МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Экономики и управления на предприятии природопользования»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(бакалаврская работа)
по направлению подготовки 09.03.03 Прикладная информатика
(квалификация – бакалавр)

На тему «Разработка мобильного приложения для учета расходов на содержание автомобиля»

Исполнитель Поскотинова Нина Николаевна

Руководитель к.г.н., доцент по кафедре «Информатики и прикладной математики» Полупанов Владимир Николаевич

«К защите допускаю»

Руководитель кафедры _____

кандидат экономических наук

Майборода Евгений Викторович

«17» 01 2026 г.

Филиал Российского государственного гидрометеорологического университета в г. Туапсе
НОРМОКОНТРОЛЬ ПРОЙДЕН
«12» 01 2026 г.
Мартынова Т.В

Туапсе
2026

ОГЛАВЛЕНИЕ

Введение

Сегодня автомобиль есть у многих, и он давно стал не роскошью, а обычным средством передвижения. Но вместе с удобством приходят и расходы: бензин, обслуживание, страховка, налоги, парковка. Эти траты накапливаются, и часто водитель даже не замечает, сколько реально уходит денег на содержание машины.

Большинство людей не ведут учёт. Кто-то записывает суммы в блокнот, кто-то – в таблицу, а кто-то просто держит всё в голове. В итоге контроль теряется: непонятно, когда пора менять масло, сколько потрачено на бензин за месяц, или почему расходы выросли.

Смартфон всегда под рукой, и именно он может помочь решить эту проблему. Мобильное приложение позволяет быстро записывать траты, видеть статистику и напоминания о важных делах. Это удобнее и надёжнее, чем ручные записи.

Поэтому мною была выбрана тема для выпускной квалификационной работы: «Разработка мобильного приложения для учета расходов на содержание автомобиля».

Актуальность работы обусловлена растущей востребованностью мобильных приложений, а также необходимостью освоения новых технологий для разработки таких приложений.

Объект исследования: кроссплатформенное мобильное приложение.

Предмет исследования: особенности разработки приложения с использованием кроссплатформенного набора инструментов Flutter.

Цель работы – разработать мобильное приложение, которое поможет пользователю учитывать расходы на содержание автомобиля и анализировать их.

Для достижения этой цели нужно решить следующие задачи:

- изучить предметную область и существующие решения;
- определить требования к приложению и структуру данных;

- разработать интерфейс и основные функции приложения;
- протестировать работу и оценить удобство использования.

Результатом работы станет мобильное приложение, которое позволит водителю вести учёт расходов и лучше понимать, сколько стоит эксплуатация автомобиля. Это поможет планировать бюджет, избегать лишних трат и следить за техническим состоянием машины.

Такой проект не только практичен, но и актуален: тема финансового учёта и осознанного потребления сейчас особенно важна. А мобильный формат делает инструмент доступным и удобным для любого пользователя.

1 Аналитическая часть

1.1 Анализ предметной области

Сегодня почти у каждого есть машина. И чем дольше её держишь, тем больше понимаешь: расходы на неё растут. Бензин, масло, ремонт, страховка, налоги, парковка – всё это деньги, и они быстро исчезают, если не следить. Многие пытаются вести записи в блокнотах, Excel или заметках на телефоне. Но такие способы неудобны: забываешь записать, теряешь данные или просто лень вносить всё вручную.

Смартфон всегда под рукой, и логично, что контроль расходов можно перевести туда. Мобильное приложение может хранить все траты, считать итоги и показывать статистику в удобной форме. Например, сколько в месяц уходит на топливо или когда пора менять масло. Так пользователь видит полную картину затрат и может заранее планировать обслуживание.

На рынке уже есть подобные приложения. Например:

- Fuelio – считает расход топлива, пробег, затраты. Простое, но часть функций платная.
- Drivvo – умеет учитывать все виды расходов, но интерфейс сложный и перегружен.
- Simply Auto – синхронизирует данные между устройствами, но плохо переведён и требует платную подписку.

Эти приложения решают задачу частично, но у них есть минусы. Интерфейсы не всегда понятные, часть функций спрятана за платной подпиской, а поддержка русского языка или валюты – не везде. Часто пользователю просто нужно простое и бесплатное решение, где можно быстро внести расходы и увидеть понятную статистику.

Так что есть смысл сделать приложение, которое будет:

- простым и лёгким для понимания;
- сохранять данные прямо на телефоне (без регистрации и лишних шагов);

- показывать итоги расходов по категориям (топливо, обслуживание, страховка и т. д.);
- напоминать о важных вещах вроде ТО или продления страховки.

С точки зрения предметной области, это приложение работает с двумя основными типами данных – финансовыми (сумма, дата, категория) и временными (когда и что нужно сделать). Всё строится вокруг владельца автомобиля, который хочет иметь под рукой историю своих затрат и отчёты в удобной форме.

1.2 Обоснование выбора задачи

Идея вести учёт расходов не нова. Но проблема в том, что большинство решений либо слишком сложные, либо неудобные. Часто пользователю не нужен большой функционал – ему просто нужно понимать, сколько денег он тратит на машину.

Я выбрала именно эту задачу, потому что она реальна и полезна. Любой водитель сталкивается с ситуацией, когда не может точно сказать, сколько стоит ему содержание автомобиля. А когда все траты разбросаны по чекам и переводам, посчитать сложно.

Мобильное приложение – логичный способ решить эту проблему. Оно всегда с собой, не требует подключения к компьютеру и может хранить данные в одном месте.

Кроме того, проект не требует дорогих технологий – достаточно стандартного набора инструментов для мобильной разработки. Это делает задачу выполнимой в рамках выпускной квалификационной работы, но при этом – практической и применимой в жизни.

Таким образом, разработка такого приложения – это не просто учебное упражнение, а реальный продукт, который может помочь людям вести учёт своих расходов и экономить деньги.

1.3 Экономико-информационная сущность задачи

Любая система учёта работает с информацией. В нашем случае – с финансовыми данными владельца автомобиля. Задача приложения – собрать эти данные, сохранить и показать их в удобной форме.

Основная информация, с которой работает приложение:

- дата операции;
- категория расхода (топливо, ремонт, страховка и т. д.);
- сумма;
- описание или комментарий;
- пробег (если нужно считать расход топлива).

На основе этих данных можно строить отчёты – например, сколько потрачено за месяц, какие категории самые затратные, как меняется расход топлива со временем. Это уже не просто записи, а полезная аналитика, помогающая принимать решения.

Экономическая суть задачи в том, чтобы сделать процесс учёта затрат простым и прозрачным. Когда человек видит реальные суммы, он лучше понимает, где можно сократить расходы. Даже небольшая экономия в месяц может со временем превратиться в заметную сумму.

С информационной стороны приложение представляет собой небольшую базу данных с удобным интерфейсом. Оно объединяет ввод информации, хранение и визуализацию в одном месте. Это делает процесс контроля расходов понятным и наглядным.

2 Проектная часть

2.1 Информационное обеспечение задачи

Чтобы приложение могло работать, ему нужно хранить и обрабатывать данные о расходах. Основная информация – это дата, сумма, категория и описание каждой траты.

Пользователь добавляет новые записи вручную, выбирает тип расхода (например, топливо, ремонт, страховка, парковка и т. д.) и при желании оставляет комментарий.

Приложение сохраняет данные в локальную базу, чтобы пользователь мог работать без интернета. Это важно, потому что не везде есть стабильное соединение, а данные должны быть доступны всегда.

Информационные потоки простые:

- Ввод данных: пользователь добавляет запись о расходе.
- Хранение: запись сохраняется в базе данных устройства.
- Обработка: приложение считает итоги и формирует отчёты.
- Вывод: пользователь видит статистику и графики.

Структура данных может включать следующие таблицы:

- Расходы (id, дата, категория, сумма, комментарий, пробег);
- Категории (id, название, иконка);
- Напоминания (id, дата, тип события, описание).

Такой набор данных покрывает все основные сценарии: добавление, редактирование, просмотр и удаление записей.

2.1.1 Обоснование проектных решений по технологическому обеспечению

Для разработки выбрана технология кроссплатформенной мобильной разработки – Flutter. Этот фреймворк позволяет создавать одно приложение, которое работает и на Android, и на iOS [30].

Почему это удобно:

- одно кодовое основание – меньше времени на поддержку;
- одинаковый внешний вид на разных устройствах;
- большой выбор готовых библиотек и компонентов;
- быстрое тестирование и сборка.

В качестве среды разработки можно использовать Android Studio – он поддерживает Flutter и дает возможность быстро проверять приложение на эмуляторе или телефоне.

2.1.2 Технологические операции

Процесс создания программного приложения представляет собой последовательность взаимосвязанных технологических операций, каждая из которых направлена на достижение конкретного результата. Разделение процесса разработки на этапы позволяет систематизировать работу, повысить её качество и обеспечить контролируемый переход от идеи к готовому программному продукту [13].

Процесс создания приложения можно разделить на несколько этапов:

- Проектирование интерфейса**.** Определяются основные экраны: главный, добавление расхода, статистика, настройки.
- Создание структуры данных. Описываются таблицы и связи между ними.
- Реализация функционала. Разработка экранов, форм ввода, логики сохранения и отображения данных.
- Добавление аналитики. Реализация расчёта сумм и графиков по категориям.
- Тестирование. Проверка стабильности и удобства интерфейса.
- Сборка и установка. Формирование установочного файла для Android и iOS.

На первом этапе выполняется проектирование пользовательского интерфейса приложения. Определяется структура основных экранов и логика взаимодействия пользователя с системой. В рамках данного этапа разрабатываются следующие экраны:

- главный экран с общей информацией о расходах;
- экран добавления новой записи о расходе;
- экран статистики и аналитики;
- экран настроек приложения.

При проектировании интерфейса учитываются принципы удобства использования, наглядности и минимального количества действий, необходимых для выполнения основных операций. Интерфейс разрабатывается с ориентацией на конечного пользователя и адаптируется под различные размеры экранов мобильных устройств [13].

На втором этапе определяется структура хранения данных приложения. Описываются основные сущности, такие как расходы, категории расходов, даты и дополнительные параметры. Формируется логическая модель данных, включающая таблицы и связи между ними [13].

Структура данных проектируется таким образом, чтобы обеспечить:

- быстрый доступ к информации;
- корректное хранение истории расходов;
- возможность расширения функционала в дальнейшем.

Для хранения данных может использоваться локальная база данных, что позволяет приложению работать автономно, без постоянного подключения к сети Интернет [13].

Следующим этапом является реализация функциональной части приложения. На этом этапе осуществляется разработка экранов, форм ввода данных и программной логики.

Реализуются следующие функции:

- добавление, редактирование и удаление записей о расходах;
- сохранение данных в базе;

- отображение информации в удобном для пользователя виде;
- навигация между экранами приложения.

Особое внимание уделяется корректной обработке пользовательского ввода и предотвращению ошибок при работе с данными.

На этапе аналитики реализуются механизмы обработки накопленных данных. Приложение выполняет автоматический расчёт суммарных расходов за выбранный период, а также формирует статистику по категориям затрат [13].

Для наглядного представления информации используются диаграммы и графики, которые позволяют пользователю быстро оценить структуру расходов и выявить наиболее затратные категории. Аналитические функции повышают практическую ценность приложения и способствуют более осознанному управлению личным бюджетом.

После реализации основного функционала проводится тестирование приложения. Целью тестирования является выявление и устранение ошибок, а также оценка стабильности и удобства пользовательского интерфейса.

В процессе тестирования проверяется:

- корректность работы всех функций;
- устойчивость приложения к ошибочному вводу данных;
- удобство навигации и восприятия интерфейса;
- стабильность работы на различных устройствах и версиях операционных систем.

Результаты тестирования используются для доработки и оптимизации приложения.

Заключительным этапом является сборка приложения и подготовка его к установке на мобильные устройства. Формируется установочный файл для платформ Android и iOS, после чего выполняется установка и финальная проверка работоспособности приложения [29].

На данном этапе также подготавливаются необходимые файлы конфигурации и проводится проверка соответствия приложения требованиям мобильных платформ.

Эти шаги позволяют перейти от идеи к работающему приложению без сложных технологий и долгих циклов разработки.

Таким образом, последовательное выполнение технологических операций позволяет организовать процесс разработки приложения без использования сложных технологий и длительных циклов разработки. Применяемый подход обеспечивает получение работоспособного, удобного и надёжного программного продукта, соответствующего поставленным требованиям [5].

2.2 Технологическое обеспечение

Для функционирования разработанного приложения не требуется специализированного или дорогостоящего оборудования. Программный продукт ориентирован на массового пользователя и предназначен для работы на стандартных мобильных устройствах.

Для использования приложения достаточно смартфона с установленной операционной системой Android или iOS. Минимальные системные требования соответствуют базовым возможностям современных мобильных устройств, что обеспечивает широкую доступность приложения для пользователей. Приложение не использует ресурсоёмкие вычисления, графические эффекты или сложные фоновые процессы, благодаря чему стабильно работает даже на устройствах бюджетного сегмента [21].

Программный продукт не предъявляет повышенных требований к объёму оперативной памяти, производительности процессора и объёму внутреннего хранилища. Все операции выполняются локально либо с минимальной нагрузкой на систему, что положительно сказывается на времени отклика интерфейса и общем удобстве использования.

Для корректной работы приложения требуется:

- смартфон или планшет;
- операционная система Android версии 8.0 и выше либо iOS версии 13 и выше;

- не менее 1 ГБ оперативной памяти;
- свободное место во внутренней памяти устройства не менее 50 МБ.

Указанные требования являются минимальными и соответствуют большинству мобильных устройств, находящихся в эксплуатации в настоящее время.

Для разработки, отладки и тестирования приложения требуется стандартное рабочее место разработчика. В качестве аппаратного обеспечения используется персональный компьютер или ноутбук со следующими характеристиками:

- процессор с тактовой частотой не ниже 2,0 ГГц;
- не менее 8 ГБ оперативной памяти;
- жёсткий диск или SSD с объёмом свободного пространства не менее 20 ГБ.

В качестве операционной системы может использоваться Windows, macOS или Linux, что обеспечивает гибкость и независимость от конкретной платформы [19].

Для создания приложения применяются современные инструменты разработки программного обеспечения:

- среда разработки Android Studio или Visual Studio Code, обеспечивающие удобную работу с исходным кодом;
- программные библиотеки и фреймворки, необходимые для создания пользовательского интерфейса и логики приложения;
- встроенный эмулятор мобильных устройств либо физическое мобильное устройство для тестирования функциональности приложения.

Использование эмулятора позволяет проверять работу приложения на различных версиях операционных систем и с разными параметрами экранов, а физическое устройство обеспечивает тестирование в реальных условиях эксплуатации.

Таким образом, технологическое обеспечение разработанного приложения не требует специализированных технических средств и основано

на общедоступных аппаратных и программных ресурсах. Минимальные системные требования, кроссплатформенность и низкое потребление ресурсов делают приложение удобным в использовании и доступным для широкого круга пользователей, а также упрощают процесс его разработки и дальнейшего сопровождения [26].

