СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ

ОС – Операционная система.

ДТП – Дорожно-транспортное происшествие.

ТО – Техническое обслуживание.

SDK – Software Development Kit.

IDE – Integrated Development Environment.

MVVM – Model-View-ViewModel.

БД – База данных.

SQL – Structured Query Language.

DAO – Data Access Object.

UI – User Interface.

API – Application Programming Interface.

ВВЕДЕНИЕ

В настоящее время с развитием современных технологий и распространением смартфонов, мобильные приложения стали неотъемлемой частью повседневной жизни человека. Пользователи могут обмениваться сообщениями с друзьями и близкими, делиться красивыми моментами своей жизни через фотографии и видео, а также всегда быть в курсе последних новостей. Более того, мы можем присоединиться к различным онлайн-сообществам, где обсуждаются общие интересы и находятся новые друзья. Все это создает ощущение связи и участия в нашей общей цифровой среде. Но все это, по большей части, относится к развлечениям и проведению досуга. Помимо перечисленного выше, мобильные приложения также могут быть полезны в быту человека или его профессиональной деятельности.

Одной из популярных категорий мобильных приложений являются органайзеры задач, которые помогают пользователям планировать, отслеживать и управлять своими обязанностями и активностями в повседневной жизни.

Темп жизни растет с каждым годом. Когда у человека много дел и ответственности, эффективное управление временем и задачами становится критически важным. Мобильные планировщики задач позволяют пользователям:

- структурировать жизненные процессы;
- устанавливать приоритеты;
- не забывать о важных событиях и дедлайнах;
- повышать свою продуктивность и эффективность.

А если пользователь такого ПО является автовладельцем, то количество задач становится еще больше.

Известно, что для автовладельцев есть особые потребности и задачи, связанные с обслуживанием и эксплуатацией автомобиля. Помимо обычных

задач, как встречи и повседневные дела, им нужно помнить о регулярной замене деталей, техническом обслуживании своего авто и прочих автомобильных вопросах. В этом ему поможет специальный планировщик, разработанный именно для этой категории граждан.

Целью данной выпускной квалификационной работы является разработка мобильного приложения для обслуживания автомобиля и планирования задач на Android.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Общая информация о мобильных устройствах и приложениях

Мобильные устройства – это портативные электронные устройства, которые позволяют пользователям осуществлять различные коммуникационные, информационные и развлекательные функции. Обычно, они имеют компактный размер и легкий вес, а также работают на базе мобильных операционных систем, таких как iOS, Android, Windows и других.

В настоящее время существует три основные категории мобильных устройств – это сотовые телефоны, смартфоны и планшеты.

Сотовые телефоны – это портативные устройства, основной целью которых является предоставление пользователям услуг голосовой связи посредством сотовой сети. [1] Первая категория активно вытесняется смартфонами, но все еще занимает существенную часть рынка. На 2023 г. количество сотовых телефонов составило около 18% от общего числа мобильных устройств. ОС таких устройств не предназначена для работы разного рода прикладных программ. Основная их цель – это голосовая связь.

Теперь, поговорим про вторую и третью категорию мобильных устройств.

Смартфоны – это многофункциональные устройства, которые объединяют в себе функции телефона, компьютера, камеры и т.д. Они обладают сенсорным экраном для ввода данных и навигации, а также поддерживают установку и использование различных мобильных приложений.

Планшеты – это устройства с большим сенсорным экраном, предназначенные для чтения электронных книг, просмотра видео, игр и других развлекательных и информационных задач. Они также поддерживают установку мобильных приложений и обладают большей производительностью в сравнении со смартфонами.

Вторая и третья категория мобильных устройств, рассмотренных в данной главе, имеют широкий спектр функций и возможностей, включая доступ к интернету, коммуникацию через социальные сети и мессенджеры, использование мобильных приложений, воспроизведение медиа-контента, навигацию, съемку фото и видео, выполнение банковских операций и многое другое. [1] Такой богатый набор возможностей при использовании мобильных устройств привлекает к себе все больше, и больше потребителей. Проведенное в 2022 году исследование “Digital 2022 Global Overview Report” указывает на то, что количество пользователей мобильных устройств составляет более 67,1% от населения мира и достигло цифры в 5,31 млрд человек. [3] Данная статистика приведена на рисунке 1.1.1.

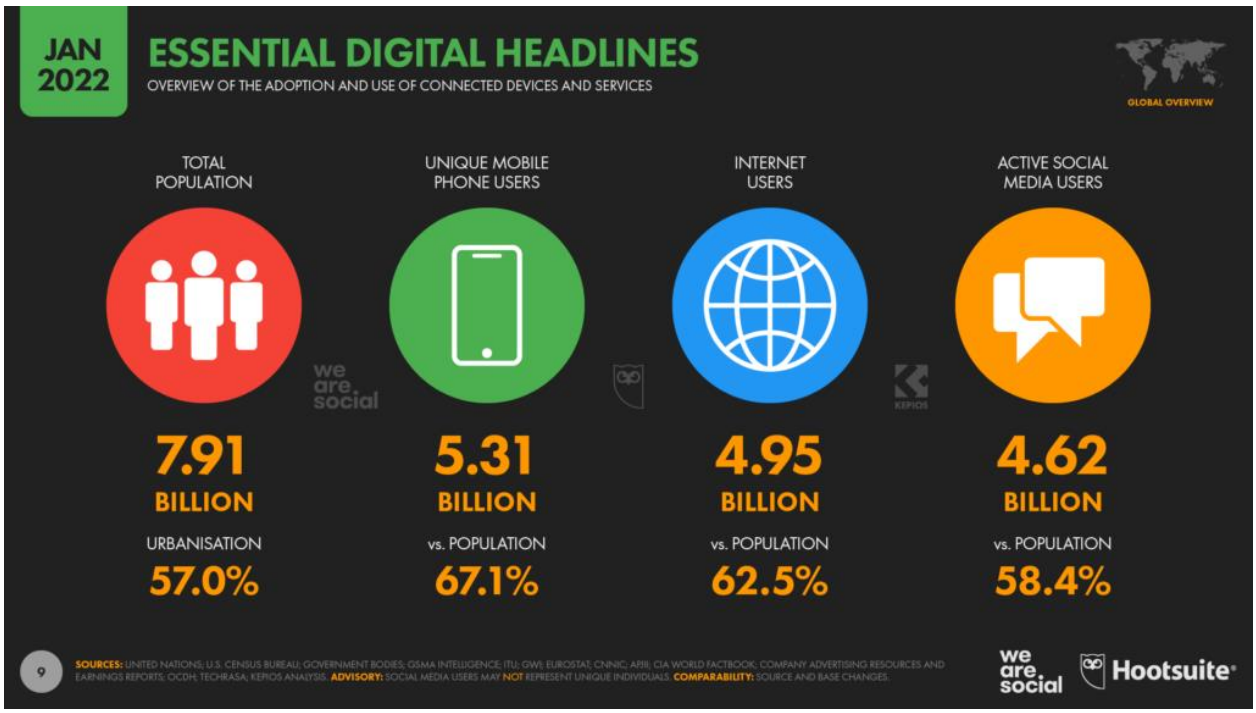

Рисунок 1.1.1 – Статистические данные исследования “Digital 2022 Global Overview Report”

У каждого мобильного устройства есть своя ОС. Мобильная ОС – это программное обеспечение, разработанное специально для управления мобильными устройствами. Она предоставляет основу для работы приложений, управляет ресурсами устройства, а также взаимодействует с

аппаратным обеспечением и предоставляет пользовательский интерфейс для взаимодействия с устройством.

Что касается популярности мобильных ОС, то здесь безоговорочно первое место достается Android. Под данной операционной системой выпущено около 69% всех мобильных устройств. [4] На рисунке 1.1.2 представлена сводная статистика использования разных ОС для мобильных устройств в мире на 2023 год.

Рисунок 1.1.1 – Доля рынка мобильных операционных систем по всему миру

Мобильные ОС обеспечивают безопасность данных и приложений, многозадачность, поддержку мультимедийных возможностей и другие функции, необходимые для работы и взаимодействия с мобильными устройствами.

Каждая мобильная ОС имеет свои особенности, архитектуру и требования к разработке приложений. Разработчики выбирают конкретную мобильную ОС в зависимости от целевой аудитории и требований проекта.

Мобильное приложение – это самостоятельней программный продукт, разрабатываемый для мобильного устройства и содержащий определенный

функционал. [2] Оно может быть загружено и установлено из официальных магазинов приложений, таких как Google Play Store для Android и App Store для iOS, либо из других источников, предоставляющих приложения сторонних разработчиков.

Мобильные приложения предназначены для выполнения определенных функций и задач, которые могут быть связаны с коммуникацией, развлечениями, образованием, финансами, здоровьем, путешествиями, социальными сетями и многим другим. Они обеспечивают доступ к информации, предоставляют удобные интерфейсы для выполнения задач и предлагают персонализацию в соответствии с потребностями и предпочтениями пользователей. Также они могут взаимодействовать с внешними сервисами и API, расширяя свои возможности за счет использования других приложений и платформ. [2]

Разработка мобильных приложений включает в себя проектирование пользовательского интерфейса, программирование, тестирование и оптимизацию приложения для работы на различных устройствах и операционных системах. Разработчики используют различные языки программирования, такие как Java, Kotlin, Swift, Objective-C, Xamarin и другие, и инструменты разработки, такие как Android Studio, Xcode, React Native и другие, для создания мобильных приложений, а также следуют рекомендациям и стандартам платформы, для которой они разрабатывают приложение.

С точки зрения функциональности мобильных приложений, их можно разделить на следующие категории:

- Развлечения;
- Бизнес;
- Социальные приложения;
- Еда;

- Путешествия;
- Спорт;
- Образование;
- Новости.

К категории бизнес, относятся финансовые приложения, инструменты для предпринимателей, а также планировщики задач, которые мы рассмотрим более детально в следующей главе.

1.2 Анализ планировщиков задач

Эффективное управление временем играет ключевую роль в достижении успеха и повышении продуктивности человека. Однако, хранение всех задач и обязанностей в голове может быть сложным процессом и подвержено риску забыть или упустить что-то важное. Раньше, люди вели дневники и ежедневники, в которых по пунктам расписывали свой распорядок и планы на день. Некоторые отмечали важные события в календаре, чтобы случайно не забыть их. Но в настоящее время, с приходом интернета, компьютеров и мобильных устройств, на замену всем способам планирования своих дел и задач пришли цифровые аналоги. Приложения для планирования задач становятся незаменимыми инструментами, помогающие людям организовать свою жизнь и профессиональную деятельность максимально эффективно.

Планировщики задач – это программы, разработанные для автоматизации процесса планирования дел и задач. Они позволяют составить расписание, запланировать события и мероприятия заранее, что облегчает планирование и предотвращает конфликты в расписании.

Основные функции планировщиков задач:

1. Создание, просмотр, редактирование и удаление задач. Пользователю дана возможность создавать новые задачи, указывая их

описание, сроки выполнения и приоритеты. При необходимости у него должна быть возможность отредактировать их или удалить.

2. Установка напоминаний. В планировщике задач должна быть реализована настраиваемая система уведомлений, которая будет напоминать пользователям о предстоящих задачах и событиях.

1.3 Общая информация об обслуживание автомобилей

Обслуживание автомобиля является одной из составляющих его эксплуатации и имеет решающее значение для его безопасности, производительности и долговечности.

У автовладельцев, помимо бытовых и рабочих задач, существует еще те, что связаны с автомобилем. Сюда относятся техническое обслуживание авто, учет износа ходовых и расходных деталей, проверка уровня технических жидкостей и т.д.

Зачастую, вся информация об актуальном износе деталей, пробеге автомобиля, а также запланированных мероприятиях по обслуживанию авто храниться исключительно в голове водителя. Данный подход неправильный и может привезти к непредсказуемому поведению автомобиля на дороге, в том числе и выходу его из строя. Согласно статистическим сведениям о дорожно-транспортных происшествиях в Российской Федерации за 9 месяцев 2021 года, которые представил “Научный центр безопасности дорожного движения Министерства внутренних дел Российской Федерации”, доля ДТП при технической неисправности авто составила 5,9%, а это 5650 из зафиксированных 96314 случаев. [10] Также стоит отметить достаточно высокий показатель тяжести последствий, равный 10,3. На рисунке 1.3.1 представлена диаграмма количества ДТП и их доли из-за основных и сопутствующих причин.

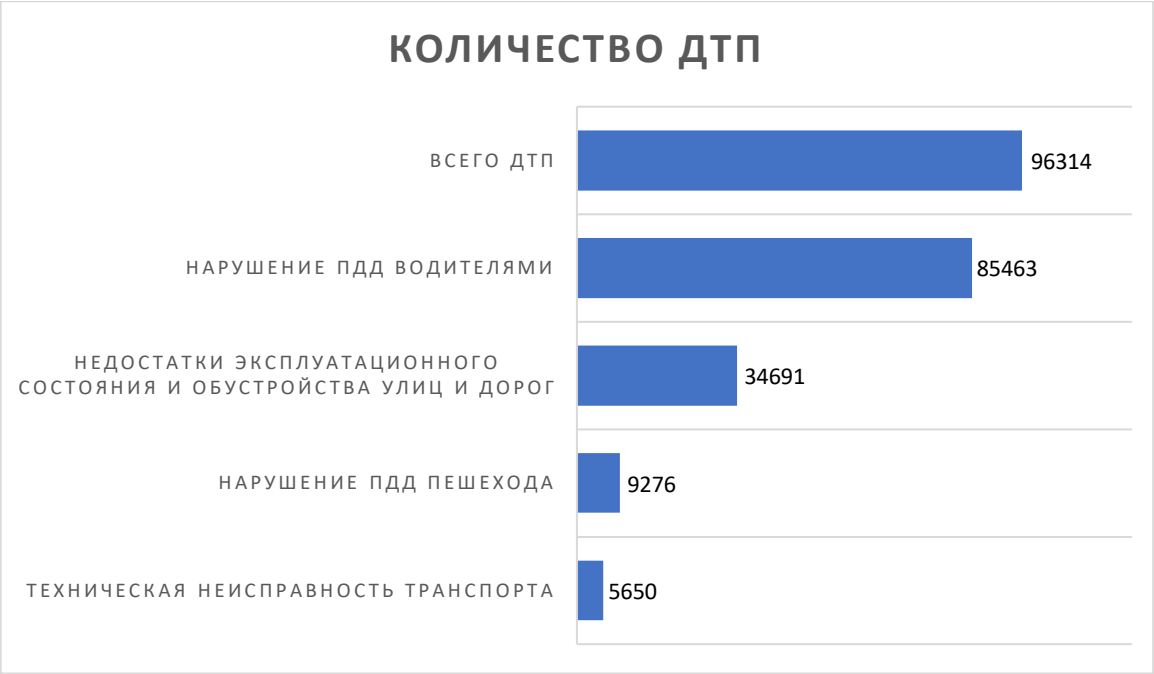

Рисунок 1.3.1 – Количество ДТП и их доля из-за основных и сопутствующих причин

Чтобы избежать такого рода ДТП, необходимо организовать правильный процесс планирования, связанный с обслуживанием автомобиля. Лучшим решением будет – использовать мобильный планировщик задач, который будет хранить всю необходимую информацию об авто и запланированных с ним событий.

1.4 Анализ существующих решений

В магазинах приложений было найдено не так много сервисов для обслуживания автомобилей и планирования задач. Среди всех существующих можно выделить следующие:

- “Обслуживание автомобиля Free”; [8]
- “Сервисная книжка автомобиля”. [9]

Обслуживание автомобиля Free – мобильное приложение, разработанное специально для автовладельцев, которые хотят вести учет состояния деталей машины.

Основные назначением данного программного продукта является автоматизация учета состояния разных механизмом авто. В приложение существует база, в которой содержатся основные механизмы автомобиля. Также, при необходимости, пользователи могут добавить отсутствующие в базе детали. Со слов разработчиков, приложение имеет интуитивно-понятный интерфейс, который лёгок в освоении.

Пример работы в приложении “Обслуживание автомобиля Free” приведен на рисунке 1.4.1.

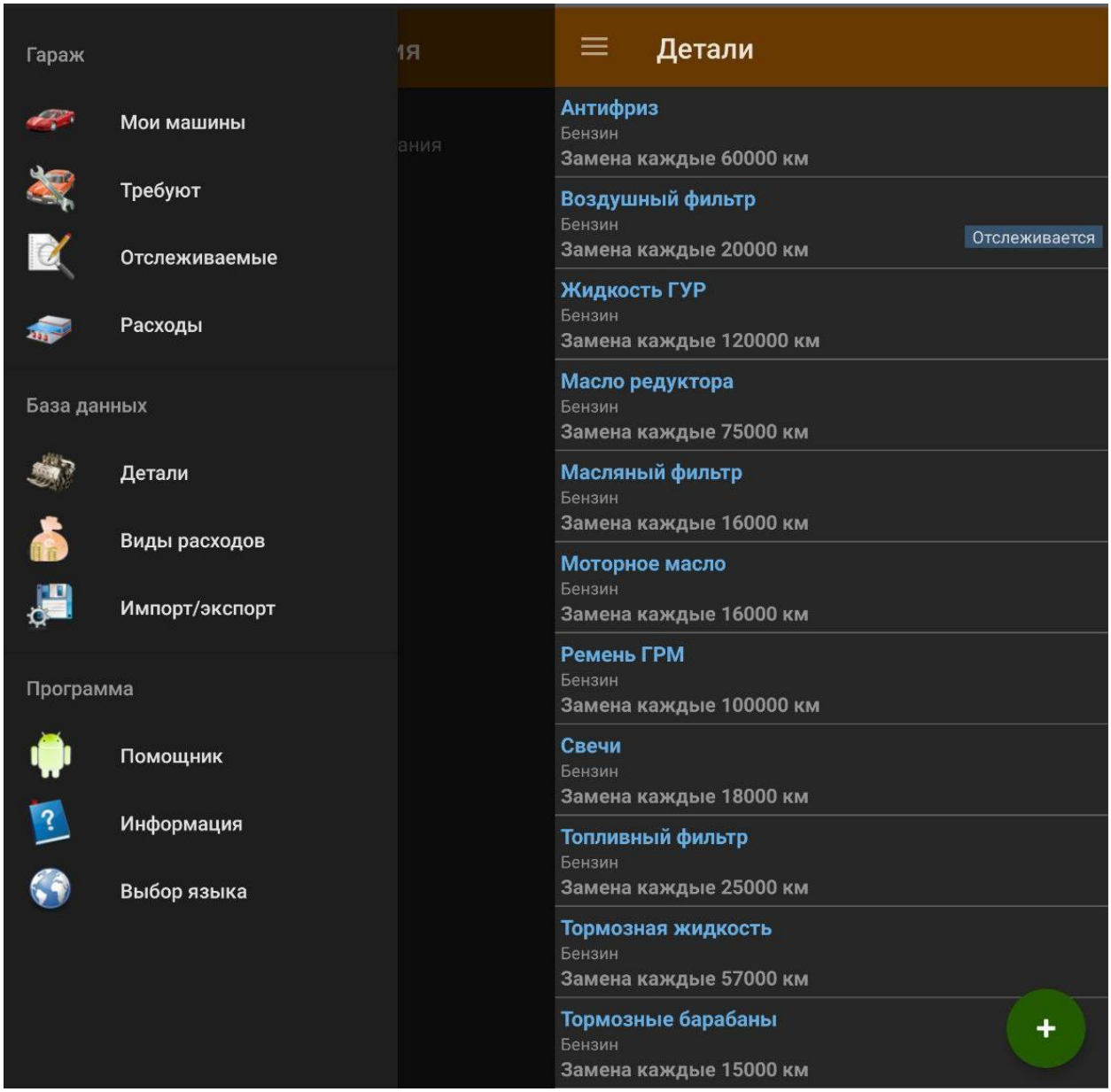

Рисунок 1.4.1 – Приложение “Обслуживание автомобиля Free”

Сервисная книжка автомобиля – это более популярное приложение, разработанное для ведения сервисной книжки автомобиля. Главное его отличие от предыдущего приложения – это наличие уведомление. В приложении можно поставить напоминания на замену необходимой детали или поездку в сервис. Также, сервис позволяет отслеживать пользователям расходы на свое транспортное средство, включая покупку новых запчастей, оплату штрафов и топлива и т.д.

