if (parsedDate.isAfter(LocalDate.now().plusDays(1))) {
                        dateError = "Дата не может быть более чем на 1 день
в будущем"
                        return@Button
                    }

                    val op = Operation(
                        id = viewModel.editingOperation?.id ?: 0,
                        type = type,
                        amount = sum,
                        category = category,
                        date = parsedDate,
                        name = name
                    )

                    viewModel.saveOperation(op)
                },
                enabled = amount.isNotEmpty() && category.isNotEmpty() &&
dateText.isNotEmpty() && selectedDate != null
            ) {
                Text("Сохранить", color = AppColors.LightColor)
            }
        },
        dismissButton = {
            TextButton(
                colors = ButtonDefaults.buttonColors(backgroundColor =
AppColors.RedColor),
                onClick = {
                    viewModel.hideOperationDialog()
                    amountError = ""
                    categoryError = ""
                    dateError = ""
                }
            ) {
                Text("Отмена", color = AppColors.LightColor)
            }
        }
    )
}
```

Листинг 10 – AddCategoryDialog.kt – Диалог добавления категорий

```
package ui.components

import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
```

```
import androidx.compose.ui.unit.dp
import ui.theme.AppColors
import viewmodel.FinanceViewModel

/**
 * Диалоговое окно для добавления новой категории (доходов или расходов).
 * Позволяет пользователю ввести название новой категории.
 *
 * @param viewModel ViewModel для управления категориями
 */
@Composable
fun AddCategoryDialog(viewModel: FinanceViewModel) {
    var categoryName by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    // Определяем тип категории для отображения в заголовке
    val categoryType = if (viewModel.isIncomeCategory) "доходов" else
"расходов"

    AlertDialog(
        onDismissRequest = {
            viewModel.hideCategoryDialog()
            categoryName = ""
            error = ""
        },
        title = { Text("Добавить категорию $categoryType") },
        text = {
            Column {
                OutlinedTextField(
                    value = categoryName,
                    onValueChange = {
                        categoryName = it
                        error = ""
                    },
                    label = { Text("Название категории") },
                    modifier = Modifier.fillMaxWidth(),
                    singleLine = true,
                    isError = error.isNotEmpty()
                )

                if (error.isNotEmpty()) {
                    Text(
                        text = error,
                        color = AppColors.RedColor,
                        style = MaterialTheme.typography.caption,
                        modifier = Modifier.padding(top = 4.dp)
                    )
                }
            }
        },
        confirmButton = {
            Button(
                colors = ButtonDefaults.buttonColors(backgroundColor =
AppColors.BlueColor),
                onClick = {
                    // Валидация: проверяем, что название не пустое
                    if (categoryName.trim().isEmpty()) {
                        error = "Введите название категории"
                        return@Button
                    }

                    // Сохраняем категорию через ViewModel
                    viewModel.saveCategory(categoryName.trim())
                    categoryName = ""
```

```
                },
                enabled = categoryName.isNotEmpty()
            ) {
                Text("Добавить", color = AppColors.LightColor)
            }
        },
        dismissButton = {
            TextButton(
                colors = ButtonDefaults.buttonColors(backgroundColor =
AppColors.RedColor),
                onClick = {
                    viewModel.hideCategoryDialog()
                    categoryName = ""
                    error = ""
                }
            ) {
                Text("Отмена", color = AppColors.LightColor)
            }
        }
    )
}
```

Состояние и управление:

Листинг 11 – FinanceState.kt – Состояние приложения

```
package viewmodel

import models.Operation

/**
 * Класс состояния приложения, содержащий все данные, необходимые для
отображения UI.
 * Состояние является иммутабельным (immutable) и обновляется через Flow в
архитектуре MVVM.
 *
 * @property operations Список всех финансовых операций (доходы и расходы)
 * @property totalIncome Общая сумма всех доходов за выбранный период
 * @property totalExpenses Общая сумма всех расходов за выбранный период
 * @property balance Текущий баланс (доходы минус расходы)
 * @property expensesByCategory Распределение расходов по категориям для
построения диаграмм
 * @property incomeCategories Список доступных категорий доходов
 * @property expenseCategories Список доступных категорий расходов
 * @property editingCategory Категория, находящаяся в процессе
редактирования (в текущей реализации не используется)
 */
data class FinanceState(
    val operations: List<Operation> = emptyList(),
    val totalIncome: Double = 0.0,
    val totalExpenses: Double = 0.0,
    val balance: Double = 0.0,
    val expensesByCategory: Map<String, Double> = emptyMap(),
    val incomeCategories: List<String> = emptyList(),
    val expenseCategories: List<String> = emptyList(),
    val editingCategory: String? = null
)
```

Листинг 12 – FinanceViewModel.kt – ViewModel приложения

```
package viewmodel
```

```
import androidx.compose.runtime.*
import java.text.DecimalFormat
import java.text.DecimalFormatSymbols
import java.util.Locale
import kotlin.math.roundToInt
import kotlin.math.pow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import models.Operation
import repository.FinanceRepository

/**
 * ViewModel для управления состоянием приложения и бизнес-логикой.
 * Следует архитектуре MVVM, обеспечивает реактивное обновление UI.
 */
class FinanceViewModel(private val repo: FinanceRepository) {
    // Создаем DecimalFormat с явным указанием локали для корректного
парсинга
    private val decimalFormat = DecimalFormat("#.##",
DecimalFormatSymbols(Locale.US))

    // Хранит текущее состояние приложения
    private val _state = MutableStateFlow(FinanceState())
    val state: StateFlow<FinanceState> = _state.asStateFlow()

    // Текущий экран приложения
    var currentScreen by mutableStateOf("Overview")
        private set

    // Флаги для отображения диалогов
    var showOperationDialog by mutableStateOf(false)
        private set
    var showAddCategoryDialog by mutableStateOf(false)
        private set

    var editingOperation by mutableStateOf<Operation?>(null)
        private set

    var isIncomeCategory by mutableStateOf(true)
        private set

    // Инициализация данных при создании
    init {
        refresh()
    }

    /**
     * Пересчитывает статистику на основе списка операций.
     *
     * @param operations Список операций для анализа
     */
    private fun calculateStatistics(operations: List<Operation>) {
        // Суммируем доходы и расходы
        val income = operations.filter { it.type == "Доход" }.sumOf {
it.amount }
        val expenses = operations.filter { it.type == "Расход" }.sumOf {
it.amount }

        // Группируем расходы по категориям для диаграммы
        val expenseOperations = operations.filter { it.type == "Расход" }
        val expByCat = expenseOperations
            .groupBy { it.category }
```

```
            .mapValues { entry ->
                val sum = entry.value.sumOf { op -> op.amount }
                sum
            }

        // Подробная отладочная информация
        println("\n=== CALCULATE STATISTICS ===")
        println("Всего операций: ${operations.size}")
        println("Операций доходов: ${operations.count { it.type == "Доход"
}}")
        println("Операций расходов: ${expenseOperations.size}")
        println("Общая сумма доходов: $income руб.")
        println("Общая сумма расходов: $expenses руб.")
        println("Баланс: ${income - expenses} руб.")

        if (expByCat.isNotEmpty()) {
            println("Детализация расходов по категориям:")
            expByCat.forEach { (category, amount) ->
                println("  - $category: $amount руб.")
            }
        } else {
            println("Расходы по категориям: нет данных")
        }
        println("============================\n")

        _state.value = _state.value.copy(
            operations = operations,
            totalIncome = income,
            totalExpenses = expenses,
            balance = income - expenses,
            expensesByCategory = expByCat,
            incomeCategories = repo.getIncomeCategories(),
            expenseCategories = repo.getExpenseCategories()
        )
    }

    /**
     * Обновляет данные из репозитория и пересчитывает статистику.
     * Вызывается при изменении данных.
     */
    fun refresh() {
        println("Обновление данных...")
        val allOps = repo.getAllOperations()
        println("Операций загружено: ${allOps.size}")
        calculateStatistics(allOps)
    }

    /**
     * Открывает диалог добавления новой операции.
     * Сбрасывает editingOperation в null для режима создания.
     */
    fun showAddOperation() {
        println("Показать диалог добавления операции")
        editingOperation = null
        showOperationDialog = true
    }

    /**
     * Открывает диалог редактирования существующей операции.
     *
     * @param op Операция для редактирования
     */
    fun showEditOperation(op: Operation) {
        println("Показать диалог редактирования операции: $op")
```

```
        editingOperation = op
        showOperationDialog = true
    }

    /**
     * Закрывает диалог операции и сбрасывает связанные состояния.
     */
    fun hideOperationDialog() {
        println("Скрыть диалог операции")
        showOperationDialog = false
        editingOperation = null
    }

    /**
     * Сохраняет операцию: добавляет новую или обновляет существующую.
     * Автоматически обновляет данные приложения после сохранения.
     *
     * @param op Операция для сохранения (при id=0 - добавление, иначе -
обновление)
     */
    fun saveOperation(op: Operation) {
        println("Сохранение операции: $op")
        try {
            if (op.id == 0) {
                repo.addOperation(op)
            } else {
                repo.updateOperation(op)
            }
            refresh()
            hideOperationDialog()
        } catch (e: Exception) {
            println("Ошибка при сохранении операции: ${e.message}")
        }
    }

    /**
     * Удаляет операцию из базы данных.
     *
     * @param op Операция для удаления
     */
    fun deleteOperation(op: Operation) {
        println("Удаление операции: $op")
        try {
            repo.deleteOperation(op.id, op.type)
            refresh()
        } catch (e: Exception) {
            println("Ошибка при удалении операции: ${e.message}")
        }
    }

    /**
     * Открывает диалог добавления новой категории.
     *
     * @param isIncome Тип добавляемой категории (true - доходы, false -
расходы)
     */
    fun showAddCategory(isIncome: Boolean) {
        println("Показать диалог добавления категории. Тип: ${if (isIncome)
"доходы" else "расходы"}")
        isIncomeCategory = isIncome
        showAddCategoryDialog = true
    }

    /**
```

```
     * Закрывает диалог добавления категории.
     */
    fun hideCategoryDialog() {
        println("Скрыть диалог категории")
        showAddCategoryDialog = false
    }

    /**
     * Сохраняет новую категорию в базу данных.
     * Проверяет уникальность имени категории перед добавлением.
     *
     * @param name Название новой категории
     */
    fun saveCategory(name: String) {
        println("Сохранение категории: '$name'. Тип: ${if (isIncomeCategory)
"доходы" else "расходы"}")
        try {
            if (name.isBlank()) {
                println("Имя категории пустое")
                return
            }

            val trimmedName = name.trim()

            // Проверяем уникальность категории
            val existingCategories = if (isIncomeCategory)
                state.value.incomeCategories
            else
                state.value.expenseCategories

            if (existingCategories.any { it.equals(trimmedName, ignoreCase =
true) }) {
                println("Категория '$trimmedName' уже существует")
                return
            }

            val success = repo.addCategory(trimmedName, isIncomeCategory)
            println("Категория добавлена. Успех: $success")

            if (success) {
                refresh()
                hideCategoryDialog()
            } else {
                println("Не удалось добавить категорию")
            }
        } catch (e: Exception) {
            println("Ошибка при добавлении категории: ${e.message}")
        }
    }

    /**
     * Навигация между экранами приложения.
     *
     * @param screen Название целевого экрана ("Overview", "History",
"Settings")
     */
    fun navigateTo(screen: String) {
        currentScreen = screen
    }

    /**
     * Удаляет категорию, если с ней не связано ни одной операции.
     *
     * @param name Название категории для удаления
```

```
     * @param isIncome Тип категории (true - доходы, false - расходы)
     */
    fun deleteCategory(name: String, isIncome: Boolean) {
        println("Удаление категории: '$name'. Тип: ${if (isIncome) "доходы"
else "расходы"}")
        try {
            // Проверяем, есть ли операции с этой категорией
            val hasOperations = if (isIncome) {
                state.value.operations.any { it.type == "Доход" &&
it.category == name }
            } else {
                state.value.operations.any { it.type == "Расход" &&
it.category == name }
            }

            if (hasOperations) {
                println("Нельзя удалить категорию '$name', так как есть
операции с этой категорией")
                return
            }

            repo.deleteCategory(name, isIncome)
            refresh()
        } catch (e: Exception) {
            println("Ошибка при удалении категории: ${e.message}")
            e.printStackTrace()
        }
    }

    private fun Double.roundTo(decimals: Int): Double {
        val multiplier = 10.0.pow(decimals)
        return (this * multiplier).roundToInt() / multiplier
    }
}

// Добавляем функцию roundToDouble как extension
fun Double.roundToDouble(): Double = kotlin.math.round(this)
```

Ресурсы и визуальные элементы:

Листинг 13 – AppColors.kt – Цветовая палитра приложения

```
package ui.theme

import androidx.compose.ui.graphics.Color

/**
 * Объект, содержащий цветовую палитру приложения Finlytics.
 * Определяет цвета для темной темы приложения.
 */
object AppColors {
    // Основной темный цвет фона приложения
    val DarkColor = Color(0xFF1E1E1E)

    // Темно-серый цвет для вторичных элементов
    val DarkGreyColor = Color(0xFF2C2C2C)

    // Светло-серый цвет для неактивных элементов
    val LightGreyColor = Color(0xFF444444)

    // Основной светлый цвет текста
    val LightColor = Color(0xFFF4F4F4)
```

```
    // Основной синий цвет для акцентных элементов и кнопок
    val BlueColor = Color(0xFF2176FF)

    // Красный цвет для ошибок и негативных действий
    val RedColor = Color(0xFFEF4444)

    // Зеленый цвет для доходов и позитивных действий
    val GreenColor = Color(0xFF4ADE80)

    // Желтый цвет для предупреждений и баланса
    val YellowColor = Color(0xFFECB324)

    /**
     * Палитра цветов для категорий доходов в диаграммах.
     * 8 оттенков сине-зеленой гаммы.
     */
    val IncomeColor1 = Color(0xFF1565C0)
    val IncomeColor2 = Color(0xFF42A5F5)
    val IncomeColor3 = Color(0xFF29B6F6)
    val IncomeColor4 = Color(0xFF26A69A)
    val IncomeColor5 = Color(0xFF66BB6A)
    val IncomeColor6 = Color(0xFF9CCC65)
    val IncomeColor7 = Color(0xFFD4E157)
    val IncomeColor8 = Color(0xFFFFCA28)