2.2.1 Обоснование проектных решений по технологическому обеспечению

При выборе технологических решений для разработки приложения учитывались требования доступности, универсальности, экономической целесообразности и удобства эксплуатации программного продукта. Основной целью являлось создание приложения, которое может использоваться максимально широким кругом пользователей без необходимости приобретения специализированного оборудования или программного обеспечения [27].

Выбор мобильной платформы обусловлен тем, что в настоящее время смартфоны являются наиболее распространёнными персональными вычислительными устройствами. Большинство пользователей постоянно имеют мобильное устройство при себе, что делает использование мобильного приложения для учёта расходов наиболее удобным и практичным. Мобильный формат позволяет оперативно вносить данные, получать актуальную информацию и анализировать расходы в любое время и в любом месте.

В качестве основной целевой платформы выбран Android, так как данная операционная система занимает значительную долю рынка мобильных устройств. Android используется на широком спектре смартфонов различных ценовых категорий, что обеспечивает высокую доступность приложения для пользователей с разным уровнем дохода. Кроме того, платформа Android не требует приобретения платных лицензий для разработки, тестирования и публикации приложений, что особенно важно в рамках учебного проекта и при ограниченных ресурсах [14].

Существенным преимуществом Android является наличие развитой экосистемы инструментов разработки, подробной документации и большого сообщества разработчиков. Это упрощает процесс создания приложения, поиска решений возникающих технических задач и дальнейшего сопровождения программного продукта.

В качестве технологического подхода к разработке рассматривается использование кроссплатформенного фреймворка Flutter. Применение Flutter позволяет создавать единый программный код, который может быть использован для сборки приложения как под Android, так и под iOS. Такой подход существенно сокращает трудозатраты на разработку и поддержку приложения, а также повышает его масштабируемость [17].

Использование Flutter обеспечивает:

- единый пользовательский интерфейс на разных платформах;
- высокую производительность за счёт компиляции в нативный код;
- возможность быстрого внесения изменений и доработок;
- расширение аудитории пользователей за счёт поддержки iOS.

Таким образом, выбранные проектные решения по технологическому обеспечению являются обоснованными и соответствуют современным тенденциям разработки мобильных приложений. Они позволяют создать универсальный, доступный и экономически целесообразный программный продукт, который может быть легко адаптирован и расширен в дальнейшем [25].

2.2.2 Комплекс технологических средств

Комплекс технологических средств включает совокупность аппаратного и программного обеспечения, необходимого для разработки, тестирования и эксплуатации программного продукта. Выбор технологических средств осуществлялся с учётом требований доступности, производительности, совместимости и экономической целесообразности [28].

Для разработки мобильного приложения используется стандартное рабочее место разработчика, включающее персональный компьютер или ноутбук. Минимальные требования к аппаратному обеспечению обеспечивают комфортную работу среды разработки и эмулятора мобильных устройств.

Для разработки необходимы следующие аппаратные средства:

- персональный компьютер или ноутбук;
- объём оперативной памяти не менее 8 ГБ, обеспечивающий стабильную работу среды разработки;
- процессор уровня Intel Core i5 или аналогичный процессор AMD, обеспечивающий достаточную вычислительную производительность;
- жёсткий диск или SSD с объёмом свободного пространства не менее 20 ГБ.

В качестве программного обеспечения для разработки используется интегрированная среда разработки Android Studio, предоставляющая полный набор инструментов для создания, отладки и сборки мобильных приложений.

Дополнительно устанавливаются:

- Android SDK, содержащий необходимые библиотеки и инструменты для разработки;
- эмулятор Android-устройств, позволяющий тестировать приложение на различных версиях операционной системы и с разными характеристиками экранов.

Использование эмулятора позволяет выявлять ошибки на ранних этапах разработки и снижает необходимость постоянного использования физических устройств [18].

Для эксплуатации разработанного приложения требуется мобильное устройство на базе операционной системы Android. Приложение ориентировано на массового пользователя и не предъявляет повышенных требований к аппаратным ресурсам.

Минимальные требования к устройству пользователя включают:

- смартфон с операционной системой Android версии 8.0 и выше;

- свободное место во внутренней памяти устройства в объёме 50–100 МБ;
- экран с минимальным разрешением 720p, обеспечивающий корректное отображение пользовательского интерфейса.

Приложение оптимизировано для работы на устройствах различных ценовых категорий и корректно функционирует на смартфонах со средними и базовыми техническими характеристиками.

Таким образом, комплекс технологических средств, используемых для разработки и эксплуатации приложения, основан на общедоступном и широко распространённом аппаратном и программном обеспечении. Это снижает затраты на реализацию проекта, упрощает процесс разработки и обеспечивает удобство использования приложения для конечного пользователя [23].

2.3 Программное обеспечение задачи

Программное обеспечение разработанного приложения представляет собой совокупность программных модулей, обеспечивающих выполнение всех функциональных задач по учёту, хранению и анализу расходов пользователя. Архитектура программного обеспечения построена по модульному принципу, что упрощает разработку, тестирование и дальнейшее сопровождение приложения.

Программное обеспечение включает:

- интерфейс приложения – экраны и элементы управления;
- модуль работы с базой данных – хранение и поиск записей;
- модуль аналитики – подсчёт расходов, построение диаграмм;
- система уведомлений – напоминания о предстоящих событиях

Пользовательский интерфейс является основным средством взаимодействия пользователя с приложением. Он включает набор экранов и элементов управления, обеспечивающих удобный и интуитивно понятный доступ ко всем функциям системы [24].

Интерфейс приложения содержит следующие основные экраны:

- главный экран с общей информацией о расходах;
- экран добавления и редактирования записей;
- экран аналитики и статистики;
- экран настроек приложения.

Элементы управления включают кнопки, поля ввода, выпадающие списки и элементы навигации. При проектировании интерфейса соблюдаются принципы простоты, наглядности и минимального количества действий, необходимых для выполнения основных операций.

Модуль работы с базой данных отвечает за хранение, обновление и поиск информации о расходах пользователя. В рамках приложения используется локальная база данных, что позволяет работать с приложением без подключения к сети Интернет [20].

Функции модуля включают:

- сохранение данных о расходах и категориях;
- выборку данных по заданным параметрам (дата, категория, сумма);
- обновление и удаление существующих записей;
- обеспечение целостности и корректности данных.

Использование локального хранилища повышает производительность приложения и обеспечивает конфиденциальность пользовательской информации.

Модуль аналитики предназначен для обработки накопленных данных и формирования аналитической информации. Он выполняет автоматический подсчёт расходов за выбранные периоды времени и по различным категориям [20].

Основные функции модуля аналитики:

- расчёт суммарных расходов за день, месяц и год;
- группировка данных по категориям;
- построение диаграмм и графиков;
- визуализация структуры расходов.

Наглядное представление данных позволяет пользователю быстро оценить финансовую ситуацию и принимать обоснованные решения по оптимизации расходов.

Система уведомлений предназначена для информирования пользователя о предстоящих событиях, связанных с эксплуатацией автомобиля и финансовыми обязательствами. Уведомления выводятся в виде стандартных системных сообщений мобильного устройства.

Система уведомлений обеспечивает:

- напоминания о прохождении технического обслуживания;
- оповещения о регулярных расходах;
- уведомления о приближении запланированных событий.

Использование уведомлений повышает практическую ценность приложения и способствует своевременному выполнению пользователем необходимых действий.

Таким образом, программное обеспечение задачи включает набор взаимосвязанных модулей, обеспечивающих полный цикл работы с данными – от ввода и хранения информации до её анализа и представления пользователю. Модульная структура приложения повышает его надёжность, удобство использования и возможности дальнейшего развития [15].

2.3.1 Обоснование проектных решений по программному обеспечению

При выборе программных решений для разработки приложения основное внимание уделялось таким критериям, как простота использования, стабильность работы, высокая производительность и возможность дальнейшего расширения функционала. Программное обеспечение должно обеспечивать корректную работу приложения на мобильных устройствах с различными техническими характеристиками и не создавать избыточной нагрузки на систему.

Для реализации программной архитектуры приложения выбрана модель Model–View–ViewModel (MVVM). Данная архитектура предполагает чёткое разделение приложения на три логических уровня:

- Model – уровень данных и бизнес-логики, отвечающий за хранение и обработку информации;
- View – уровень пользовательского интерфейса, отображающий данные и принимающий ввод от пользователя;
- ViewModel – связующий уровень, обеспечивающий взаимодействие между моделью и представлением.

Использование архитектуры MVVM позволяет повысить читаемость и структурированность программного кода, упростить тестирование отдельных компонентов и снизить связанность между элементами системы. Благодаря разделению логики и интерфейса упрощается сопровождение приложения и внедрение новых функций без значительных изменений существующего кода [24].

В качестве средства хранения данных выбрана база данных SQLite. Данное решение обусловлено тем, что SQLite является встроенной в операционную систему Android и не требует установки дополнительных серверных компонентов. Использование локальной базы данных обеспечивает высокую скорость работы приложения и возможность его функционирования в автономном режиме [3].

При использовании кроссплатформенного фреймворка Flutter для работы с SQLite применяется библиотека sqflite, которая предоставляет удобный интерфейс для выполнения операций создания таблиц, добавления, обновления и выборки данных. Данная библиотека широко используется, имеет хорошую документацию и поддерживается сообществом разработчиков [27].

Приложение спроектировано таким образом, что все основные функции доступны в офлайн-режиме. Это является важным преимуществом, поскольку пользователь может вести учёт расходов без подключения к сети Интернет.

Хранение данных локально также повышает уровень конфиденциальности пользовательской информации.

При этом архитектура приложения допускает возможность дальнейшего расширения функциональности, в частности добавления механизма резервного копирования данных в облачное хранилище. Такое решение может быть реализовано в будущем без существенной переработки архитектуры и позволит повысить надёжность хранения данных.

Таким образом, выбранные проектные решения по программному обеспечению являются обоснованными и соответствуют современным требованиям к мобильным приложениям. Использование архитектуры MVVM, локальной базы данных SQLite и офлайн-режима работы обеспечивает стабильность, быстродействие и удобство использования приложения, а также создаёт основу для его дальнейшего развития [23].

2.3.2 Архитектура программного обеспечения

Архитектура программного обеспечения разработанного приложения построена на принципах разделения ответственности и модульности. Такой подход позволяет повысить надёжность системы, упростить сопровождение кода и обеспечить возможность дальнейшего расширения функциональности без существенных изменений существующей структуры.

В основе архитектуры приложения лежит многоуровневая модель, включающая три основных логических слоя: слой данных, слой бизнес-логики и слой представления. Каждый слой выполняет строго определённые функции и взаимодействует с другими слоями через чётко определённые интерфейсы [22].

Архитектура приложения построена по принципу разделения ответственности и включает три основных слоя:

- слой данных, отвечающий за работу с локальной базой данных SQLite;

- слой бизнес-логики, реализованный с использованием паттерна Provider;
- слой представления, включающий экраны и пользовательские интерфейсы.

Такой подход обеспечивает модульность, улучшает читаемость кода и облегчает дальнейшее расширение функциональности приложения.

Слой данных отвечает за хранение, получение и обновление информации, используемой приложением. В рамках данного слоя реализована работа с локальной базой данных SQLite, которая обеспечивает надёжное и быстрое хранение пользовательских данных.

Функции слоя данных включают:

- создание и инициализацию структуры базы данных;
- выполнение операций добавления, чтения, обновления и удаления данных;
- обеспечение целостности и корректности хранимой информации.

Доступ к базе данных осуществляется через специализированные классы и методы, что изолирует слой хранения данных от остальных компонентов приложения и повышает безопасность и стабильность работы системы.

Слой бизнес-логики реализует основные правила обработки данных и управления состоянием приложения. В данном проекте бизнес-логика построена с использованием паттерна Provider, который широко применяется в разработке приложений на Flutter [28].

Provider обеспечивает централизованное управление состоянием приложения и позволяет эффективно организовать обмен данными между слоем данных и слоем представления. Использование данного паттерна снижает связанность компонентов и упрощает обновление пользовательского интерфейса при изменении данных.

К функциям слоя бизнес-логики относятся:

- обработка пользовательских действий;
- взаимодействие со слоем данных;

- выполнение вычислений и аналитических операций;
- управление состоянием экранов приложения.

Слой представления отвечает за визуальное отображение информации и взаимодействие с пользователем. Он включает набор экранов и элементов пользовательского интерфейса, реализованных с использованием стандартных компонентов мобильной платформы.

В данном слое реализуются:

- главный экран приложения;
- формы ввода и редактирования данных;
- экраны аналитики и статистики;
- элементы навигации и управления.

Слой представления не содержит бизнес-логики и работает исключительно с данными, предоставляемыми слоем бизнес-логики, что повышает читаемость кода и упрощает модификацию интерфейса без затрагивания логики приложения.

Взаимодействие между слоями осуществляется последовательно:

- пользователь выполняет действие в пользовательском интерфейсе;
- слой представления передаёт запрос в слой бизнес-логики;
- слой бизнес-логики обрабатывает запрос и при необходимости обращается к слою данных;
- полученные данные передаются обратно в слой представления для отображения.

Такой механизм взаимодействия обеспечивает чёткое разделение обязанностей и предотвращает прямую зависимость между слоями.

Таким образом, выбранная архитектура программного обеспечения обеспечивает модульность, расширяемость и устойчивость приложения. Использование разделения на слои, а также применение паттерна Provider позволяют создать структурированную и легко сопровождаемую систему, соответствующую современным требованиям к мобильным приложениям [16].

2.3.3 Описание программных модулей

- Модуль расходов. Добавление, редактирование и удаление записей.
- Модуль категорий. Управление типами затрат (топливо, ремонт, страховка).
- Модуль статистики. Построение диаграмм и расчёт итогов.
- Модуль напоминаний. Отображение уведомлений о событиях (ТО, страховка).
- Модуль настроек. Изменение валюты, темы оформления, резервное копирование.

После проверки необходимых системных требований и установки Flutter, Android Studio, SQLlite Studio, а также всех необходимых плагинов. Приступаем к созданию самого приложения.

Первым делом открываем Android Studio и выбираем «New Flutter Project», как показано на рисунке 1.

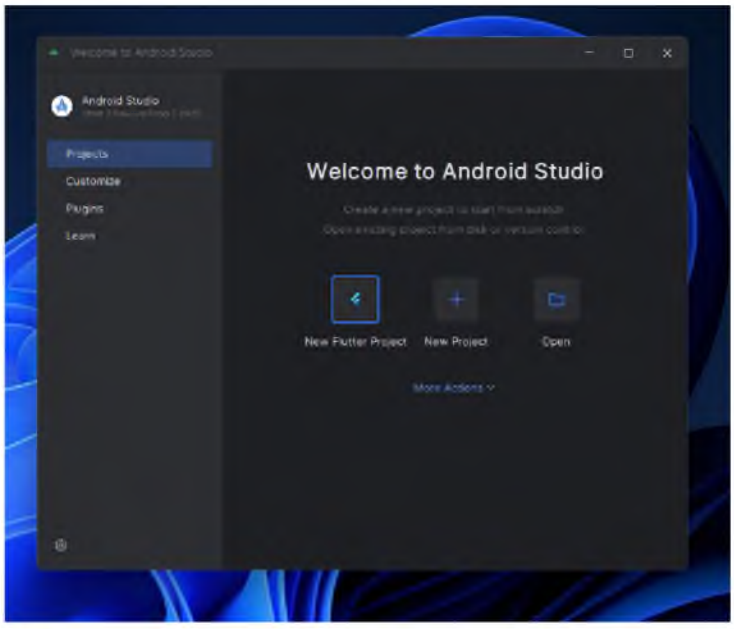

Рисунок 1 – Начало создания проекта

Далее необходимо указать путь к корневой папке Flutter, указываем путь (рисунок 2).