Пример работы в приложении “Сервисная книжка автомобиля” приведен на рисунке 1.4.2.

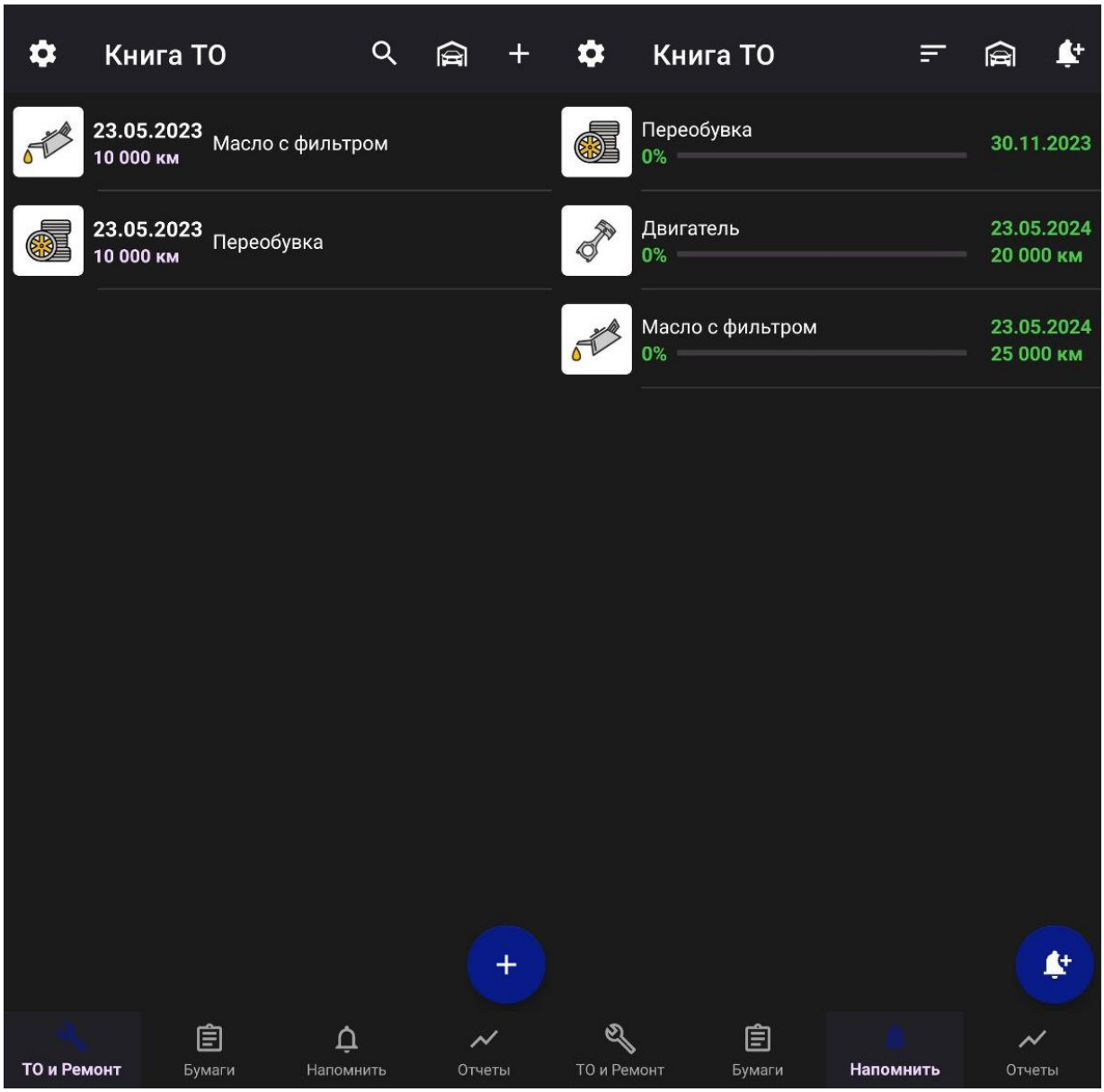

Рисунок 1.4.2 – Приложение “Сервисная книжка автомобиля”

Проведя анализ рассмотренных программных средств, можно отметить некоторые недостатки. В первом приложении, отсутствует возможность планирования и организации задач, связанных с автомобилем. Например, отсутствуют необходимые напоминания об окончании срока эксплуатации деталей авто и об указании актуального пробега. Также, в приложении “Обслуживание автомобиля Free” нет возможности создавать категории для групп деталей. Это сильно влияет на скорость поиска нужной детали и понижает степень удобства использования приложения.

В приложении “Сервисная книжка автомобиля”, как и в прошлом приложении, отсутствуют категории деталей. Кроме того, в приложении сложно вести учет износа деталей автомобиля, т.к. для каждой замененной детали необходимо создавать соответствующую запись в разделе “ТО и Ремонт”, а также отдельно создавать уведомление о будущей замене в разделе “Напомнить”. Еще одним недостатком является отсутствие хранения пробега автомобиля.

В таблице 1.4.3 приведен сравнительный анализ рассмотренных приложений.

Таблица 1.4.3 – Сравнительный анализ существующих решений

Функционал	**Существующие решения**	
	Обслуживание автомобиля Free	**Сервисная книжка автомобиля**
Хранения категорий деталей	-	-
Хранение деталей	+	+
Хранение пробега автомобиля	+	-
Учет износа деталей по актуальному пробегу	+	-

Продолжение таблицы 1.4.3.

Возможность планирования задач и событий	-	-
Уведомления об износе деталей	-	+
Уведомления для событий	-	-

Анализируя приложения “Обслуживание автомобиля Free” и “Сервисная книжка автомобиля” были выявлены их достоинства и недостатки, которые будут учтены при разработке мобильного приложения для обслуживания автомобиля и планирования задач на Android.

Глава 2. ОПИСАНИЕ МЕТОДОВ И ТЕХНОЛОГИЙ РАЗРАБОТКИ

2.1 Операционная система Android

Android – это всемирно известная открытая операционная система, основанная на Linux-ядре. [5] Она была разработана в 2005 году американским программистом и инженером Энди Рубином. В этом же году данная ОС была выкуплена компанией Google за 130 миллионов долларов. Спустя всего 3 года, после многочисленных доработок, было выпущено первое устройство под данной ОС. Как уже было сказано в главе “Общая информация о мобильных устройствах и приложениях”, большая часть всех телефон и других устройств в настоящее время выпускается на базе ОС Android. Это происходит из-за того, что данная ОС распространяется бесплатно и находится в свободном доступе для всех разработчиков мобильных устройств. Именно поэтому цены на такие устройства постоянно уменьшаются, что делает их более доступными для большинства населения планеты.

Ниже представлен список главных преимуществ ОС Android перед ее конкурентами:

- Открытая платформа;
- Огромный выбор мобильных устройств;
- Разнообразие приложений;
- Возможность кастомизации своего устройства;
- Наличие виджетов;
- Многозадачность;
- Интеграция с сервисами Google.

Остановимся на огромном разнообразии приложения для Android. В Google Play существует 34 категории мобильных приложений, в которых размещены, согласно статистике сервиса Statista, более 3.5 миллионов приложений. [12] Данная статистика приведена на рисунке 2.1.1.

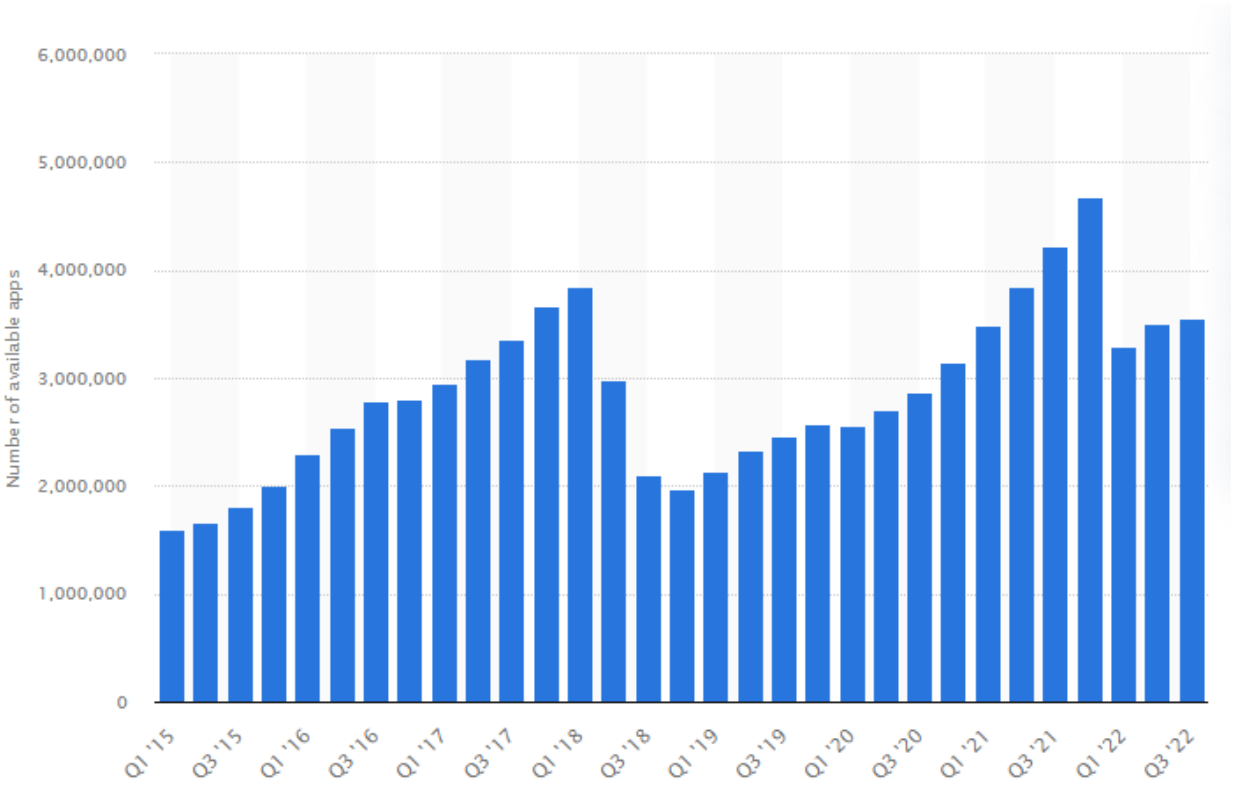

Рисунок 2.1.2 – Statista: количество доступных приложений по состоянию на 2022 год

На 2022 год количество скачиваний приложения из Google Play достигло цифры в 27.5 миллионов. [11] В то время, у App Store цифра заметно ниже и равна 9 миллионам. На рисунке 2.1.2 представлена статистика Statista по загрузке приложений на смартфоны.

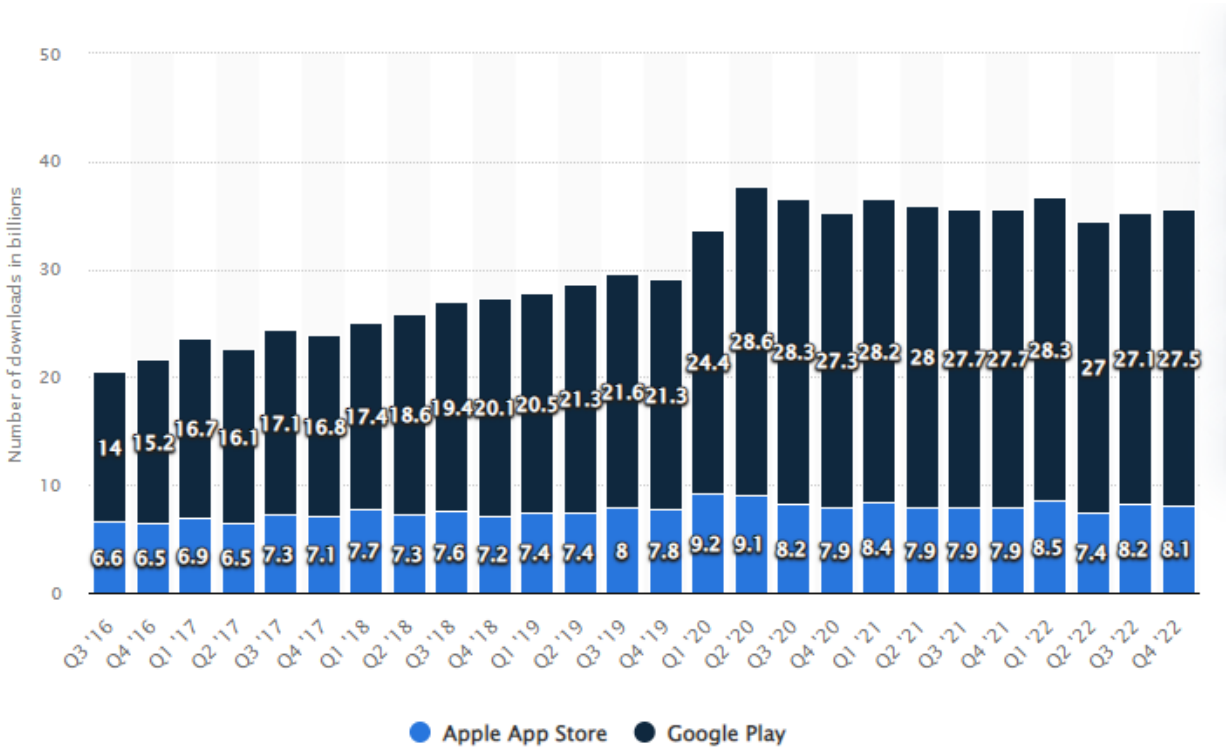

Рисунок 2.1.2 – Statista: загрузка приложений на смартфоны

Операционная система Android предлагает широкие возможности для создания разнообразных мобильных приложений. Например, компания Google выпустила целый набор инструментов для разработчиков – Android SDK.

Все приложения под Android пишутся на языках Java и Kotlin. В качестве среды разработки большинство программистов выбирают Android Studio за счет ее удобства и широкого выбора инструментов для разработки. [2]

Подводя итог, можно сказать, что операционная система Android является популярной платформой для разработки мобильных приложений. Открытость, многообразие устройств, богатые возможности разработки и интеграция с сервисами Google делают ее привлекательным выбором для разработчиков.

2.2 Постановка задачи и требований к мобильному приложению

Главной задачей, поставленной в данной выпускной квалификационной работе, является разработка мобильного приложения на базе операционной системы Android, где пользователь сможет вести учет деталей автомобиля и их износа, а также планировать разнообразные задачи, связанные с обслуживанием машины.

Для решения этой задачи должны быть выполнены такие пункты, как:

- анализ предметной области;
- анализ существующих решений;
- анализ возможных сред разработки необходимого мобильного приложения;
- формулировка требований к разрабатываемому мобильному приложению;
- написание мобильного приложения в выбранной среде разработки;
- формулировка руководства пользователя;
- тестирование мобильного приложения.

Функционал, который должен быть реализован в рамках выбранной темы выпускной квалификационной работы:

- Учет актуального пробега авто;
- Учет деталей авто;
- Автоматический подсчет пройденного километража для каждой детали;
- Календарь событий, в котором возможно добавить необходимые задачи;
- Возможность удалять и редактировать категории, детали и события в календаре;
- Уведомления для календаря событий;
- Интервальные уведомления для указания актуального пробега авто;
- Уведомления о необходимой замене детали при приближении максимального срока ее эксплуатации.

При запуске приложения, перед пользователем должно открываться главное меню со списком категорий деталей. При нажатии на категорию, должен открываться экран со списком деталей, выбранной категории. Каждая

деталь должна иметь свое название, срок максимальной эксплуатации (в километрах), срок текущей эксплуатации (в километрах), марка и краткое описание. Также для категорий и деталей должны быть реализованы функции их добавления, редактирования и удаления.

Помимо главного экрана, должны быть реализованы следующие экраны приложения, доступные из главного меню:

- Экран “События” с календарем, на котором можно добавлять, редактировать и удалять события. При нажатии на дату в календаре должен открываться список всех событий для выбранного дня. Оно должно состоять из названия, краткого описания и дата и времени его наступления.
- Экран “Настройки” с регулировкой напоминаний о необходимом обновлении актуального пробега. Пользователь должен иметь возможность указывать интервал (в днях с текущего момента), когда он хотел бы получать эти уведомления. Также в этом экране должно происходить указание актуального пробега автомобиля пользователя.

Также, должны быть реализованы дополнительные экраны:

- Экран создания-редактирования категории деталей;
- Экран создания-редактирования детали автомобиля;
- Экран создания-редактирования события в календаре.