    /**
     * Палитра цветов для категорий расходов в диаграммах.
     * 8 оттенков красно-фиолетовой гаммы.
     */
    val ExpensesColor1 = Color(0xFFD32F2F)
    val ExpensesColor2 = Color(0xFFFF5722)
    val ExpensesColor3 = Color(0xFFFF9800)
    val ExpensesColor4 = Color(0xFFFFB74D)
    val ExpensesColor5 = Color(0xFFF06292)
    val ExpensesColor6 = Color(0xFFBA68C8)
    val ExpensesColor7 = Color(0xFF7E57C2)
    val ExpensesColor8 = Color(0xFF5C6BC0)
}
```

Листинг 14 – FinlyticsIconPack.kt – Контейнер векторных иконок

```
package ui.theme.icons

import androidx.compose.ui.graphics.vector.ImageVector
import ui.theme.icons.finlyticsiconpack.Add
import ui.theme.icons.finlyticsiconpack.Close
import ui.theme.icons.finlyticsiconpack.Date
import ui.theme.icons.finlyticsiconpack.Delete
import ui.theme.icons.finlyticsiconpack.Edit
import ui.theme.icons.finlyticsiconpack.Expenses
import ui.theme.icons.finlyticsiconpack.History
import ui.theme.icons.finlyticsiconpack.Income
import ui.theme.icons.finlyticsiconpack.Left
import ui.theme.icons.finlyticsiconpack.Minus
import ui.theme.icons.finlyticsiconpack.Plus
import ui.theme.icons.finlyticsiconpack.Right
import ui.theme.icons.finlyticsiconpack.Settings
import ui.theme.icons.finlyticsiconpack.Statistic
import ui.theme.icons.finlyticsiconpack.Wallet
import kotlin.collections.List as ____KtList

/**
```

```
 * Объект-контейнер для пользовательских векторных иконок приложения
Finlytics.
 * Предоставляет доступ ко всем иконкам, используемым в пользовательском
интерфейсе.
 */
public object FinlyticsIconPack

/**
 * Список всех доступных векторных иконок в пакете.
 */
private var __AllIcons: ____KtList<ImageVector>? = null

public val FinlyticsIconPack.AllIcons: ____KtList<ImageVector>
  get() {
    if (__AllIcons != null) {
      return __AllIcons!!
    }
    __AllIcons= listOf(Add, Close, Date, Delete, Edit, Expenses, History,
Income, Left, Minus, Plus,
        Right, Settings, Statistic, Wallet)
    return __AllIcons!!
  }
```

Листинг 15 – Add.kt – Иконка "Добавить"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.StrokeJoin.Companion.Miter
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Добавить".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 28x28 dp.
 */
public val FinlyticsIconPack.Add: ImageVector
    get() {
        if (_add != null) {
            return _add!!
        }
        _add = Builder(name = "Add", defaultWidth = 28.0.dp, defaultHeight =
28.0.dp, viewportWidth
                = 28.0f, viewportHeight = 28.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 3.0f, strokeLineCap = Round,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(14.0f, 4.9f)
                verticalLineTo(23.1f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
```

```
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 3.0f, strokeLineCap = Round,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(4.9f, 14.0f)
                lineTo(23.1f, 14.0f)
            }
        }
        .build()
        return _add!!
    }

private var _add: ImageVector? = null
```

Листинг 16 – Close.kt – Иконка "Закрыть"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Закрыть".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 19x19 dp.
 */
public val FinlyticsIconPack.Close: ImageVector
    get() {
        if (_close != null) {
            return _close!!
        }
        _close = Builder(name = "Close", defaultWidth = 19.0.dp,
defaultHeight = 19.0.dp,
                viewportWidth = 19.0f, viewportHeight = 19.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 3.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(1.5f, 1.5f)
                lineTo(17.333f, 17.333f)
                moveTo(1.5f, 17.333f)
                lineTo(17.333f, 1.5f)
            }
        }
        .build()
        return _close!!
    }

private var _close: ImageVector? = null
```

Листинг 17 – Date.kt – Иконка "Дата"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Дата".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 18x20 dp.
 */
public val FinlyticsIconPack.Date: ImageVector
    get() {
        if (_date != null) {
            return _date!!
        }
        _date = Builder(name = "Date", defaultWidth = 18.0.dp, defaultHeight
= 20.0.dp,
                viewportWidth = 18.0f, viewportHeight = 20.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(1.0f, 7.0f)
                horizontalLineTo(17.0f)
                moveTo(1.0f, 7.0f)
                verticalLineTo(15.8f)
                curveTo(1.0f, 16.92f, 1.0f, 17.48f, 1.218f, 17.908f)
                curveTo(1.41f, 18.284f, 1.715f, 18.59f, 2.092f, 18.782f)
                curveTo(2.519f, 19.0f, 3.079f, 19.0f, 4.197f, 19.0f)
                horizontalLineTo(13.803f)
                curveTo(14.921f, 19.0f, 15.48f, 19.0f, 15.907f, 18.782f)
                curveTo(16.284f, 18.59f, 16.59f, 18.284f, 16.782f, 17.908f)
                curveTo(17.0f, 17.48f, 17.0f, 16.921f, 17.0f, 15.804f)
                verticalLineTo(7.0f)
                moveTo(1.0f, 7.0f)
                verticalLineTo(6.2f)
                curveTo(1.0f, 5.08f, 1.0f, 4.52f, 1.218f, 4.092f)
                curveTo(1.41f, 3.715f, 1.715f, 3.41f, 2.092f, 3.218f)
                curveTo(2.52f, 3.0f, 3.08f, 3.0f, 4.2f, 3.0f)
                horizontalLineTo(5.0f)
                moveTo(17.0f, 7.0f)
                verticalLineTo(6.197f)
                curveTo(17.0f, 5.079f, 17.0f, 4.519f, 16.782f, 4.092f)
                curveTo(16.59f, 3.715f, 16.284f, 3.41f, 15.907f, 3.218f)
                curveTo(15.48f, 3.0f, 14.92f, 3.0f, 13.8f, 3.0f)
                horizontalLineTo(13.0f)
                moveTo(13.0f, 1.0f)
                verticalLineTo(3.0f)
                moveTo(13.0f, 3.0f)
                horizontalLineTo(5.0f)
                moveTo(5.0f, 1.0f)
                verticalLineTo(3.0f)
            }
        }
```

```
        .build()
        return _date!!
    }

private var _date: ImageVector? = null
```

Листинг 18 – Delete.kt – Иконка "Удалить"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Удалить".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 28x28 dp.
 */
public val FinlyticsIconPack.Delete: ImageVector
    get() {
        if (_delete != null) {
            return _delete!!
        }
        _delete = Builder(name = "Delete", defaultWidth = 28.0.dp,
defaultHeight = 28.0.dp,
                viewportWidth = 28.0f, viewportHeight = 28.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(8.587f, 9.333f)
                verticalLineTo(20.4f)
                curveTo(8.587f, 21.505f, 9.482f, 22.4f, 10.587f, 22.4f)
                horizontalLineTo(17.787f)
                curveTo(18.891f, 22.4f, 19.787f, 21.505f, 19.787f, 20.4f)
                verticalLineTo(9.333f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(6.72f, 9.333f)
                horizontalLineTo(21.653f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
```

```
pathFillType = NonZero) {
                moveTo(9.52f, 9.333f)
                lineTo(11.387f, 5.6f)
                horizontalLineTo(16.987f)
                lineTo(18.853f, 9.333f)
            }
        }
        .build()
        return _delete!!
    }

private var _delete: ImageVector? = null
```

Листинг 19 – Edit.kt – Иконка "Редактировать"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Butt
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.StrokeJoin.Companion.Miter
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Редактировать".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 28x28 dp.
 */
public val FinlyticsIconPack.Edit: ImageVector
    get() {
        if (_edit != null) {
            return _edit!!
        }
        _edit = Builder(name = "Edit", defaultWidth = 28.0.dp, defaultHeight
= 28.0.dp,
                viewportWidth = 28.0f, viewportHeight = 28.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(21.784f, 14.7f)
                verticalLineTo(17.512f)
                curveTo(21.784f, 20.274f, 19.545f, 22.512f, 16.784f,
22.512f)
                horizontalLineTo(11.16f)
                curveTo(8.399f, 22.512f, 6.16f, 20.274f, 6.16f, 17.512f)
                verticalLineTo(11.888f)
                curveTo(6.16f, 9.127f, 8.399f, 6.888f, 11.16f, 6.888f)
                horizontalLineTo(13.972f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Butt,
```

```
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(18.275f, 6.455f)
                curveTo(19.055f, 5.674f, 20.321f, 5.673f, 21.102f, 6.454f)
                lineTo(22.024f, 7.376f)
                curveTo(22.798f, 8.15f, 22.806f, 9.404f, 22.042f, 10.188f)
                lineTo(16.212f, 16.169f)
                curveTo(15.648f, 16.748f, 14.874f, 17.075f, 14.065f,
17.075f)
                lineTo(12.965f, 17.075f)
                curveTo(12.113f, 17.075f, 11.432f, 16.364f, 11.467f,
15.511f)
                lineTo(11.517f, 14.343f)
                curveTo(11.548f, 13.591f, 11.861f, 12.879f, 12.392f,
12.348f)
                lineTo(18.275f, 6.455f)
                close()
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(17.209f, 7.642f)
                lineTo(20.694f, 11.126f)
            }
        }
        .build()
        return _edit!!
    }

private var _edit: ImageVector? = null
```

Листинг 20 – Expenses.kt – Иконка "Расходы"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Расходы".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 24x16 dp.
 */
public val FinlyticsIconPack.Expenses: ImageVector
    get() {
        if (_expenses != null) {
            return _expenses!!
        }
        _expenses = Builder(name = "Expenses", defaultWidth = 24.0.dp,
defaultHeight = 16.0.dp,
                viewportWidth = 24.0f, viewportHeight = 16.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
```

```
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(23.0f, 15.0f)
                lineTo(14.962f, 6.688f)
                curveTo(14.817f, 6.538f, 14.744f, 6.463f, 14.68f, 6.405f)
                curveTo(13.636f, 5.453f, 12.057f, 5.453f, 11.013f, 6.404f)
                curveTo(10.948f, 6.463f, 10.875f, 6.538f, 10.73f, 6.688f)
                curveTo(10.586f, 6.837f, 10.514f, 6.912f, 10.449f, 6.971f)
                curveTo(9.405f, 7.922f, 7.825f, 7.922f, 6.781f, 6.971f)
                curveTo(6.717f, 6.912f, 6.645f, 6.837f, 6.501f, 6.689f)
                lineTo(1.0f, 1.0f)
                moveTo(23.0f, 15.0f)
                lineTo(22.999f, 6.6f)
                moveTo(23.0f, 15.0f)
                horizontalLineTo(14.75f)
            }
        }
        .build()
        return _expenses!!
    }

private var _expenses: ImageVector? = null
```

Листинг 21 – History.kt – Иконка "История"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Butt
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "История".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 28x28 dp.
 */
public val FinlyticsIconPack.History: ImageVector
    get() {
        if (_history != null) {
            return _history!!
        }
        _history = Builder(name = "History", defaultWidth = 28.0.dp,
defaultHeight = 28.0.dp,
                viewportWidth = 28.0f, viewportHeight = 28.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
```

```
                moveTo(7.833f, 24.5f)
                curveTo(6.729f, 24.5f, 5.833f, 23.605f, 5.833f, 22.5f)
                verticalLineTo(3.5f)
                horizontalLineTo(16.333f)
                lineTo(22.167f, 9.333f)
                verticalLineTo(22.5f)
                curveTo(22.167f, 23.605f, 21.271f, 24.5f, 20.167f, 24.5f)
                horizontalLineTo(7.833f)
                close()
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Butt,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(15.167f, 3.5f)
                verticalLineTo(10.5f)
                horizontalLineTo(22.167f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(10.5f, 15.167f)
                horizontalLineTo(17.5f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(10.5f, 19.833f)
                horizontalLineTo(17.5f)
            }
        }
        .build()
        return _history!!
    }

private var _history: ImageVector? = null
```

Листинг 22 – Income.kt – Иконка "Доходы"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Доходы".
 * Используется в пользовательском интерфейсе приложения Finlytics.
```

```
 * Размер по умолчанию: 24x16 dp.
 */
public val FinlyticsIconPack.Income: ImageVector
    get() {
        if (_income != null) {
            return _income!!
        }
        _income = Builder(name = "Income", defaultWidth = 24.0.dp,
defaultHeight = 16.0.dp,
                viewportWidth = 24.0f, viewportHeight = 16.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(23.0f, 1.0f)
                lineTo(14.962f, 9.313f)
                curveTo(14.817f, 9.462f, 14.744f, 9.537f, 14.68f, 9.595f)
                curveTo(13.636f, 10.547f, 12.057f, 10.547f, 11.013f, 9.596f)
                curveTo(10.948f, 9.537f, 10.875f, 9.462f, 10.73f, 9.312f)
                curveTo(10.586f, 9.163f, 10.514f, 9.088f, 10.449f, 9.029f)
                curveTo(9.405f, 8.078f, 7.825f, 8.078f, 6.781f, 9.029f)
                curveTo(6.717f, 9.088f, 6.645f, 9.163f, 6.501f, 9.311f)
                lineTo(1.0f, 15.0f)
                moveTo(23.0f, 1.0f)
                lineTo(22.999f, 9.4f)
                moveTo(23.0f, 1.0f)
                horizontalLineTo(14.75f)
            }
        }
        .build()
        return _income!!
    }

private var _income: ImageVector? = null
```

Листинг 23 – Left.kt – Иконка "Влево"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Влево".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 20x20 dp.
 */
public val FinlyticsIconPack.Left: ImageVector
    get() {
        if (_left != null) {
            return _left!!
```

```
        }
        _left = Builder(name = "Left", defaultWidth = 20.0.dp, defaultHeight
= 20.0.dp,
                viewportWidth = 20.0f, viewportHeight = 20.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(11.667f, 14.167f)
                lineTo(7.5f, 10.0f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(7.5f, 10.0f)
                lineTo(11.667f, 5.833f)
            }
        }
        .build()
        return _left!!
    }

private var _left: ImageVector? = null
```

Листинг 24 – Minus.kt – Иконка "Минус"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin.Companion.Miter
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Минус".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 17x3 dp.
 */
public val FinlyticsIconPack.Minus: ImageVector
    get() {
        if (_minus != null) {
            return _minus!!
        }
        _minus = Builder(name = "Minus", defaultWidth = 17.0.dp,
defaultHeight = 3.0.dp,
                viewportWidth = 17.0f, viewportHeight = 3.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 3.0f, strokeLineCap = Round,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(1.5f, 1.5f)
```