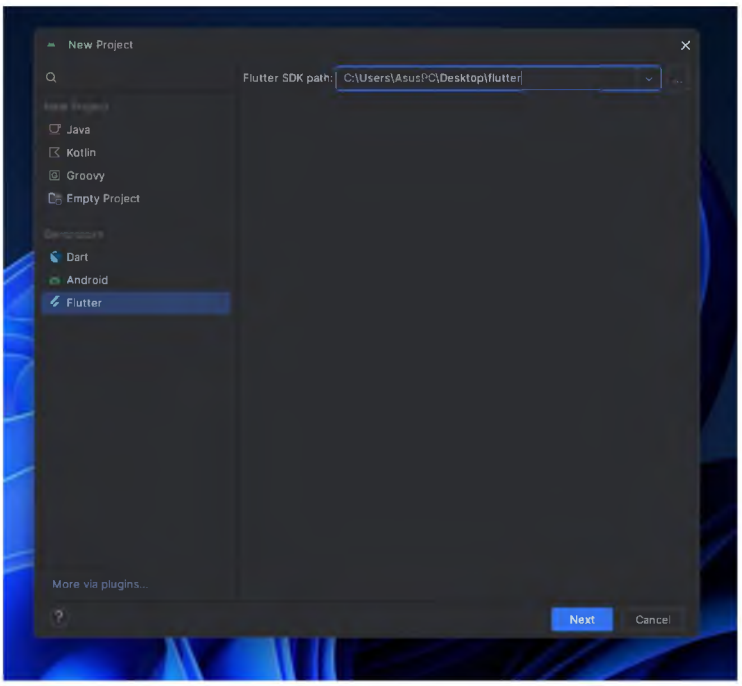

Рисунок 2 – Путь к корневой папкеFlutter

На финальном этапе настройки необходимо указать имя проекта, язык программирования для разработки, а также платформы, на которые будет разрабатываться проект. Это показано ниже на рисунке 3.

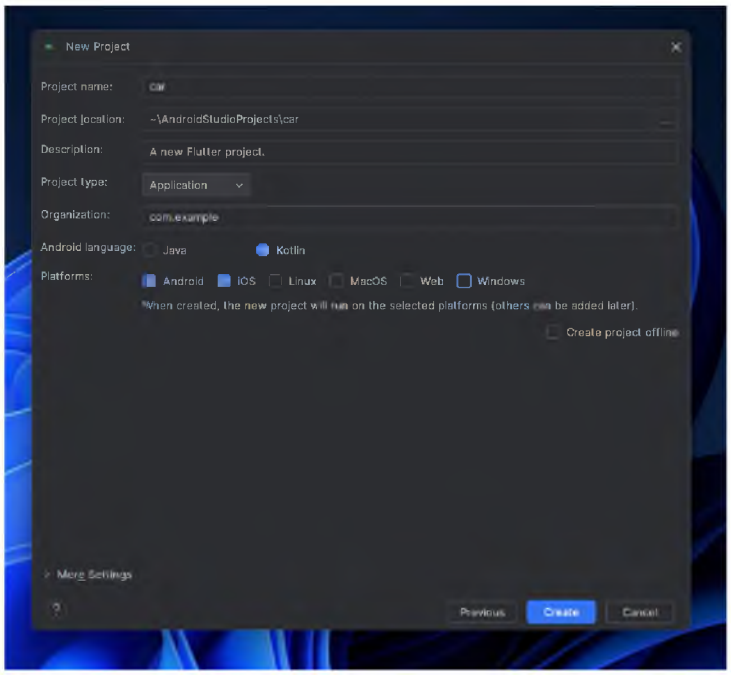

Рисунок 3 – Финальная настройка проекта

В итоге перед нами будет окно, как показано на рисунке 4.

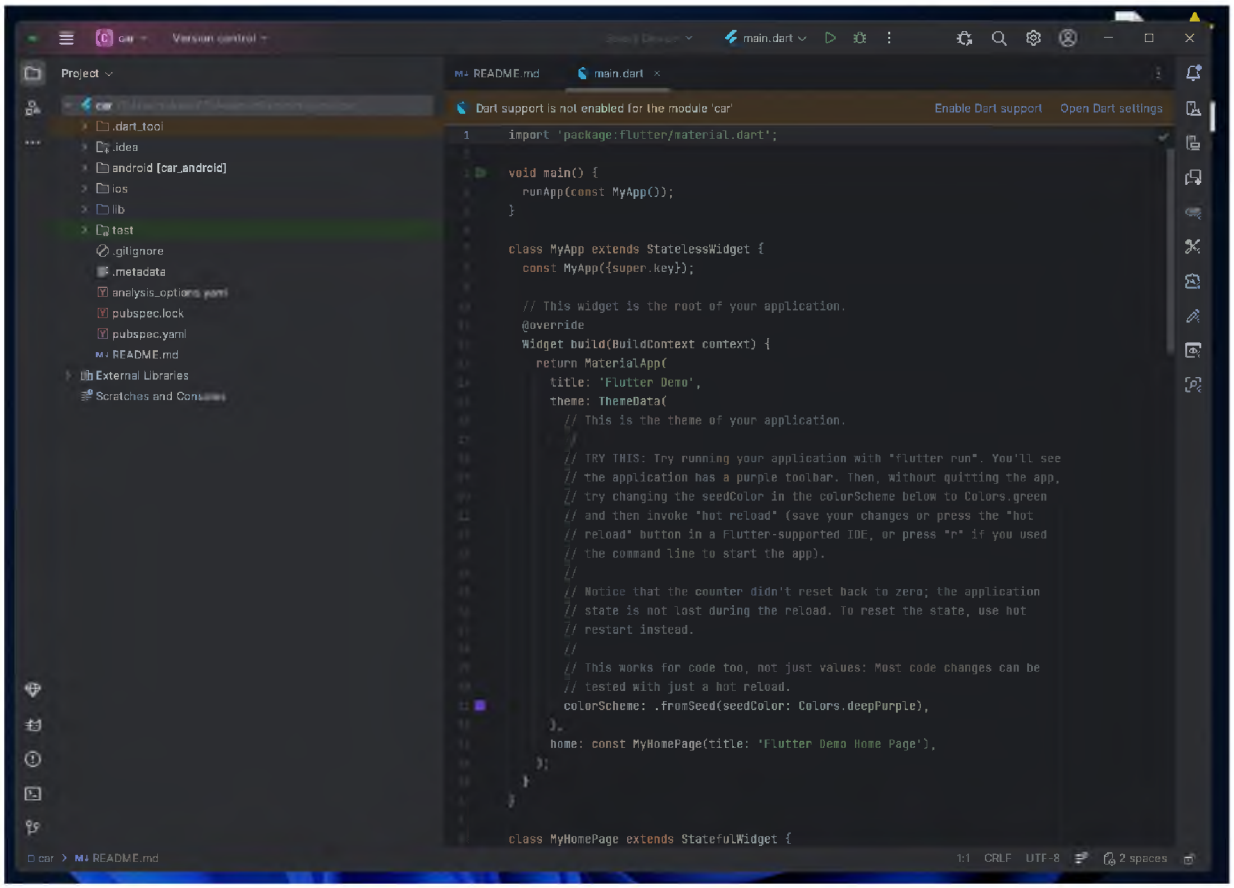

Рисунок 4 – Итог создания проекта

Для начала основного проектирования нужно создать структуру проекта. Структура проекта – это основа, это скелет программы. Структура для моего проекта показана на рисунке 5. В ней такое разделение:

- data – работа с БД
- providers – состояние
- screens – экраны
- widgets – переиспользуемые UI-компоненты

Где, например, lib/ – папка, main.dart – файл. Для создания папки или файла необходимо:

- Нажать ПКМ на папке, где необходимо создать объект;
- Откроется контекстное меню, в нем выбрать «File» (файл) или «Directory» (папка) в зависимости от того, что необходимо создать;
- Ввести название объекта.

Более детальное создание показано на рисунках 6 и 7.

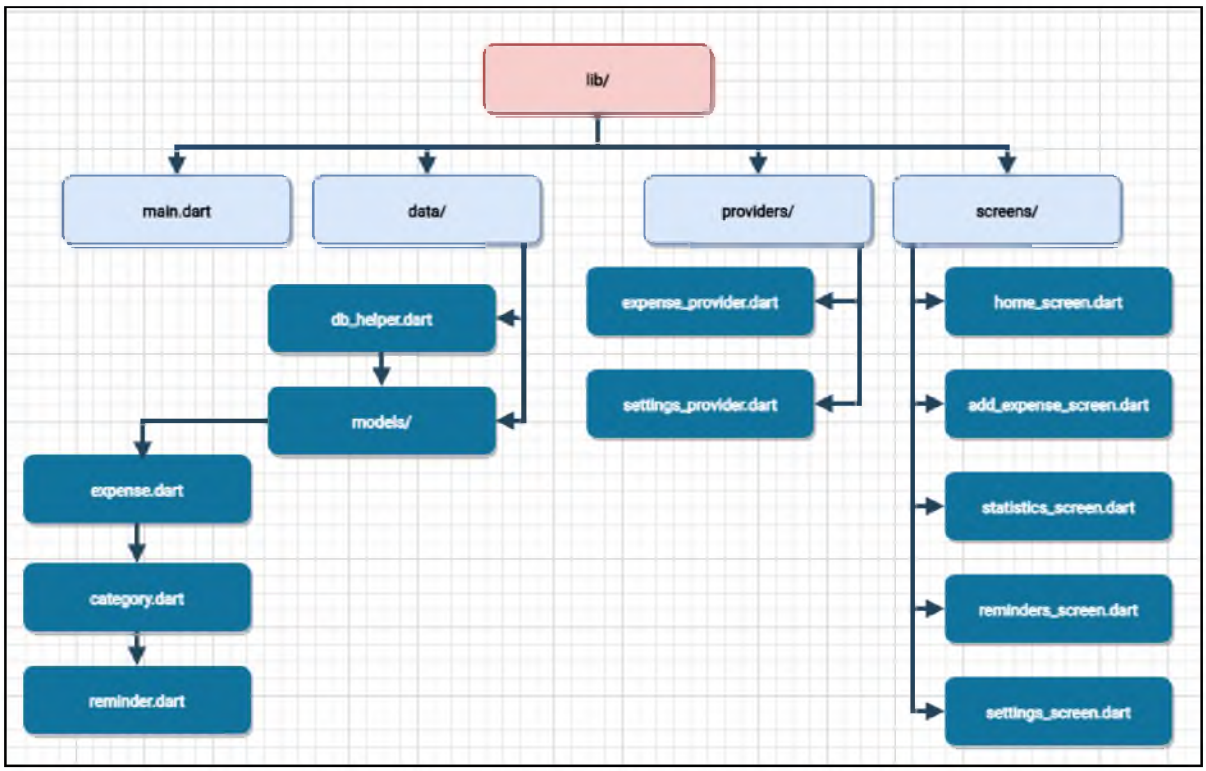

Рисунок 5 – Структура проекта

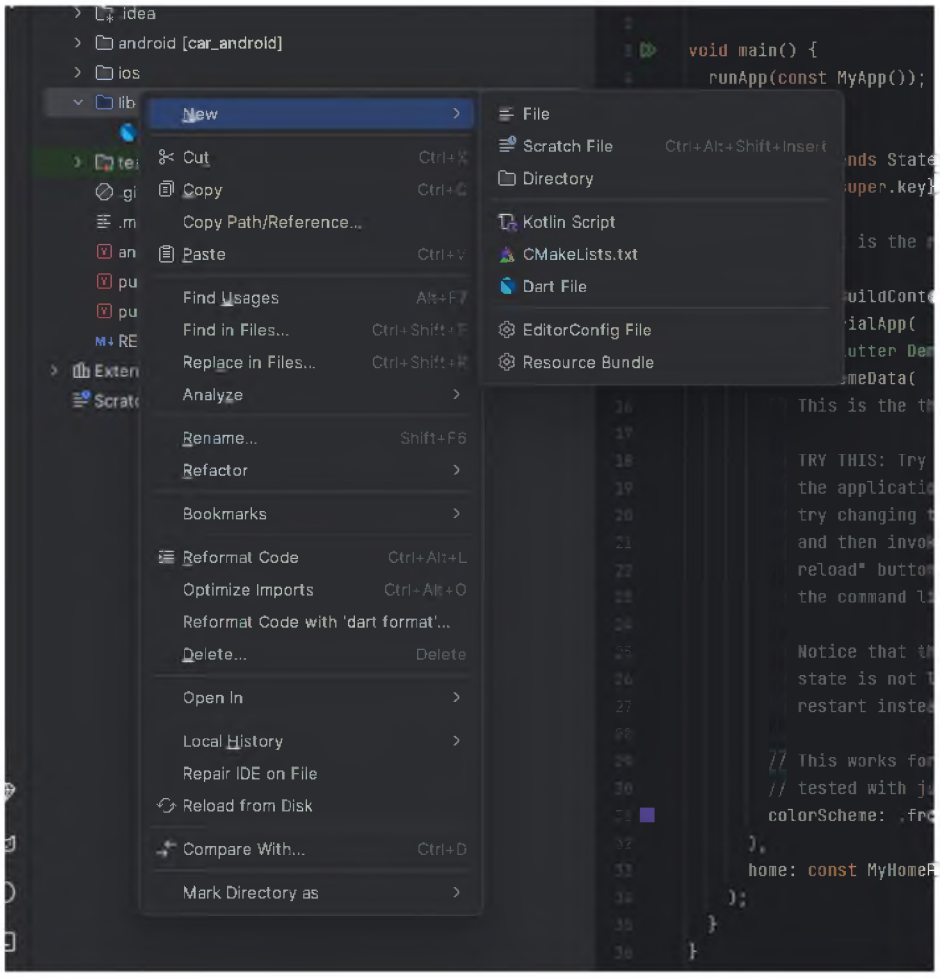

Рисунок 6 – Первая часть создания объекта

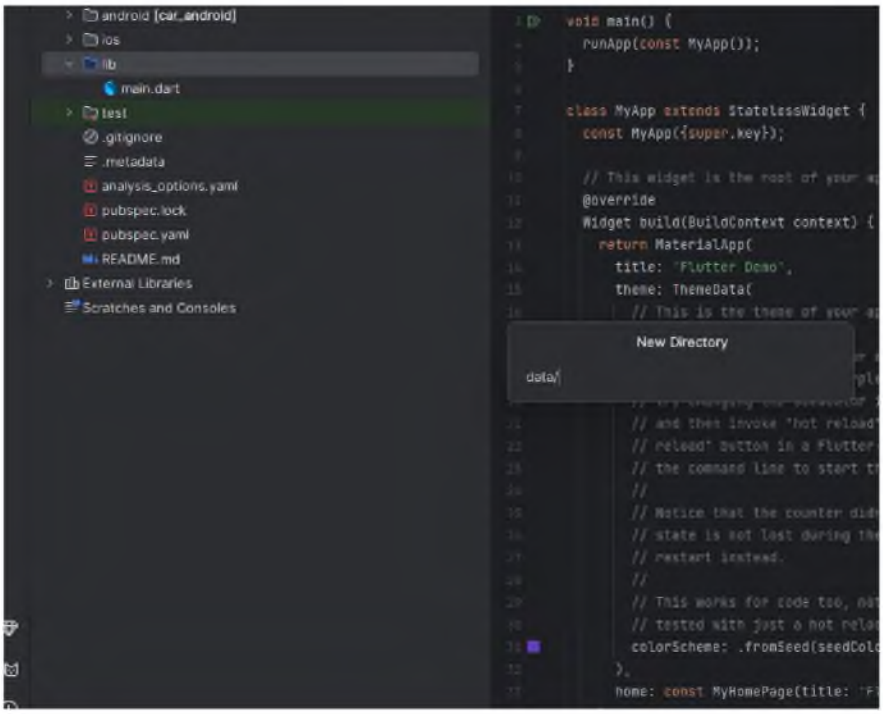

Рисунок 7 – Вторая часть создания объекта

После создания всех необходимых объектов, готовая структура выглядит так (рисунок 8).