Расчет износа любой детали должен автоматически формироваться исходя из актуального пробега автомобиля.

Уведомления об необходимой замене деталей должны приходить по достижению 60 %, 75 %, 80 %, 90 % и 100 % от указанного срока максимальной эксплуатации.

Ниже представлен список требований к интерфейсу приложения:

- Интерфейс должен быть разработан с учетом разрешения экранов мобильных устройств на базе ОС Android;
- Приложение должно поддерживать как вертикальную, так и горизонтальную ориентацию экрана;
- При входе в приложение, перед пользователем должен открываться экран “Категории”;
- Навигация в приложении должна осуществляться с помощью “Tab bar” внизу экрана;
- Реализация списков категорий и деталей должна быть выполнена с помощью компонента RecyclerView с собственными разработанными классами адаптеров и layout_item.xml для основных элементов на экранах со списками.

Учитывая поставленные задачи и требуемые функции, разработка мобильного приложения для обслуживания автомобиля и планирования задач на Android целесообразна именно на языке Kotlin.

2.3 Описание используемого языка программирования Kotlin

Kotlin – это объектно-ориентированный язык программирования для платформы Java, разработанная в 2011 году компанией JetBrains. Он предназначен для создания разнообразных приложений, включая:

- мобильные приложения для платформы Android;
- серверные приложения;
- веб-приложения. [7]

Kotlin является статически типизированным языком, который работает поверх виртуальной машины Java (JVM) и полностью совместим с существующими Java-библиотеками и инфраструктурой. [6]

Также, Kotlin будет полезен в написании кроссплатформенных приложений. В нем есть специальная технология Multiplatform, благодаря которой возможно написать приложения под разные ОС.

Все приложения до 2017 года писались на Java, а после начал набирать популярность уже Kotlin. На сегодняшний день, более 80% приложений в Play Market написаны на данном языке.

Ниже приведем список преимуществ использования языка Kotlin для написания приложений:

- Совместимость с Java;
- Краткий и ясный синтаксис;
- Автоматическая проверка кода на этапе компиляции;
- Расширения функций;
- Функциональное программирование;
- Поддержка корутин;
- Широкий выбор инструментов разработки;
- Безопасная работа с null-значениями;
- Активное сообщество разработчиков.

В настоящее время выделяют 3 основных IDE для разработки приложений:

- IntelliJ IDEA;
- Eclipse;
- Android Studio.

Глава 3. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

3.1 Особенности реализации и архитектуры

Для разработки мобильного приложения для обслуживания автомобиля и планирования задач на Android был выбран объектно-ориентированный язык программирования Kotlin.

В качестве среды разработки использовалась IDE Android Studio.

Для работы с БД в приложении была выбрана библиотека Room.

Для скорости работы приложения, удобства в его написании и тестировании был выбран паттерн MVVM. MVVM — это паттерн разработки, позволяющий разделить приложение на три основные функциональные части:

- Model – основная логика работы приложения;
- View – пользовательский интерфейс приложения;
- ViewModel – связь между Model и View через механизм привязки данных.

Архитектура мобильного приложения представлена на рисунке 3.1.1.

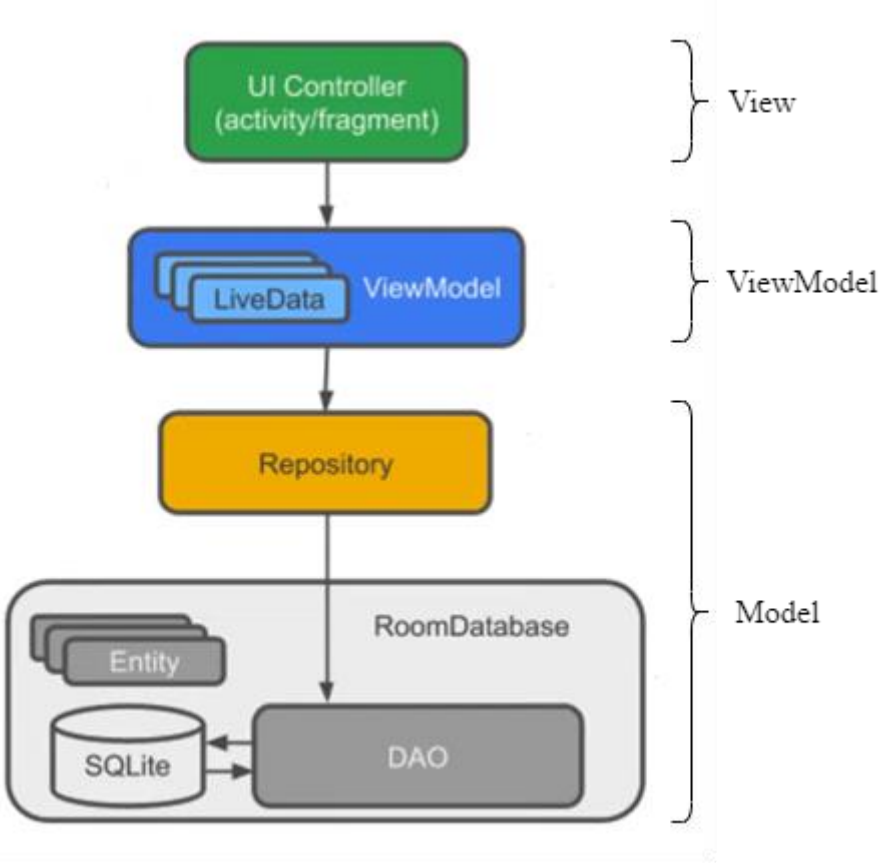

Рисунок 3.3.1 – Архитектура мобильного приложения

3.2 Разработка макета интерфейса приложения

3.2.1 Графические ресурсы (drawable)

В разрабатываемом приложении реализованы drawable ресурсы, которые хранятся в виде файлов разрешения .xml и служат для рендеринга необходимых элементов экранных форм в виде векторных изображений. Их список и описание представлены ниже:

- ic_add – векторное изображение кнопки добавления объектов;
- ic_calendar – векторное изображение кнопки меню для перехода на экран “События”;
- ic_categories – векторное изображение кнопки меню для перехода на экран “Категории”;
- ic_setting – векторное изображение кнопки меню для перехода на экран “Настройки”;
- ic_launcher_background – векторное изображение заднего плана (фона) иконки приложения;
- ic_launcher_foreground – векторное изображение переднего плана иконки приложения;
- ic_repair – векторное изображение элемента, отображающего событие в календаре.

3.2.2 Ресурсы макетов (layout)

В приложении был реализованы ресурсы макетов layout, которые определяют структуру пользовательского интерфейса. Они также хранятся в виде файлов с разрешением .xml. Их список и описание представлены ниже:

- activity_add_edit_category – макет экрана добавления и редактирования категория;
- activity_add_edit_event – макет экрана добавления и редактирования событий;
- activity_add_edit_part – макет экрана добавления и редактирования деталей;

- activity_category_parts – макет экрана со списком деталей, выбранной пользователем категории;
- activity_date_events – макет экрана со списком событий;
- activity_main – макет меню приложения;
- auto_event_rv_item – макет отображения информации о событии;
- category_rv_item – макет отображения информации о категории;
- part_rv_item – макет отображения информации о детали;
- fragment_categories – макет экрана со списком категорий;
- fragment_events – макет экрана календаря событий;
- root_preference – макет экрана настроек приложения.

3.2.3 Ресурсы простых значений (values)

Values предназначены для хранения ресурсов разных типов. Ниже перечислим те, что используются в приложении:

- arrays – хранит ресурсы типа массив строк;
- colors – содержит цветовые ресурсы;
- lc_launcher_background – хранит ресурс цвета заднего плана (фона) иконки приложения;
- themes – содержит ресурсы визуального стиля приложения;
- themes (night) – содержит ресурсы визуального стиля приложения, но для темного режима мобильного устройства.

3.3 Разработка локальной БД

Базы данных организованы с помощью библиотеки Room.

Room – это библиотека, созданная компанией Google. Она обеспечивает свободный доступ к БД, используя все возможности SQLite. Она достаточно проста в понимании и требует от разработчиков минимум действий: написать структуру данных, их вид и способы взаимодействия.

Преимущества библиотеки Room:

- Проверка SQL-запросов на этапе компиляции;

- Предоставляет комфортные для разработчиков аннотации, с помощью которых происходит взаимодействие с данными;
- Упрощенные способы миграции БД.

Room включает в себя три главных составляющие:

- Класс БД;
- Объекты данных;
- Объекты доступа к данным (DAO).

На рисунке 3.3.1 представлена схема архитектуры библиотеки Room.

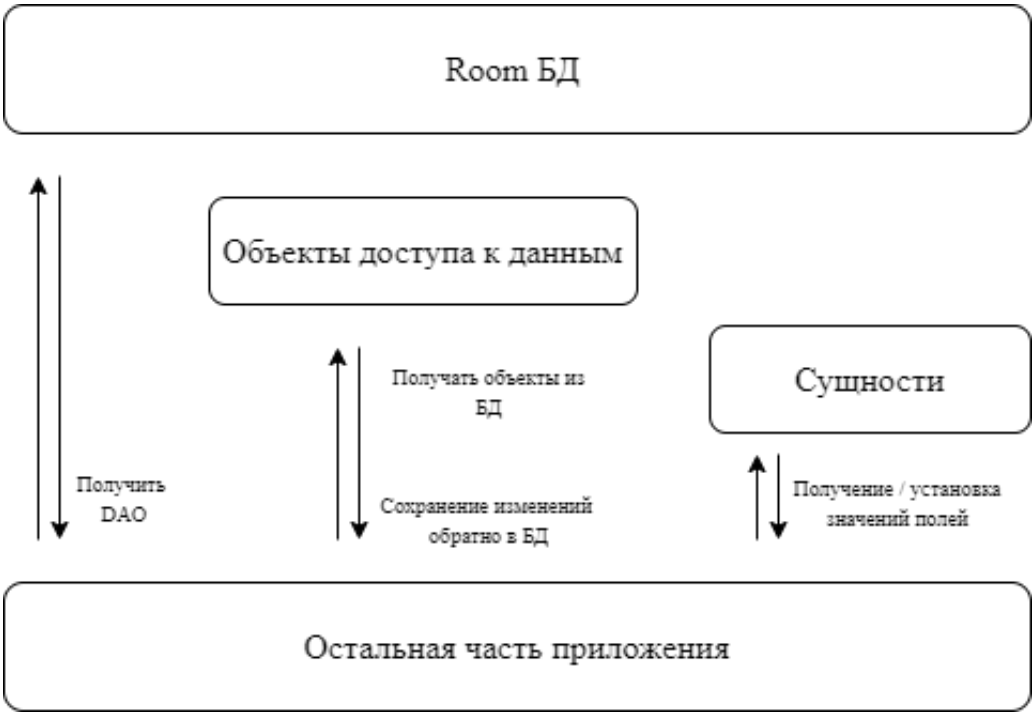

Рисунок 3.3.1 – Схема архитектуры библиотеки Room

3.3.1 Классы сущностей (entities)

В ходе разработки мобильного приложения было организовано 3 класса сущностей:

- AutoEvent – класс сущности для событий;
- Category – класс сущности для категорий;
- Part – класс сущности для деталей категории.

Они отвечают за организацию БД с помощью библиотеки Room. Организация таблиц происходит за счет конструкторов, которые написаны для каждой сущности.

На рисунке 3.3.1.1 представлены все сущности и их взаимоотношения.

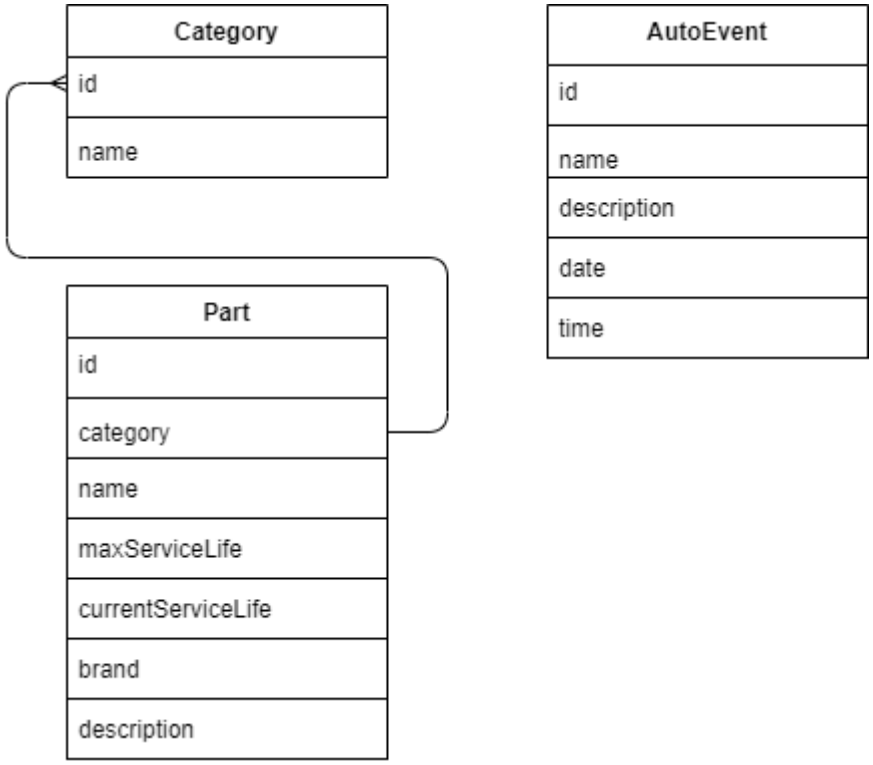

Рисунок 3.3.1.1 – Сущности и их отношения

3.3.2 Таблица autoEventsTable

Таблица autoEventsTable содержит информацию о событиях (рис. 3.3.2.1).

autoEventsTable	
id	Int
name	String
description	String
date	String
time	String

Рисунок 3.3.2.1 – Таблица autoEventsTable

Ниже представлено описание полей таблицы:

- id – уникальный идентификатор события;
- name – название события;
- description – краткое описание события;

- date – дата события;
- time – время события.

3.3.3 Таблица categoriesTable

Таблица categoriesTable содержит информацию о категориях деталей (рис. 3.3.3.1).

categoriesTable	
id	Int
name	String

Рисунок 3.3.3.1 – Таблица categoriesTable

Ниже представлено описание полей таблицы:

- id – уникальный идентификатор категории;
- name – название категории.

3.3.4 Таблица partsTable

Таблица partsTable содержит информацию о автомобильных деталях (рис. 3.3.4.1).

partsTable	
id	Int
category	Int
name	String
maxServiceLife	Int
currentServiceLife	Int
brand	String
description	String

Рисунок 3.3.4.1 – Таблица partsTable

Ниже представлено описание полей таблицы:

- id – уникальный идентификатор события;

- category – идентификатор категории детали;
- name – название детали;
- maxServiceLife – срок максимальной эксплуатации детали в километрах;
- currentServiceLife – срок текущей эксплуатации детали в километрах;
- brand – марка детали;
- description – краткое описание детали.

Кроме перечисленных полей, таблица partsTable содержит один внешний ключ для связи с таблицей categoriesTable. Это необходимо для создания отношений категория-деталь. При удалении категории, все ее детали будут также удалены.

Пример работы БД в готовом приложении представлен на рисунке 3.3.4.2.

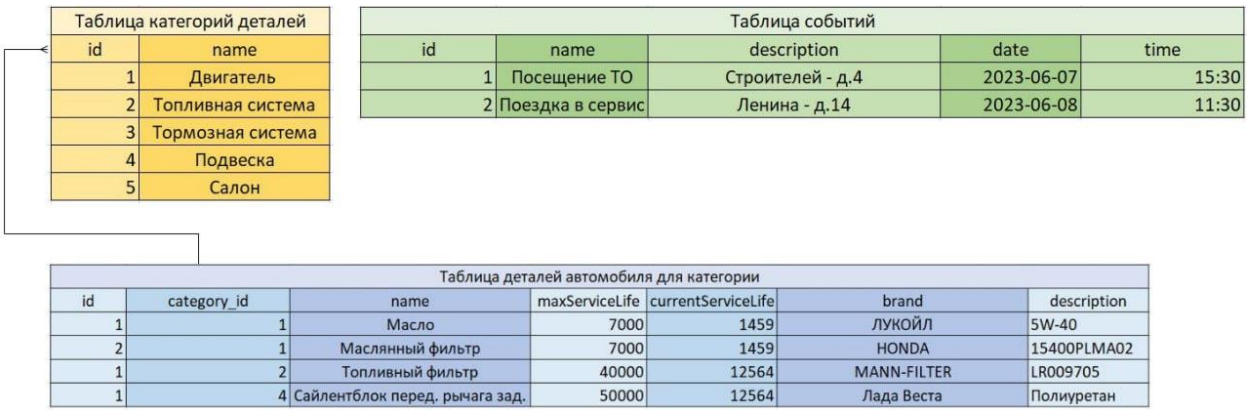

Таблица категорий деталей

id	name
1	Двигатель
2	Топливная система
3	Тормозная система
4	Подвеска
5	Салон

Таблица событий

id	name	description	date	time
1	Посещение ТО	Строителей - д.4	2023-06-07	15:30
2	Поездка в сервис	Ленина - д.14	2023-06-08	11:30

Таблица деталей автомобиля для категории

id	category_id	name	maxServiceLife	currentServiceLife	brand	description
1	1	Масло	7000	1459	ЛУКОЙЛ	5W-40
2	1	Маслянный фильтр	7000	1459	HONDA	15400PLMA02
1	2	Топливный фильтр	40000	12564	MANN-FILTER	LR009705
1	4	Сайлентблок перед. рычага зад.	50000	12564	Лада Веста	Полиуретан

Рисунок 3.3.4.2 – Пример работы БД

3.4 Описание работы приложения

Данный раздел содержит информацию об основной логике работы приложения и классах для взаимодействия с БД.

3.4.1 Классы объектов доступа к данным (DAO)

Объекты доступа к данным (DAO) необходимы для взаимодействия с созданной БД.

В ходе написания приложения, было реализовано 3 таких класса:

- AutoEventDao – DAO-интерфейс для выполнения операций над объектами класса AutoEvent (событие) в БД;
- CategoryDao – DAO-интерфейс для выполнения операций над объектами класса Category (категория) в БД;
- PartDao – DAO-интерфейс для выполнения операций над объектами класса Part (запчасть) в БД.