```
                lineTo(15.5f, 1.5f)
            }
        }
        .build()
        return _minus!!
    }

private var _minus: ImageVector? = null
```

Листинг 25 – Plus.kt – Иконка "Плюс"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.StrokeJoin.Companion.Miter
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Плюс".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 17x17 dp.
 */
public val FinlyticsIconPack.Plus: ImageVector
    get() {
        if (_plus != null) {
            return _plus!!
        }
        _plus = Builder(name = "Plus", defaultWidth = 17.0.dp, defaultHeight
= 17.0.dp,
                viewportWidth = 17.0f, viewportHeight = 17.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 3.0f, strokeLineCap = Round,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(8.5f, 1.5f)
                verticalLineTo(15.5f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 3.0f, strokeLineCap = Round,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(1.5f, 8.5f)
                horizontalLineTo(15.5f)
            }
        }
        .build()
        return _plus!!
    }

private var _plus: ImageVector? = null
```

Листинг 26 – Right.kt – Иконка "Вправо"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Вправо".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 20x20 dp.
 */
public val FinlyticsIconPack.Right: ImageVector
    get() {
        if (_right != null) {
            return _right!!
        }
        _right = Builder(name = "Right", defaultWidth = 20.0.dp,
defaultHeight = 20.0.dp,
                viewportWidth = 20.0f, viewportHeight = 20.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(8.333f, 14.167f)
                lineTo(12.5f, 10.0f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(12.5f, 10.0f)
                lineTo(8.333f, 5.833f)
            }
        }
        .build()
        return _right!!
    }

private var _right: ImageVector? = null
```

Листинг 27 – Settings.kt – Иконка "Настройки"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Butt
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.StrokeJoin.Companion.Miter
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Настройки".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 28x28 dp.
 */
public val FinlyticsIconPack.Settings: ImageVector
    get() {
        if (_settings != null) {
            return _settings!!
        }
        _settings = Builder(name = "Settings", defaultWidth = 28.0.dp,
defaultHeight = 28.0.dp,
                viewportWidth = 28.0f, viewportHeight = 28.0f).apply {
            path(fill = SolidColor(Color(0xFFF4F4F4)), stroke = null,
strokeLineWidth = 0.0f,
                    strokeLineCap = Butt, strokeLineJoin = Miter,
strokeLineMiter = 4.0f,
                    pathFillType = NonZero) {
                moveTo(14.062f, 3.652f)
                lineTo(14.063f, 2.652f)
                horizontalLineTo(14.062f)
                verticalLineTo(3.652f)
                close()
                moveTo(18.444f, 6.512f)
                lineTo(17.944f, 7.377f)
                lineTo(17.944f, 7.378f)
                lineTo(18.444f, 6.512f)
                close()
                moveTo(21.851f, 13.389f)
                lineTo(20.851f, 13.389f)
                verticalLineTo(13.389f)
                horizontalLineTo(21.851f)
                close()
                moveTo(18.443f, 20.264f)
                lineTo(17.943f, 19.397f)
                lineTo(17.943f, 19.397f)
                lineTo(18.443f, 20.264f)
                close()
                moveTo(14.062f, 23.128f)
                lineTo(14.062f, 24.128f)
                lineTo(14.063f, 24.128f)
                lineTo(14.062f, 23.128f)
                close()
                moveTo(9.683f, 20.264f)
                lineTo(10.183f, 19.397f)
                lineTo(10.183f, 19.397f)
                lineTo(9.683f, 20.264f)
                close()
                moveTo(6.274f, 13.389f)
                lineTo(7.274f, 13.389f)
                lineTo(7.274f, 13.389f)
```

```
                lineTo(6.274f, 13.389f)
                close()
                moveTo(9.682f, 6.513f)
                lineTo(10.182f, 7.379f)
                lineTo(10.182f, 7.378f)
                lineTo(9.682f, 6.513f)
                close()
                moveTo(11.06f, 4.655f)
                lineTo(12.034f, 4.885f)
                lineTo(11.06f, 4.655f)
                close()
                moveTo(11.677f, 3.946f)
                lineTo(11.433f, 2.977f)
                lineTo(11.677f, 3.946f)
                close()
                moveTo(7.565f, 19.948f)
                lineTo(7.34f, 18.973f)
                lineTo(7.565f, 19.948f)
                close()
                moveTo(6.656f, 19.709f)
                lineTo(7.417f, 19.06f)
                lineTo(6.656f, 19.709f)
                close()
                moveTo(11.679f, 22.834f)
                lineTo(11.435f, 23.803f)
                lineTo(11.679f, 22.834f)
                close()
                moveTo(11.062f, 22.124f)
                lineTo(12.035f, 21.895f)
                lineTo(11.062f, 22.124f)
                close()
                moveTo(17.062f, 22.123f)
                lineTo(18.036f, 22.352f)
                lineTo(17.062f, 22.123f)
                close()
                moveTo(16.445f, 22.833f)
                lineTo(16.201f, 21.863f)
                lineTo(16.445f, 22.833f)
                close()
                moveTo(21.466f, 19.711f)
                lineTo(20.706f, 19.061f)
                lineTo(21.466f, 19.711f)
                close()
                moveTo(23.25f, 10.165f)
                lineTo(24.194f, 9.834f)
                lineTo(23.25f, 10.165f)
                close()
                moveTo(22.902f, 11.147f)
                lineTo(23.542f, 11.915f)
                lineTo(22.902f, 11.147f)
                close()
                moveTo(21.463f, 7.065f)
                lineTo(20.703f, 7.715f)
                lineTo(21.463f, 7.065f)
                close()
                moveTo(7.568f, 6.827f)
                lineTo(7.343f, 7.802f)
                lineTo(7.568f, 6.827f)
                close()
                moveTo(22.903f, 15.631f)
                lineTo(22.263f, 16.4f)
                lineTo(22.903f, 15.631f)
                close()
                moveTo(17.065f, 4.655f)
```

```
                lineTo(18.038f, 4.425f)
                lineTo(17.065f, 4.655f)
                close()
                moveTo(14.062f, 3.652f)
                lineTo(14.062f, 4.652f)
                curveTo(14.802f, 4.652f, 15.519f, 4.744f, 16.204f, 4.916f)
                lineTo(16.448f, 3.946f)
                lineTo(16.692f, 2.977f)
                curveTo(15.849f, 2.765f, 14.968f, 2.652f, 14.063f, 2.652f)
                lineTo(14.062f, 3.652f)
                close()
                moveTo(17.065f, 4.655f)
                lineTo(16.092f, 4.885f)
                curveTo(16.331f, 5.896f, 16.97f, 6.815f, 17.944f, 7.377f)
                lineTo(18.444f, 6.512f)
                lineTo(18.944f, 5.646f)
                curveTo(18.469f, 5.371f, 18.156f, 4.924f, 18.038f, 4.425f)
                lineTo(17.065f, 4.655f)
                close()
                moveTo(18.444f, 6.512f)
                lineTo(17.944f, 7.378f)
                curveTo(18.838f, 7.894f, 19.851f, 8.015f, 20.78f, 7.801f)
                lineTo(20.556f, 6.827f)
                lineTo(20.332f, 5.852f)
                curveTo(19.873f, 5.958f, 19.381f, 5.898f, 18.944f, 5.646f)
                lineTo(18.444f, 6.512f)
                close()
                moveTo(21.463f, 7.065f)
                lineTo(20.703f, 7.715f)
                curveTo(21.398f, 8.528f, 21.946f, 9.468f, 22.307f, 10.497f)
                lineTo(23.25f, 10.165f)
                lineTo(24.194f, 9.834f)
                curveTo(23.75f, 8.569f, 23.076f, 7.413f, 22.223f, 6.415f)
                lineTo(21.463f, 7.065f)
                close()
                moveTo(22.902f, 11.147f)
                lineTo(22.262f, 10.379f)
                curveTo(21.402f, 11.096f, 20.851f, 12.178f, 20.851f,
13.389f)
                lineTo(21.851f, 13.389f)
                lineTo(22.851f, 13.39f)
                curveTo(22.851f, 12.798f, 23.118f, 12.269f, 23.542f,
11.915f)
                lineTo(22.902f, 11.147f)
                close()
                moveTo(21.851f, 13.389f)
                horizontalLineTo(20.851f)
                curveTo(20.851f, 14.602f, 21.403f, 15.683f, 22.263f, 16.4f)
                lineTo(22.903f, 15.631f)
                lineTo(23.543f, 14.863f)
                curveTo(23.118f, 14.509f, 22.851f, 13.981f, 22.851f,
13.389f)
                horizontalLineTo(21.851f)
                close()
                moveTo(23.251f, 16.612f)
                lineTo(22.308f, 16.281f)
                curveTo(21.948f, 17.309f, 21.4f, 18.249f, 20.706f, 19.061f)
                lineTo(21.466f, 19.711f)
                lineTo(22.227f, 20.36f)
                curveTo(23.079f, 19.363f, 23.752f, 18.208f, 24.195f,
16.943f)
                lineTo(23.251f, 16.612f)
                close()
                moveTo(20.558f, 19.949f)
```

```
                lineTo(20.784f, 18.975f)
                curveTo(19.853f, 18.76f, 18.838f, 18.881f, 17.943f, 19.397f)
                lineTo(18.443f, 20.264f)
                lineTo(18.943f, 21.13f)
                curveTo(19.381f, 20.877f, 19.874f, 20.817f, 20.333f,
20.923f)
                lineTo(20.558f, 19.949f)
                close()
                moveTo(18.443f, 20.264f)
                lineTo(17.943f, 19.397f)
                curveTo(16.968f, 19.961f, 16.328f, 20.881f, 16.089f,
21.894f)
                lineTo(17.062f, 22.123f)
                lineTo(18.036f, 22.352f)
                curveTo(18.153f, 21.852f, 18.467f, 21.405f, 18.943f,
21.129f)
                lineTo(18.443f, 20.264f)
                close()
                moveTo(16.445f, 22.833f)
                lineTo(16.201f, 21.863f)
                curveTo(15.517f, 22.035f, 14.801f, 22.128f, 14.062f,
22.128f)
                lineTo(14.062f, 23.128f)
                lineTo(14.063f, 24.128f)
                curveTo(14.968f, 24.128f, 15.848f, 24.014f, 16.689f,
23.802f)
                lineTo(16.445f, 22.833f)
                close()
                moveTo(14.062f, 23.128f)
                verticalLineTo(22.128f)
                curveTo(13.323f, 22.128f, 12.607f, 22.036f, 11.923f,
21.864f)
                lineTo(11.679f, 22.834f)
                lineTo(11.435f, 23.803f)
                curveTo(12.277f, 24.015f, 13.157f, 24.128f, 14.062f,
24.128f)
                verticalLineTo(23.128f)
                close()
                moveTo(11.062f, 22.124f)
                lineTo(12.035f, 21.895f)
                curveTo(11.798f, 20.882f, 11.159f, 19.961f, 10.183f,
19.397f)
                lineTo(9.683f, 20.264f)
                lineTo(9.182f, 21.129f)
                curveTo(9.659f, 21.404f, 9.971f, 21.852f, 10.088f, 22.352f)
                lineTo(11.062f, 22.124f)
                close()
                moveTo(9.683f, 20.264f)
                lineTo(10.183f, 19.397f)
                curveTo(9.287f, 18.881f, 8.272f, 18.758f, 7.34f, 18.973f)
                lineTo(7.565f, 19.948f)
                lineTo(7.791f, 20.922f)
                curveTo(8.249f, 20.816f, 8.744f, 20.876f, 9.183f, 21.129f)
                lineTo(9.683f, 20.264f)
                close()
                moveTo(6.656f, 19.709f)
                lineTo(7.417f, 19.06f)
                curveTo(6.723f, 18.248f, 6.177f, 17.308f, 5.817f, 16.282f)
                lineTo(4.873f, 16.612f)
                lineTo(3.93f, 16.943f)
                curveTo(4.373f, 18.207f, 5.045f, 19.362f, 5.896f, 20.358f)
                lineTo(6.656f, 19.709f)
                close()
                moveTo(5.222f, 15.631f)
```

```
                lineTo(5.862f, 16.4f)
                curveTo(6.722f, 15.683f, 7.274f, 14.602f, 7.274f, 13.389f)
                horizontalLineTo(6.274f)
                horizontalLineTo(5.274f)
                curveTo(5.274f, 13.981f, 5.007f, 14.509f, 4.582f, 14.863f)
                lineTo(5.222f, 15.631f)
                close()
                moveTo(6.274f, 13.389f)
                lineTo(7.274f, 13.389f)
                curveTo(7.274f, 12.177f, 6.723f, 11.095f, 5.862f, 10.378f)
                lineTo(5.222f, 11.147f)
                lineTo(4.582f, 11.915f)
                curveTo(5.007f, 12.269f, 5.274f, 12.798f, 5.274f, 13.39f)
                lineTo(6.274f, 13.389f)
                close()
                moveTo(4.874f, 10.165f)
                lineTo(5.817f, 10.496f)
                curveTo(6.178f, 9.468f, 6.726f, 8.528f, 7.421f, 7.716f)
                lineTo(6.661f, 7.066f)
                lineTo(5.901f, 6.416f)
                curveTo(5.048f, 7.413f, 4.374f, 8.569f, 3.93f, 9.834f)
                lineTo(4.874f, 10.165f)
                close()
                moveTo(7.568f, 6.827f)
                lineTo(7.343f, 7.802f)
                curveTo(8.273f, 8.016f, 9.287f, 7.895f, 10.182f, 7.379f)
                lineTo(9.682f, 6.513f)
                lineTo(9.182f, 5.646f)
                curveTo(8.745f, 5.899f, 8.252f, 5.959f, 7.793f, 5.853f)
                lineTo(7.568f, 6.827f)
                close()
                moveTo(9.682f, 6.513f)
                lineTo(10.182f, 7.378f)
                curveTo(11.156f, 6.815f, 11.795f, 5.896f, 12.034f, 4.885f)
                lineTo(11.06f, 4.655f)
                lineTo(10.087f, 4.426f)
                curveTo(9.969f, 4.925f, 9.657f, 5.372f, 9.181f, 5.647f)
                lineTo(9.682f, 6.513f)
                close()
                moveTo(11.677f, 3.946f)
                lineTo(11.922f, 4.916f)
                curveTo(12.606f, 4.744f, 13.323f, 4.652f, 14.062f, 4.652f)
                verticalLineTo(3.652f)
                verticalLineTo(2.652f)
                curveTo(13.157f, 2.652f, 12.276f, 2.765f, 11.433f, 2.977f)
                lineTo(11.677f, 3.946f)
                close()
                moveTo(11.06f, 4.655f)
                lineTo(12.034f, 4.885f)
                curveTo(12.038f, 4.869f, 12.042f, 4.856f, 12.046f, 4.848f)
                curveTo(12.05f, 4.839f, 12.052f, 4.839f, 12.047f, 4.844f)
                curveTo(12.042f, 4.85f, 12.03f, 4.863f, 12.008f, 4.878f)
                curveTo(11.985f, 4.894f, 11.955f, 4.908f, 11.922f, 4.916f)
                lineTo(11.677f, 3.946f)
                lineTo(11.433f, 2.977f)
                curveTo(10.619f, 3.182f, 10.216f, 3.879f, 10.087f, 4.426f)
                lineTo(11.06f, 4.655f)
                close()
                moveTo(5.222f, 11.147f)
                lineTo(5.862f, 10.378f)
                curveTo(5.848f, 10.366f, 5.836f, 10.354f, 5.829f, 10.344f)
                curveTo(5.821f, 10.334f, 5.821f, 10.33f, 5.823f, 10.336f)
                curveTo(5.825f, 10.342f, 5.832f, 10.361f, 5.833f, 10.392f)
                curveTo(5.835f, 10.424f, 5.83f, 10.46f, 5.817f, 10.496f)
```

```
                lineTo(4.874f, 10.165f)
                lineTo(3.93f, 9.834f)
                curveTo(3.615f, 10.731f, 4.106f, 11.519f, 4.582f, 11.915f)
                lineTo(5.222f, 11.147f)
                close()
                moveTo(7.565f, 19.948f)
                lineTo(7.34f, 18.973f)
                curveTo(7.324f, 18.977f, 7.31f, 18.979f, 7.301f, 18.979f)
                curveTo(7.291f, 18.979f, 7.29f, 18.978f, 7.297f, 18.979f)
                curveTo(7.304f, 18.981f, 7.321f, 18.986f, 7.345f, 18.999f)
                curveTo(7.368f, 19.013f, 7.394f, 19.033f, 7.417f, 19.06f)
                lineTo(6.656f, 19.709f)
                lineTo(5.896f, 20.358f)
                curveTo(6.44f, 20.996f, 7.243f, 21.049f, 7.791f, 20.922f)
                lineTo(7.565f, 19.948f)
                close()
                moveTo(11.679f, 22.834f)
                lineTo(11.923f, 21.864f)
                curveTo(11.957f, 21.872f, 11.987f, 21.886f, 12.01f, 21.902f)
                curveTo(12.032f, 21.917f, 12.044f, 21.93f, 12.049f, 21.936f)
                curveTo(12.054f, 21.941f, 12.052f, 21.941f, 12.048f,
21.932f)
                curveTo(12.044f, 21.924f, 12.039f, 21.911f, 12.035f,
21.895f)
                lineTo(11.062f, 22.124f)
                lineTo(10.088f, 22.352f)
                curveTo(10.217f, 22.9f, 10.62f, 23.598f, 11.435f, 23.803f)
                lineTo(11.679f, 22.834f)
                close()
                moveTo(17.062f, 22.123f)
                lineTo(16.089f, 21.894f)
                curveTo(16.085f, 21.91f, 16.08f, 21.923f, 16.076f, 21.931f)
                curveTo(16.072f, 21.94f, 16.071f, 21.94f, 16.075f, 21.935f)
                curveTo(16.08f, 21.929f, 16.092f, 21.916f, 16.114f, 21.901f)
                curveTo(16.137f, 21.886f, 16.167f, 21.872f, 16.201f,
21.863f)
                lineTo(16.445f, 22.833f)
                lineTo(16.689f, 23.802f)
                curveTo(17.503f, 23.597f, 17.907f, 22.899f, 18.036f,
22.352f)
                lineTo(17.062f, 22.123f)
                close()
                moveTo(21.466f, 19.711f)
                lineTo(20.706f, 19.061f)
                curveTo(20.729f, 19.035f, 20.754f, 19.014f, 20.778f,
19.001f)
                curveTo(20.801f, 18.987f, 20.819f, 18.982f, 20.826f,
18.981f)
                curveTo(20.833f, 18.979f, 20.832f, 18.98f, 20.822f, 18.98f)
                curveTo(20.813f, 18.98f, 20.8f, 18.979f, 20.784f, 18.975f)
                lineTo(20.558f, 19.949f)
                lineTo(20.333f, 20.923f)
                curveTo(20.88f, 21.05f, 21.683f, 20.997f, 22.227f, 20.36f)
                lineTo(21.466f, 19.711f)
                close()
                moveTo(23.25f, 10.165f)
                lineTo(22.307f, 10.497f)
                curveTo(22.294f, 10.461f, 22.289f, 10.424f, 22.291f,
10.392f)
                curveTo(22.292f, 10.361f, 22.299f, 10.343f, 22.301f,
10.336f)
                curveTo(22.303f, 10.33f, 22.303f, 10.334f, 22.295f, 10.344f)
                curveTo(22.288f, 10.354f, 22.276f, 10.367f, 22.262f,
10.379f)
```

```
                lineTo(22.902f, 11.147f)
                lineTo(23.542f, 11.915f)
                curveTo(24.018f, 11.518f, 24.508f, 10.731f, 24.194f, 9.834f)
                lineTo(23.25f, 10.165f)
                close()
                moveTo(4.873f, 16.612f)
                lineTo(5.817f, 16.282f)
                curveTo(5.83f, 16.318f, 5.834f, 16.354f, 5.833f, 16.386f)
                curveTo(5.832f, 16.417f, 5.825f, 16.436f, 5.823f, 16.442f)
                curveTo(5.821f, 16.448f, 5.821f, 16.444f, 5.829f, 16.434f)
                curveTo(5.836f, 16.424f, 5.847f, 16.412f, 5.862f, 16.4f)
                lineTo(5.222f, 15.631f)
                lineTo(4.582f, 14.863f)
                curveTo(4.106f, 15.259f, 3.615f, 16.046f, 3.93f, 16.943f)
                lineTo(4.873f, 16.612f)
                close()
                moveTo(20.556f, 6.827f)
                lineTo(20.78f, 7.801f)
                curveTo(20.796f, 7.798f, 20.81f, 7.796f, 20.819f, 7.796f)
                curveTo(20.828f, 7.796f, 20.829f, 7.797f, 20.822f, 7.796f)
                curveTo(20.815f, 7.794f, 20.798f, 7.789f, 20.775f, 7.776f)
                curveTo(20.751f, 7.762f, 20.726f, 7.742f, 20.703f, 7.715f)
                lineTo(21.463f, 7.065f)
                lineTo(22.223f, 6.415f)
                curveTo(21.679f, 5.78f, 20.878f, 5.726f, 20.332f, 5.852f)
                lineTo(20.556f, 6.827f)
                close()
                moveTo(6.661f, 7.066f)
                lineTo(7.421f, 7.716f)
                curveTo(7.398f, 7.742f, 7.372f, 7.762f, 7.349f, 7.776f)
                curveTo(7.326f, 7.789f, 7.308f, 7.794f, 7.301f, 7.796f)
                curveTo(7.294f, 7.797f, 7.295f, 7.796f, 7.305f, 7.797f)
                curveTo(7.314f, 7.797f, 7.327f, 7.798f, 7.343f, 7.802f)
                lineTo(7.568f, 6.827f)
                lineTo(7.793f, 5.853f)
                curveTo(7.246f, 5.727f, 6.445f, 5.78f, 5.901f, 6.416f)
                lineTo(6.661f, 7.066f)
                close()
                moveTo(22.903f, 15.631f)
                lineTo(22.263f, 16.4f)
                curveTo(22.278f, 16.412f, 22.289f, 16.424f, 22.296f,
16.434f)
                curveTo(22.304f, 16.444f, 22.304f, 16.448f, 22.302f,
16.442f)
                curveTo(22.3f, 16.436f, 22.293f, 16.417f, 22.292f, 16.386f)
                curveTo(22.291f, 16.354f, 22.295f, 16.317f, 22.308f,
16.281f)
                lineTo(23.251f, 16.612f)
                lineTo(24.195f, 16.943f)
                curveTo(24.51f, 16.046f, 24.019f, 15.259f, 23.543f, 14.863f)
                lineTo(22.903f, 15.631f)
                close()
                moveTo(16.448f, 3.946f)
                lineTo(16.204f, 4.916f)
                curveTo(16.17f, 4.908f, 16.14f, 4.894f, 16.117f, 4.878f)
                curveTo(16.095f, 4.863f, 16.083f, 4.85f, 16.078f, 4.845f)
                curveTo(16.073f, 4.839f, 16.075f, 4.84f, 16.079f, 4.848f)
                curveTo(16.083f, 4.856f, 16.088f, 4.869f, 16.092f, 4.885f)
                lineTo(17.065f, 4.655f)
                lineTo(18.038f, 4.425f)
                curveTo(17.909f, 3.879f, 17.506f, 3.182f, 16.692f, 2.977f)
                lineTo(16.448f, 3.946f)
                close()
            }
```

```
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 2.0f, strokeLineCap = Butt,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(14.0f, 14.0f)
                moveToRelative(3.043f, 0.0f)
                arcToRelative(3.043f, 3.043f, 0.0f, true, false, -6.087f,
0.0f)
                arcToRelative(3.043f, 3.043f, 0.0f, true, false, 6.087f,
0.0f)
            }
        }
        .build()
        return _settings!!
    }

private var _settings: ImageVector? = null
```