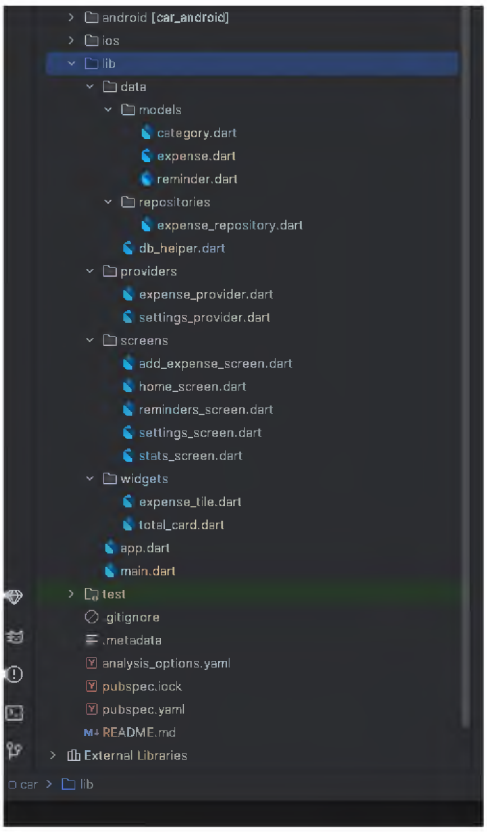

Рисунок 8 – Готовая структура проекта

Далее необходимо настроить зависимости. Для этого открываем файл «pubspec.yaml» и указываем версии плагинов (рисунок 9). Для сохранения файла нажимаем кнопку «Pub get». Полный программный код можно увидеть в приложении 1.

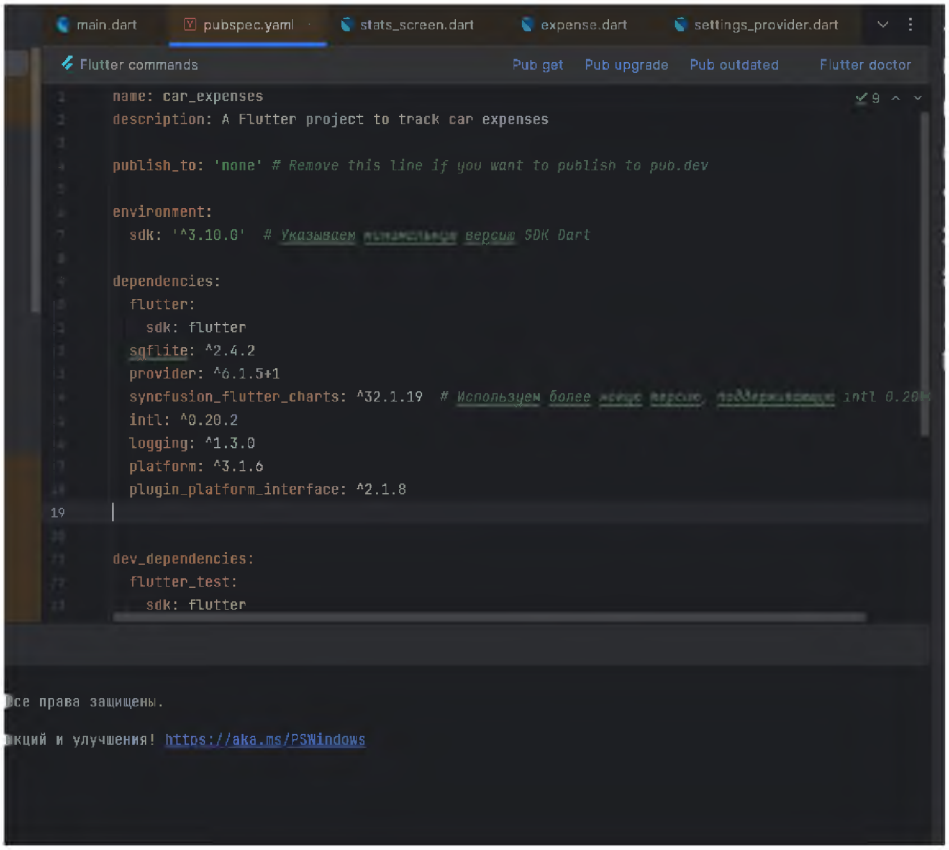

Рисунок 9 – Подключение зависимостей

Далее для корректной работы в дальнейшем нужно создать базу данных, а также подключить ее к проекту.

Для наглядного проектирования и проверки структуры базы данных на этапе разработки использовалась программа SQLiteStudio – графический инструмент для работы с базами данных SQLite. Данный инструмент позволяет создавать таблицы, выполнять SQL-запросы и визуально контролировать структуру базы данных [3].

Процесс создания базы данных выполнялся следующим образом:

- Запуск программы SQLiteStudio.
- В главном меню выбор пункта Database → Add a database.

- В открывшемся окне указание имени файла базы данных, например car_expenses.db, и выбор места его сохранения.
- После подтверждения база данных добавляется в список доступных баз в левой части окна программы.

После создания файла базы данных производится создание таблиц с помощью SQL-запросов.

Для хранения информации о финансовых затратах создаётся таблица «expenses». Таблица содержит основные данные о каждом расходе.

SQL-запрос создания таблицы:

```
CREATE TABLE expenses (
id INTEGER PRIMARY KEY AUTOINCREMENT,
date TEXT NOT NULL,
category TEXT NOT NULL,
amount REAL NOT NULL,
comment TEXT,
mileage INTEGER
);
```

Назначение полей:

- id – уникальный идентификатор записи;
- date – дата совершения расхода;
- category – категория расхода;
- amount – сумма затрат;
- comment – дополнительная информация;
- mileage – пробег автомобиля на момент расхода.

Для хранения категорий расходов создаётся таблица «categories».

SQL-запрос создания таблицы:

```
CREATE TABLE categories (
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT NOT NULL,
icon TEXT
```

);

Таблица позволяет централизованно управлять категориями и использовать их в пользовательском интерфейсе.

Для реализации функциональности напоминаний создаётся таблица «reminders».

SQL-запрос создания таблицы:

```
CREATE TABLE reminders (
id INTEGER PRIMARY KEY AUTOINCREMENT,
date TEXT NOT NULL,
type TEXT NOT NULL,
comment TEXT
);
```

Данная таблица используется для хранения информации о будущих событиях, связанных с обслуживанием автомобиля.

Созданная структура базы данных используется в мобильном приложении и полностью соответствует требованиям поставленной задачи.

Для хранения информации о расходах и событиях обслуживания автомобиля была спроектирована реляционная база данных SQLite, состоящая из трёх взаимосвязанных таблиц.

Таблица расходов предназначена для хранения информации обо всех финансовых операциях. Она содержит следующие поля: уникальный идентификатор записи, дату расхода, категорию, сумму, комментарий и пробег автомобиля на момент совершения расхода. Наличие поля пробега позволяет в дальнейшем рассчитывать средний расход топлива.

Таблица категорий используется для хранения пользовательских категорий расходов. Выделение категорий в отдельную таблицу упрощает масштабирование приложения и изменение пользовательского интерфейса.

Таблица напоминаний предназначена для хранения информации о будущих событиях, связанных с обслуживанием автомобиля, таких как техническое обслуживание, замена масла или продление страховки.

Спроектированная структура базы данных является минимальной, но достаточной для реализации всего заявленного функционала.

Чтобы подключить созданную базу данных к проекту, нужно зайти в файл «db_helper.dart» и написать код (рисунок 10). Полный программный код можно увидеть в приложении 2.

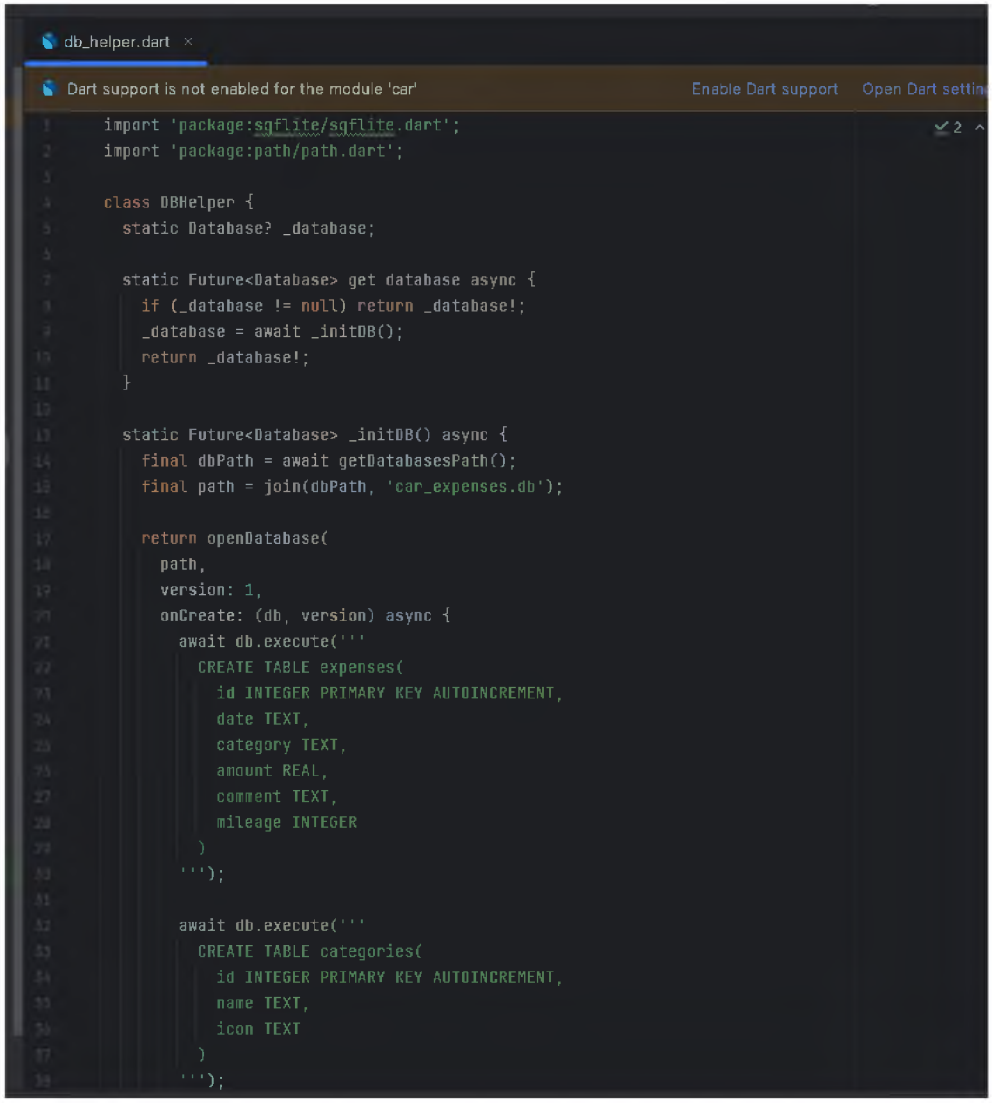

Рисунок 10 – Подключение БД к проекту

Следующий важный шаг – это создание моделей для расходов, категорий и напоминаний. Для этого необходимо зайти в «expense.dart» (рисунок 11), «category.dart» (рисунок 12), «reminder.dart» (рисунок 13) и ввести следующие программные коды. Полный программный код можно увидеть в приложениях 3, 4.