Перечисленные DAO-интерфейсы содержат следующие аннотации для работы с БД:

- @Insert – вставка списка объектов или одного объекта;
- @Update – обновление объекта;
- @Delete – удаление объекта;
- @Query – получение списка объектов.

3.4.2 Класс для хранения БД (AppDatabase)

Абстрактный класс AppDatabase определяет конфигурацию БД и служит для организации доступа к данным. В нем мы регистрируем сущности и DAO-интерфейсы.

На вход абстрактный класс AppDatabase получает классы сущностей БД.

Также с помощью данного класса происходит подготовка данных для загрузки в БД при первом запуске.

3.4.3 Интерфейс исполнителя (executors)

Использование интерфейса Executor позволяет создать поток ввода-вывода при первом обращении к БД и заполнить таблицу предварительно подготовленными данными. В нашем случае это будут пять категорий деталей автомобиля.

3.5 Разработка UI классов

3.5.1 Классы пользовательских действий (activities)

Activity – это главный компонент при разработке приложения. Он отвечает за взаимодействие пользователя с приложением, а также за

навигацию внутри приложения. Activity представляет собой окно с пользовательским интерфейсом

В приложении были разработаны следующие activity:

- MainActivity – класс главного activity приложения, который включает фрагмент, содержащий нижнюю навигацию и основной загружаемый фрагмент, который содержит список категорий;
- AddEditCategoryActivity – класс activity для создания и редактирования категории;
- AddEditEventActivity – класс activity для создания и редактирования события;
- AddEditPartActivity – класс activity для создания и редактирования детали;
- CategoryPartsActivity – класс activity для списка запчастей выбранной категории;
- DateEventsActivity – класс activity для списка событий выбранной даты.

3.5.2 Классы фрагментов (fragments)

Фрагменты – это составные пользовательского интерфейса. Они представляют собой определенную операцию или интерфейс, которые являются частью более крупного activity и не могут использоваться без него.

За интерфейс отвечают 3 фрагмента между которыми можно переключаться в главном activity:

- CategoriesFragment – фрагмент, отвечающий за вывод списка категорий;
- EventsFragment – фрагмент, отвечающий за вывод календаря с событиями;
- SettingsFragment – фрагмент настроек, отвечающий за хранение интервала между оповещениями об обновлении пробега и за хранение значения пробега.

3.5.3 Классы управления данными (ViewModels)

ViewModels – это классы, отвечающие за управление данными и их предоставление UI компонентам. Они помогают отделять логику работы с данными от интерфейса пользователя. Все действия пользователя идут через ViewModel, исключая изменение данных в Activity.

По ходу написания приложения было разработано пять классов ViewModel:

- AutoEventViewModel – класс для хранения и взаимодействия с данными Activity и Fragment событий;
- AutoEventViewModelFactory – класс-фабрика для того, чтобы создавать объекты AutoEventViewModel с дополнительными параметрами;
- CategoryViewModel – класс для хранения и взаимодействия с данными Activity и Fragment категорий;
- PartViewModel – класс для хранения и взаимодействия с данными Activity и Fragment деталей;
- PartViewModelFactory – класс-фабрика для того, чтобы создавать объекты PartViewModel с дополнительными параметрами.

Классы ViewModels напрямую связаны с репозиториями, о которых будет рассказано в следующем разделе.

3.5.4 Классы репозиториев (repositories)

Классы repositories позволяют приложению подключаться к различным источникам данных, после чего они отправляются в основной пользовательский интерфейс. Источник данных в нашем приложение – это локальная БД.

В нашем приложении они позволяют получить разные наборы данных для случаев, когда нам нужны, например, только запчасти определенной категории или все запчасти. Каждый из созданных репозиториев содержит различные методы для взаимодействия с БД.

По ходу написания приложения было разработано три класса репозиториев:

- AutoEventRepository – класс-репозиторий для выполнения операций над объектами класса AutoEvent (событие);
- CategoryRepository – класс-репозиторий для выполнения операций над объектами класса Category (категория);
- PartRepository – класс-репозиторий для выполнения операций над объектами класса Part (запчасть).

Классы ViewModels и репозитории нужны в первую очередь для реализации архитектуры MVVM.

3.5.5 Классы адаптеры (adapters)

В activity и фрагментах используются классы адаптеры. В разрабатываемом приложении они позволяют выводить объекты в виде карточек в перелистываемом списке – RecyclerView, а также обрабатывают действия над этими карточками: короткий клик и длинный клик.

Ниже представлен список разработанных классов адаптеров:

- AutoEventRVAdapter – класс адаптер для вывода списка событий в RecyclerView;
- CategoryRVAdapter – класс адаптер для вывода списка категорий в RecyclerView;
- PartRVAdapter – класс адаптер для вывода списка деталей в RecyclerView.

3.6 Разработка уведомлений в приложении

3.6.1 Класс немедленных оповещений (NotificationHelper)

Класс NotificationHelper позволяет создавать немедленные оповещения по запросу. В разрабатываемом приложении он предназначен для оповещений об износе деталей.

3.6.2 Классы отложенных оповещений (alarms)

Классы alarms позволяют организовать в нашем приложении подсистему отложенных уведомлений. Они будут появляться в строке уведомлений, при определенном сигнале.

По ходу написания кода, были разработаны три класса alarms:

- AlarmHelper – главный класс-помощник для создания отложенных напоминаний;
- EventReceiver – класс-получатель (слушатель) отложенных напоминаний о событиях;
- MileageRemindReceiver – класс-получатель (слушатель) отложенных напоминаний об обновлении пробега.

3.7 Манифест приложения

Манифест – это список правил, без которых приложение не будет работать. Все правила находятся в файле AndroidManifest.xml.

Ниже представлен список правил, которые были внесены для корректной работы приложения:

- Правила для резервного копирования и восстановления данных;
- Ссылка на значок приложения;
- Заголовок пользовательских экранов;
- Правила для работы с RTL-макетами;
- Ссылка на файл с визуальными стилями приложения;
- Описание компонентов activity;
- Описание <intent-filter> для главного экрана приложения;
- Описание <receiver> для уведомлений.

3.8 Руководство пользователя

Для начала работы с мобильным приложением для обслуживания автомобиля и планирования задач на Android необходимо скачать его в Play Market, когда оно будет доступно, или же на данном этапе разработке скачать и установить специальный apk-файл (рис. 3.4.1).

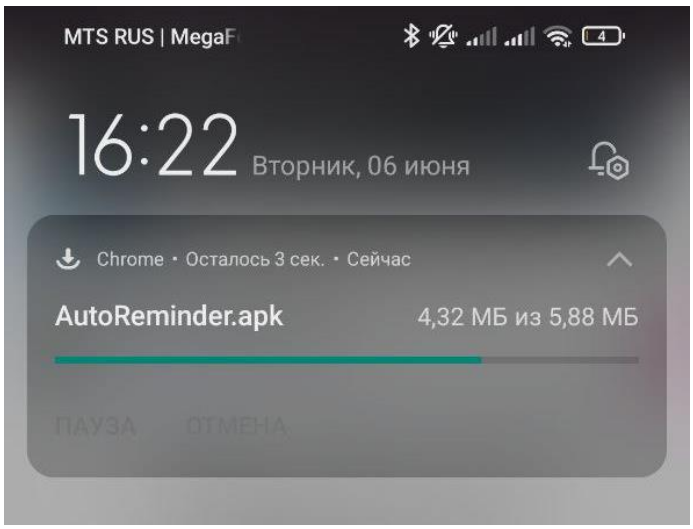

Рисунок 3.4.1 – Скивание apk-файла

После окончания скачивания файла, необходимо устновить приложение на устройство. Для это запускаем apk-файл и нажимаем кнопку установить (рис. 3.4.2).

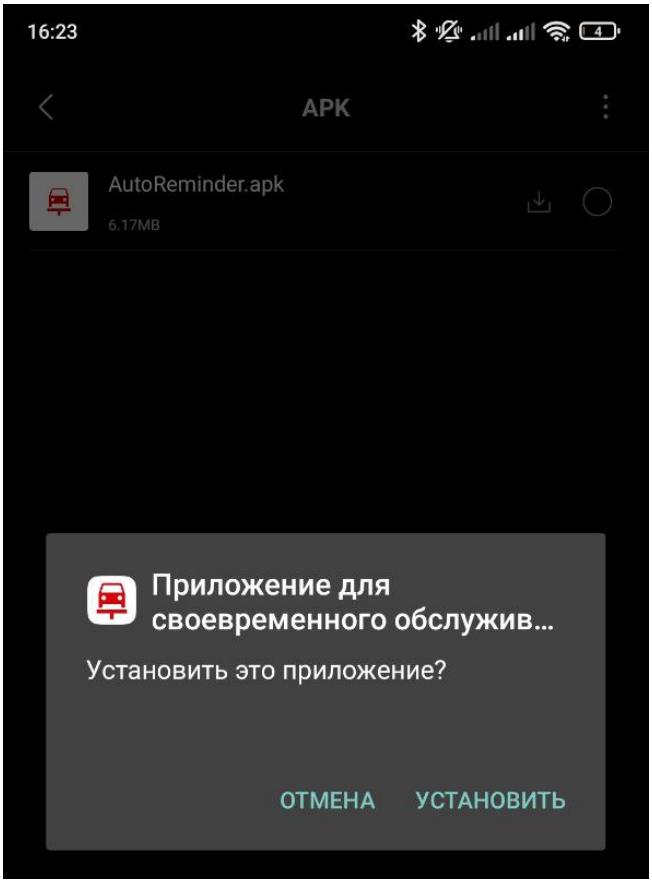

Рисунок 3.4.2 – Установка приложения

После загрузки приложения, можно открыть его, кликнув на соответствующий ярлык (рис. 3.4.3). Приложение успешно установлено и готово к работе.

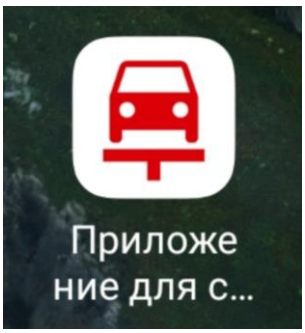

Рисунок 3.4.3 – Ярлык приложения

Далее, перед пользователем откроется экран “Категории” (рис. 3.4.4). В нем расположен динамический список с предустановленными категориями деталей.

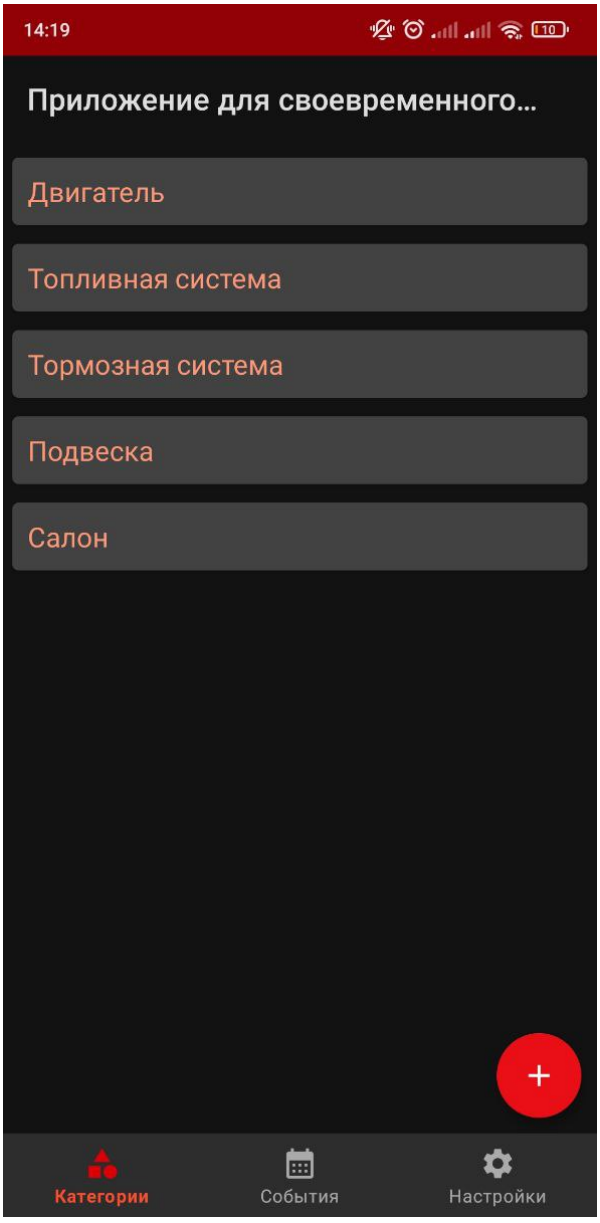

Рисунок 3.4.4 – Экран “Категории”

Здесь пользователь может:

- открыть существующую категорию для дальнейшей работы с деталями автомобиля, выполнив короткий клик по ее названию;
- добавить новую категорию, нажав на кнопку “+” в правом нижнем углу экрана;
- открыть экран редактирования и удаления категории, выполнив долгий клик по ее названию;
- перейти в другие экраны приложения.

Перейдем в категорию “Двигатель”. Как видно по рисунку 3.4.5, в данную категорию уже были добавлены детали ранее.

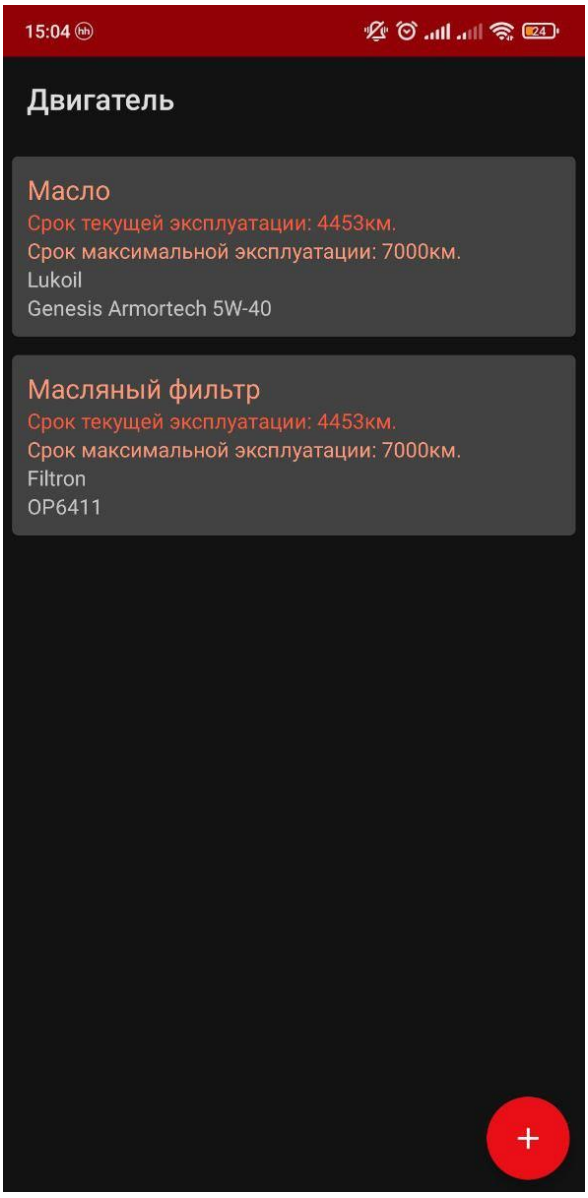

Рисунок 3.4.5 – Экран со списком деталей категории “Двигатель”

На данном экране пользователь может:

- ознакомиться со списком деталей автомобиля для выбранной категории;
- добавить новую деталь, нажав на кнопку “+” в правом нижнем углу экрана;
- открыть экран редактирования детали, выполнив короткий клик по ней;
- удалить деталь, выполнив долгий клик по ней.

По достижении определенного пробега детали пользователю будут приходить уведомления об ее износе (рис. 3.4.6).

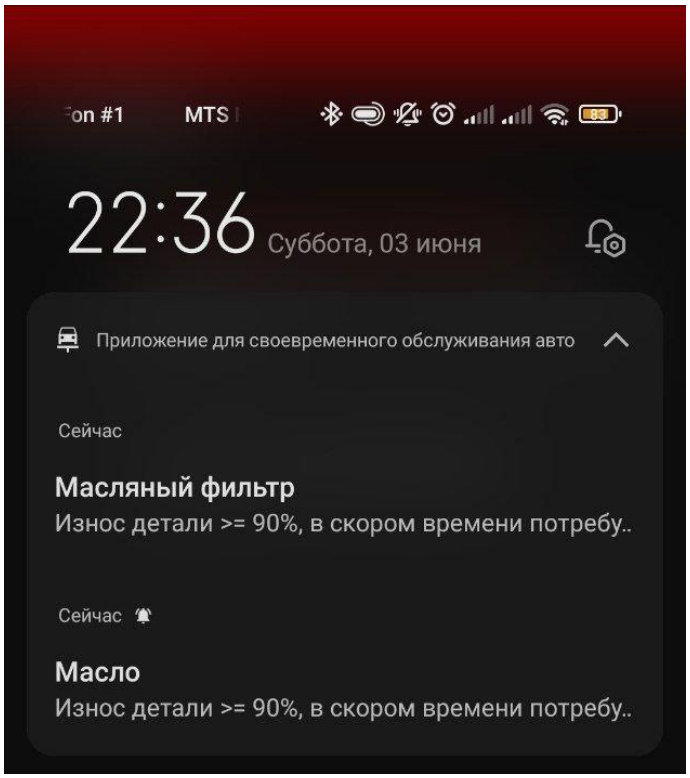

Рисунок 3.4.6 – Уведомление об износе детали

Чтобы перейти на экран “События”, необходимо вернуться обратно в “Категории” и выбрать в нижнем меню “События”. После этого, перед пользователем откроется календарь событий (рис. 3.4.7).