Листинг 28 – Statistic.kt – Иконка "Статистика"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Butt
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.StrokeJoin.Companion.Miter
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Статистика".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 28x28 dp.
 */
public val FinlyticsIconPack.Statistic: ImageVector
    get() {
        if (_statistic != null) {
            return _statistic!!
        }
        _statistic = Builder(name = "Statistic", defaultWidth = 28.0.dp,
defaultHeight = 28.0.dp,
                viewportWidth = 28.0f, viewportHeight = 28.0f).apply {
            path(fill = SolidColor(Color(0xFFF4F4F4)), stroke = null,
strokeLineWidth = 0.0f,
                    strokeLineCap = Butt, strokeLineJoin = Miter,
strokeLineMiter = 4.0f,
                    pathFillType = NonZero) {
                moveTo(16.227f, 5.9f)
                curveTo(16.227f, 5.072f, 15.556f, 4.4f, 14.727f, 4.4f)
                curveTo(13.899f, 4.4f, 13.227f, 5.072f, 13.227f, 5.9f)
                horizontalLineTo(14.727f)
                horizontalLineTo(16.227f)
                close()
                moveTo(14.727f, 19.9f)
```

```
                horizontalLineTo(16.227f)
                verticalLineTo(5.9f)
                horizontalLineTo(14.727f)
                horizontalLineTo(13.227f)
                verticalLineTo(19.9f)
                horizontalLineTo(14.727f)
                close()
            }
            path(fill = SolidColor(Color(0xFFF4F4F4)), stroke = null,
strokeLineWidth = 0.0f,
                    strokeLineCap = Butt, strokeLineJoin = Miter,
strokeLineMiter = 4.0f,
                    pathFillType = NonZero) {
                moveTo(22.242f, 10.45f)
                curveTo(22.242f, 9.622f, 21.57f, 8.95f, 20.742f, 8.95f)
                curveTo(19.913f, 8.95f, 19.242f, 9.622f, 19.242f, 10.45f)
                horizontalLineTo(20.742f)
                horizontalLineTo(22.242f)
                close()
                moveTo(20.742f, 19.9f)
                horizontalLineTo(22.242f)
                verticalLineTo(10.45f)
                horizontalLineTo(20.742f)
                horizontalLineTo(19.242f)
                verticalLineTo(19.9f)
                horizontalLineTo(20.742f)
                close()
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFF4F4F4)),
                    strokeLineWidth = 3.0f, strokeLineCap = Round,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(3.5f, 4.5f)
                verticalLineTo(20.05f)
                curveTo(3.5f, 21.707f, 4.843f, 23.05f, 6.5f, 23.05f)
                horizontalLineTo(24.751f)
            }
            path(fill = SolidColor(Color(0xFFF4F4F4)), stroke = null,
strokeLineWidth = 0.0f,
                    strokeLineCap = Butt, strokeLineJoin = Miter,
strokeLineMiter = 4.0f,
                    pathFillType = NonZero) {
                moveTo(10.614f, 15.35f)
                curveTo(10.614f, 14.521f, 9.942f, 13.85f, 9.114f, 13.85f)
                curveTo(8.285f, 13.85f, 7.614f, 14.521f, 7.614f, 15.35f)
                horizontalLineTo(9.114f)
                horizontalLineTo(10.614f)
                close()
                moveTo(9.114f, 19.9f)
                horizontalLineTo(10.614f)
                verticalLineTo(15.35f)
                horizontalLineTo(9.114f)
                horizontalLineTo(7.614f)
                verticalLineTo(19.9f)
                horizontalLineTo(9.114f)
                close()
            }
        }
        .build()
        return _statistic!!
    }

private var  _statistic: ImageVector? = null
```

Листинг 29 – Wallet.kt – Иконка "Кошелек"

```
package ui.theme.icons.finlyticsiconpack

import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.PathFillType
import androidx.compose.ui.graphics.PathFillType.Companion.NonZero
import androidx.compose.ui.graphics.SolidColor
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.StrokeCap.Companion.Butt
import androidx.compose.ui.graphics.StrokeCap.Companion.Round
import androidx.compose.ui.graphics.StrokeJoin
import androidx.compose.ui.graphics.StrokeJoin.Companion.Miter
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.graphics.vector.ImageVector.Builder
import androidx.compose.ui.graphics.vector.path
import androidx.compose.ui.unit.dp
import ui.theme.icons.FinlyticsIconPack

/**
 * Векторное изображение иконки "Кошелек".
 * Используется в пользовательском интерфейсе приложения Finlytics.
 * Размер по умолчанию: 22x22 dp.
 */
public val FinlyticsIconPack.Wallet: ImageVector
    get() {
        if (_wallet != null) {
            return _wallet!!
        }
        _wallet = Builder(name = "Wallet", defaultWidth = 22.0.dp,
defaultHeight = 22.0.dp,
                viewportWidth = 22.0f, viewportHeight = 22.0f).apply {
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFECB324)),
                    strokeLineWidth = 2.0f, strokeLineCap = Butt,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(6.0f, 4.17f)
                lineTo(16.0f, 4.17f)
                arcTo(5.0f, 5.0f, 0.0f, false, true, 21.0f, 9.17f)
                lineTo(21.0f, 15.17f)
                arcTo(5.0f, 5.0f, 0.0f, false, true, 16.0f, 20.17f)
                lineTo(6.0f, 20.17f)
                arcTo(5.0f, 5.0f, 0.0f, false, true, 1.0f, 15.17f)
                lineTo(1.0f, 9.17f)
                arcTo(5.0f, 5.0f, 0.0f, false, true, 6.0f, 4.17f)
                close()
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFECB324)),
                    strokeLineWidth = 2.0f, strokeLineCap = Butt,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(18.0f, 4.67f)
                curveTo(18.0f, 2.346f, 15.868f, 0.609f, 13.592f, 1.077f)
                lineTo(4.992f, 2.848f)
                curveTo(2.668f, 3.326f, 1.0f, 5.372f, 1.0f, 7.745f)
                lineTo(1.0f, 11.17f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFECB324)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
```