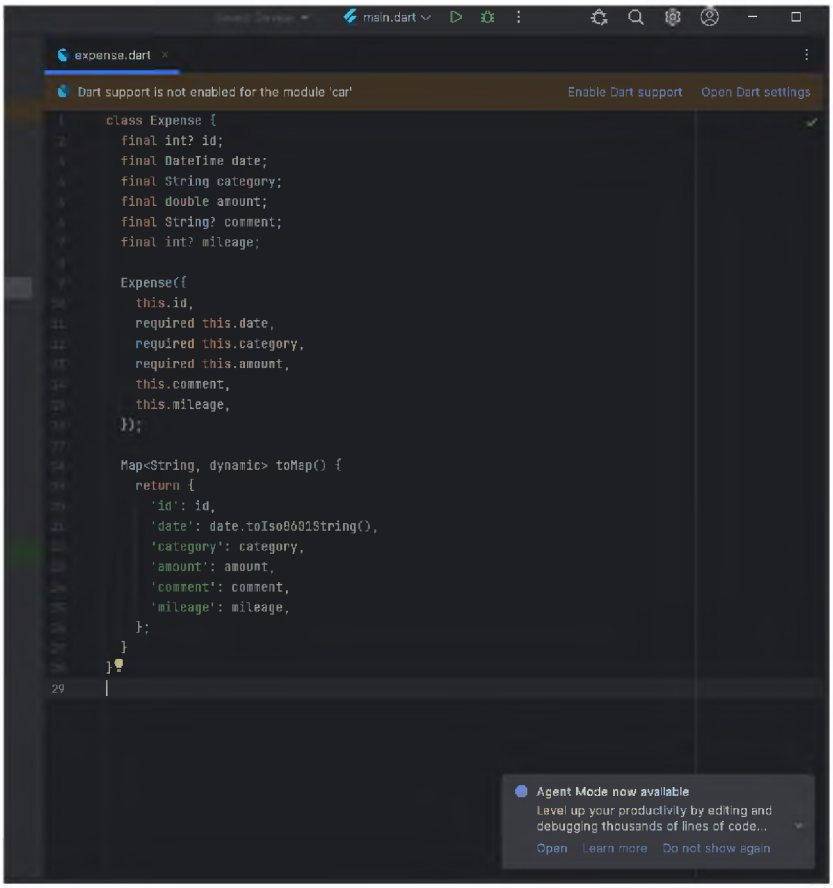

Рисунок 11 - Expense.dart

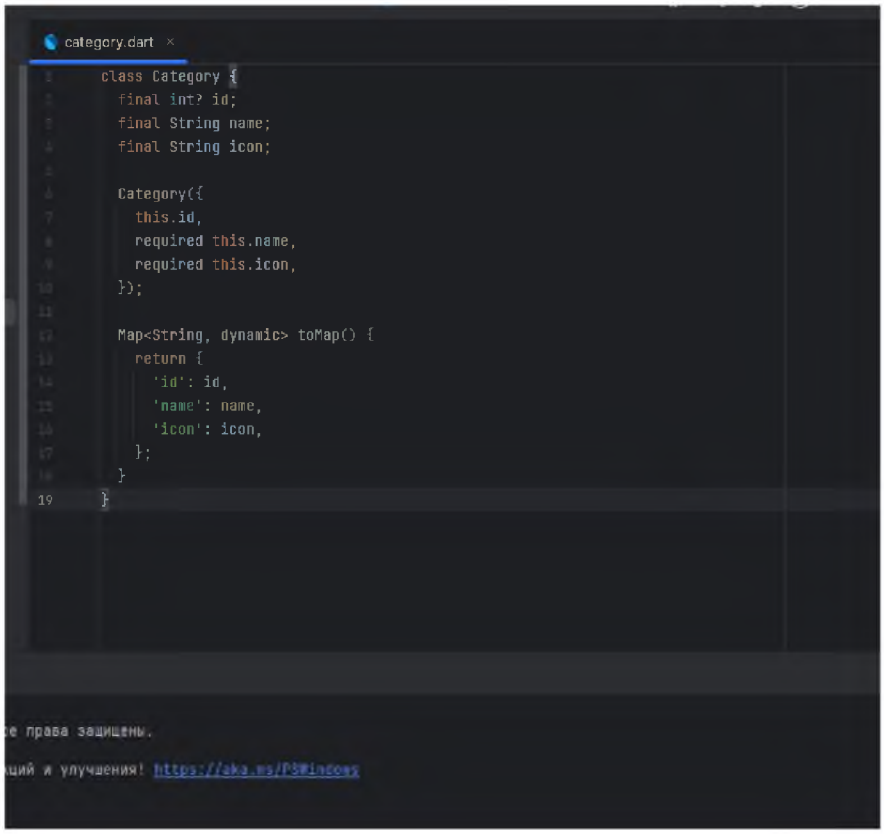

Рисунок 12 - Category.dart

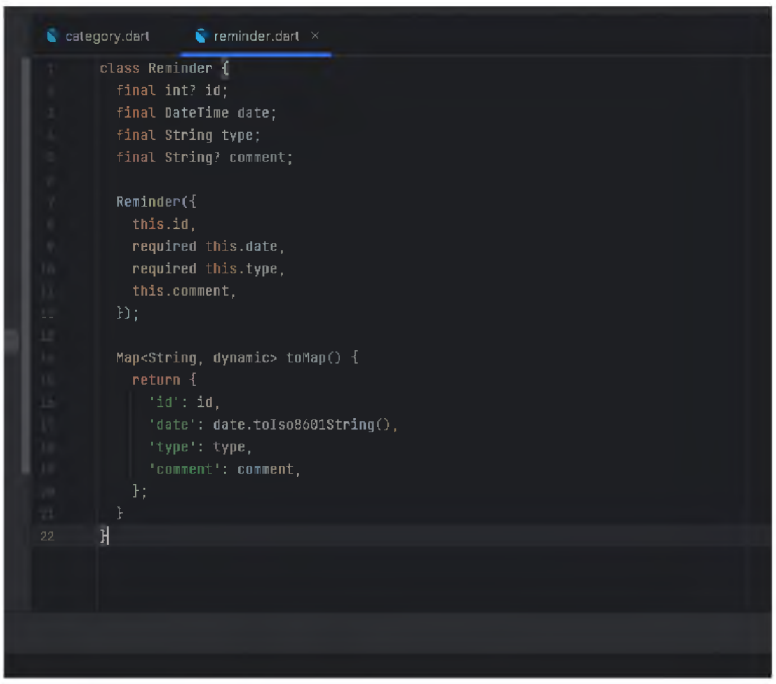

Рисунок 13 - Reminder.dart

После нужно заполнить «Provider». Provider - это связка базы данных и интерфейса. Заполненные файлы показаны ниже на рисунках 14 и 15. Полный программный код можно увидеть в приложениях 5 и 6.

Рисунок 14 - Expense_provider.dart

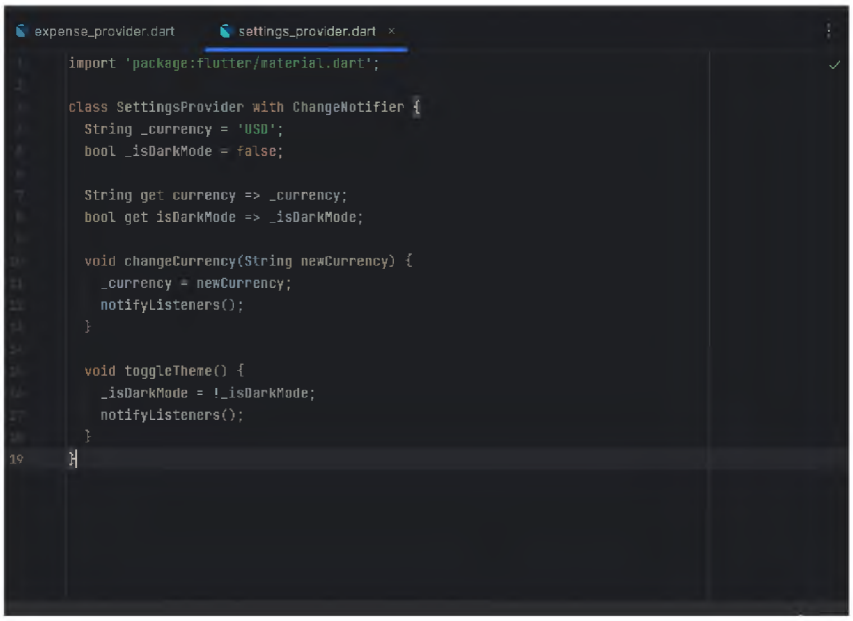

Рисунок 15 - Settings_provider.dart

Далее создаем необходимые экраны.

Главный экран отображает список последних расходов и общую сумму затрат за текущий месяц. Его основная задача – предоставить пользователю быстрый обзор текущих расходов без необходимости перехода в другие разделы приложения (рисунок 16). Полный программный код можно увидеть в приложении 7.

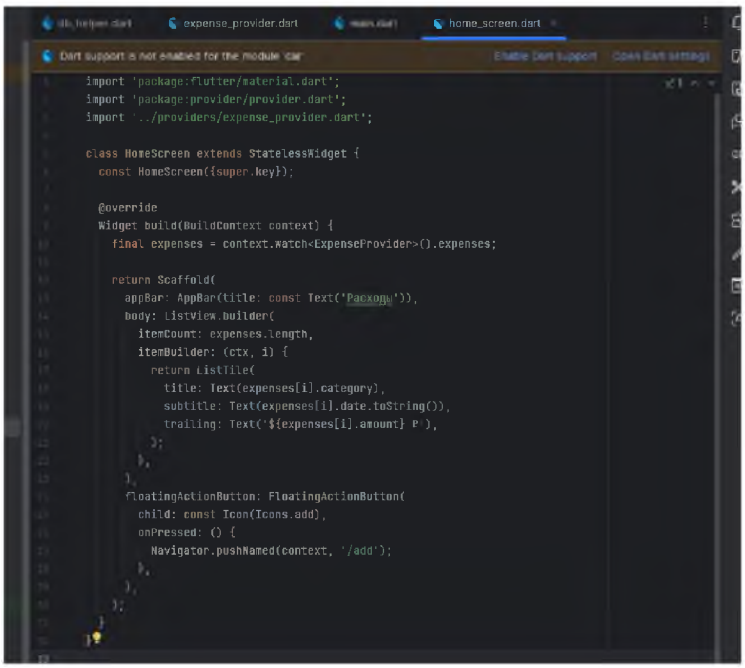

Рисунок 16 – Программный код «Главный экран»

Экран добавления расхода реализован в виде простой формы ввода. Пользователь выбирает категорию расхода, вводит сумму, дату, при необходимости пробег автомобиля и комментарий. После подтверждения данные сохраняются в базе данных (рисунок 17). Полный программный код можно увидеть в приложении 8.

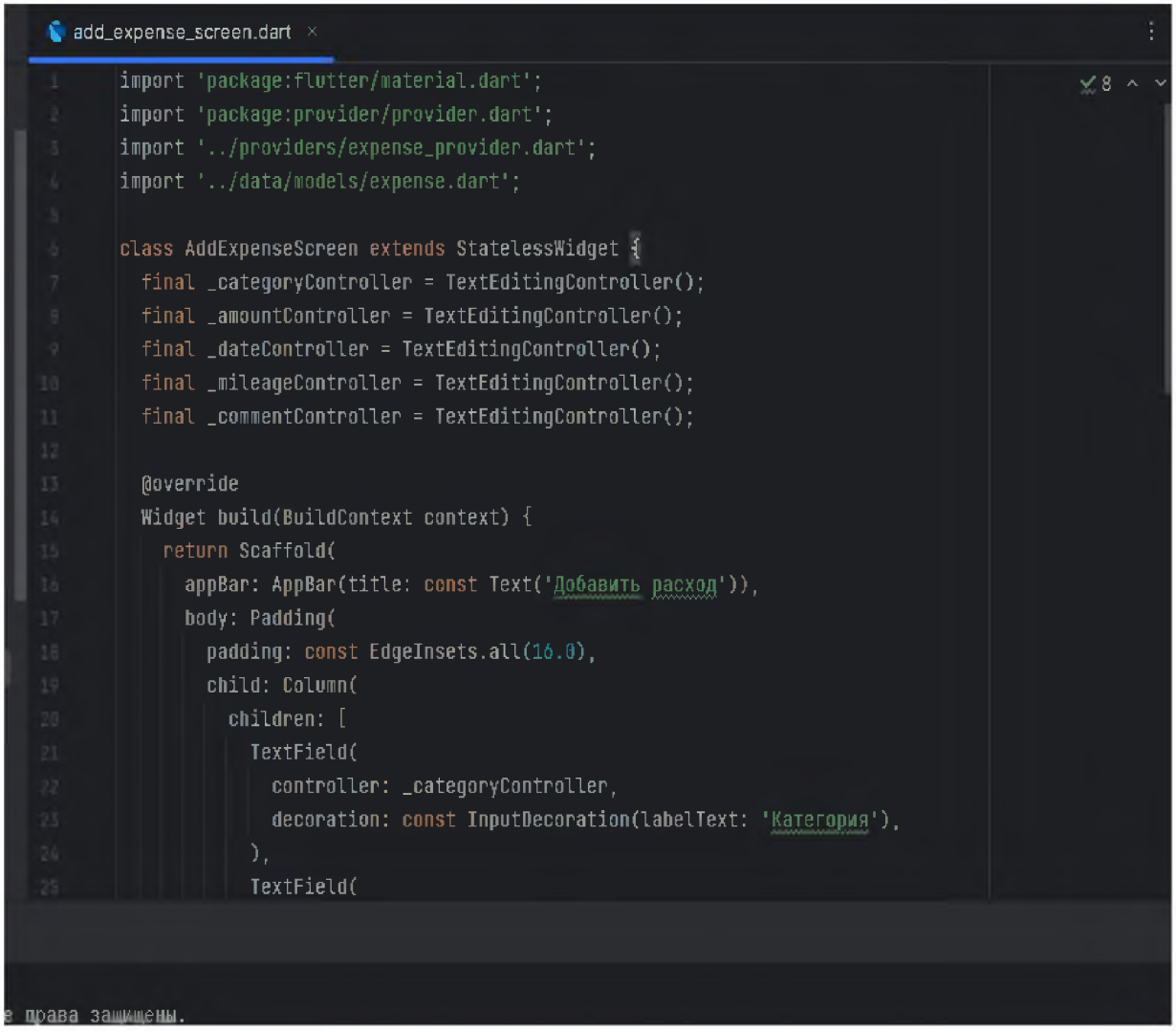

Рисунок 17 – Программный код «Добавление расходов»

Экран статистики предназначен для анализа расходов. На экране отображаются круговые и столбчатые диаграммы, демонстрирующие распределение расходов по категориям и по временным периодам. При наличии данных о пробеге дополнительно рассчитывается средний расход топлива (рисунок 18). Полный программный код можно увидеть в приложении 9.

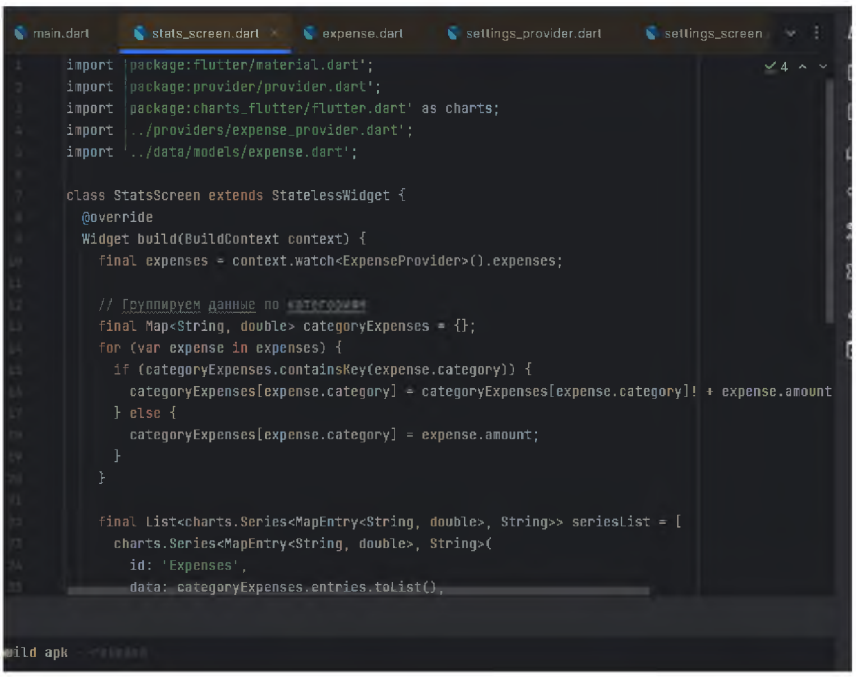

Рисунок 18 – Программный код «Статистика»

Экран настроек предоставляет возможность выбора валюты, темы оформления и единиц измерения, что позволяет адаптировать приложение под индивидуальные предпочтения пользователя (рисунок 19). Полный программный код можно увидеть в приложении 10.