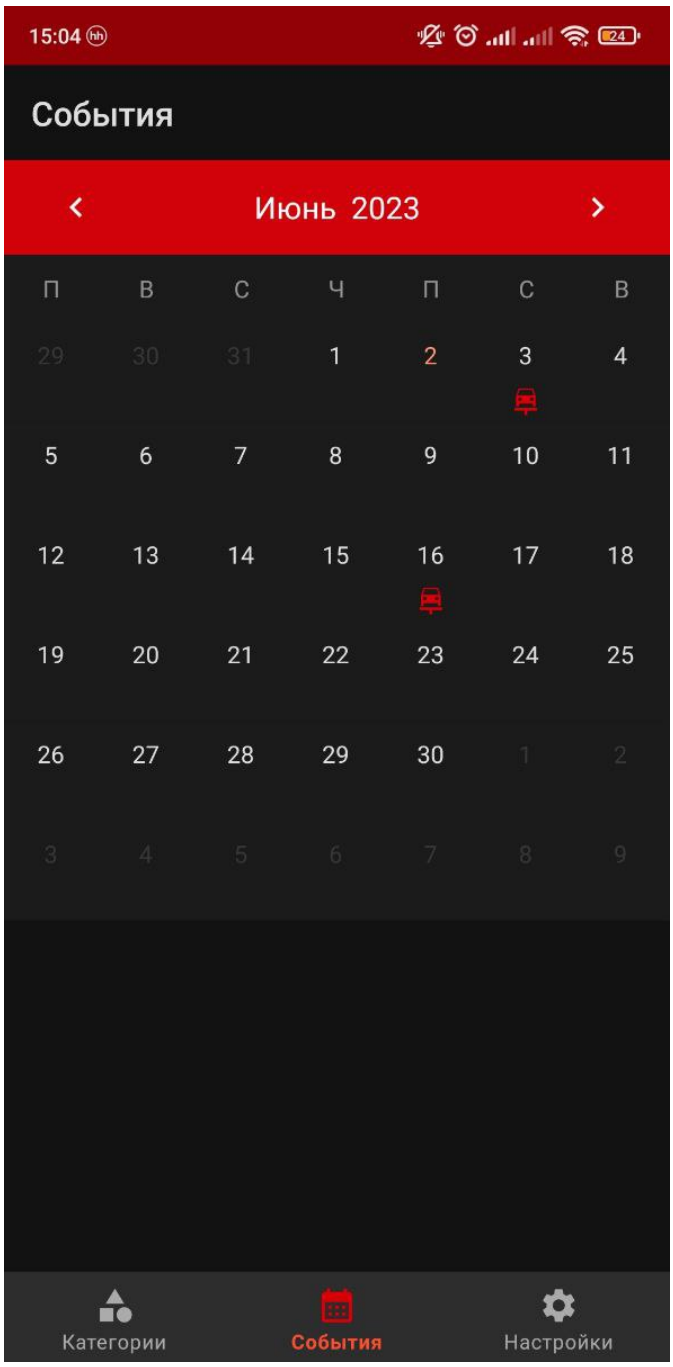

Рисунок 3.4.7 – Экран “События”

Для добавления события, необходимо выбрать дату и нажать на нее в календаре. После этого перед пользователем откроется пустой список, либо список с событиями для выбранной даты (рис. 3.4.8).

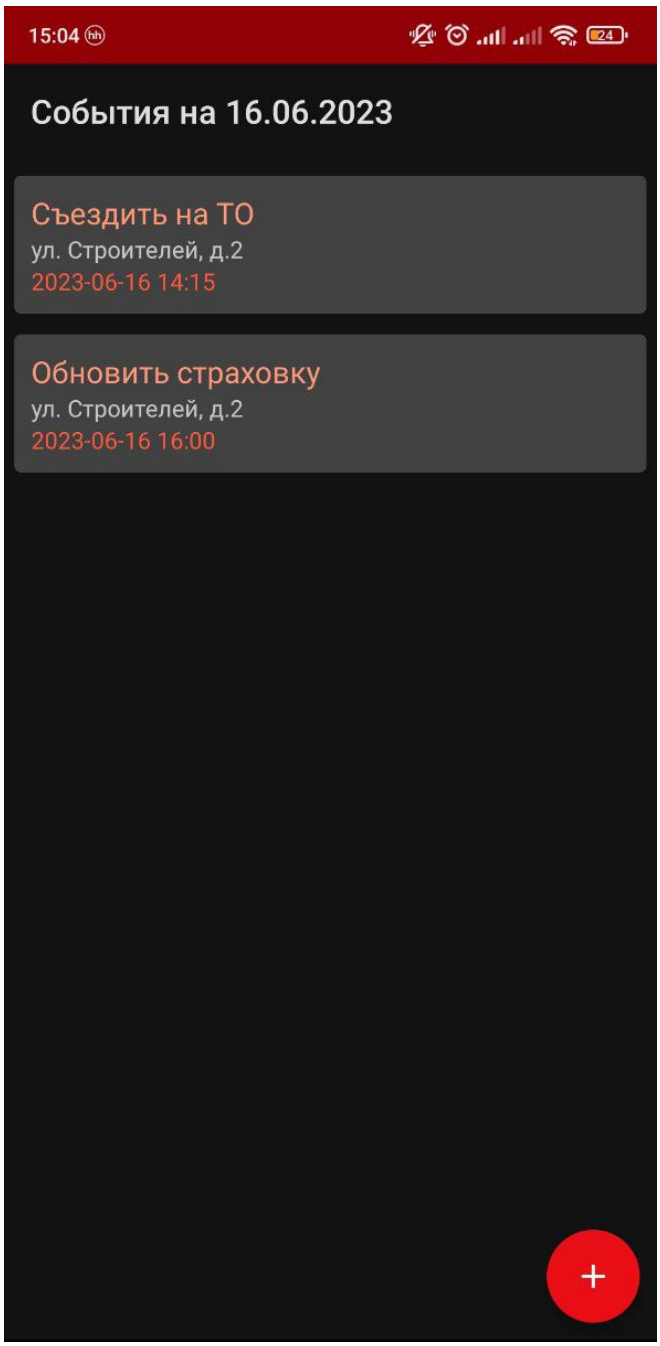

Рисунок 3.4.8 – Экран со списком событий

На данном экране пользователь может:

- ознакомиться со списком существующих событий для выбранной даты;
- добавить новое событие, нажав на кнопку “+” в правом нижнем углу экрана;
- открыть экран редактирования события, выполнив короткий клик по ней;
- удалить событие, выполнив долгий клик по ней.

Если для даты есть хотя бы одно событие, то она будет помечена специальным красным элементом в календаре.

За день и в день наступления события пользователь получит соответствующие уведомления (рис. 3.4.9).

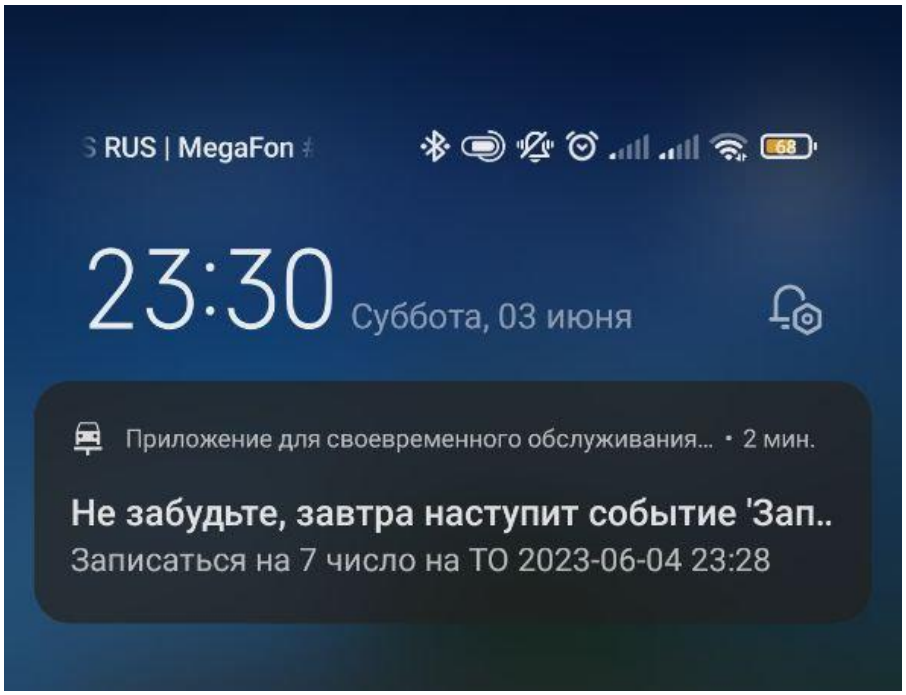

Рисунок 3.4.9 – Уведомление для события

Чтобы открыть последний экран “Настройки”, необходимо нажать в меню специальную одноименную кнопку. В данном экране пользователь настраивает интервал напоминаний для обновления пробега и изменяет, при необходимости, актуальный пробег своего автомобиля (рис. 3.4.10).

Рисунок 3.4.10 – Экран “Настройки”

Уведомления для обновления пробега продемонстрированы на рисунке 3.4.11.

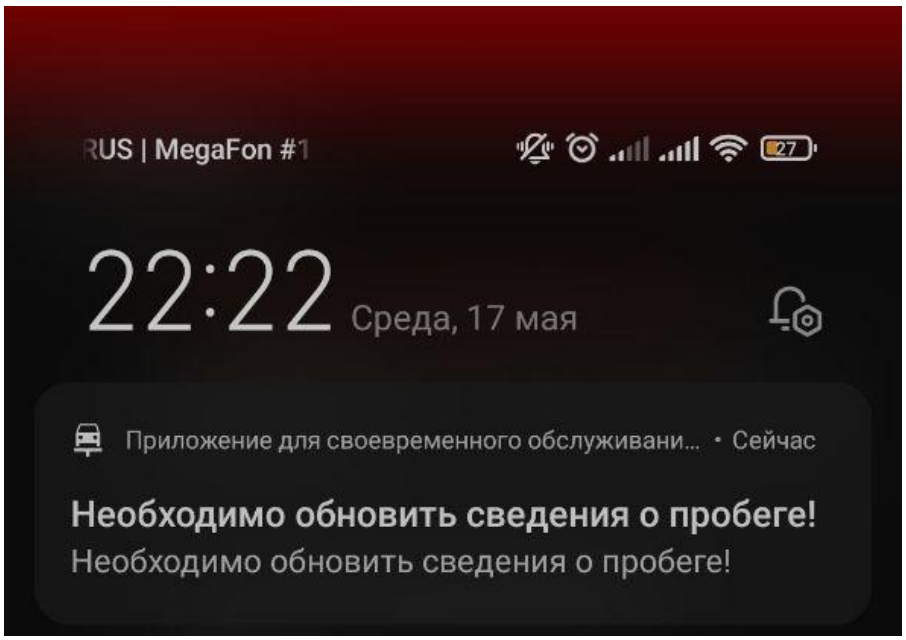

Рисунок 3.4.11 – Уведомления для обновления пробега

3.9 Тестирование мобильного приложения

3.8.1 Функциональное тестирование приложения

В рамках данной дипломной работы, учитывая все особенности финальной версии разрабатываемого приложения, было проведено функциональное тестирование методом черного ящика.

Функциональное тестирование является одним из видов тестирования программного обеспечения, направленным на проверку функциональности системы. Оно оценивает соответствие программного продукта требованиям и ожиданиям пользователей, а также его способность выполнять заданные функции.

Метод черного ящика означает, что специалист-тестировщик не обладает информацией о внутренней структуре и реализации программного продукта. Он тестирует программу, основываясь только на ее внешнем поведении и функциональности.

Для проведения тестирования был составлен список сценариев, представленный в таблице 4.1.1.

Таблица 4.1.1 – Состав и содержание сценариев испытания

№ п/п	Наименование сценария	Технические требования	Содержание	Критерий выполнения
1	Запуск приложения	**Требования к функциональным характеристикам.** Приложение должно эксплуатироваться на мобильных устройствах пользователей.	Запустить приложение	Методика считается выполненной, если будет открыто главное окно приложения. Методика считается невыполненной, если главное окно не будет открыто.
2	Создание категории	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю создавать категории.	Нажать на кнопку “+” справа снизу в главном меню. Ввести название категории. Нажать кнопку сохранить.	Методика считается выполненной, если будет добавлена новая категория в список категорий. Методика считается невыполненной, если категория не будет добавлена в список категорий.
3	Редактирование категории	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю редактировать категории.	Зажать категорию в главном меню. Отредактировать название категории. Нажать кнопку сохранить.	Методика считается выполненной, если название категории изменится. Методика считается невыполненной, если название категории не изменится.

Продолжение таблицы 4.1.1.

4	Удаление категории	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю удалять категории.	Выполнить продолжительное нажатие на категорию в главном меню. Нажать кнопку удалить.	Методика считается выполненной, если будет удалена выбранная категория из списка категорий. Методика считается невыполненной, если выбранная категория не будет удалена из списка категорий.
5	Выбор категории	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю выбирать категории.	Выбрать категорию в главном меню приложения.	Методика считается выполненной, если откроется выбранная категория. Методика считается невыполненной, если категория не будет открыта.
6	Создание детали	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю создавать детали.	Нажать на кнопку “+” справа снизу в выбранной категории. Ввести информацию о детали. Нажать кнопку сохранить.	Методика считается выполненной, если будет добавлена новая деталь в список всех деталей категории. Методика считается невыполненной, если деталь не будет добавлена в список всех деталей категории.

Продолжение таблицы 4.1.1.

7	Редактирование детали	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю редактировать детали.	Нажать на деталь в выбранной категории. Отредактировать информацию о детали. Нажать кнопку сохранить.	Методика считается выполненной, если информация о детали в списке всех деталей категории изменится. Методика считается невыполненной, если информация о детали не изменится в списке всех деталей категории.
8	Удаление детали	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю удалять детали.	Выполнить продолжительное нажатие на деталь в выбранной категории.	Методика считается выполненной, если будет удалена выбранная деталь из списка всех деталей категории. Методика считается невыполненной, если выбранная деталь не будет удалена из списка всех деталей категории.
9	Открытие экрана “События”	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю открывать вкладку “События” в меню.	Нажать на вкладку “События” внизу экрана.	Методика считается выполненной, если будет открыт экран “События”. Методика считается невыполненной, если экран “События” не будет открыт.

Продолжение таблицы 4.1.1.

10	Создание события	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю создавать события.	Выбрать дату в календаре событий. Нажать на кнопку “+” справа снизу для выбранной даты. Ввести информацию о событие. Нажать кнопку сохранить.	Методика считается выполненной, если будет добавлено новое событие в список всех событий для выбранной даты. Методика считается невыполненной, если новое событие не будет добавлена в список всех событий для выбранной даты.
11	Редактирование события	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю редактировать события.	Нажать на событие для выбранной даты. Отредактировать информацию о событие. Нажать кнопку сохранить.	Методика считается выполненной, если будет изменена информация о событие. Методика считается невыполненной, если информация о событие не будет изменена.
12	Удаление события	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю удалять события.	Выполнить продолжительное нажатие на событие для выбранной даты.	Методика считается выполненной, если будет удалено выбранное событие из списка всех событий для выбранной даты. Методика считается невыполненной, если выбранное событие не будет удалено из списка всех событий для выбранной даты.

Продолжение таблицы 4.1.1.

13	Открытие экрана “Настройки”	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю открывать вкладку “Найстройки” в меню.	Нажать на вкладку “Настройки” внизу экрана.	Методика считается выполненной, если будет открыт экран “Настройки”. Методика считается невыполненной, если экран “Настройки” не будет открыт.
14	Указание интервала напоминаний для обновления пробега	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю указывать интервал напоминаний для обновления пробега автомобиля.	Нажать на кнопку “Интервал напоминаний (в днях)”. Указать интервал напоминаний.	Методика считается выполненной, если значение интервала напоминаний будет изменено. Методика считается невыполненной, если значение интервала напоминаний не будет изменено.
15	Указание актуального пробега автомобиля	**Требования к функциональным характеристикам.** Приложение должно позволять пользователю указывать актуальный пробег автомобиля.	Нажать на кнопку “Текущий пробег автомобиля”. Указать текущий пробег автомобиля.	Методика считается выполненной, если значение актуального пробега автомобиля будет изменено. Методика считается невыполненной, если значение актуального пробега автомобиля не будет изменено.

Продолжение таблицы 4.1.1.

16	Проверка встроенных данных в локальной БД	**Требования к функциональным характеристикам.** Приложение должно отображать встроенные данные на экране приложения.	Проверить отображение встроенных категорий во вкладке “Категории”.	Методика считается выполненной, если встроенные категории отображены на экране. Методика считается невыполненной, если встроенные категории не будут отображены на экране.
17	Добавление данных в локальную БД	**Требования к функциональным характеристикам.** Приложение должно отображать внесенные пользователем данные на экране приложения.	Добавить новую категорию. Проверить отображение новой категории во вкладке “Категории”.	Методика считается выполненной, если новая категория отображена на экране. Методика считается невыполненной, если новая категория не будут отображена на экране.

3.8.2 Результаты тестирования приложения

Результаты проведения испытаний в рамках функционального тестирования методом черного ящика представлены в таблице 4.2.1.

Таблица 4.2.1 – Результаты проведения испытаний

№ п/п	Наименование сценария	Содержание	Участники	Результат выполнения
1	Запуск приложения	Запустить приложение	Славинский Б.Д.	Методика выполнена успешно
2	Создание категории	Нажать на кнопку “+” справа снизу в главном меню. Ввести название категории. Нажать кнопку сохранить.	Славинский Б.Д.	Методика выполнена успешно

Продолжение таблицы 4.2.1.

3	Редактирование категории	Зажать категорию в главном меню. Отредактировать название категории. Нажать кнопку сохранить.	Славинский Б.Д.	Методика выполнена успешно
4	Удаление категории	Выполнить продолжительное нажатие на категорию в главном меню. Нажать кнопку удалить.	Славинский Б.Д.	Методика выполнена успешно
5	Выбор категории	Выбрать категорию в главном меню приложения.	Славинский Б.Д.	Методика выполнена успешно
6	Создание детали	Нажать на кнопку “+” справа снизу в выбранной категории. Ввести информацию о детали. Нажать кнопку сохранить.	Славинский Б.Д.	Методика выполнена успешно
7	Редактирование детали	Нажать на деталь в выбранной категории. Отредактировать информацию о детали. Нажать кнопку сохранить.	Славинский Б.Д.	Методика выполнена успешно
8	Удаление детали	Выполнить продолжительное нажатие на деталь в выбранной категории.	Славинский Б.Д.	Методика выполнена успешно
9	Открытие экрана “События”	Нажать на вкладку “События” внизу экрана.	Славинский Б.Д.	Методика выполнена успешно

Продолжение таблицы 4.2.1.