```
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(5.0f, 15.67f)
                horizontalLineTo(11.0f)
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFECB324)),
                    strokeLineWidth = 2.0f, strokeLineCap = Butt,
strokeLineJoin = Miter,
                    strokeLineMiter = 4.0f, pathFillType = NonZero) {
                moveTo(14.0f, 12.17f)
                curveTo(14.0f, 10.789f, 15.119f, 9.67f, 16.5f, 9.67f)
                horizontalLineTo(21.0f)
                verticalLineTo(14.67f)
                horizontalLineTo(16.5f)
                curveTo(15.119f, 14.67f, 14.0f, 13.55f, 14.0f, 12.17f)
                close()
            }
            path(fill = SolidColor(Color(0x00000000)), stroke =
SolidColor(Color(0xFFECB324)),
                    strokeLineWidth = 2.0f, strokeLineCap = Round,
strokeLineJoin =
                    StrokeJoin.Companion.Round, strokeLineMiter = 4.0f,
pathFillType = NonZero) {
                moveTo(16.5f, 12.17f)
                horizontalLineTo(16.7f)
            }
        }
        .build()
        return _wallet!!
    }

private var _wallet: ImageVector? = null
```

3.2.2 Backend

В контексте офлайн-приложения "Finlytics" под backend понимается слой данных и бизнес-логики, работающий локально на устройстве пользователя.

Архитектурный паттерн: MVVM (Model-View-ViewModel)

Структура backend-слоя:

1. Модель данных:

   • Operation.kt – data-класс, представляющий модель финансовой операции с полями: id, тип, сумма, категория, дата и описание.

2. Репозиторий и вспомогательные утилиты:

- FinanceRepository.kt – центральный репозиторий, абстрагирующий доступ к данным и обеспечивающий единую точку взаимодействия между ViewModel и базой данных;
- LoggingConfig.kt – утилита для настройки корректного логирования с поддержкой UTF-8.

3. Работа с базой данных SQLite:
   - Конфигурация и инициализация БД:
     - DatabaseConfig.kt – управляет подключением к SQLite базе данных, автоматически находит или создает файл БД в доступных локациях;
     - DatabaseInitializer.kt – отвечает за создание всех необходимых таблиц при первом запуске приложения и обеспечение целостности схемы БД;
     - DefaultCategories.kt – предоставляет стандартные категории доходов и расходов, инициализирует их при первом запуске приложения.
   - Операции с категориями (CRUD):
     - GetIncomeCategories.kt – получение всех категорий доходов из базы данных;
     - GetExpensesCategories.kt – получение всех категорий расходов из базы данных;
     - AddCategory.kt – добавление новых категорий доходов и расходов;
     - DeleteCategory.kt – удаление категорий доходов и расходов по их идентификаторам.
   - Операции с транзакциями доходов (CRUD):
     - AddIncomeTransaction.kt – добавление новых транзакций доходов в базу данных;
     - GetIncomeTransactions.kt – получение всех транзакций доходов с информацией о категориях;

◦ UpdateIncomeTransaction.kt – обновление существующих транзакций доходов;

◦ DeleteIncomeTransaction.kt – удаление транзакций доходов по их идентификаторам.

• Операции с транзакциями расходов (CRUD):

◦ AddExpensesTransaction.kt – добавление новых транзакций расходов в базу данных;

◦ GetExpensesTransactions.kt – получение всех транзакций расходов с информацией о категориях;

◦ UpdateExpensesTransaction.kt – обновление существующих транзакций расходов;

◦ DeleteExpensesTransaction.kt – удаление транзакций расходов по их идентификаторам.

Листинги компонентов интерфейса:

Модель данных:

Листинг 30 – Operation.kt – Модель данных операции

```
package models

import java.time.LocalDate

/**
 * Модель данных, представляющая финансовую операцию (доход или расход).
 *
 * @property id Уникальный идентификатор операции в базе данных
 * @property type Тип операции: "Доход" или "Расход"
 * @property amount Сумма операции (в рублях)
 * @property category Категория операции
 * @property date Дата совершения операции
 * @property name Описание операции
 */
data class Operation(
    val id: Int,
    val type: String,
    val amount: Double,
    val category: String,
    val date: LocalDate,
    val name: String?
)
```

Репозиторий и вспомогательные утилиты:

Листинг 31 – FinanceRepository.kt – Репозиторий данных

```
package repository
```

```
import db.database_request.*
import models.Operation
import java.time.LocalDate

/**
 * Репозиторий для работы с финансовыми данными.
 * Служит прослойкой между ViewModel и базой данных.
 */
class FinanceRepository {
    /**
     * Инициализирует базу данных при создании репозитория.
     */
    init {
        DatabaseInitializer.createTablesIfNotExist()
    }

    /**
     * Получает все операции (доходы и расходы) из базы данных.
     *
     * @return Отсортированный по дате список всех операций
     */
    fun getAllOperations(): List<Operation> {
        println("\n=== REPOSITORY: ПОЛУЧЕНИЕ ВСЕХ ОПЕРАЦИЙ ===")

        // Получаем операции доходов и преобразуем в модель Operation
        val income = GetIncomeTransactions.getAll().map { row ->
            Operation(
                id = row.id,
                type = "Доход",
                amount = row.amount,
                category = row.category,
                date = LocalDate.parse(row.date),
                name = row.name
            )
        }

        // Получаем операции расходов и преобразуем в модель Operation
        val expenses = GetExpensesTransactions.getAll().map { row ->
            Operation(
                id = row.id,
                type = "Расход",
                amount = row.amount,
                category = row.category,
                date = LocalDate.parse(row.date),
                name = row.name
            )
        }

        // Подробная отладка
        println("Доходы получены: ${income.size} операций")
        println("Расходы получены: ${expenses.size} операций")

        if (income.isNotEmpty()) {
            println("Последняя операция дохода: ${income.first()}")
        }
        if (expenses.isNotEmpty()) {
            println("Последняя операция расхода: ${expenses.first()}")
        }

        // Соединяем и сортируем по дате (сначала новые)
        val allOps = (income + expenses).sortedByDescending { it.date }
        println("Загружено операций: доходов=${income.size},
расходов=${expenses.size}, всего=${allOps.size}")
        println("==============================================\n")
```

```
        return allOps
    }

    /**
     * Получает операции за указанный временной период.
     *
     * @param from Начальная дата периода (включительно)
     * @param to Конечная дата периода (включительно)
     * @return Отфильтрованный список операций в указанном диапазоне дат
     */
    fun getOperations(from: LocalDate, to: LocalDate): List<Operation> {
        println("Получение операций с $from по $to")
        return getAllOperations().filter { it.date in from..to }
    }

    /**
     * Добавляет новую операцию в базу данных.
     *
     * @param op Операция для добавления
     * @return Добавленная операция с присвоенным ID
     * @throws IllegalArgumentException Если категория не найдена
     * @throws RuntimeException Если не удалось добавить операцию
     */
    fun addOperation(op: Operation): Operation {
        println("\n=== ДОБАВЛЕНИЕ ОПЕРАЦИИ ===")
        println("Операция: $op")
        println("Тип операции: ${op.type}")
        println("Категория: ${op.category}")
        println("Сумма: ${op.amount}")
        println("Дата: ${op.date}")
        println("Описание: ${op.name}")
        println("================================\n")

        return if (op.type == "Доход") {
            // Получаем ID категории доходов
            val catId = GetIncomeCategories.getIdByName(op.category)
                ?: throw IllegalArgumentException("Категория доходов
'${op.category}' не найдена")

            // Добавляем транзакцию в базу данных
            val success = AddIncomeTransaction.addIncomeTransaction(
                name = op.name,
                sum = op.amount,
                categoryId = catId,
                date = op.date.toString(),
            )

            if (!success) {
                throw RuntimeException("Не удалось добавить операцию
дохода")
            }

            // Возвращаем операцию с ID, полученным из базы
            op.copy(id = GetIncomeTransactions.getLastId())
        } else {
            // Получаем ID категории расходов
            val catId = GetExpensesCategories.getIdByName(op.category)
                ?: throw IllegalArgumentException("Категория расходов
'${op.category}' не найдена")

            // Добавляем транзакцию в базу данных
            val success = AddExpensesTransaction.addExpensesTransaction(
                name = op.name,
                sum = op.amount,
```

```
                categoryId = catId,
                date = op.date.toString()
            )

            if (!success) {
                throw RuntimeException("Не удалось добавить операцию
расхода")
            }

            // Возвращаем операцию с ID, полученным из базы
            op.copy(id = GetExpensesTransactions.getLastId())
        }
    }

    /**
     * Обновляет существующую операцию в базе данных.
     *
     * @param op Операция с обновленными данными (должен быть задан
корректный id)
     * @throws RuntimeException Если операция не может быть обновлена
     */
    fun updateOperation(op: Operation) {
        println("Обновление операции: $op")
        val success = if (op.type == "Доход") {
            UpdateIncomeTransaction.update(
                transactionId = op.id,
                name = op.name,
                sum = op.amount,
                categoryName = op.category,
                date = op.date.toString()
            )
        } else {
            UpdateExpensesTransaction.update(
                transactionId = op.id,
                name = op.name,
                sum = op.amount,
                categoryName = op.category,
                date = op.date.toString()
            )
        }

        if (!success) {
            throw RuntimeException("Не удалось обновить операцию")
        }
    }

    /**
     * Удаляет операцию из базы данных по идентификатору и типу.
     *
     * @param id Уникальный идентификатор операции
     * @param type Тип операции ("Доход" или "Расход")
     * @throws RuntimeException Если операция не может быть удалена
     */
    fun deleteOperation(id: Int, type: String) {
        println("Удаление операции: id=$id, type=$type")
        val success = if (type == "Доход") {
            DeleteIncomeTransaction.deleteIncomeTransaction(id)
        } else {
            DeleteExpensesTransaction.deleteExpensesTransaction(id)
        }

        if (!success) {
            throw RuntimeException("Не удалось удалить операцию")
        }
```

```
    }

    /**
     * Получает список всех категорий доходов.
     *
     * @return Список названий категорий доходов
     */
    fun getIncomeCategories(): List<String> {
        println("Получение категорий доходов...")
        val categories = GetIncomeCategories.getAllNames()
        println("Категории доходов: $categories")
        return categories
    }

    /**
     * Получает список всех категорий расходов.
     *
     * @return Список названий категорий расходов
     */
    fun getExpenseCategories(): List<String> {
        println("Получение категорий расходов...")
        val categories = GetExpensesCategories.getAllNames()
        println("Категории расходов: $categories")
        return categories
    }

    /**
     * Добавляет новую категорию в базу данных.
     *
     * @param name Название новой категории
     * @param isIncome Тип категории (true - доходы, false - расходы)
     * @return true если категория успешно добавлена, false в случае ошибки
     */
    fun addCategory(name: String, isIncome: Boolean): Boolean {
        println("Добавление категории: name='$name', isIncome=$isIncome")
        return if (isIncome) {
            AddCategory.addIncomeCategory(name)
        } else {
            AddCategory.addExpensesCategory(name)
        }
    }

    /**
     * Удаляет категорию из базы данных.
     *
     * @param name Название категории для удаления
     * @param isIncome Тип категории (true - доходы, false - расходы)
     * @throws IllegalArgumentException Если категория не найдена
     * @throws RuntimeException Если категория не может быть удалена
     */
    fun deleteCategory(name: String, isIncome: Boolean) {
        println("Удаление категории: name='$name', isIncome=$isIncome")
        val id = if (isIncome) GetIncomeCategories.getIdByName(name)
        else GetExpensesCategories.getIdByName(name)

        if (id == null) {
            throw IllegalArgumentException("Категория '$name' не найдена")
        }

        val success = if (isIncome) {
            DeleteCategory.deleteIncomeCategory(id)
        } else {
            DeleteCategory.deleteExpensesCategory(id)
        }
```

```
        if (!success) {
            throw RuntimeException("Не удалось удалить категорию '$name'")
        }
    }
}
```

Листинг 32 – LoggingConfig.kt – Конфигурация логирования

```
package utils

import java.io.PrintStream
import java.nio.charset.StandardCharsets

/**
 * Утилита для настройки корректного логирования с поддержкой UTF-8.
 * Решает проблемы с отображением русских символов в консоли.
 */
object LoggingConfig {
    /**
     * Настраивает System.out и System.err для корректной работы с UTF-8.
     * Устанавливает системные свойства кодировки.
     */
    fun setupLogging() {
        // Устанавливаем UTF-8 кодировку для System.out и System.err
        System.setOut(PrintStream(System.out, true, StandardCharsets.UTF_8))
        System.setErr(PrintStream(System.err, true, StandardCharsets.UTF_8))

        // Устанавливаем системную кодировку
        System.setProperty("file.encoding", "UTF-8")
        System.setProperty("sun.stdout.encoding", "UTF-8")
        System.setProperty("sun.stderr.encoding", "UTF-8")

        println("Логирование настроено. Кодировка:
${System.getProperty("file.encoding")}")
    }
}
```

Конфигурация и инициализация БД:

Листинг 33 – DatabaseConfig.kt – Конфигурация подключения к SQLite

```
package db.database_request

import java.io.File

/**
 * Конфигурация подключения к базе данных SQLite.
 * Автоматически находит или создает файл базы данных в доступных локациях.
 */
object DatabaseConfig {
    private const val DB_NAME = "Finlytics.db"

    /**
     * URL подключения к базе данных SQLite.
     * Ищет существующий файл БД или создает новый в текущей директории.
     */
    val DB_URL: String by lazy {
        // Пробуем несколько путей для поиска базы данных
        val possiblePaths = listOf(
            File(System.getProperty("user.dir"), DB_NAME).absolutePath, //
```

```
Текущая директория
            File("src/main/kotlin/db/database", DB_NAME).absolutePath,  //
Путь при разработке
            File("db/database", DB_NAME).absolutePath,                  //
Альтернативный путь
            DB_NAME
// Просто имя файла
        )

        var dbPath = possiblePaths.firstOrNull { File(it).exists() } ?:
possiblePaths[0]