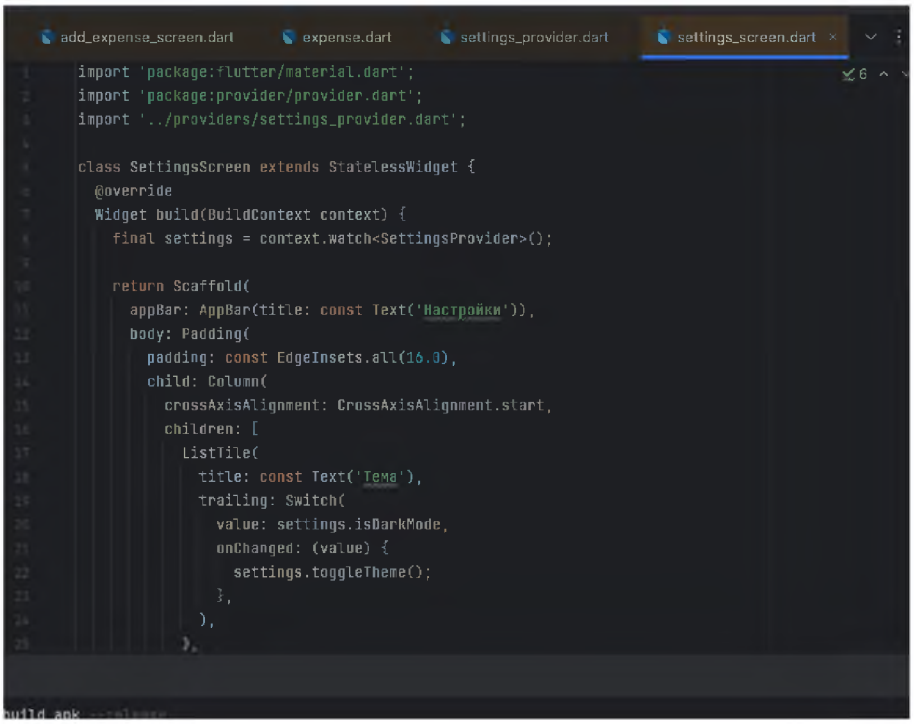

Рисунок 19 – Программный код «Настройки»

2.4 Руководство пользования

2.4.1 Описание интерфейса

Приложение состоит из нескольких основных экранов:

- Главный экран. Показывает общие итоги расходов и последние записи.
- Добавление расхода. Простая форма: дата, категория, сумма, комментарий и пробег (по желанию).
- Статистика. Графики по категориям.
- Настройки. Выбор темы, валюты и других параметров.

Навигация простая: сверху меню с иконками для перехода между разделами.

2.4.2 Порядок работы

Порядок работы:

- После установки пользователь открывает приложение и видит главный экран (рисунок 20).
- Чтобы добавить запись, нажимает кнопку «+» и вводит данные (рисунок 21, 22).
- Запись сохраняется и сразу отображается в списке (рисунок 23).
- В разделе «Статистика» можно выбрать период и посмотреть расходы по категориям (рисунок 24).
- Все данные сохраняются на устройстве и доступны в любое время.

Также доступен экран настроек. В нем можно выбрать тему, валюту и другие параметры.

Для того чтобы его открыть необходимо в главном меню нажать иконку шестеренки. После этого откроется экран с настройками, как показано на рисунке 25.

Рисунок 20- Главный экран (пустая страница)

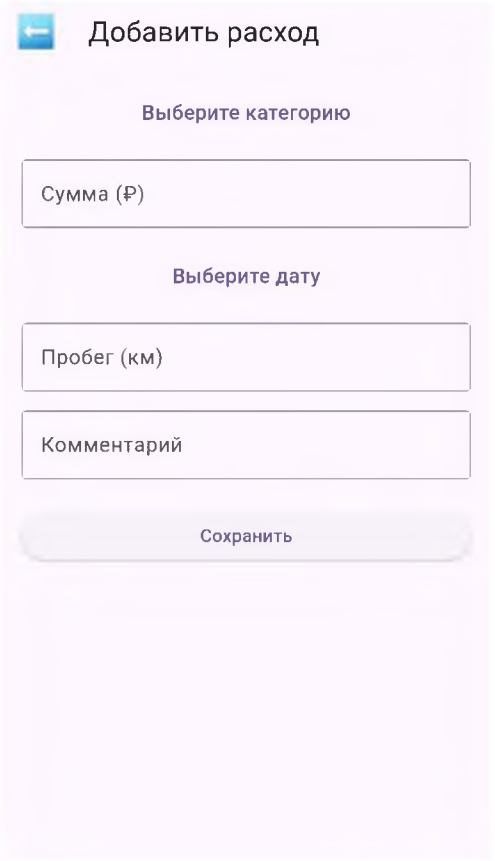

Рисунок 21- Добавление расхода (пустая страница)

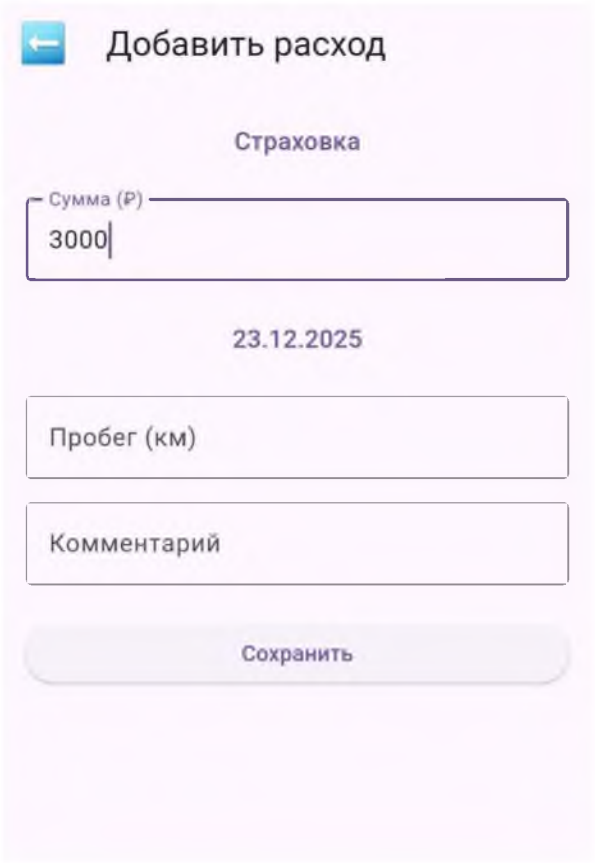

Рисунок 22 – Добавление расхода (страница с данными)

Рисунок 23 – Главный экран (полная страница)

Рисунок 24 – Статистика

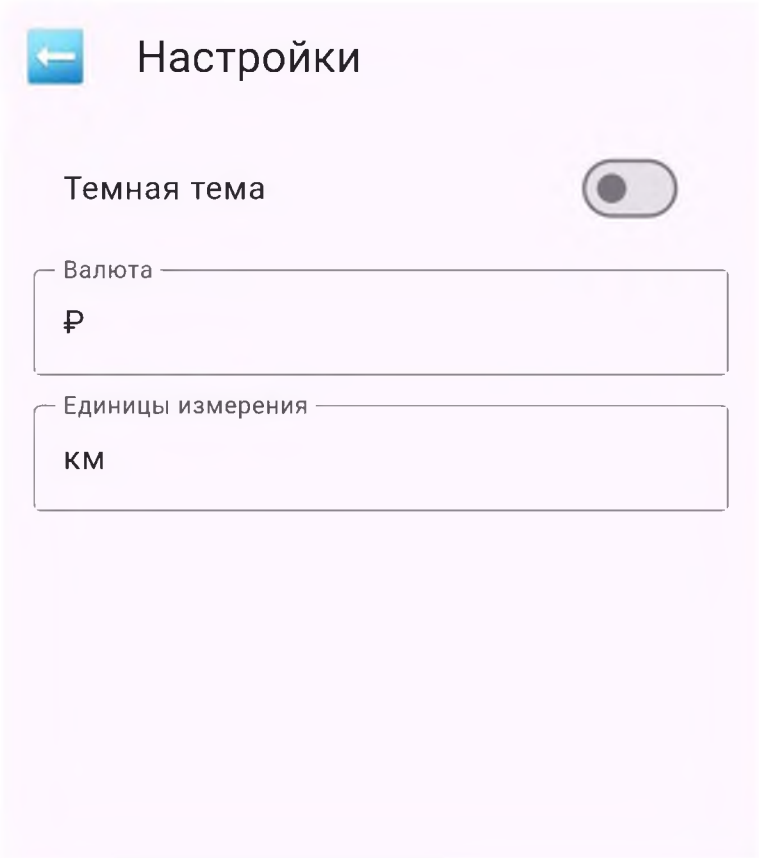

Рисунок 25 - Настройки

3 Обоснование экономической эффективности результатов ВКР

3.1 Выбор и обоснование методики расчета экономической эффективности

Экономическая эффективность программного продукта отражает степень оправданности затрат на его разработку и внедрение по отношению к получаемому эффекту. Оценка эффективности является обязательным этапом при выполнении выпускной квалификационной работы, так как позволяет подтвердить практическую значимость проекта.

Разрабатываемое приложение предназначено для учёта автомобильных расходов и не предполагает коммерческой реализации в рамках данной работы. В связи с этим классические показатели эффективности, такие как прибыль, рентабельность или чистый дисконтированный доход, в данном случае не применяются. Основной экономический эффект от внедрения приложения выражается не в получении дохода, а в снижении трудозатрат пользователя и оптимизации его расходов.

Для оценки экономической эффективности целесообразно использовать методику сравнения трудозатрат, которая широко применяется при анализе информационных систем и автоматизированных рабочих мест. Суть методики заключается в сравнении времени, затрачиваемого пользователем на выполнение одних и тех же операций до и после внедрения программного продукта.

В качестве базового варианта рассматривается ручной способ ведения учёта расходов (в тетради или таблице), а в качестве проектного варианта – использование разработанного мобильного приложения. Разница во времени выполнения операций интерпретируется как экономия рабочего времени пользователя. Для получения стоимостного выражения экономического эффекта сэкономленное время переводится в денежный эквивалент на основе средней стоимости часа времени пользователя.

Помимо прямой экономии времени, учитывается косвенный экономический эффект, который проявляется в:

- снижении необоснованных и импульсивных расходов;
- более точном планировании бюджета;
- своевременном прохождении технического обслуживания автомобиля;
- снижении вероятности штрафов и внеплановых затрат.

Таким образом, выбранная методика позволяет комплексно оценить экономическую эффективность проекта с учётом специфики некоммерческого программного продукта и наглядно показать полезность приложения для конечного пользователя.

3.2 Расчет показателей экономической эффективности

Для проведения расчётов примем следующие исходные данные, основанные на типичном сценарии использования приложения водителем. Они отображены ниже, в таблице 1.

Таблица 1 – Исходные данные для расчёта

Показатель	Значение
Количество записей о расходах в месяц	20
Время на одну запись вручную	5 минут
Время на одну запись в приложении	1 минута
Средняя стоимость 1 часа времени пользователя	300 руб.
Продолжительность расчётного периода	1 год

При выполнении расчётов экономической эффективности был принят ряд допущений. В частности, предполагается стабильное количество записей о расходах в течение года, а также постоянная средняя стоимость часа времени пользователя. Не учитываются индивидуальные различия в уровне дохода и

частоте использования приложения. Несмотря на указанные ограничения, полученные результаты позволяют объективно оценить экономический эффект от внедрения приложения.

Экономическая эффективность разработанного приложения формируется под воздействием ряда факторов, связанных с особенностями его функциональности, архитектуры и принципов работы. Эти факторы позволяют снизить как прямые, так и косвенные затраты пользователя, а также повысить удобство эксплуатации программного продукта.

Для наглядности основные факторы экономической эффективности представлены в таблице 2.

Таблица 2 – Факторы экономической эффективности приложения

Фактор	Влияние на экономическую эффективность
Автоматизация учёта расходов	Сокращение трудозатрат пользователя
Локальное хранение данных	Отсутствие затрат на серверную инфраструктуру
Работа в офлайн-режиме	Независимость от интернет-соединения
Простой и интуитивный интерфейс	Снижение времени ввода и вероятности ошибок

Автоматизация процесса ведения учёта расходов является одним из ключевых факторов экономической эффективности приложения. Использование автоматизированного ввода и обработки данных позволяет значительно сократить время, затрачиваемое пользователем на выполнение рутинных операций. В отличие от ручного учёта, приложение автоматически подсчитывает суммы, группирует расходы по категориям и формирует статистику, что снижает трудоёмкость работы и уменьшает вероятность ошибок.

Хранение данных в локальной базе данных SQLite позволяет отказаться от использования серверных решений и облачной инфраструктуры. Это

исключает дополнительные затраты на аренду серверов, обслуживание баз данных и передачу данных по сети. Кроме того, локальное хранение повышает производительность приложения и обеспечивает конфиденциальность пользовательской информации, что также является важным фактором при оценке эффективности.

Возможность работы приложения без подключения к сети Интернет обеспечивает его независимость от внешних условий. Пользователь может вносить данные и просматривать статистику в любое время и в любом месте, не расходуя мобильный трафик. Это особенно актуально в условиях ограниченного или нестабильного интернет-соединения и способствует повышению практической ценности приложения.

Простой и логически организованный пользовательский интерфейс снижает время, необходимое для освоения приложения и выполнения основных операций. Минимальное количество действий при добавлении записей о расходах позволяет ускорить процесс ввода данных и снизить когнитивную нагрузку на пользователя. Это напрямую влияет на экономию времени и повышает общую эффективность использования приложения.

Таким образом, совокупность рассмотренных факторов обеспечивает положительный экономический эффект от использования приложения. Автоматизация учёта, отсутствие затрат на серверные ресурсы, возможность работы в офлайн-режиме и удобный интерфейс делают приложение экономически целесообразным и практичным инструментом для контроля расходов.

Рассчитаем затраты времени на ведение учёта расходов вручную и с использованием приложения. Более точные расчеты показаны на таблице 3.

Таблица 3 – Сравнение трудозатрат

Показатель	Ручной учёт	Учёт с приложением
Время на одну запись	5 мин	1 мин
Количество записей в месяц	20	20

Продолжение таблицы 3

Общее время в месяц	100 мин	20 мин
Общее время в месяц	1,5 часа	0,33 часа

Экономия времени в месяц составляет:

100 – 20 = 80 минут, что эквивалентно примерно 1,3 часа.

Переведём сэкономленное время в денежное выражение. При средней стоимости часа времени пользователя 300 рублей экономия составит:

1,3 × 300 = 390 рублей в месяц.

Годовой экономический эффект от экономии времени:

390 × 12 = 4680 рублей в год.