10	Создание события	Выбрать дату в календаре событий. Нажать на кнопку “+” справа снизу для выбранной даты. Ввести информацию о событие. Нажать кнопку сохранить.	Славинский Б.Д.	Методика выполнена успешно
11	Редактирование события	Нажать на событие для выбранной даты. Отредактировать информацию о событие. Нажать кнопку сохранить.	Славинский Б.Д.	Методика выполнена успешно
12	Удаление события	Выполнить продолжительное нажатие на событие для выбранной даты.	Славинский Б.Д.	Методика выполнена успешно
13	Открытие экрана “Настройки”	Нажать на вкладку “Настройки” внизу экрана.	Славинский Б.Д.	Методика выполнена успешно
14	Указание интервала напоминаний для обновления пробега	Нажать на кнопку “Интервал напоминаний (в днях)”. Указать интервал напоминаний.	Славинский Б.Д.	Методика выполнена успешно
15	Указание актуального пробега автомобиля	Нажать на кнопку “Текущий пробег автомобиля”. Указать текущий пробег автомобиля.	Славинский Б.Д.	Методика выполнена успешно
16	Проверка встроенных данных в локальной БД	Проверить отображение встроенных категорий во вкладке “Категории”.	Славинский Б.Д.	Методика выполнена успешно

Продолжение таблицы 4.2.1.

17	Добавление данных в локальную БД	Добавить новую категорию. Проверить отображение новой категории во вкладке “Категории”.	Славинский Б.Д.	Методика выполнена успешно

ЗАКЛЮЧЕНИЕ

В ходе данной выпускной квалификационной работы было разработано мобильное приложение для обслуживания автомобиля и планирования задач на Android.

Разработка была выполнена “с нуля”, без использования каких-либо готовых решений.

В первую очередь был проведен анализ предметной области, а именно:

1. Анализ мобильных устройств и приложений;
2. Анализ приложений для планирования задач;
3. Сбор информации об обслуживание автомобилей;
4. Анализ существующих решений на рынке.

После этого, был сформирован список требований к разрабатываемому в данной выпускной квалификационной работе приложению. Также были выбраны язык и среда разработки: Kotlin и IDE Android Studio соотвественно.

Следующий этап заключался в разработке приложения для обслуживания автомобиля и планирования задач на Android. Здесь проводился анализ и выбор подходящих для разработки библиотек, а также формировалась будущая архитектура приложения. В итоге, был написан код для клиентской и серверной части приложения.

Для работы с БД в приложении была выбрана библиотека Room. Для скорости работы приложения, удобства в его написании и тестировании был выбран паттерн MVVM.

Все требования к мобильному приложению были учтены, а функционал, который должен быть реализован в рамках темы выпускной квалификационной работы, успешно разработан и протестирован. По результатам тестирования – баги найдены не были.

Для понимания взаимодействия пользователя с приложением было разработано руководство пользователя.

Реализованные возможности мобильного приложения позволяют пользователям контролировать износ всех деталей своего автомобиля, а также

вести календарь событий, чтобы не забыть о важных задачах, связанных с машиной.

В случае дальнейшего развития данного приложения, будут выделены в приоритет и учтены основные принципы и идеи заложенные в целях задачи без существенных изменений. Локальная база данных будет пополняться популярными среди пользователей категориями и деталями автомобиля, чтобы пользователь не тратил на это свое время. Также, планируется разработка сетевой версии приложения для подключения и сбора важной информации с автомобиля пользователя.

Для усовершенствования мобильного приложения планируется разработка профиля пользователя, в котором он сможет хранить важные автомобильные документы и дополнительную информацию об автомобиле, а также добавление возможности переключаться между несколькими автомобилями для тех пользователей, чей автопарк состоит не из одной машины.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Соколова, В. В. Разработка мобильных приложений : учебное пособие для среднего профессионального образования / В.В. Соколова. — Москва : Издательство Юрайт, 2023. – 175 с.
2. Виноградов, Д. В. Разработка мобильных приложений и облачные сервисы : учебное пособие / Д. В. Виноградов ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2022. – 235 с.
3. DIGITAL 2022: GLOBAL OVERVIEW REPORT [Электронный ресурс]. URL: https://datareportal.com/reports/digital-2022-global-overview-report
4. Mobile Operating System Market Share Worldwide [Электронный ресурс]. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide
5. Официальная документация Android [Электронный ресурс]. URL: https://developer.android.com/docs
6. Официальная документация Kotlin [Электронный ресурс]. URL: https://kotlinlang.org/docs/home.html
7. Жемеров Д., Исакова С. Kotlin в действии. / пер. с англ. Киселев А. Н. - М.: ДМК Пресс, 2018. - 402 с.
8. Приложения в Google Play - Обслуживание автомобиля Free [Электронный ресурс]. URL: https://play.google.com/store/apps/details?id=com.replacement.free
9. Приложения в Google Play - Сервисная книжка автомобиля [Электронный ресурс]. URL: https://play.google.com/store/apps/details?id=com.cars.android.carapps.carnotes
10. Дорожно-транспортная аварийность в Российской Федерации за 9 месяцев 2021 года. Информационно-аналитический обзор. – М.: ФКУ «НЦ БДД МВД России», 2021. - 39 с.
11. Number of iOS and Google Play app downloads as of Q4 2022 [Электронный ресурс]. URL: https://www.statista.com/statistics/695094/quarterly-number-of-mobile-app-downloads-store/

12. Google Play: number of available apps as of Q3 2022 [Электронный ресурс]. URL: https://www.statista.com/statistics/289418/number-of-available-apps-in-the-google-play-store-quarter/

ПРИЛОЖЕНИЕ

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.AutoReminderApp"
        tools:targetApi="31">
        <activity
            android:name=".ui.activities.AddEditEventActivity"
            android:exported="false" />
        <activity
            android:name=".ui.activities.DateEventsActivity"
            android:exported="false" />
        <activity
```

```
            android:name=".ui.activities.AddEditPartActivity"
            android:exported="false" />
        <activity
            android:name=".ui.activities.CategoryPartsActivity"
            android:exported="false" />
        <activity
            android:name=".ui.activities.AddEditCategoryActivity"
            android:exported="false" />
        <activity
            android:name=".ui.activities.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".alarms.EventReceiver" />
        <receiver android:name=".alarms.MileageRemindReceiver" />
    </application>
</manifest>
```

AlarmHelper.kt

```
package com.a.autoreminderapp.alarms


import android.app.AlarmManager
import android.app.PendingIntent
```

```
import android.content.Context
import android.content.Intent
import com.a.autoreminderapp.data.entities.AutoEvent
import java.text.DateFormat
import java.text.SimpleDateFormat
import java.util.*

class AlarmHelper {
    companion object {
        fun makeAlarmsForEvent(context: Context, autoEvent: AutoEvent) {
            makeAlarmForEvent(context, autoEvent, 0)
            makeAlarmForEvent(context, autoEvent, 1)
        }

        fun removeAlarmsForEvent(context: Context, autoEvent: AutoEvent){
            removeAlarmForEvent(context, autoEvent, 0)
            removeAlarmForEvent(context, autoEvent, 1)
        }

        private fun makeAlarmForEvent(context: Context, autoEvent: AutoEvent, additional: Int) {
            val am = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager
            val intent = Intent(context, EventReceiver::class.java)
            if (additional != 1) {
                intent.putExtra("eventName", autoEvent.name)
            } else {
```

```
            intent.putExtra(
                "eventName",
                "Не забудьте, завтра наступит событие '" + autoEvent.name + "'"
            )
        }
        intent.putExtra("eventDescription", autoEvent.description)
        intent.putExtra("eventDate", autoEvent.date)
        intent.putExtra("eventTime", autoEvent.time)
        val dateFormat: DateFormat =
            SimpleDateFormat("yyyy-MM-dd hh:mm", Locale.getDefault())
        val eventDate: Date = dateFormat.parse(autoEvent.date + " " + autoEvent.time) as Date
        val requestCode: Long = eventDate.time + autoEvent.id + additional
        val pendingIntent = PendingIntent.getBroadcast(
            context,
            requestCode.toInt(),
            intent,
            PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
        )
        val calendar = Calendar.getInstance()
        calendar.time = eventDate
        calendar.add(Calendar.DATE, -additional)
        am[AlarmManager.RTC_WAKEUP, calendar.timeInMillis] = pendingIntent
    }

    fun makeAlarmForMileageRemind(context: Context, daysInterval: Int) {
```

```
    val am = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager
    val intent = Intent(context, MileageRemindReceiver::class.java)
    val calendar = Calendar.getInstance()
    val requestCode: Int = 1
    val pendingIntent = PendingIntent.getBroadcast(
        context,
        requestCode,
        intent,
        PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
    )
    val dayMillis = AlarmManager.INTERVAL_DAY
    am.setRepeating(
        AlarmManager.RTC_WAKEUP,
        calendar.timeInMillis,
        dayMillis * daysInterval,
        pendingIntent
    )
}

fun removeAlarmForEvent(context: Context, autoEvent: AutoEvent, additional: Int){
    val intent = Intent(context, EventReceiver::class.java)
    val dateFormat: DateFormat =
        SimpleDateFormat("yyyy-MM-dd hh:mm", Locale.getDefault())
    val eventDate: Date = dateFormat.parse(autoEvent.date + " " + autoEvent.time) as Date
    val requestCode: Long = eventDate.time + autoEvent.id + additional
```

```
            val pendingIntent = PendingIntent.getBroadcast(
                context,
                requestCode.toInt(),
                intent,
                PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
            )
            val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as
AlarmManager
            alarmManager.cancel(pendingIntent)
        }

        fun removeAlarmForMileageRemind(context: Context) {
            val intent = Intent(context, MileageRemindReceiver::class.java)
            val requestCode: Int = 1
            val pendingIntent = PendingIntent.getBroadcast(
                context,
                requestCode,
                intent,
                PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
            )
            val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as
AlarmManager
            alarmManager.cancel(pendingIntent)
        }
    }
}
```

EventReceiver.kt

```
package com.a.autoreminderapp.alarms


import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.os.Build
import androidx.core.app.NotificationCompat
import com.a.autoreminderapp.R


class EventReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        val notification_id = System.currentTimeMillis().toInt()
        val eventName = intent!!.getStringExtra("eventName")
        val eventDescription = intent!!.getStringExtra("eventDescription")
        val eventDate = intent!!.getStringExtra("eventDate")
        val eventTime = intent!!.getStringExtra("eventTime")
        val notificationManager =
            context!!.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
        val mBuilder = NotificationCompat.Builder(context, "notify")
        mBuilder.setContentTitle("$eventName")
```

```
        mBuilder.setContentText("$eventDescription\n$eventDate $eventTime")
        mBuilder.setSmallIcon(R.drawable.ic_repair)
        mBuilder.priority = Notification.PRIORITY_HIGH
        mBuilder.build().flags = Notification.FLAG_NO_CLEAR or
Notification.PRIORITY_HIGH


        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channelId = "channel_id"
            val channel = NotificationChannel(
                channelId,
                "notifications_channel",
                NotificationManager.IMPORTANCE_HIGH
            )
            channel.enableVibration(true)
            notificationManager.createNotificationChannel(channel)
            mBuilder.setChannelId(channelId)
        }


        val notification = mBuilder.build()
        notificationManager.notify(notification_id, notification)
    }
}
```

MileageRemindReceiver.kt

```
package com.a.autoreminderapp.alarms
```

```
import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.os.Build
import androidx.core.app.NotificationCompat
import com.a.autoreminderapp.R

class MileageRemindReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        val notification_id = System.currentTimeMillis().toInt()
        val notificationManager =
            context!!.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
        val mBuilder = NotificationCompat.Builder(context, "notify")
        mBuilder.setContentTitle("Необходимо обновить сведения о пробеге!")
        mBuilder.setContentText("Необходимо обновить сведения о пробеге!")
        mBuilder.setSmallIcon(R.drawable.ic_repair)
        mBuilder.priority = Notification.PRIORITY_HIGH
        mBuilder.build().flags = Notification.FLAG_NO_CLEAR or
Notification.PRIORITY_HIGH

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channelId = "channel_id"
```

```
            val channel = NotificationChannel(
                channelId,
                "notifications_channel",
                NotificationManager.IMPORTANCE_HIGH
            )
            channel.enableVibration(true)
            notificationManager.createNotificationChannel(channel)
            mBuilder.setChannelId(channelId)
        }

        val notification = mBuilder.build()
        notificationManager.notify(notification_id, notification)
    }
}
```

AutoEventDao.kt

```
package com.a.autoreminderapp.data.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.a.autoreminderapp.data.entities.AutoEvent

@Dao
interface AutoEventDao {

    @Insert
```

```
    fun insertList(data: List<AutoEvent>)

    @Insert
    suspend fun insert(autoEvent: AutoEvent)

    @Update
    suspend fun update(autoEvent: AutoEvent)

    @Delete
    suspend fun delete(autoEvent: AutoEvent)

    @Query("Select * from autoEventsTable order by id ASC")
    fun getAll(): LiveData<List<AutoEvent>>

    @Query("Select * from autoEventsTable where date = :date order by id ASC")
    fun getAllByDate(date: String): LiveData<List<AutoEvent>>
}
```

CategoryDao.kt

```
package com.a.autoreminderapp.data.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.a.autoreminderapp.data.entities.Category

@Dao
```

```
interface CategoryDao {

    @Insert
    fun insertList(data: List<Category>)

    @Insert
    suspend fun insert(category: Category)

    @Update
    suspend fun update(category: Category)

    @Delete
    suspend fun delete(category: Category)

    @Query("Select * from categoriesTable order by id ASC")
    fun getAll(): LiveData<List<Category>>
}
```

PartDao.kt

```
package com.a.autoreminderapp.data.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.a.autoreminderapp.data.entities.Part

@Dao
```

```
interface PartDao {

    @Insert
    fun insertList(data: List<Part>)

    @Insert
    suspend fun insert(part: Part)

    @Update
    suspend fun update(part: Part)

    @Delete
    suspend fun delete(part: Part)

    @Query("Select * from partsTable order by id ASC")
    fun getAll(): LiveData<List<Part>>

    @Query("Select * from partsTable order by id ASC")
    fun getPartsList(): List<Part>

    @Query("Select * from partsTable where category = :categoryId order by id ASC")
    fun getAllByCategory(categoryId: Int): LiveData<List<Part>>
}
```

AutoEvent.kt

```
package com.a.autoreminderapp.data.entities
```

```
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "autoEventsTable")
class AutoEvent(
    @ColumnInfo(name = "name") var name: String,
    @ColumnInfo(name = "description") var description: String,
    @ColumnInfo(name = "date") var date: String,
    @ColumnInfo(name = "time") var time: String,
    ) {
    @PrimaryKey(autoGenerate = true)
    var id = 0
}
```

Category.kt

```
package com.a.autoreminderapp.data.entities

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "categoriesTable")
class Category(
    @ColumnInfo(name = "name") var name: String,
```

```
    ) {
    @PrimaryKey(autoGenerate = true)
    var id = 0
}
```

Part.kt

```
package com.a.autoreminderapp.data.entities


import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.ForeignKey.Companion.CASCADE
import androidx.room.PrimaryKey


@Entity(
    tableName = "partsTable",
    foreignKeys = [
        ForeignKey(
            onDelete = CASCADE,
            entity = Category::class,
            parentColumns = ["id"],
            childColumns = ["category"]
        )]
)
class Part(
    @ColumnInfo(name = "category") val category: Int,
```

```
    @ColumnInfo(name = "name") var name: String,
    @ColumnInfo(name = "maxServiceLife") var maxServiceLife: Int,
    @ColumnInfo(name = "currentServiceLife") var currentServiceLife: Int,
    @ColumnInfo(name = "brand") var brand: String,
    @ColumnInfo(name = "description") var description: String,
    ) {
    @PrimaryKey(autoGenerate = true)
    var id = 0
}
```

AutoEventRepository.kt

```
package com.a.autoreminderapp.data.repositories

import androidx.lifecycle.LiveData
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.data.dao.AutoEventDao

class AutoEventRepository(private val autoEventDao: AutoEventDao, val date: String) {
    val allAutoEvents: LiveData<List<AutoEvent>> = if (date == "") {
        autoEventDao.getAll()
    } else {
        autoEventDao.getAllByDate(date)
    }

    suspend fun insert(autoEvent: AutoEvent) {
        autoEventDao.insert(autoEvent)
```

```
    }

    suspend fun update(autoEvent: AutoEvent) {
        autoEventDao.update(autoEvent)
    }

    suspend fun delete(autoEvent: AutoEvent) {
        autoEventDao.delete(autoEvent)
    }
}
```

CategoryRepository.kt

```
package com.a.autoreminderapp.data.repositories

import androidx.lifecycle.LiveData
import com.a.autoreminderapp.data.entities.Category
import com.a.autoreminderapp.data.dao.CategoryDao

class CategoryRepository(private val categoriesDao: CategoryDao) {
    val allCategories: LiveData<List<Category>> = categoriesDao.getAll()

    suspend fun insert(category: Category){
        categoriesDao.insert(category)
    }

    suspend fun update(category: Category){
```

```
        categoriesDao.update(category)
    }

    suspend fun delete(category: Category){
        categoriesDao.delete(category)
    }
}
```

PartRepository.kt

```
package com.a.autoreminderapp.data.repositories

import android.content.Context
import androidx.lifecycle.LiveData
import com.a.autoreminderapp.data.entities.Part
import com.a.autoreminderapp.data.dao.PartDao
import com.a.autoreminderapp.notifications.NotificationHelper

class PartRepository(private val partDao: PartDao, val categoryId: Int) {
    val allParts: LiveData<List<Part>> = if (categoryId == -1){
        partDao.getAll()
    } else {
        partDao.getAllByCategory(categoryId)
    }

    suspend fun insert(part: Part){
        partDao.insert(part)
```

```
    }

    suspend fun update(part: Part){
        partDao.update(part)
    }

    suspend fun delete(part: Part){
        partDao.delete(part)
    }

    suspend fun updatePartsByMileage(previousMileage: Int, newMileage: Int, context: Context){
        for (part: Part in partDao.getPartsList()) {
            part.currentServiceLife = newMileage - previousMileage + part.currentServiceLife
            partDao.update(part)
            if (part.currentServiceLife > part.maxServiceLife){
                val text = "Превышен срок максимльной эксплуатации детали, немедленно замените"
                NotificationHelper.makeMileageNotification(context, part, text)
            }
            else if (part.currentServiceLife.toFloat() / part.maxServiceLife.toFloat() >= 0.9){
                val text = "Износ детали >= 90%, в скором времени потребуется замена"
                NotificationHelper.makeMileageNotification(context, part, text)
            }
            else if (part.currentServiceLife.toFloat() / part.maxServiceLife.toFloat() >= 0.8){
                val text = "Износ детали >= 80%, в скором времени потребуется замена"
```