        // Если файл не существует, создадим его в текущей директории
        val dbFile = File(dbPath)
        if (!dbFile.exists()) {
            dbFile.parentFile?.mkdirs()
            try {
                dbFile.createNewFile()
                println("Создана новая база данных: ${dbFile.absolutePath}")
            } catch (e: Exception) {
                println("Ошибка при создании базы данных: ${e.message}")
                // Если не удалось создать по этому пути, пробуем другой
                dbPath = possiblePaths[1]
                val altFile = File(dbPath)
                altFile.parentFile?.mkdirs()
                altFile.createNewFile()
                println("Создана альтернативная база данных:
${altFile.absolutePath}")
            }
        }

        println("Используется база данных: ${dbFile.absolutePath}")
        "jdbc:sqlite:$dbPath"
    }
}
```

Листинг 34 – DatabaseInitializer.kt – Инициализация БД и создание таблиц

```
package db.database_request

import java.sql.DriverManager
import java.io.File

/**
 * Инициализатор базы данных. Создает таблицы при первом запуске приложения.
 * Добавляет категории по умолчанию.
 */
object DatabaseInitializer {
    // Используем конфигурацию для получения пути к БД
    private val DB_URL = DatabaseConfig.DB_URL

    init {
        // Создаем директорию для базы данных, если ее нет
        val dbFile = File("src/main/kotlin/db/database/Finlytics.db")
        dbFile.parentFile.mkdirs()
        println("Путь к БД: ${dbFile.absolutePath}")
        println("БД существует: ${dbFile.exists()}")
    }

    /**
     * Создает все необходимые таблицы в базе данных, если они не
существуют.
     * Выполняется при каждом запуске приложения для обеспечения целостности
```

```
схемы БД.
     */
    fun createTablesIfNotExist() {
        // SQL-запросы для создания таблиц
        val sql = """
            CREATE TABLE IF NOT EXISTS Income_categories (
                id_income_category INTEGER PRIMARY KEY AUTOINCREMENT,
                income_category_name TEXT NOT NULL UNIQUE
            );
            CREATE TABLE IF NOT EXISTS expenses_categories (
                id_expenses_category INTEGER PRIMARY KEY AUTOINCREMENT,
                expenses_category_name TEXT NOT NULL UNIQUE
            );
            CREATE TABLE IF NOT EXISTS Income_transactions (
                id_income_transaction INTEGER PRIMARY KEY AUTOINCREMENT,
                income_transaction_name TEXT,
                income_transaction_sum REAL NOT NULL,
                id_income_category INTEGER,
                income_transaction_date TEXT NOT NULL,
                FOREIGN KEY (id_income_category) REFERENCES
Income_categories(id_income_category) ON DELETE RESTRICT
            );
            CREATE TABLE IF NOT EXISTS expenses_transactions (
                id_expenses_transaction INTEGER PRIMARY KEY AUTOINCREMENT,
                expenses_transaction_name TEXT,
                expenses_transaction_sum REAL NOT NULL,
                id_expenses_category INTEGER,
                expenses_transaction_date TEXT NOT NULL,
                FOREIGN KEY (id_expenses_category) REFERENCES
expenses_categories(id_expenses_category) ON DELETE RESTRICT
            );
        """.trimIndent()

        try {
            DriverManager.getConnection(DB_URL).use { conn ->
                println("\n=== DATABASE INITIALIZATION ===")
                println("Подключение к БД успешно: $DB_URL")

                // Проверяем существующие таблицы
                val metaData = conn.metaData
                val tables = metaData.getTables(null, null, "%", null)
                println("Существующие таблицы в БД:")
                while (tables.next()) {
                    println("  - ${tables.getString("TABLE_NAME")}")
                }
                tables.close()

                conn.createStatement().use { stmt ->
                    // Выполняем каждый запрос отдельно (разделены точкой с
запятой)
                    sql.split(";").forEach { query ->
                        if (query.trim().isNotEmpty()) {
                            stmt.execute(query.trim())
                            println("Выполнен запрос:
${query.trim().take(50)}...")
                        }
                    }
                }
                println("Таблицы созданы/проверены успешно")
                println("================================\n")

                // Добавляем категории по умолчанию
                DefaultCategories.initializeDefaultCategories()
            }
```

```
        } catch (e: Exception) {
            println("\n!!! ОШИБКА ИНИЦИАЛИЗАЦИИ БД !!!")
            println("Сообщение: ${e.message}")
            println("Трассировка:")
            e.printStackTrace()
            println("================================\n")
        }
    }
}
```

Листинг 35 – DefaultCategories.kt – Заполнение категорий по умолчанию

```
package db.database_request

/**
 * Предоставляет стандартные категории доходов и расходов.
 * Инициализирует их при первом запуске приложения.
 */
object DefaultCategories {
    // Стандартные категории доходов
    private val defaultIncomeCategories = listOf(
        "Зарплата",
        "Стипендия",
        "Фриланс",
        "Инвестиции",
        "Подарок"
    )

    // Стандартные категории расходов
    private val defaultExpenseCategories = listOf(
        "Еда",
        "Транспорт",
        "Жилье",
        "Развлечения",
        "Образование",
        "Здоровье",
        "Одежда",
        "Коммунальные услуги"
    )

    /**
     * Добавляет стандартные категории в базу данных, если их еще нет.
     * Вызывается при инициализации базы данных.
     */
    fun initializeDefaultCategories() {
        println("Инициализация категорий по умолчанию...")

        // Добавляем категории доходов
        var incomeCount = 0
        defaultIncomeCategories.forEach { category ->
            if (!incomeCategoryExists(category)) {
                val success = AddCategory.addIncomeCategory(category)
                if (success) incomeCount++
            }
        }
        println("Добавлено категорий доходов: $incomeCount")

        // Добавляем категории расходов
        var expenseCount = 0
        defaultExpenseCategories.forEach { category ->
            if (!expenseCategoryExists(category)) {
                val success = AddCategory.addExpensesCategory(category)
                if (success) expenseCount++
```

```
            }
        }
        println("Добавлено категорий расходов: $expenseCount")
    }

    /**
     * Проверяет существование категории доходов.
     */
    private fun incomeCategoryExists(name: String): Boolean {
        return GetIncomeCategories.getAllNames().any { it.equals(name,
ignoreCase = true) }
    }

    /**
     * Проверяет существование категории расходов.
     */
    private fun expenseCategoryExists(name: String): Boolean {
        return GetExpensesCategories.getAllNames().any { it.equals(name,
ignoreCase = true) }
    }
}
```

Операции с категориями (CRUD):

Листинг 36 – GetIncomeCategories.kt – Получение категорий доходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для получения категорий доходов из базы данных.
 */
object GetIncomeCategories {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Получает все категории доходов из базы данных.
     * Категории возвращаются отсортированными по названию.
     *
     * @return Список пар (id категории, название категории)
     */
    fun getAll(): List<Pair<Int, String>> {
        val list = mutableListOf<Pair<Int, String>>()
        val sql = "SELECT id_income_category, income_category_name FROM
Income_categories ORDER BY income_category_name"
        try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.createStatement().use { stmt ->
                    val rs = stmt.executeQuery(sql)
                    while (rs.next()) {
                        list.add(rs.getInt(1) to rs.getString(2))
                    }
                    println("Загружено категорий доходов: ${list.size}")
                }
            }
        } catch (e: Exception) {
            println("Ошибка при получении категорий доходов: ${e.message}")
        }
        return list
    }
```

```
    /**
     * Получает список названий всех категорий доходов.
     *
     * @return Список названий категорий
     */
    fun getAllNames() = getAll().map { it.second }

    /**
     * Получает идентификатор категории доходов по её названию.
     *
     * @param name Название категории
     * @return Идентификатор категории или null, если категория не найдена
     */
    fun getIdByName(name: String) = getAll().find { it.second == name
}?.first
}
```

Листинг 37 – GetExpensesCategories.kt – Получение категорий расходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для получения транзакций расходов из базы данных.
 * Выполняет JOIN с таблицей категорий для получения полной информации.
 */
object GetExpensesTransactions {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Модель строки результата запроса транзакций расходов.
     *
     * @property id Уникальный идентификатор транзакции
     * @property name Название транзакции (может быть null)
     * @property amount Сумма транзакции
     * @property date Дата транзакции в формате строки
     * @property category Название категории транзакции
     */
    data class Row(
        val id: Int,
        val name: String?,
        val amount: Double,
        val date: String,
        val category: String
    )

    /**
     * Получает все транзакции расходов из базы данных.
     * Транзакции возвращаются в порядке убывания даты (от новых к старым).
     *
     * @return Список всех транзакций расходов с информацией о категориях
     */
    fun getAll(): List<Row> {
        val list = mutableListOf<Row>()
        val sql = """
            SELECT et.id_expenses_transaction, et.expenses_transaction_name,
et.expenses_transaction_sum,
                   et.expenses_transaction_date, ec.expenses_category_name
            FROM expenses_transactions et
            JOIN expenses_categories ec ON et.id_expenses_category =
ec.id_expenses_category
            ORDER BY et.expenses_transaction_date DESC
```

```
        """.trimIndent()
        try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.createStatement().use { stmt ->
                    val rs = stmt.executeQuery(sql)
                    while (rs.next()) {
                        list.add(Row(rs.getInt(1), rs.getString(2),
rs.getDouble(3), rs.getString(4), rs.getString(5)))
                    }
                }
            }
        } catch (e: Exception) { }
        return list
    }

    /**
     * Получает максимальный идентификатор транзакции расхода.
     * Используется для получения ID только что добавленной транзакции.
     *
     * @return Максимальный ID или 0, если транзакций нет
     */
    fun getLastId() = getAll().maxOfOrNull { it.id } ?: 0
}
```

Листинг 38 – AddCategory.kt – Добавление новых категорий

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для управления категориями в базе данных.
 * Предоставляет функции добавления категорий доходов и расходов.
 */
object AddCategory {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Добавляет новую категорию доходов в базу данных.
     * Валидирует входные данные и логирует процесс выполнения.
     *
     * @param name Название категории доходов
     * @return true если категория успешно добавлена, false в случае ошибки
     */
    fun addIncomeCategory(name: String): Boolean {
        println("Добавление категории доходов: '$name'")
        if (name.isBlank()) {
            println("Имя категории пустое")
            return false
        }

        val sql = "INSERT INTO Income_categories (income_category_name)
VALUES (?)"
        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setString(1, name.trim())
                    val rowsAffected = pstmt.executeUpdate()
                    val result = rowsAffected > 0
                    println("Категория доходов '$name' добавлена. Затронуто
строк: $rowsAffected, успех: $result")
                    result
                }
```

```
            }
        } catch (e: Exception) {
            println("Ошибка при добавлении категории доходов '$name':
${e.message}")
            e.printStackTrace()
            false
        }
    }

    /**
     * Добавляет новую категорию расходов в базу данных.
     * Валидирует входные данные и логирует процесс выполнения.
     *
     * @param name Название категории расходов
     * @return true если категория успешно добавлена, false в случае ошибки
     */
    fun addExpensesCategory(name: String): Boolean {
        println("Добавление категории расходов: '$name'")
        if (name.isBlank()) {
            println("Имя категории пустое")
            return false
        }

        val sql = "INSERT INTO expenses_categories (expenses_category_name)
VALUES (?)"
        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setString(1, name.trim())
                    val rowsAffected = pstmt.executeUpdate()
                    val result = rowsAffected > 0
                    println("Категория расходов '$name' добавлена. Затронуто
строк: $rowsAffected, успех: $result")
                    result
                }
            }
        } catch (e: Exception) {
            println("Ошибка при добавлении категории расходов '$name':
${e.message}")
            e.printStackTrace()
            false
        }
    }
}
```

Листинг 39 – DeleteCategory.kt – Удаление категорий

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для удаления категорий из базы данных.
 * Предоставляет функции удаления категорий доходов и расходов по
идентификатору.
 */
object DeleteCategory {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Удаляет категорию доходов по её идентификатору.
     *
     * @param categoryId Идентификатор удаляемой категории доходов
```

```
     * @return true если категория успешно удалена, false в случае ошибки
     */
    fun deleteIncomeCategory(categoryId: Int): Boolean {
        if (categoryId <= 0) return false
        val sql = "DELETE FROM Income_categories WHERE id_income_category =
?"
        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setInt(1, categoryId)
                    val rowsAffected = pstmt.executeUpdate()
                    println("Удалена категория доходов с ID $categoryId.
Затронуто строк: $rowsAffected")
                    rowsAffected > 0
                }
            }
        } catch (e: Exception) {
            println("Ошибка при удалении категории доходов: ${e.message}")
            e.printStackTrace()
            false
        }
    }

    /**
     * Удаляет категорию расходов по её идентификатору.
     *
     * @param categoryId Идентификатор удаляемой категории расходов
     * @return true если категория успешно удалена, false в случае ошибки
     */
    fun deleteExpensesCategory(categoryId: Int): Boolean {
        if (categoryId <= 0) return false
        val sql = "DELETE FROM expenses_categories WHERE
id_expenses_category = ?"
        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setInt(1, categoryId)
                    val rowsAffected = pstmt.executeUpdate()
                    println("Удалена категория расходов с ID $categoryId.
Затронуто строк: $rowsAffected")
                    rowsAffected > 0
                }
            }
        } catch (e: Exception) {
            println("Ошибка при удалении категории расходов: ${e.message}")
            e.printStackTrace()
            false
        }
    }
}
```

Операции с транзакциями доходов (CRUD):

Листинг 40 – AddIncomeTransaction.kt – Добавление транзакций доходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для добавления транзакций доходов в базу данных.
 * Реализует операцию INSERT для таблицы Income_transactions.
```

```
 */
object AddIncomeTransaction {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Добавляет новую транзакцию дохода в базу данных.
     *
     * @param name Название транзакции (может быть null)
     * @param sum Сумма транзакции (должна быть положительной)
     * @param categoryId Идентификатор категории дохода (должен быть > 0)
     * @param date Дата транзакции в формате строки (ГГГГ-ММ-ДД)
     * @return true если транзакция успешно добавлена, false в случае ошибки
или некорректных данных
     */
    fun addIncomeTransaction(name: String?, sum: Double, categoryId: Int,
date: String): Boolean {
        if (sum <= 0 || categoryId <= 0) return false
        val sql = """
            INSERT INTO Income_transactions
            (income_transaction_name, income_transaction_sum,
id_income_category, income_transaction_date)
            VALUES (?, ?, ?, ?)
        """.trimIndent()
        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setString(1, name)
                    pstmt.setDouble(2, sum)
                    pstmt.setInt(3, categoryId)
                    pstmt.setString(4, date)
                    pstmt.executeUpdate() > 0
                }
            }
        } catch (e: Exception) {
            false
        }
    }
}
```

Листинг 41 – GetIncomeTransactions.kt – Получение транзакций доходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для получения транзакций доходов из базы данных.
 * Выполняет JOIN с таблицей категорий для получения полной информации.
 */
object GetIncomeTransactions {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Модель строки результата запроса транзакций доходов.
     *
     * @property id Уникальный идентификатор транзакции
     * @property name Название транзакции (может быть null)
     * @property amount Сумма транзакции
     * @property date Дата транзакции в формате строки
     * @property category Название категории транзакции
     */
    data class Row(
        val id: Int,
```