Дополнительный экономический эффект достигается за счёт функциональных возможностей приложения:

- напоминаний о техническом обслуживании;
- контроля регулярных расходов;
- анализа структуры затрат.

Использование приложения позволяет пользователю избегать штрафов, просрочек и незапланированных расходов. По экспертной оценке, косвенная экономия может составлять 2000–3000 рублей в год. На таблице 4 показан примерный расчет сэкономленной суммы.

Таблица 4 – Совокупный экономический эффект

Вид экономии	Сумма, руб./год
Экономия времени	4680
Косвенная экономия	2000–3000
Итого	≈ 7000–8000

Для дальнейших расчётов примем усреднённое значение годового экономического эффекта равным 7000 рублей.

Затраты на разработку приложения включают только трудозатраты разработчика, так как проект создаётся в рамках учебной деятельности. В таблице 5 приведена информация по затратам на разработку.

Таблица 5 – Затраты на разработку

Показатель	Значение
Количество часов разработки	80 часов
Стоимость 1 часа работы	300 руб.
Общая стоимость разработки	24 000 руб.

Срок окупаемости проекта определяется как отношение затрат на разработку к годовому экономическому эффекту:

24 000 / 7 000 ≈ 3,4 года.

Более наглядно срок окупаемости можно посмотреть на рисунке 26.

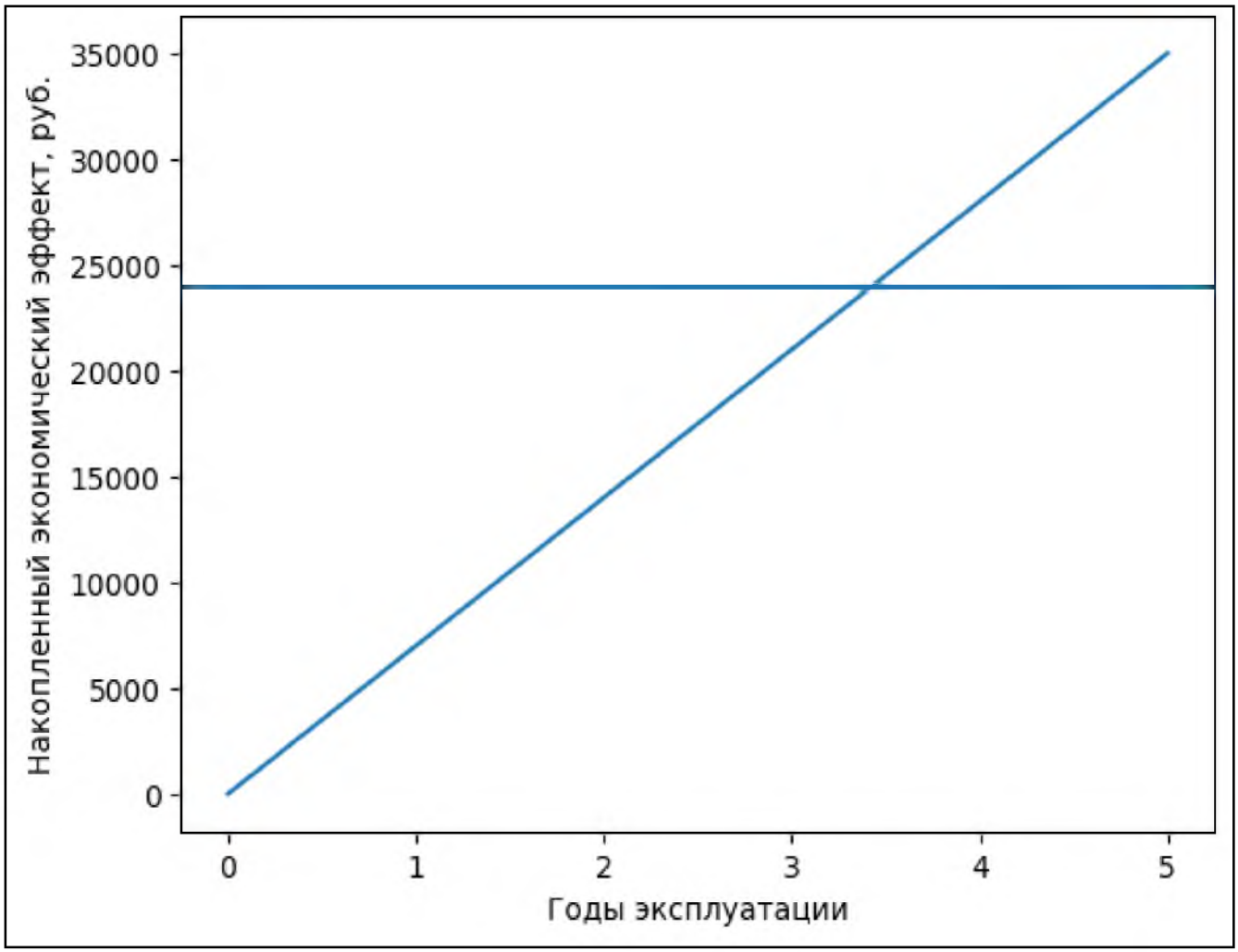

Рисунок 26 – Срок окупаемости приложения

Для учебного программного продукта данный срок окупаемости является допустимым и свидетельствует о практической ценности проекта.

Использование приложения позволяет перейти от ручного способа ведения учёта к автоматизированному. При ручном учёте возрастает вероятность ошибок при подсчётах, потери данных и несвоевременного анализа расходов. Автоматизация учёта снижает влияние человеческого фактора и повышает точность финансовой информации.

Помимо прямого экономического эффекта, использование приложения обеспечивает ряд нематериальных преимуществ, к которым относятся повышение удобства учёта, снижение когнитивной нагрузки на пользователя, улучшение финансовой дисциплины и повышение осознанности при принятии решений о расходах.

На основании проведённых расчётов можно сделать вывод, что разработанное приложение является экономически эффективным. Его использование позволяет существенно сократить затраты времени пользователя, снизить уровень необоснованных расходов и повысить удобство контроля автомобильных затрат. Даже при индивидуальном использовании приложение окупает затраты на разработку в течение нескольких лет, а при распространении среди большого числа пользователей суммарный экономический эффект многократно возрастает.

Заключение

В ходе выполнения выпускной квалификационной работы была поставлена и решена задача разработки мобильного приложения для учёта расходов на содержание автомобиля.

В первой части проведён анализ предметной области. Было показано, что существующие решения не всегда удобны и часто требуют оплаты. Пользователям не хватает простого инструмента, где можно быстро записать траты и посмотреть статистику.

Во второй части разработано собственное решение. Определены структура данных, архитектура приложения, описаны интерфейс и функции. Приложение хранит все данные локально, работает офлайн и показывает наглядную статистику расходов.

В третьей части рассчитана экономическая эффективность. Было установлено, что приложение позволяет экономить время и деньги, а значит, имеет практическую ценность.

Разработанное приложение может использоваться любым владельцем автомобиля для контроля своих затрат и планирования бюджета. В дальнейшем его можно доработать – добавить синхронизацию с облаком, экспорт отчётов и расширенную аналитику.

Проект показал, что даже простое мобильное решение может реально помогать людям управлять своими финансами и принимать более осознанные решения.

В рамках выпускной квалификационной работы было разработано мобильное приложение для учёта расходов на содержание автомобиля. При выборе инструментов учитывались требования к кроссплатформенности, автономной работе приложения, простоте сопровождения и отсутствию необходимости серверной инфраструктуры.

В качестве основного инструмента разработки был выбран фреймворк Flutter. Данный фреймворк позволяет создавать мобильные приложения для

операционных систем Android и iOS на основе единой кодовой базы, что существенно сокращает время разработки и упрощает поддержку программного продукта.

Языком программирования выбран Dart, который используется во Flutter как основной. Он обладает строгой типизацией, высокой производительностью и хорошо подходит для разработки пользовательских интерфейсов.

Для разработки и тестирования приложения использовалась интегрированная среда разработки Android Studio, предоставляющая инструменты для написания кода, отладки и запуска эмуляторов мобильных устройств.

Для хранения данных была выбрана локальная база данных SQLite, функционирующая непосредственно на устройстве пользователя. Такой подход обеспечивает автономную работу приложения без подключения к сети Интернет.

Для реализации функциональности были использованы следующие плагины Flutter:

- sqflite – для взаимодействия с базой данных SQLite;
- provider – для управления состоянием приложения;
- charts_flutter – для визуализации статистических данных в виде диаграмм.

Выбранный стек технологий является бесплатным, широко распространённым и полностью соответствует поставленным задачам.

Разработанное приложение может использоваться любым владельцем автомобиля для контроля своих затрат и планирования бюджета. В дальнейшем его можно доработать – добавить синхронизацию с облаком, экспорт отчётов и расширенную аналитику.

Проект показал, что даже простое мобильное решение может реально помогать людям управлять своими финансами и принимать более осознанные решения.

Список литературы

1. Android Studio и Flutter: интеграция и лучшие практики / А. И. Николаев, С. М. Воронов. – Москва: Лань, 2023. – 344 с. – Текст: непосредственный (дата обращения: 05.06.2025).

2. Android Studio: разработка мобильных приложений с Flutter / И. Р. Сидоров, А. В. Петров. – Нижний Новгород: ННГУ, 2022. – 320 с. – Текст: непосредственный (дата обращения: 23.07.2025).

3. Dart и Flutter: от новичка до профессионала / С. В. Михайлов. – Ростов-на-Дону: Феникс, 2022. – 416 с. – (Серия «Библиотека программиста»). – ISBN 978-5-222-35678-9. – Текст: непосредственный (дата обращения: 20.04.2025).

4. Flutter и Dart: разработка кроссплатформенных приложений / коллектив авторов. – Москва: Питер, 2021. – 384 с. – (Библиотека разработчика). – ISBN 978-5-496-03892-7. – Текст: непосредственный (дата обращения: 29.05.2025).

5. Flutter: создание интерактивных интерфейсов / В. А. Смирнов. – Москва: Инфра-М, 2023. – 288 с. – Текст: непосредственный (дата обращения: 16.01.2025).

6. Flutter-разработка: от идеи до публикации приложения / О. В. Тихомиров, Н. С. Колесников. – Москва: Диалектика, 2024. – 416 с. – (Серия «Профессиональное программирование»). – ISBN 978-5-907144-88-8. – Текст: непосредственный (дата обращения: 21.05.2025).

7. Flutter-разработка: продвинутый курс / под ред. М. А. Лебедева. – Москва: Диалектика, 2023. – 368 с. – (Серия «Профессиональное программирование»). – ISBN 978-5-907144-56-7. – Текст: непосредственный (дата обращения: 21.10.2025).

8. Бакетт, К. Dart в действии: практическое руководство / К. Бакетт ; пер. с англ. – 2-е изд., эл. – Москва: ДМК Пресс, 2023. – 529 с. – ISBN 978-5-89818-314-1. – Текст: электронный // Znanium [сайт]. – URL:

https://znanium.ru/catalog/document?pid=2102601 (дата обращения: 01.08.2025).

9. Дарт для начинающих: практическое руководство по языку программирования Dart / под ред. И. В. Смирнова. – Москва: Диалектика, 2021. – 320 с. – (Серия «Программирование для всех»). – ISBN 978-5-907144-12-3. – Текст: непосредственный (дата обращения: 20.07.2025).

10. Жао, Э. SQL. Pocket guide, 4-е изд. / Э. Жао. – Москва: Питер, 2026. – 320 с. – ISBN 978-601-08-3728-7. – Текст: непосредственный (дата обращения: 20.07.2025).

11. Иванов, Д. Н. Основы работы с SQLite в мобильных приложениях / Д. Н. Иванов, Т. П. Лебедева. – Казань: Изд-во КФУ, 2021. – 196 с. – Текст: непосредственный (дата обращения: 01.03.2025).

12. Как пользоваться Android Studio [Электронный ресурс] // practicum.yandex.ru – Режим доступа: https://practicum.yandex.ru/blog/kak-polzovatsya-android-studio/, свободный. – Загл. с экрана (дата обращения: 12.03.2025).

13. Кузнецов, М. В. SQLite: практическое руководство по работе с базами данных / М. В. Кузнецов. – Москва: Издательство «Солон-Пресс», 2023. – 288 с. – (Серия «Библиотека программиста»). – ISBN 978-5-91359-487-2. – Текст: непосредственный (дата обращения: 05.01.2025).

14. Нестеров, С. А. Базы данных: учеб. / С. А. Нестеров. – 2-е изд., перераб. и доп. – Москва: Издательство Юрайт, 2025. – 258 с. – (Высшее образование). – ISBN 978-5-534-18107-4. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/560753 (дата обращения: 19.12.2025).

15. Николаев, С. Р. Dart и Flutter: программирование интерфейсов и работа с данными / С. Р. Николаев. – Москва: Диалектика, 2025. – 512 с. – ISBN 978-5-907144-99-4. – Текст: непосредственный (дата обращения: 08.07.2025).

16. Основы Flutter: пошаговое руководство для начинающих / А. Н. Кузнецов. – Санкт-Петербург: БХВ-Петербург, 2022. – 256 с. – (Серия

«Библиотека разработчика»). – Текст: непосредственный (дата обращения: 08.06.2025).

17. Первые шаги в мобильной разработке с Flutter [Электронный ресурс] // tproger.ru – Режим доступа: https://tproger.ru/articles/pervye-wagi-v-mobilnoj-razrabotke-s-flutter, свободный. – Загл. с экрана (дата обращения: 17.07.2025).

18. Петров, А. И. Программирование на Dart: от основ к практике / А. И. Петров, Е. С. Волков. – Санкт-Петербург: БХВ-Петербург, 2024. – 416 с. – ISBN 978-5-9775-7023-3. – Текст: непосредственный (дата обращения: 26.07.2025).

19. Практикум по Flutter и Dart: 50 упражнений для закрепления навыков / Е. С. Воронина. – Екатеринбург: УрГУПС, 2023. – 192 с. – Текст: непосредственный (дата обращения: 07.11.2025).

20. Программирование на Dart для Flutter-разработки: учебное пособие / Е. П. Воронин. – Екатеринбург: УрФУ, 2021. – 160 с. – Текст: непосредственный (дата обращения: 27.02.2025).

21. Разработка Android-приложения на Java: пошаговая инструкция для новичков [Электронный ресурс] // selectel.ru – Режим доступа: https://selectel.ru/blog/mobile-app-development/, свободный. – Загл. с экрана (дата обращения: 08.01.2025).

22. Разработка кроссплатформенных приложений на Flutter / коллектив авторов. – Новосибирск: НГТУ, 2022. – 304 с. – Текст: непосредственный (дата обращения: 18.10.2025).

23. Разработка под Android с использованием Android Studio: учебник для вузов / В. С. Смирнов, Н. А. Попов. – Москва: Юрайт, 2023. – 608 с. – (Высшее образование). – ISBN 978-5-534-12345-6. – Текст: электронный (дата обращения: 03.03.2025).