```
                NotificationHelper.makeMileageNotification(context, part, text)
            }
            else if (part.currentServiceLife.toFloat() / part.maxServiceLife.toFloat() >= 0.75){
                val text = "Износ детали >= 75%, в скором времени потребуется замена"
                NotificationHelper.makeMileageNotification(context, part, text)
            }
            else if (part.currentServiceLife.toFloat() / part.maxServiceLife.toFloat() >= 0.6){
                val text = "Износ детали >= 60%, в скором времени потребуется замена"
                NotificationHelper.makeMileageNotification(context, part, text)
            }

        }
    }
}
```

AppDatabase.kt

```
package com.a.autoreminderapp.data

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.sqlite.db.SupportSQLiteDatabase
import com.a.autoreminderapp.data.dao.AutoEventDao
import com.a.autoreminderapp.data.dao.CategoryDao
import com.a.autoreminderapp.data.dao.PartDao
```

```
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.data.entities.Category
import com.a.autoreminderapp.data.entities.Part
import com.a.autoreminderapp.utils.ioThread

@Database(entities = [Category::class, Part::class, AutoEvent::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase(){

    abstract fun getAutoEventDao(): AutoEventDao
    abstract fun getCategoryDao(): CategoryDao
    abstract fun getPartDao(): PartDao

    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase =
            INSTANCE ?: synchronized(this) {
                INSTANCE ?: buildDatabase(context).also { INSTANCE = it }
            }

        private fun buildDatabase(context: Context) =
            Room.databaseBuilder(context.applicationContext,
                AppDatabase::class.java, "autoreminder_database")
                .addCallback(object : RoomDatabase.Callback() {
```

```
                override fun onCreate(db: SupportSQLiteDatabase) {
                    super.onCreate(db)
                    ioThread {
getDatabase(context).getCategoryDao().insertList(PREPOPULATE_CATEGORIES)
                    }
                }
            })
            .build()

        val PREPOPULATE_CATEGORIES = listOf(
            Category("Двигатель"),
            Category("Топливная система"),
            Category("Тормозная система"),
            Category("Подвеска"),
            Category("Салон"),
        )
    }
}
```

NotificationHelper.kt

```
package com.a.autoreminderapp.notifications

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
```

```
import android.os.Build
import androidx.core.app.NotificationCompat
import com.a.autoreminderapp.R
import com.a.autoreminderapp.data.entities.Part

class NotificationHelper {
    companion object {
        fun makeMileageNotification(context: Context, part: Part, text: String) {
            val notification_id = System.currentTimeMillis().toInt()

            val notificationManager =
                context!!.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
            val mBuilder = NotificationCompat.Builder(context, "notify")
            mBuilder.setContentTitle("${part.name}")
            mBuilder.setContentText(text)
            mBuilder.setSmallIcon(R.drawable.ic_repair)
            mBuilder.priority = Notification.PRIORITY_HIGH
            mBuilder.build().flags = Notification.FLAG_NO_CLEAR or
Notification.PRIORITY_HIGH

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                val channelId = "channel_id"
                val channel = NotificationChannel(
                    channelId,
                    "notifications_channel",
```

```
                NotificationManager.IMPORTANCE_HIGH
            )
            channel.enableVibration(true)
            notificationManager.createNotificationChannel(channel)
            mBuilder.setChannelId(channelId)
        }

        val notification = mBuilder.build()
        notificationManager.notify(notification_id, notification)
    }
  }
}
```

AddEditCategoryActivity.kt

```
package com.a.autoreminderapp.ui.activities

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.lifecycle.ViewModelProvider
import com.a.autoreminderapp.R
```

```
import com.a.autoreminderapp.data.entities.Category
import com.a.autoreminderapp.ui.viewmodels.CategoryViewModel

class AddEditCategoryActivity : AppCompatActivity() {

    lateinit var etCategoryName: EditText
    lateinit var btnSave: Button
    lateinit var btnRemove: Button

    lateinit var vmCategory: CategoryViewModel

    var categoryId = -1


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_add_edit_category)

        etCategoryName = findViewById(R.id.etCategoryName)
        btnSave = findViewById(R.id.btnSave)
        btnRemove = findViewById(R.id.btnRemove)

        vmCategory = ViewModelProvider(
            this,
            ViewModelProvider.AndroidViewModelFactory.getInstance(application)
```

```
        ).get(CategoryViewModel::class.java)


        val action = intent.getStringExtra("action")
        if (action.equals("edit")){
            title = "Изменение категории"
            val name = intent.getStringExtra("name")
            categoryId = intent.getIntExtra("id", -1)
            etCategoryName.setText(name)
            btnRemove.visibility = View.VISIBLE
            if (name != null){
                val category: Category = Category(name)
                category.id = categoryId
                btnRemove.setOnClickListener{
                    vmCategory.deleteCategory(category)
                    Toast.makeText(this, "Категория '$name' удалена",
Toast.LENGTH_LONG).show()
                    startActivity(Intent(applicationContext, MainActivity::class.java))
                    finish()
                }
            }
        } else {
            title = "Добавление категории"
            btnSave.text = "Сохранить"
        }
```

```
        btnSave.setOnClickListener{
            try {
                if (etCategoryName.text.toString() == ""){
                    Toast.makeText(this, "Не указано название категории!",
Toast.LENGTH_SHORT).show()
                    throw java.lang.IllegalArgumentException("Не указано название категории!")
                }
                val name = etCategoryName.text.toString()

                if (action.equals("edit")) {
                    val updatedCategory = Category(name)
                    updatedCategory.id = categoryId
                    vmCategory.updateCategory(updatedCategory)
                    Toast.makeText(this, "Категория обновлена", Toast.LENGTH_LONG).show()
                } else {
                    vmCategory.addCategory(Category(name))
                    Toast.makeText(this, "Категория $name добавлена",
Toast.LENGTH_LONG).show()
                }
                startActivity(Intent(applicationContext, MainActivity::class.java))
                finish()
            } catch (e: java.lang.Exception){
                Log.d("Exception", e.message.toString())
            }
        }
```

```
    }
}
```

AddEditEventActivity.kt

```
package com.a.autoreminderapp.ui.activities

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.TimePicker
import android.widget.Toast
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import com.a.autoreminderapp.R
import com.a.autoreminderapp.alarms.AlarmHelper
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.ui.viewmodels.AutoEventViewModel
import com.a.autoreminderapp.ui.viewmodels.AutoEventViewModelFactory
import java.text.SimpleDateFormat
import java.util.*

class AddEditEventActivity : AppCompatActivity() {
```

```
lateinit var etEventName: EditText
lateinit var etDescription: EditText
lateinit var tpTime: TimePicker
lateinit var btnSave: Button

var autoEventId = -1

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_add_edit_event)

    etEventName = findViewById(R.id.etEventName)
    etDescription = findViewById(R.id.etDescription)
    tpTime = findViewById(R.id.tpTime)
    tpTime.setIs24HourView(true)
    btnSave = findViewById(R.id.btnSave)

    val dateStr = intent.getStringExtra("date")
    if (dateStr != null){
        val vmAutoEvent: AutoEventViewModel by viewModels {
            AutoEventViewModelFactory(application, dateStr)
        }
        val action = intent.getStringExtra("action")
        if (action.equals("edit")){
            title = "Изменение события"
```

```
            autoEventId = intent.getIntExtra("id", -1)
            val name = intent.getStringExtra("name")
            val timeStr = intent.getStringExtra("time")
            val description = intent.getStringExtra("description")
            etEventName.setText(name)
            etDescription.setText(description)

            val sdf = SimpleDateFormat("hh:mm", Locale.getDefault())
            val calendar = Calendar.getInstance()
            calendar.time = sdf.parse(timeStr)
            tpTime.hour = calendar.get(Calendar.HOUR_OF_DAY)
            tpTime.minute = calendar.get(Calendar.MINUTE)
        } else {
            title = "Добавление события"
            btnSave.text = "Сохранить"
        }


        btnSave.setOnClickListener{
            try {
                if (etEventName.text.toString() == ""){
                    throw java.lang.IllegalArgumentException("Необходимо указать название
события!")
                }
                if (etDescription.text.toString() == ""){
```

```
            throw java.lang.IllegalArgumentException("Необходимо указать краткое описание события!")
        }
        val name = etEventName.text.toString()
        val description = etDescription.text.toString()
        val hourStr = if (tpTime.hour.toString().length == 1){
            "0" + tpTime.hour.toString()
        } else {
            tpTime.hour.toString()
        }
        val minuteStr = if (tpTime.minute.toString().length == 1){
            "0" + tpTime.minute.toString()
        } else {
            tpTime.minute.toString()
        }
        val timeStr = "$hourStr:$minuteStr"

        if (action.equals("edit")) {
            val updatedAutoEvent = AutoEvent(name, description, dateStr, timeStr)
            updatedAutoEvent.id = autoEventId
            vmAutoEvent.updateAutoEvent(updatedAutoEvent)
            Toast.makeText(this, "Событие '$name' обновлено", Toast.LENGTH_LONG).show()
            AlarmHelper.makeAlarmsForEvent(this, updatedAutoEvent)
        } else {
            val autoEvent: AutoEvent = AutoEvent(name, description, dateStr, timeStr)
```

```
                vmAutoEvent.addAutoEvent(autoEvent)
                Toast.makeText(this, "Событие '$name' добавлено",
Toast.LENGTH_LONG).show()
                AlarmHelper.makeAlarmsForEvent(this, autoEvent)
            }

            val intent = Intent(this@AddEditEventActivity, DateEventsActivity::class.java)
            intent.putExtra("dateTime", dateStr)
            startActivity(intent)
            finish()
        } catch (e: java.lang.Exception){
            Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }
  }
}
```

AddEditPartActivity.kt

```
package com.a.autoreminderapp.ui.activities

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
```

```
import android.widget.TextView
import android.widget.Toast
import androidx.activity.viewModels
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.a.autoreminderapp.R
import com.a.autoreminderapp.data.entities.Category
import com.a.autoreminderapp.data.entities.Part
import com.a.autoreminderapp.ui.adapters.*
import com.a.autoreminderapp.ui.viewmodels.CategoryViewModel
import com.a.autoreminderapp.ui.viewmodels.PartViewModelFactory
import com.a.autoreminderapp.ui.viewmodels.PartViewModel

class AddEditPartActivity : AppCompatActivity(), CategoryClickInterface,
CategoryLongClickInterface {

    lateinit var tvCategoryName: TextView
    lateinit var rvCategories: RecyclerView
    lateinit var etPartName: EditText
    lateinit var etMaxServiceLife: EditText
    lateinit var etCurrentServiceLife: EditText
    lateinit var etBrand: EditText
    lateinit var etDescription: EditText
```

```
lateinit var btnSave: Button

lateinit var vmCategory: CategoryViewModel
lateinit var vmPart: PartViewModel

var categoryId = -1
var partId = -1

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_add_edit_part)

    tvCategoryName = findViewById(R.id.tvCategoryName)
    rvCategories = findViewById(R.id.rvCategories)
    etPartName = findViewById(R.id.etPartName)
    etMaxServiceLife = findViewById(R.id.etMaxServiceLife)
    etCurrentServiceLife = findViewById(R.id.etCurrentServiceLife)
    etBrand = findViewById(R.id.etBrand)
    etDescription = findViewById(R.id.etDescription)
    btnSave = findViewById(R.id.btnSave)
    rvCategories = findViewById(R.id.rvCategories)

    rvCategories.layoutManager = LinearLayoutManager(this)
    val categoryRVAdapter = CategoryRVAdapter(this, this, this)
    rvCategories.adapter = categoryRVAdapter
```

```
vmCategory = ViewModelProvider(
    this,
    ViewModelProvider.AndroidViewModelFactory.getInstance(application)
).get(CategoryViewModel::class.java)

vmCategory.allCategories.observe(this, Observer { list ->
    list?.let {
        if (categoryRVAdapter != null){
            categoryRVAdapter.updateList(it)
        }
    }
})

val vmPart: PartViewModel by viewModels {
    PartViewModelFactory(application, categoryId)
}

val action = intent.getStringExtra("action")
if (action.equals("edit")){
    title = "Изменение детали"
    partId = intent.getIntExtra("id", 0)
    categoryId = intent.getIntExtra("category", 0)
    val categoryName = intent.getStringExtra("categoryName")
    val name = intent.getStringExtra("name")
```

```
    val maxServiceLife = intent.getIntExtra("maxServiceLife", 0)
    val currentServiceLife = intent.getIntExtra("currentServiceLife", 0)
    val brand = intent.getStringExtra("brand")
    val description = intent.getStringExtra("description")
    tvCategoryName.text = categoryName
    etPartName.setText(name)
    etMaxServiceLife.setText(maxServiceLife.toString())
    etCurrentServiceLife.setText(currentServiceLife.toString())
    etBrand.setText(brand)
    etDescription.setText(description)
} else {
    title = "Добавление детали"
    categoryId = intent.getIntExtra("category", 0)
    val categoryName = intent.getStringExtra("categoryName")
    tvCategoryName.text = categoryName
    btnSave.text = "Сохранить"
}

btnSave.setOnClickListener{
    try {
        if (categoryId == -1){
            throw java.lang.IllegalArgumentException("Необходимо выбрать категорию!")
        }
        if (etPartName.text.toString() == ""){
            throw java.lang.IllegalArgumentException("Не указано название детали!")
```

```
            }
            if (etMaxServiceLife.text.toString() == ""){
                throw java.lang.IllegalArgumentException("Не указан срок максимальной
эксплуатации!")
            }
            if (etCurrentServiceLife.text.toString() == ""){
                throw java.lang.IllegalArgumentException("Не указан срок текущей
эксплуатации!")
            }
            if (etBrand.text.toString() == ""){
                throw java.lang.IllegalArgumentException("Не указана марка!")
            }
            if (etDescription.text.toString() == ""){
                throw java.lang.IllegalArgumentException("Не введено краткое описание!")
            }

            val name = etPartName.text.toString()
            val maxServiceLife = etMaxServiceLife.text.toString().toInt()
            val currentServiceLife = etCurrentServiceLife.text.toString().toInt()
            val brand = etBrand.text.toString()
            val description = etDescription.text.toString()

            if (action.equals("edit")) {
                val updatedPart = Part(categoryId, name, maxServiceLife, currentServiceLife,
brand, description)
                updatedPart.id = partId
```

```
                vmPart.updatePart(updatedPart)
                Toast.makeText(this, "Деталь '$name' обновлена",
Toast.LENGTH_LONG).show()
            } else {
                vmPart.addPart(Part(categoryId, name, maxServiceLife, currentServiceLife, brand,
description))
                Toast.makeText(this, "Деталь '$name' добавлена",
Toast.LENGTH_LONG).show()
            }
            val intent = Intent(this@AddEditPartActivity, CategoryPartsActivity::class.java)
            intent.putExtra("id", categoryId)
            intent.putExtra("name", tvCategoryName.text.toString())
            startActivity(intent)
            finish()
        } catch (e: java.lang.Exception){
            Toast.makeText(this, e.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }
}

override fun onClick(category: Category) {
    categoryId = category.id
    tvCategoryName.text = category.name
}

override fun onLongClick(category: Category) {
```

```
        categoryId = category.id
        tvCategoryName.text = category.name
    }
}
```

CategoryPartsActivity.kt

```
package com.a.autoreminderapp.ui.activities

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import androidx.activity.viewModels
import androidx.lifecycle.Observer
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.a.autoreminderapp.R
import com.a.autoreminderapp.data.entities.Part
import com.a.autoreminderapp.ui.adapters.PartClickInterface
import com.a.autoreminderapp.ui.adapters.PartLongClickInterface
import com.a.autoreminderapp.ui.adapters.PartRVAdapter
import com.a.autoreminderapp.ui.viewmodels.PartViewModelFactory
import com.a.autoreminderapp.ui.viewmodels.PartViewModel
import com.google.android.material.floatingactionbutton.FloatingActionButton
```

```
class CategoryPartsActivity : AppCompatActivity(), PartClickInterface, PartLongClickInterface
{

    lateinit var rvParts: RecyclerView
    lateinit var fabAdd: FloatingActionButton

    var categoryId = -1

    val vmPart: PartViewModel by viewModels {
        PartViewModelFactory(application, categoryId)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_category_parts)


        val categoryName = intent.getStringExtra("name")

        title = categoryName

        categoryId = intent.getIntExtra("id", -1)

        rvParts = findViewById(R.id.rvParts)
        fabAdd = findViewById(R.id.fabAddPart)
```

```
    rvParts.layoutManager = LinearLayoutManager(application)
    val partRVAdapter = PartRVAdapter(application, this, this)

    rvParts.adapter = partRVAdapter

    vmPart.allParts.observe(this, Observer { list ->
        list?.let {
            partRVAdapter.updateList(it)
        }
    } )

    fabAdd.setOnClickListener{
        val intent = Intent(this@CategoryPartsActivity, AddEditPartActivity::class.java)
        intent.putExtra("category", categoryId)
        intent.putExtra("categoryName", title)
        startActivity(intent)
        finish()
    }
}

override fun onClick(part: Part) {
    val intent = Intent(this@CategoryPartsActivity, AddEditPartActivity::class.java)
    intent.putExtra("action", "edit")
    intent.putExtra("id", part.id)
```

```
            intent.putExtra("category", categoryId)
            intent.putExtra("categoryName", title)
            intent.putExtra("name", part.name)
            intent.putExtra("maxServiceLife", part.maxServiceLife)
            intent.putExtra("currentServiceLife", part.currentServiceLife)
            intent.putExtra("brand", part.brand)
            intent.putExtra("description", part.description)
            startActivity(intent)
        }

        override fun onLongClick(part: Part) {
            vmPart.deletePart(part)
            Toast.makeText(this, "Деталь '${part.name}' удалена", Toast.LENGTH_LONG).show()
        }
    }
```

DateEventsActivity.kt

```
package com.a.autoreminderapp.ui.activities

import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.ViewModelProvider
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
```