```
        val name: String?,
        val amount: Double,
        val date: String,
        val category: String
    )

    /**
     * Получает все транзакции доходов из базы данных.
     * Транзакции возвращаются в порядке убывания даты (от новых к старым).
     *
     * @return Список всех транзакций доходов с информацией о категориях
     */
    fun getAll(): List<Row> {
        val list = mutableListOf<Row>()
        val sql = """
            SELECT it.id_income_transaction, it.income_transaction_name,
it.income_transaction_sum,
                   it.income_transaction_date, ic.income_category_name
            FROM Income_transactions it
            JOIN Income_categories ic ON it.id_income_category =
ic.id_income_category
            ORDER BY it.income_transaction_date DESC
        """.trimIndent()
        try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.createStatement().use { stmt ->
                    val rs = stmt.executeQuery(sql)
                    while (rs.next()) {
                        list.add(Row(rs.getInt(1), rs.getString(2),
rs.getDouble(3), rs.getString(4), rs.getString(5)))
                    }
                }
            }
        } catch (e: Exception) { }
        return list
    }

    /**
     * Получает максимальный идентификатор транзакции дохода.
     * Используется для получения ID только что добавленной транзакции.
     *
     * @return Максимальный ID или 0, если транзакций нет
     */
    fun getLastId() = getAll().maxOfOrNull { it.id } ?: 0
}
```

Листинг 42 – UpdateIncomeTransaction.kt – Обновление транзакций доходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для обновления транзакций доходов в базе данных.
 */
object UpdateIncomeTransaction {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Обновляет существующую транзакцию дохода.
     *
     * @param transactionId Идентификатор обновляемой транзакции (должен
быть > 0)
```

```
     * @param name Новое название транзакции (может быть null)
     * @param sum Новая сумма транзакции (должна быть положительной)
     * @param categoryName Новое название категории (должна существовать в
базе)
     * @param date Новая дата транзакции в формате строки (ГГГГ-ММ-ДД)
     * @return true если транзакция успешно обновлена, false в случае ошибки
или некорректных данных
     */
    fun update(
        transactionId: Int,
        name: String? = null,
        sum: Double,
        categoryName: String,
        date: String
    ): Boolean {
        val categoryId = GetIncomeCategories.getIdByName(categoryName) ?:
return false
        if (transactionId <= 0 || sum <= 0 || categoryId <= 0) return false

        val sql = """
            UPDATE Income_transactions
            SET income_transaction_name = ?,
                income_transaction_sum = ?,
                id_income_category = ?,
                income_transaction_date = ?
            WHERE id_income_transaction = ?
        """.trimIndent()

        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setString(1, name)
                    pstmt.setDouble(2, sum)
                    pstmt.setInt(3, categoryId)
                    pstmt.setString(4, date)
                    pstmt.setInt(5, transactionId)
                    pstmt.executeUpdate() > 0
                }
            }
        } catch (e: Exception) {
            false
        }
    }
}
```

Листинг 43 – DeleteIncomeTransaction.kt – Удаление транзакций доходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для удаления транзакций доходов из базы данных.
 */
object DeleteIncomeTransaction {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Удаляет транзакцию дохода по её идентификатору.
     *
     * @param transactionId Идентификатор удаляемой транзакции (должен быть
> 0)
     * @return true если транзакция успешно удалена, false в случае ошибки
```

```
или некорректного ID
     */
    fun deleteIncomeTransaction(transactionId: Int): Boolean {
        if (transactionId <= 0) return false

        val sql = "DELETE FROM Income_transactions WHERE
id_income_transaction = ?"

        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setInt(1, transactionId)
                    pstmt.executeUpdate() > 0
                }
            }
        } catch (e: Exception) {
            false
        }
    }
}
```

Операции с транзакциями расходов (CRUD):

Листинг 44 – AddExpensesTransaction.kt – Добавление транзакций расходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для добавления транзакций расходов в базу данных.
 * Реализует операцию INSERT для таблицы expenses_transactions.
 */
object AddExpensesTransaction {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Добавляет новую транзакцию расхода в базу данных.
     *
     * @param name Название транзакции (может быть null)
     * @param sum Сумма транзакции (должна быть положительной)
     * @param categoryId Идентификатор категории расхода (должен быть > 0)
     * @param date Дата транзакции в формате строки (ГГГГ-ММ-ДД)
     * @return true если транзакция успешно добавлена, false в случае ошибки
или некорректных данных
     */
    fun addExpensesTransaction(name: String?, sum: Double, categoryId: Int,
date: String): Boolean {
        if (sum <= 0 || categoryId <= 0) return false
        val sql = """
            INSERT INTO expenses_transactions
            (expenses_transaction_name, expenses_transaction_sum,
id_expenses_category, expenses_transaction_date)
            VALUES (?, ?, ?, ?)
        """.trimIndent()
        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setString(1, name)
                    pstmt.setDouble(2, sum)
                    pstmt.setInt(3, categoryId)
                    pstmt.setString(4, date)
```

```
                    pstmt.executeUpdate() > 0
                }
            }
        } catch (e: Exception) {
            false
        }
    }
}
```

Листинг 45 – GetExpensesTransactions.kt – Получение транзакций расходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для получения категорий расходов из базы данных.
 */
object GetExpensesCategories {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Получает все категории расходов из базы данных.
     * Категории возвращаются отсортированными по названию.
     *
     * @return Список пар (id категории, название категории)
     */
    fun getAll(): List<Pair<Int, String>> {
        val list = mutableListOf<Pair<Int, String>>()
        val sql = "SELECT id_expenses_category, expenses_category_name FROM
expenses_categories ORDER BY expenses_category_name"
        try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.createStatement().use { stmt ->
                    val rs = stmt.executeQuery(sql)
                    while (rs.next()) {
                        list.add(rs.getInt(1) to rs.getString(2))
                    }
                    println("Загружено категорий расходов: ${list.size}")
                }
            }
        } catch (e: Exception) {
            println("Ошибка при получении категорий расходов: ${e.message}")
        }
        return list
    }

    /**
     * Получает список названий всех категорий расходов.
     *
     * @return Список названий категорий
     */
    fun getAllNames() = getAll().map { it.second }

    /**
     * Получает идентификатор категории расходов по её названию.
     *
     * @param name Название категории
     * @return Идентификатор категории или null, если категория не найдена
     */
    fun getIdByName(name: String) = getAll().find { it.second == name
}?.first
}
```

Листинг 46 – UpdateExpensesTransaction.kt – Обновление транзакций расходов

```kt
package db.database_request

import java.sql.DriverManager

/**
 * Объект для обновления транзакций расходов в базе данных.
 */
object UpdateExpensesTransaction {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Обновляет существующую транзакцию расхода.
     *
     * @param transactionId Идентификатор обновляемой транзакции (должен быть > 0)
     * @param name Новое название транзакции (может быть null)
     * @param sum Новая сумма транзакции (должна быть положительной)
     * @param categoryName Новое название категории (должна существовать в базе)
     * @param date Новая дата транзакции в формате строки (ГГГГ-ММ-ДД)
     * @return true если транзакция успешно обновлена, false в случае ошибки или некорректных данных
     */
    fun update(
        transactionId: Int,
        name: String? = null,
        sum: Double,
        categoryName: String,
        date: String
    ): Boolean {
        val categoryId = GetExpensesCategories.getIdByName(categoryName) ?: return false
        if (transactionId <= 0 || sum <= 0 || categoryId <= 0) return false

        val sql = """
            UPDATE expenses_transactions
            SET expenses_transaction_name = ?,
                expenses_transaction_sum = ?,
                id_expenses_category = ?,
                expenses_transaction_date = ?
            WHERE id_expenses_transaction = ?
        """.trimIndent()

        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setString(1, name)
                    pstmt.setDouble(2, sum)
                    pstmt.setInt(3, categoryId)
                    pstmt.setString(4, date)
                    pstmt.setInt(5, transactionId)
                    pstmt.executeUpdate() > 0
                }
            }
        } catch (e: Exception) {
            false
        }
    }
```

```
}
```

Листинг 47 – DeleteExpensesTransaction.kt – Удаление транзакций расходов

```
package db.database_request

import java.sql.DriverManager

/**
 * Объект для удаления транзакций расходов из базы данных.
 */
object DeleteExpensesTransaction {
    private val DB_URL = DatabaseConfig.DB_URL

    /**
     * Удаляет транзакцию расхода по её идентификатору.
     *
     * @param transactionId Идентификатор удаляемой транзакции (должен быть
> 0)
     * @return true если транзакция успешно удалена, false в случае ошибки
или некорректного ID
     */
    fun deleteExpensesTransaction(transactionId: Int): Boolean {
        if (transactionId <= 0) return false

        val sql = "DELETE FROM expenses_transactions WHERE
id_expenses_transaction = ?"