24. Рындина, С. В. Цифровые технологии управления получением, хранением, передачей и обработкой больших данных: SQLite: учебно-методическое пособие / С. В. Рындина. – Пенза: Изд-во ПГУ, 2023. –

82 с. – УДК 004.6 (075). – Текст: электронный // URL: https://elib.pnzgu.ru/files/eb/LQaV1OD7DFsp.pdf (дата обращения: 26.01.2025).

25. Смирнов, В. А. Dart для кроссплатформенной разработки: учебное пособие / В. А. Смирнов. – Екатеринбург: Изд-во УрФУ, 2022. – 272 с. – ISBN 978-5-7996-3456-1. – Текст: непосредственный (дата обращения: 13.11.2025).

26. Создание мобильных приложений с Flutter: практическое руководство / под ред. Д. А. Иванова. – Казань: КФУ, 2022. – 288 с. – Текст: непосредственный (дата обращения: 19.02.2025).

27. Соколова, В. В. Вычислительная техника и информационные технологии. Разработка мобильных приложений: учеб. / В. В. Соколова. – Москва: Издательство Юрайт, 2025. – 160 с. – (Высшее образование). – ISBN 978-5-534-16302-5. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/561336 (дата обращения: 17.08.2025).

28. Соколова, В. В. Разработка мобильных приложений: учеб. / В. В. Соколова. – Москва: Издательство Юрайт, 2025. – 160 с. – (Профессиональное образование). – ISBN 978-5-534-16868-6. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/566082 (дата обращения: 14.07.2025).

29. Чернышев, С. А. Основы Flutter / С. А. Чернышев, Ю. М. Петров, С. П. Ильин, П. А. Гершевич. – Санкт-Петербург: Издательство «Питер», 2026. – 688 с. – (Библиотека программиста). – ISBN 978-5-4461-4469-3 (дата обращения: 22.07.2025).

30. Что такое Android Studio и как ей пользоваться [Электронный ресурс] // skillbox.ru – Режим доступа: https://skillbox.ru/media/code/chto-takoe-android-studio-i-kak-ey-polzovatsya/, свободный. – Загл. с экрана (дата обращения: 04.11.2025).

Приложение 1

Подключение зависимостей

```
name: car_expenses_app
description: Приложение для учета расходов на автомобиль
publish_to: "none"
version: 1.0.0+1

environment:
  sdk: ">=3.10.0 <4.0.0"

dependencies:
  flutter:
    sdk: flutter
  provider: ^6.1.2
  sqflite: ^2.3.0
  path: ^1.9.0
  fl_chart: ^0.66.0-beta.1
  google_fonts: ^5.0.0

dev_dependencies:
  flutter_test:
    sdk: flutter
```

Приложение 2

Инициализация БД

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

class DBHelper {
  DBHelper._privateConstructor();
  static final DBHelper instance = DBHelper._privateConstructor();

  static Database? _database;

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDB('car_expenses.db');
    return _database!;
  }

  Future<Database> _initDB(String filePath) async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, filePath);

    return await openDatabase(path, version: 1, onCreate: _createDB);
  }

  Future<void> _createDB(Database db, int version) async {
    await db.execute('''
      CREATE TABLE expenses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT NOT NULL,
        category TEXT NOT NULL,
        amount REAL NOT NULL,
        comment TEXT,
        mileage REAL
      )
    ''');
  }
}
```

Приложение 3

Модель данных «Расход»

```
class Expense {
  final int? id;
  final DateTime date;
  final String category;
  final double amount;
  final String comment;
  final double? mileage;

  Expense({
    this.id,
    required this.date,
    required this.category,
    required this.amount,
    required this.comment,
    this.mileage,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'date': date.toIso8601String(),
      'category': category,
      'amount': amount,
      'comment': comment,
      'mileage': mileage,
    };
  }

  factory Expense.fromMap(Map<String, dynamic> map) {
    return Expense(
      id: map['id'],
      date: DateTime.parse(map['date']),
      category: map['category'],
      amount: map['amount'],
      comment: map['comment'],
      mileage: map['mileage'] != null ? map['mileage'] as double : null,
    );
  }
}
```

Приложение 4

Модель данных «Напоминание»

```
class Reminder {
  final int? id;
  final DateTime date;
  final String type;
  final String comment;

  Reminder({
    this.id,
    required this.date,
    required this.type,
    required this.comment,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'date': date.toIso8601String(),
      'type': type,
      'comment': comment,
    };
  }

  factory Reminder.fromMap(Map<String, dynamic> map) {
    return Reminder(
      id: map['id'],
      date: DateTime.parse(map['date']),
      type: map['type'],
      comment: map['comment'],
    );
  }
}
```

Приложение 5

Управление состоянием 1

```
import 'package:flutter/material.dart';
import '../data/db_helper.dart';
import '../data/models/expense.dart';

class ExpenseProvider extends ChangeNotifier {
  List<Expense> _expenses = [];
  List<Expense> get expenses => _expenses;

  Future<void> loadExpenses() async {
    final db = await DBHelper.instance.database;
    final data = await db.query('expenses', orderBy: 'date DESC');
    _expenses = data.map((e) => Expense.fromMap(e)).toList();
    notifyListeners();
  }

  Future<void> addExpense(Expense expense) async {
    final db = await DBHelper.instance.database;
    await db.insert('expenses', expense.toMap());
    await loadExpenses();
  }
}
```

Приложение 6

Управление состоянием 2

```
import 'package:flutter/material.dart';

class SettingsProvider extends ChangeNotifier {
  bool _isDarkMode = false;
  String _currency = '₽';
  String _unit = 'км';

  bool get isDarkMode => _isDarkMode;
  String get currency => _currency;
  String get unit => _unit;

  void toggleTheme() {
    _isDarkMode = !_isDarkMode;
    notifyListeners();
  }

  void setCurrency(String currency) {
    _currency = currency;
    notifyListeners();
  }

  void setUnit(String unit) {
    _unit = unit;
    notifyListeners();
  }
```

Приложение 7

Главный экран

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/expense_provider.dart';
import '../providers/settings_provider.dart';
class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});
  @override
  Widget build(BuildContext context) {
    final expenseProvider = context.watch<ExpenseProvider>();
    final settings = context.watch<SettingsProvider>();
    return Scaffold(
      appBar: AppBar(
        title: const Text('Мои расходы'),
        actions: [
          // Кнопка "Статистика" через текст / emoji
          TextButton(
            onPressed: () => Navigator.pushNamed(context, '/stats'),
            child: const Text('📊', style: TextStyle(fontSize: 24)),
          ),
          // Кнопка "Настройки" через текст / emoji
          TextButton(
            onPressed: () => Navigator.pushNamed(context, '/settings'),
            child: const Text('⚙', style: TextStyle(fontSize: 24)),
          ),
        ],
      ),
      body: expenseProvider.expenses.isEmpty
          ? const Center(child: Text('Нет расходов'))
          : ListView.builder(
        itemCount: expenseProvider.expenses.length,
        itemBuilder: (context, index) {
          final e = expenseProvider.expenses[index];
          return ListTile(
            title: Text('${e.category}: ${e.amount} ${settings.currency}'),
            subtitle: Text(
                '${e.comment} | Пробег: ${e.mileage ?? '-'} ${settings.unit} |
${e.date.day}.${e.date.month}.${e.date.year}'),
          );
```

```
  },
    ),
    // Кнопка "Добавить расход" через текст
    floatingActionButton: FloatingActionButton(
      onPressed: () => Navigator.pushNamed(context, '/add'),

      tooltip: 'Добавить расход',
      child: const Text('+', style: TextStyle(fontSize: 28)),
    ),
  );
 }
}
```

Приложение 8

Экран добавления расходов

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/expense_provider.dart';
import '../providers/settings_provider.dart';
import '../data/models/expense.dart';
class AddExpenseScreen extends StatefulWidget {
  const AddExpenseScreen({super.key});
  @override
  State<AddExpenseScreen> createState() => _AddExpenseScreenState();
}
class _AddExpenseScreenState extends State<AddExpenseScreen> {
  final _amountController = TextEditingController();
  final _commentController = TextEditingController();
  String? _selectedCategory;
  DateTime? _selectedDate;
  double? _mileage;
  @override
  void dispose() {
    _amountController.dispose();
    _commentController.dispose();
    super.dispose();
  }
  @override
  Widget build(BuildContext context) {
    final settings = context.watch<SettingsProvider>();
    return Scaffold(
      appBar: AppBar(
        leading: TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('←', style: TextStyle(fontSize: 24, color: Colors.white)),
        ),
        title: const Text('Добавить расход'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            // Выбор категории
            TextButton(
              onPressed: () async {
```

```
            final selected = await showDialog<String>(
              context: context,
              builder: (ctx) => SimpleDialog(

                title: const Text('Выберите категорию'),
                children: [
                  SimpleDialogOption(onPressed: () => Navigator.pop(ctx, 'Топливо'),
child: const Text('Топливо')),
                  SimpleDialogOption(onPressed: () => Navigator.pop(ctx, 'Ремонт'),
child: const Text('Ремонт')),
                  SimpleDialogOption(onPressed: () => Navigator.pop(ctx,
'Страховка'), child: const Text('Страховка')),
                  SimpleDialogOption(onPressed: () => Navigator.pop(ctx,
'Прочее'), child: const Text('Прочее')),
                ],
              ),
            );
            if (selected != null) {
              setState(() {
                _selectedCategory = selected;
              });
            }
          },
          child: Text(
            _selectedCategory ?? 'Выберите категорию',
            style: const TextStyle(fontSize: 16),
          ),
        ),
        const SizedBox(height: 16),
        // Сумма
        TextField(
          controller: _amountController,
          keyboardType: TextInputType.number,
          decoration: InputDecoration(
            labelText: 'Сумма (${settings.currency})',
            border: const OutlineInputBorder(),
          ),
        ),
        const SizedBox(height: 16),
        // Выбор даты
        TextButton(
          onPressed: () async {
            final pickedDate = await showDatePicker(
              context: context,
```

```
            initialDate: DateTime.now(),
            firstDate: DateTime(2000),

            lastDate: DateTime(2100),
          );
          if (pickedDate != null) {
            setState(() {
              _selectedDate = pickedDate;
            });
          }
        },
        child: Text(
          _selectedDate != null
              ?
'${_selectedDate!.day}.${_selectedDate!.month}.${_selectedDate!.year}'
              : 'Выберите дату',
          style: const TextStyle(fontSize: 16),
        ),
      ),
      const SizedBox(height: 16),
      // Пробег
      TextField(
        keyboardType: TextInputType.number,
        decoration: InputDecoration(
                  labelText: 'Пробег (${settings.unit})',
          border: const OutlineInputBorder(),
        ),
        onChanged: (value) {
          _mileage = double.tryParse(value);
        },
      ),
      const SizedBox(height: 16),
      // Комментарий
      TextField(
        controller: _commentController,
        decoration: const InputDecoration(
          labelText: 'Комментарий',
          border: OutlineInputBorder(),
        ),
      ),
      const SizedBox(height: 24),
      // Кнопка сохранить
      SizedBox(
        width: double.infinity,
        child: ElevatedButton(
```

```
            onPressed: _saveExpense,
            child: const Text('Сохранить'),
          ),
        ),
      ],
    ),
  ),
 );
}
void _saveExpense() {
 if (_selectedCategory == null || _amountController.text.isEmpty || _selectedDate
== null) {
  ScaffoldMessenger.of(context).showSnackBar(
   const SnackBar(content: Text('Заполните категорию, сумму и дату')),
  );
  return;
 }
 final expense = Expense(
  category: _selectedCategory!,
  amount: double.tryParse(_amountController.text) ?? 0,
  date: _selectedDate!,
  comment: _commentController.text,
  mileage: _mileage,
 );
 context.read<ExpenseProvider>().addExpense(expense);
 Navigator.pop(context);
}
```

Приложение 9

Экран статистики

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:fl_chart/fl_chart.dart';
import '../providers/expense_provider.dart';
import '../providers/settings_provider.dart';
class StatisticsScreen extends StatelessWidget {
  const StatisticsScreen({super.key});
  @override
  Widget build(BuildContext context) {
    final expenseProvider = context.watch<ExpenseProvider>();
    final settings = context.watch<SettingsProvider>();
    final stats = _calculateStats(expenseProvider);
    return Scaffold(
      appBar: AppBar(
        leading: TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('←', style: TextStyle(fontSize: 24, color: Colors.white)),
        ),
        title: const Text('Статистика'),
      ),
      body: stats.isEmpty
          ? const Center(child: Text('Нет данных'))
          : Padding(
        padding: const EdgeInsets.all(16.0),
        child: PieChart(
          PieChartData(
            sections: stats
                .map((stat) => PieChartSectionData(
              value: stat.total,
              title:
              '${stat.category}\n${stat.total.toStringAsFixed(0)} ${settings.currency}',
              color: _getColor(stat.category),
              radius: 60,
              titleStyle: const TextStyle(fontSize: 14, fontWeight: FontWeight.bold, color:
Colors.white),
            ))
                .toList(),
            sectionsSpace: 2,
            centerSpaceRadius: 40,
          ),
```

```
        ),
      ),
    );


  }
  List<CategoryStat> _calculateStats(ExpenseProvider provider) {
    final Map<String, double> totals = {};
    for (var e in provider.expenses) {
      totals[e.category] = (totals[e.category] ?? 0) + e.amount;
    }
    return totals.entries
        .map((entry) => CategoryStat(entry.key, entry.value))
        .toList();
  }

  Color _getColor(String category) {
    final colors = [
      Colors.blue,
      Colors.red,
      Colors.green,
      Colors.orange,
      Colors.purple,
      Colors.teal,
      Colors.brown,
    ];
    return colors[category.hashCode % colors.length];
  }
}
class CategoryStat {
  final String category;
  final double total;
  CategoryStat(this.category, this.total);
}
```

Приложение 10

Экран настроек

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/settings_provider.dart';
class SettingsScreen extends StatelessWidget {
  const SettingsScreen({super.key});
  @override
  Widget build(BuildContext context) {
    final settings = context.watch<SettingsProvider>();
    return Scaffold(
      appBar: AppBar(
        leading: TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('←', style: TextStyle(fontSize: 24, color: Colors.white)),
        ),
        title: const Text('Настройки'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            SwitchListTile(
              title: const Text('Темная тема'),
              value: settings.isDarkMode,
              onChanged: (val) => settings.toggleTheme(),
            ),
            const SizedBox(height: 16),
            TextField(
              decoration: InputDecoration(
                labelText: 'Валюта',
                border: const OutlineInputBorder(),
              ),
              controller: TextEditingController(text: settings.currency),
              onSubmitted: (val) => settings.setCurrency(val),
            ),
            const SizedBox(height: 16),
            TextField(
              decoration: InputDecoration(
                labelText: 'Единицы измерения',
                border: const OutlineInputBorder(),
              ),
```

```
            controller: TextEditingController(text: settings.unit),
            onSubmitted: (val) => settings.setUnit(val),
          ),


        ],
      ),
    ),
  );
 }
}
```