        return try {
            DriverManager.getConnection(DB_URL).use { conn ->
                conn.prepareStatement(sql).use { pstmt ->
                    pstmt.setInt(1, transactionId)
                    pstmt.executeUpdate() > 0
                }
            }
        } catch (e: Exception) {
            false
        }
    }
}
```

3.2.3 База данных

В качестве системы управления базами данных была выбрана SQLite, что обусловлено минимальными требованиями к настройке и соответствием функциональных возможностей проекту.

Использование SQLite оправдано её лёгкостью и достаточностью для данного проекта, что позволяет избежать избыточности более сложных СУБД.

Структура БД:

- Таблица **income_categories**
  - **id_income_category**
  - **income_category_name**

- Таблица **income_transactions**
  - **id_income_transaction**
  - **income_transaction_name**
  - **income_transaction_sum**
  - **id_income_category**
  - **income_transaction_date**
- Таблица **expenses_categories**
  - **id_expenses_category**
  - **expenses_category_name**
- Таблица **expenses_transactions**
  - **id_expenses_transaction**
  - **expenses_transaction_name**
  - **expenses_transaction_sum**
  - **expenses_id_category**
  - **expenses_transaction_date**

3.3 Тестирование приложения

Тестирование проводилось вручную для определения успешности и исправности работы программы. Примеры работы приложения показаны на рисунках ниже.

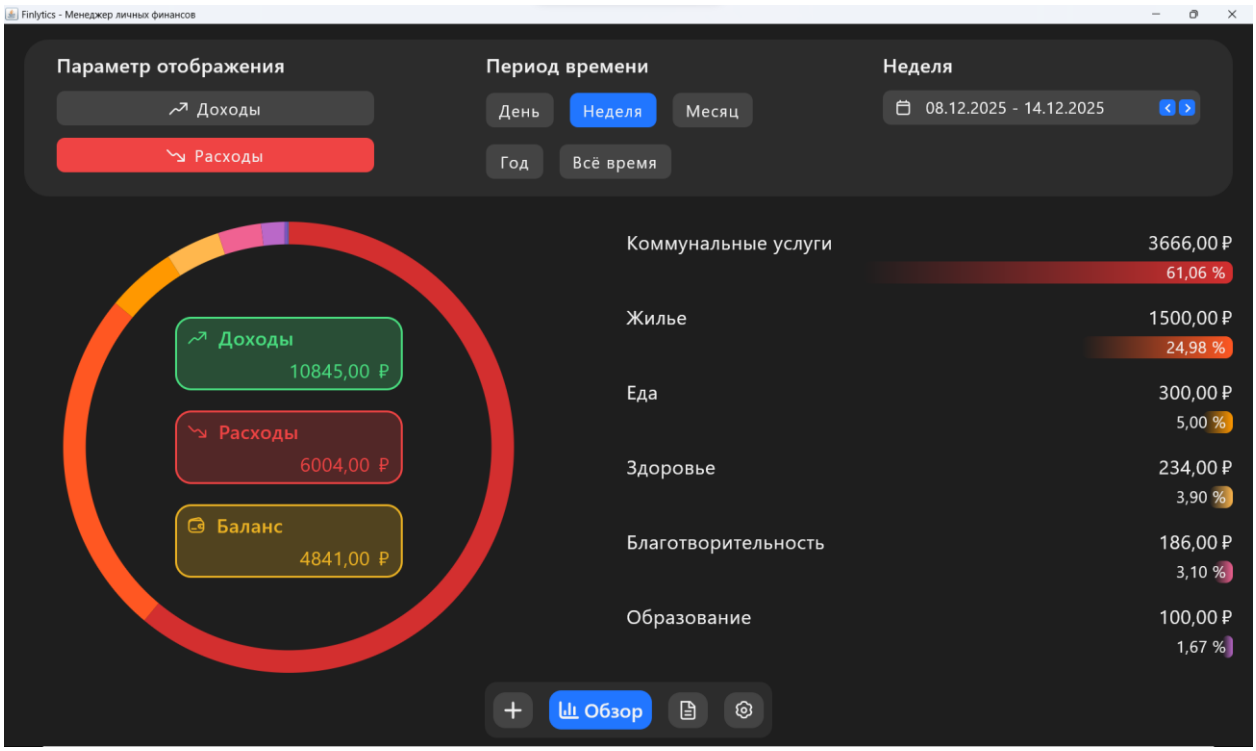

Рисунок 17 — Главный экран приложения (расходы)

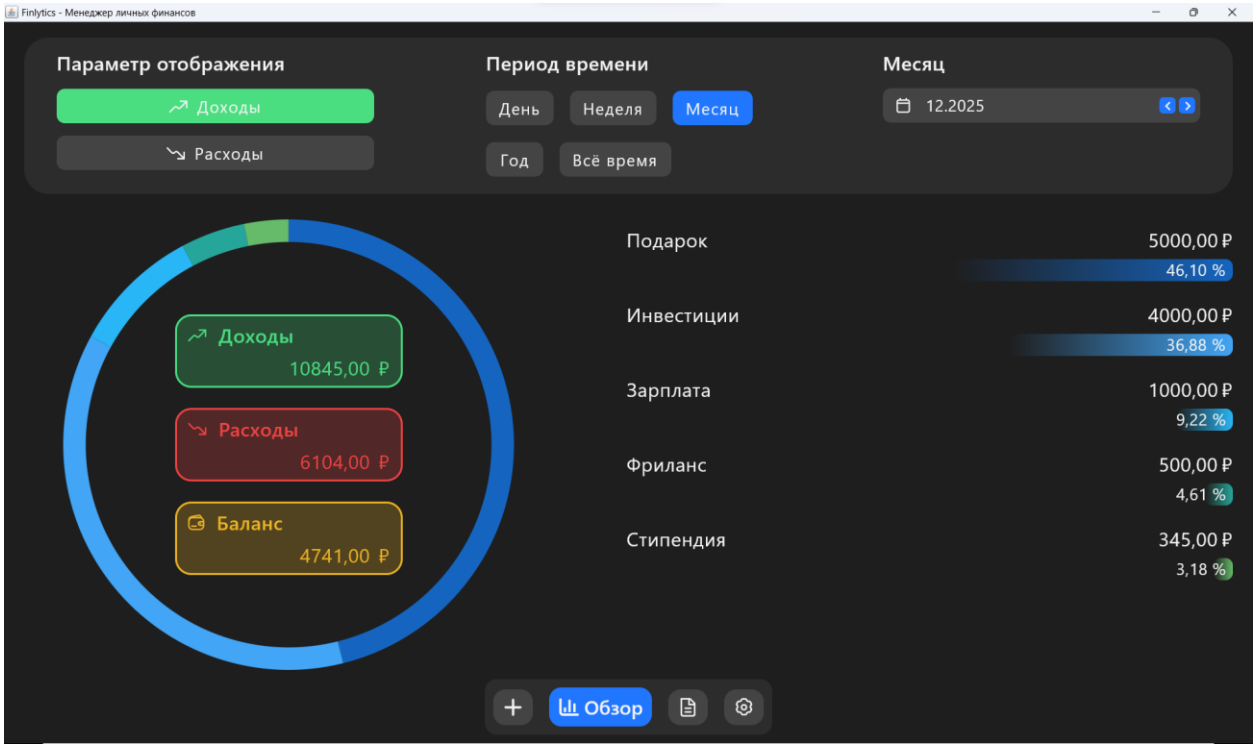

Рисунок 18 — Главный экран приложения (доходы)

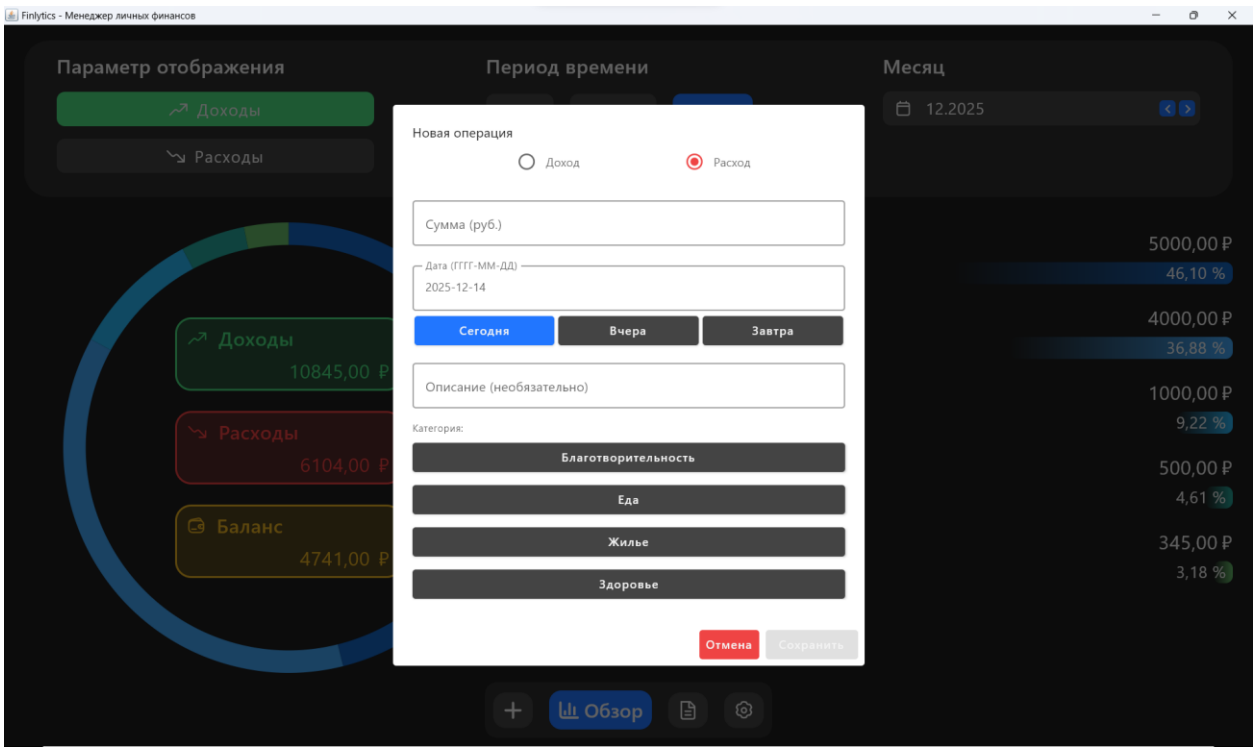

Рисунок 19 — Окно добавления транзакции

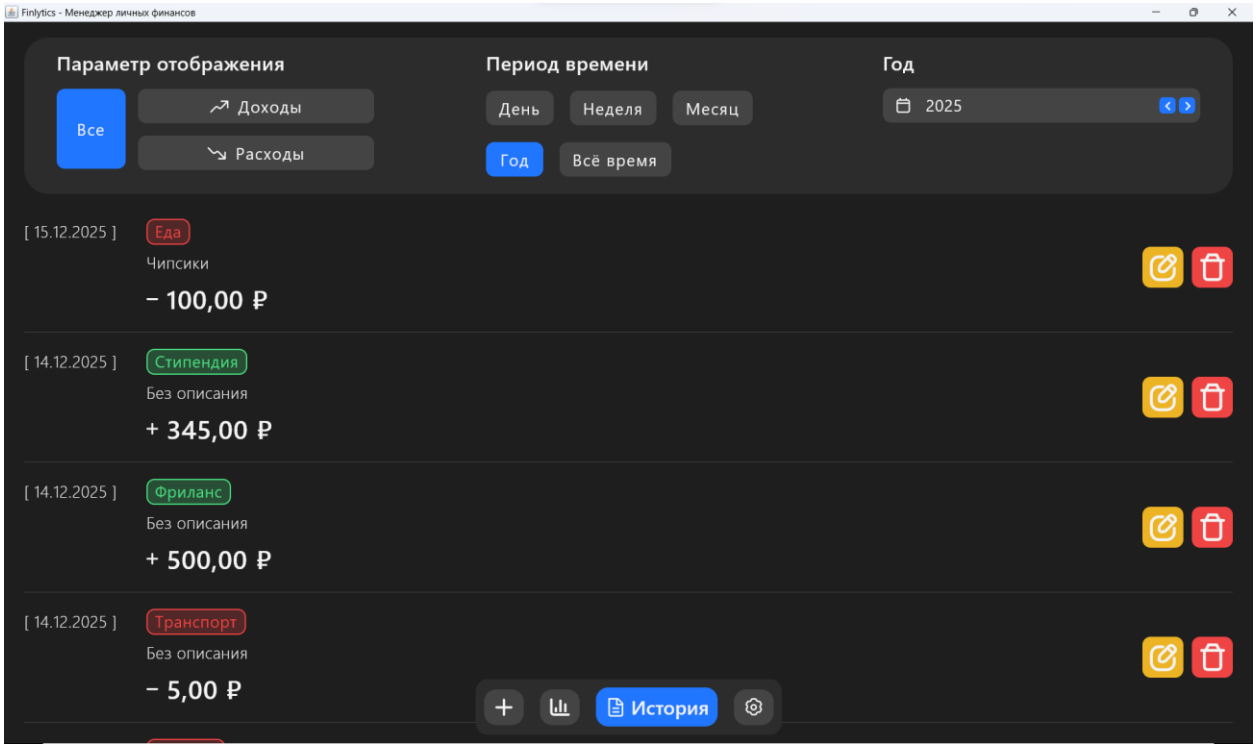

Рисунок 20 — История транзакций

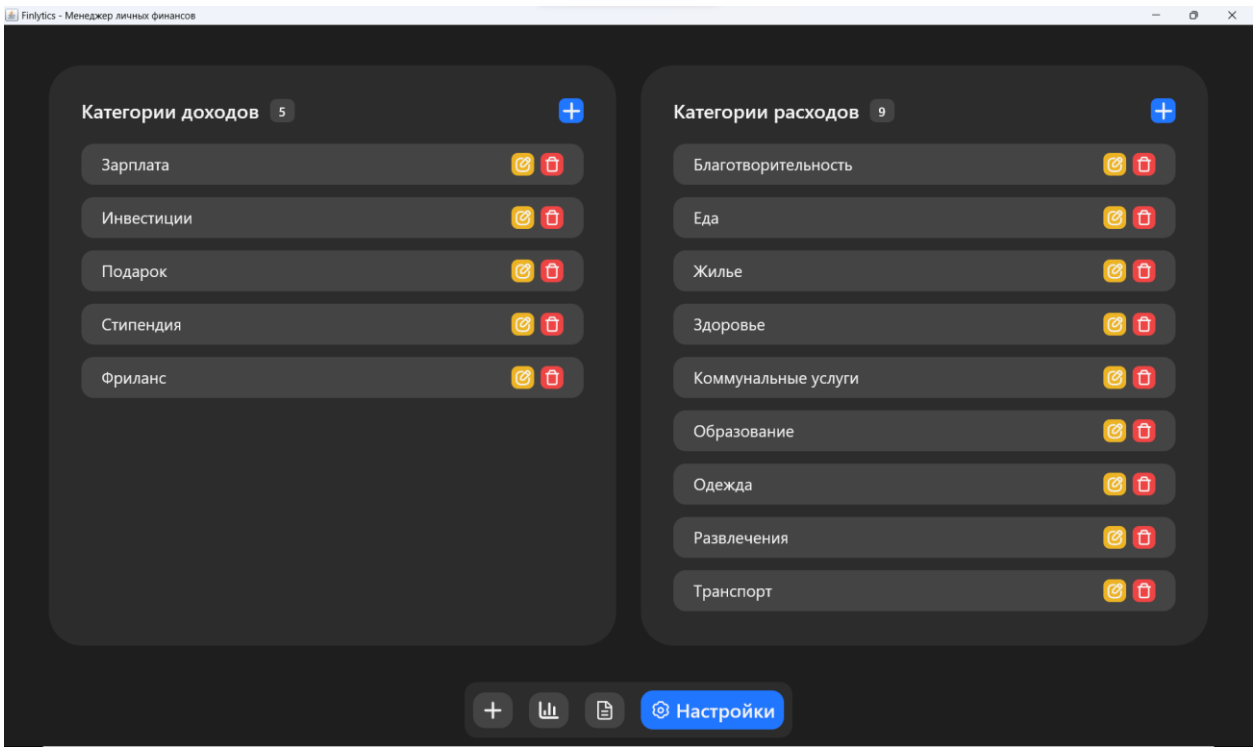

Рисунок 21 — Настройки приложения

3.4 Документирование кода

Документирование кода выполнено с использованием KDoc - стандартной системы документации для Kotlin. Все публичные классы,

функции и свойства имеют подробные комментарии с описанием назначения, параметров, возвращаемых значений и возможных исключений.

Принципы документирования:

- все публичные API документированы – каждый класс и функция имеют KDoc комментарии;
- подробное описание параметров – для каждого параметра указано назначение и ограничения;
- примеры использования – для сложных функций добавлены примеры вызова;
- описание исключений – документированы все возможные исключения и условия их возникновения.

Документация для пользователей:

- README.md – содержит полное описание проекта, требования к системе, инструкции по сборке и запуску;
- встроенные подсказки в UI – все элементы интерфейса имеют интуитивно понятные названия и всплывающие подсказки;
- валидация ввода – система сообщает пользователю об ошибках ввода в реальном времени;
- контекстные меню – для сложных операций предусмотрены диалоговые окна с инструкциями.

ЗАКЛЮЧЕНИЕ

В рамках проекта "Finlytics: Офлайн-менеджер финансов" было успешно разработано десктопное приложение для управления личными финансами на языке Kotlin с использованием современных технологий и подходов.

Достигнутые результаты:

- реализован полнофункциональный менеджер финансов с возможностью учета доходов и расходов, категоризацией операций и аналитикой;
- создан интуитивно понятный пользовательский интерфейс на основе Compose for Desktop с темной темой и акцентными цветами;
- разработана отказоустойчивая архитектура MVVM, обеспечивающая четкое разделение ответственности компонентов;
- реализована офлайн-работа приложения с использованием SQLite в качестве локального хранилища данных;
- внедрена система визуализации данных с кастомными диаграммами для наглядного представления финансовой статистики;
- обеспечена кросс-платформенность - приложение работает на Windows, macOS и Linux.

Ключевые технические достижения:

- применение реактивного программирования с Kotlin Flows для автоматического обновления UI;
- реализация кастомных Compose-компонентов (диаграммы, диалоговые окна с валидацией);
- создание модульной структуры проекта с четким разделением на слои (UI, ViewModel, Repository, Data);
- интеграция SQLite с автоматической миграцией и инициализацией данных;

- настройка системы логирования с поддержкой UTF-8 для русскоязычного интерфейса.

Практическая значимость:

Разработанное приложение представляет собой готовое к использованию решение для студентов и других пользователей, нуждающихся в простом и эффективном инструменте для контроля личного бюджета без необходимости подключения к интернету. Особенностью приложения является его полная автономность - все данные хранятся локально, что обеспечивает конфиденциальность и доступность в любое время.

Ссылка на GitHub-репозиторий разработанного проекта: https://github.com/Hohrandrey/Finlytics

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 34.602—2020 ("Техническое задание на создание автоматизированной системы") является стандартом, который определяет общие требования к разработке автоматизированных систем (АС). Он используется в разных отраслях, включая медицину, и применим для разработки автоматизированных информационных систем (АИС).

2. ГОСТ 34.201—2020. Межгосударственный стандарт. Информационные технологии. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем: Приказом Федерального агентства по техническому регулированию и метрологии № 1521-ст от 19 ноября 2021 г.: дата введения 2022-01-01. – М.: Российский институт стандартизации, 2021. – 10 с.

3. Блоштин А.В., Лаптев Д.В. Современные подходы к проектированию клиент-серверных приложений // Программные системы: теория и приложения. 2021. №2. URL: https://elibrary.ru/item.asp?id=46523678 (дата обращения: 26.11.2025)

4. Android Developers — официальный сайт документации по разработке на Kotlin: https://developer.android.com/kotlin (дата обращения: 26.11.2025).

5. Kotlin Documentation — полный справочник по языку Котлин: https://kotlinlang.org/docs (дата обращения: 26.11.2025).

6. Жемеров С., Исакова С. Kotlin в действии. 2-е издание. М.: Питер, 2022. URL: https://www.piter.com/product/kotlin-v-dejstvii (дата обращения: 26.11.2025).

7. Документация JUnit — для тестирования Kotlin-приложений: https://junit.org/junit5 (дата обращения: 26.11.2025).