```
import com.a.autoreminderapp.R
import com.a.autoreminderapp.alarms.AlarmHelper
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.ui.adapters.AutoEventClickInterface
import com.a.autoreminderapp.ui.adapters.AutoEventLongClickInterface
import com.a.autoreminderapp.ui.adapters.AutoEventRVAdapter
import com.a.autoreminderapp.ui.viewmodels.AutoEventViewModel
import com.a.autoreminderapp.ui.viewmodels.AutoEventViewModelFactory
import com.google.android.material.floatingactionbutton.FloatingActionButton
import java.text.SimpleDateFormat
import java.util.*

class DateEventsActivity : AppCompatActivity(), AutoEventClickInterface,
    AutoEventLongClickInterface {

    lateinit var fabAdd: FloatingActionButton
    lateinit var rvAutoEvents: RecyclerView
    lateinit var vmAutoEvent: AutoEventViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_date_events)

        fabAdd = findViewById(R.id.fabAddAutoEvent)
        rvAutoEvents = findViewById(R.id.rvEvents)
```

```
val sdfSource = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
val sdfTitle = SimpleDateFormat("dd.MM.yyyy", Locale.getDefault())
var dateTimeStr = intent.getStringExtra("dateTime")
val dateTime = sdfSource.parse(dateTimeStr)

title = "События на " + sdfTitle.format(dateTime)



val autoEventRVAdapter = AutoEventRVAdapter(this, this, this)
rvAutoEvents.layoutManager = LinearLayoutManager(this)
rvAutoEvents.adapter = autoEventRVAdapter

if (dateTimeStr == null){
    dateTimeStr = "";
}
vmAutoEvent = ViewModelProvider(
    this,
    AutoEventViewModelFactory(application, dateTimeStr)
).get(AutoEventViewModel::class.java)

vmAutoEvent.allAutoEvents.observe(this, androidx.lifecycle.Observer { list ->
    list?.let {
        if (autoEventRVAdapter != null) {
            autoEventRVAdapter.updateList(it)
```

```
            }
        }
    })

    fabAdd.setOnClickListener{
        val intent = Intent(this@DateEventsActivity, AddEditEventActivity::class.java)
        intent.putExtra("date", dateTimeStr)
        startActivity(intent)
    }
}

override fun onClick(autoEvent: AutoEvent) {
    val intent = Intent(this@DateEventsActivity, AddEditEventActivity::class.java)
    intent.putExtra("action", "edit")
    intent.putExtra("id", autoEvent.id)
    intent.putExtra("name", autoEvent.name)
    intent.putExtra("date", autoEvent.date)
    intent.putExtra("time", autoEvent.time)
    intent.putExtra("description", autoEvent.description)
    startActivity(intent)

}

override fun onLongClick(autoEvent: AutoEvent) {
    vmAutoEvent.deleteAutoEvent(autoEvent)
```

```
        AlarmHelper.removeAlarmsForEvent(this, autoEvent)
        Toast.makeText(this, "Событие '${autoEvent.name}' удалено",
Toast.LENGTH_LONG).show()
    }
}
```

MainActivity.kt

```
package com.a.autoreminderapp.ui.activities

import android.Manifest
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.fragment.app.Fragment
import com.a.autoreminderapp.R
import com.a.autoreminderapp.alarms.AlarmHelper
import com.a.autoreminderapp.ui.fragments.CategoriesFragment
import com.a.autoreminderapp.ui.fragments.EventsFragment
import com.a.autoreminderapp.ui.fragments.SettingsFragment
import com.google.android.material.bottomnavigation.BottomNavigationView

class MainActivity : AppCompatActivity() {

    lateinit var bottomNav : BottomNavigationView
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        if (ActivityCompat.checkSelfPermission(
                this,
                Manifest.permission.POST_NOTIFICATIONS
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            requestPermissions(arrayOf(Manifest.permission.POST_NOTIFICATIONS), 1)
        }
    }

    loadFragment(CategoriesFragment())
    bottomNav = findViewById<BottomNavigationView>(R.id.bottomNav)
    bottomNav.setOnItemSelectedListener {
        when (it.itemId) {
            R.id.bmCategories -> {
                loadFragment(CategoriesFragment())
                title = "Категории"
                true
            }
            R.id.bmCalendar -> {
```

```
                loadFragment(EventsFragment())
                title = "События"
                true
            }
            R.id.bmSettings -> {
                title = "Настройки"
                loadFragment(SettingsFragment())
                true
            }
            else -> throw AssertionError()
        }
    }
}

private fun loadFragment(fragment: Fragment){
    val transaction = supportFragmentManager.beginTransaction()
    transaction.replace(R.id.container,fragment)
    transaction.commit()
}
}
```

AutoEventRVAdapter.kt

```
package com.a.autoreminderapp.ui.adapters

import android.content.Context
import android.view.LayoutInflater
```

```
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.a.autoreminderapp.R
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.data.entities.Category

class AutoEventRVAdapter(
    val context: Context,
    val autoEventClickInterface: AutoEventClickInterface,
    val autoEventLongClickInterface: AutoEventLongClickInterface
) : RecyclerView.Adapter<AutoEventRVAdapter.ViewHolder>() {
    private val allAutoEvents = ArrayList<AutoEvent>()
    inner class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val tvAutoEventName = itemView.findViewById<TextView>(R.id.tvAutoEventName)
        val tvAutoEventDescription =
itemView.findViewById<TextView>(R.id.tvAutoEventDescription)
        val tvDateTime = itemView.findViewById<TextView>(R.id.tvDateTime)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(
                R.layout.auto_event_rv_item, parent, false
        )
```

```
    return ViewHolder(itemView)
}

override fun getItemCount(): Int {
    return allAutoEvents.size
}

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val autoEvent = allAutoEvents[position]

    holder.tvAutoEventName.text = autoEvent.name
    holder.tvAutoEventDescription.text = autoEvent.description
    holder.tvDateTime.text = autoEvent.date + " " + autoEvent.time

    holder.itemView.setOnClickListener{
        autoEventClickInterface.onClick(autoEvent)
    }

    holder.itemView.setOnLongClickListener {
        autoEventLongClickInterface.onLongClick(autoEvent)
        true
    }
}

fun updateList(newList: List<AutoEvent>){
```

```
        allAutoEvents.clear()
        allAutoEvents.addAll(newList)
        notifyDataSetChanged()
    }
}

interface AutoEventClickInterface {
    // функция обработки нажатия
    fun onClick(autoEvent: AutoEvent)
}

interface AutoEventLongClickInterface {
    fun onLongClick(autoEvent: AutoEvent)
}
```

CategoryRVAdapter.kt

```
package com.a.autoreminderapp.ui.adapters

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.a.autoreminderapp.R
```

```
import com.a.autoreminderapp.data.entities.Category

class CategoryRVAdapter(
    val context: Context,
    val categoryClickInterface: CategoryClickInterface,
    val categoryLongClickInterface: CategoryLongClickInterface
) : RecyclerView.Adapter<CategoryRVAdapter.ViewHolder>() {
    private val allCategories = ArrayList<Category>()

    inner class ViewHolder(itemView: View): RecyclerView.ViewHolder(itemView) {
        val tvCategoryName = itemView.findViewById<TextView>(R.id.tvCategoryName)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(
            R.layout.category_rv_item, parent, false
        )
        return ViewHolder(itemView)
    }

    override fun getItemCount(): Int {
        return allCategories.size
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
```

```
        val category = allCategories[position]

        holder.tvCategoryName.text = category.name
        holder.itemView.setOnClickListener{
            categoryClickInterface.onClick(category)
        }

        holder.itemView.setOnLongClickListener {
            categoryLongClickInterface.onLongClick(category)
            true
        }
    }

    fun updateList(newList: List<Category>){
        allCategories.clear()
        allCategories.addAll(newList)
        notifyDataSetChanged()
    }
}

interface CategoryClickInterface {
    fun onClick(category: Category)
}

interface CategoryLongClickInterface {
```

```
    fun onLongClick(category: Category)
}
```

PartRVAdapter.kt

```
package com.a.autoreminderapp.ui.adapters

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.a.autoreminderapp.R
import com.a.autoreminderapp.data.entities.Part

class PartRVAdapter(
    val context: Context,
    val partClickInterface: PartClickInterface,
    val partLongClickInterface: PartLongClickInterface
) : RecyclerView.Adapter<PartRVAdapter.ViewHolder>() {

    private val allParts = ArrayList<Part>()

    inner class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val tvName = itemView.findViewById<TextView>(R.id.tvName)
        val tvMaxServiceLife = itemView.findViewById<TextView>(R.id.tvMaxServiceLife)
```

```
        val tvCurrentServiceLife =
itemView.findViewById<TextView>(R.id.tvCurrentServiceLife)
        val tvBrand = itemView.findViewById<TextView>(R.id.tvBrand)
        val tvDescription = itemView.findViewById<TextView>(R.id.tvDescription)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(
            R.layout.part_rv_item, parent, false
        )
        return ViewHolder(itemView)
    }

    override fun getItemCount(): Int {
        return allParts.size
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val part = allParts[position]

        holder.tvName.text = part.name
        holder.tvMaxServiceLife.text = "Срок максимальной эксплуатации: " +
part.maxServiceLife.toString() + "км."
        holder.tvCurrentServiceLife.text = "Срок текущей эксплуатации: " +
part.currentServiceLife.toString()  + "км."
        holder.tvBrand.text = part.brand
```

```
        holder.tvDescription.text = part.description

        holder.itemView.setOnClickListener{
            partClickInterface.onClick(part)
        }

        holder.itemView.setOnLongClickListener {
            partLongClickInterface.onLongClick(part)
            true
        }
    }

    fun updateList(newList: List<Part>){
        allParts.clear()
        allParts.addAll(newList)
        notifyDataSetChanged()
    }
}

interface PartClickInterface {
    // функция обработки нажатия
    fun onClick(part: Part)
}

interface PartLongClickInterface {
```

```
    fun onLongClick(part: Part)
}
```

CategoriesFragment.kt

```
package com.a.autoreminderapp.ui.fragments


import android.content.Intent
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.a.autoreminderapp.ui.activities.AddEditCategoryActivity
import com.a.autoreminderapp.ui.activities.CategoryPartsActivity
import com.a.autoreminderapp.R
import com.a.autoreminderapp.data.entities.Category
import com.a.autoreminderapp.ui.adapters.CategoryClickInterface
import com.a.autoreminderapp.ui.adapters.CategoryLongClickInterface
import com.a.autoreminderapp.ui.adapters.CategoryRVAdapter
import com.a.autoreminderapp.ui.viewmodels.CategoryViewModel
import com.google.android.material.floatingactionbutton.FloatingActionButton
```

```
class CategoriesFragment : Fragment(), CategoryClickInterface, CategoryLongClickInterface {

    lateinit var vmCategory: CategoryViewModel
    lateinit var rvCategories: RecyclerView
    lateinit var fabAdd: FloatingActionButton

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_categories, container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        rvCategories = view.findViewById(R.id.rvCategories)
        fabAdd = view.findViewById(R.id.fabAddCategory)
```

```
        rvCategories.layoutManager = LinearLayoutManager(context)
        val categoryRVAdapter = context?.let { CategoryRVAdapter(it, this, this) }
        rvCategories.adapter = categoryRVAdapter

        vmCategory = ViewModelProvider(
            this,
ViewModelProvider.AndroidViewModelFactory.getInstance(requireActivity().application)
        ).get(CategoryViewModel::class.java)

        vmCategory.allCategories.observe(viewLifecycleOwner, Observer { list ->
            list?.let {
                if (categoryRVAdapter != null){
                    categoryRVAdapter.updateList(it)
                }
            }
        })

        fabAdd.setOnClickListener{
            val intent = Intent(context, AddEditCategoryActivity::class.java)
            startActivity(intent)
        }
    }

    override fun onClick(category: Category) {
```

```
        val intent = Intent(context, CategoryPartsActivity::class.java)
        intent.putExtra("id", category.id)
        intent.putExtra("name", category.name)
        startActivity(intent)
    }

    override fun onLongClick(category: Category) {
        val intent = Intent(context, AddEditCategoryActivity::class.java)
        intent.putExtra("action", "edit")
        intent.putExtra("id", category.id)
        intent.putExtra("name", category.name)
        startActivity(intent)
    }
}
```

EventsFragment.kt

```
package com.a.autoreminderapp.ui.fragments

import android.content.Intent
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
```

```
import com.a.autoreminderapp.ui.activities.DateEventsActivity
import com.a.autoreminderapp.R
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.ui.viewmodels.AutoEventViewModel
import com.a.autoreminderapp.ui.viewmodels.AutoEventViewModelFactory
import com.a.autoreminderapp.ui.viewmodels.CategoryViewModel
import com.applandeo.materialcalendarview.CalendarView
import com.applandeo.materialcalendarview.EventDay
import com.applandeo.materialcalendarview.listeners.OnDayClickListener
import java.text.SimpleDateFormat
import java.util.*

class EventsFragment : Fragment() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_events, container, false)
    }
```

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    val calendarView = view.findViewById<CalendarView>(R.id.calendarView)

    val vmAutoEvent: AutoEventViewModel by viewModels {
        AutoEventViewModelFactory(requireActivity().application, "")
    }

    val eventDayList: MutableList<EventDay> = mutableListOf<EventDay>()

    val sdf = SimpleDateFormat("yyyy-MM-dd hh:mm", Locale.getDefault())


    vmAutoEvent.allAutoEvents.observe(viewLifecycleOwner, Observer { list ->
        for (autoEvent: AutoEvent in list) {
            val calendar = Calendar.getInstance()
            calendar.time = sdf.parse(autoEvent.date + " " + autoEvent.time)
            eventDayList += EventDay(calendar, R.drawable.ic_repair)
        }
        calendarView.setEvents(eventDayList)
    })

    calendarView.setOnDayClickListener(object : OnDayClickListener {
```

```
            override fun onDayClick(eventDay: EventDay) {
                val intent = Intent(context, DateEventsActivity::class.java)
                val sdf = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
                intent.putExtra("dateTime", sdf.format(eventDay.calendar.time))
                startActivity(intent)
            }
        })
    }
}
```

SettingsFragment.kt

```
package com.a.autoreminderapp.ui.fragments

import android.os.Bundle
import android.text.InputType
import android.widget.Toast
import androidx.fragment.app.viewModels
import androidx.preference.EditTextPreference
import androidx.preference.Preference
import androidx.preference.PreferenceFragmentCompat
import com.a.autoreminderapp.R
import com.a.autoreminderapp.alarms.AlarmHelper
import com.a.autoreminderapp.ui.viewmodels.PartViewModel
import com.a.autoreminderapp.ui.viewmodels.PartViewModelFactory

class SettingsFragment : PreferenceFragmentCompat() {
```

```
    val vmPart: PartViewModel by viewModels {
        PartViewModelFactory(requireActivity().application, -1)
    }

    override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
        setPreferencesFromResource(R.xml.root_preferences, rootKey)

        val etPreferInterval : EditTextPreference? = findPreference("interval")
        val etPreferMileage: EditTextPreference? = findPreference("mileage")

        etPreferInterval?.setOnBindEditTextListener { editText ->
            editText.inputType = InputType.TYPE_CLASS_NUMBER
        }

        etPreferMileage?.setOnBindEditTextListener { editText ->
            editText.inputType = InputType.TYPE_CLASS_NUMBER
        }

        if (etPreferInterval != null) {
            etPreferInterval.onPreferenceChangeListener = Preference.OnPreferenceChangeListener
{ preference, newValue ->
                val previousMileageStr = etPreferMileage?.text.toString()
                val previousMileage: Int = if (previousMileageStr != ""){
                    previousMileageStr.toInt()
```

```
            } else {
                0
            }
            if (newValue.toString() != ""){
                val interval = newValue.toString().toInt()
                if (interval > 0){
                    AlarmHelper.makeAlarmForMileageRemind(requireContext(), interval)
                    Toast.makeText(requireContext(), "Интервал обновления пробега изменен!",
Toast.LENGTH_SHORT).show()
                }
                else if (interval == 0){
                    AlarmHelper.removeAlarmForMileageRemind(requireContext())
                    Toast.makeText(requireContext(), "Напоминания о обновлении пробега
отключены!", Toast.LENGTH_SHORT).show()
                }
            }
            true
        }
    }

    if (etPreferMileage != null){
        etPreferMileage.onPreferenceChangeListener =
Preference.OnPreferenceChangeListener{ preference, newValue ->
            val previousMileage: Int = if (etPreferMileage.text.toString() != ""){
                etPreferMileage.text.toString().toInt()
            } else {
```

```
                0
            }
            val newMileage: Int = if (newValue.toString() != ""){
                newValue.toString().toInt()
            } else {
                0
            }
            vmPart.updatePartsByMileage(previousMileage, newMileage)
            Toast.makeText(requireContext(), "Пробег изменен!",
Toast.LENGTH_SHORT).show()
            true
        }
    }
  }
}
```

AutoEventViewModel.kt

```
package com.a.autoreminderapp.ui.viewmodels


import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import com.a.autoreminderapp.data.AppDatabase
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.data.repositories.AutoEventRepository
import kotlinx.coroutines.Dispatchers
```

```
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.launch

class AutoEventViewModel(application: Application, val date: String) :
AndroidViewModel(application) {
    val allAutoEvents: LiveData<List<AutoEvent>>
    val repository: AutoEventRepository

    init {
        val dao = AppDatabase.getDatabase(application).getAutoEventDao()
        repository = AutoEventRepository(dao, date)
        allAutoEvents = repository.allAutoEvents
    }

    fun addAutoEvent(autoEvent: AutoEvent) = viewModelScope.launch(Dispatchers.IO) {
        repository.insert(autoEvent)
    }

    fun updateAutoEvent(autoEvent: AutoEvent) = viewModelScope.launch(Dispatchers.IO) {
        repository.update(autoEvent)
    }

    fun deleteAutoEvent(autoEvent: AutoEvent) = viewModelScope.launch(Dispatchers.IO) {
        repository.delete(autoEvent)
    }
```

```
}
```

AutoEventViewModelFactory.kt

```
package com.a.autoreminderapp.ui.viewmodels


import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import com.a.autoreminderapp.data.AppDatabase
import com.a.autoreminderapp.data.entities.AutoEvent
import com.a.autoreminderapp.data.repositories.AutoEventRepository
import kotlinx.coroutines.Dispatchers
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.launch


class AutoEventViewModel(application: Application, val date: String) :
AndroidViewModel(application) {
    val allAutoEvents: LiveData<List<AutoEvent>>
    val repository: AutoEventRepository


    init {
        val dao = AppDatabase.getDatabase(application).getAutoEventDao()
        repository = AutoEventRepository(dao, date)
        allAutoEvents = repository.allAutoEvents
    }
```

```
    fun addAutoEvent(autoEvent: AutoEvent) = viewModelScope.launch(Dispatchers.IO) {
        repository.insert(autoEvent)
    }

    fun updateAutoEvent(autoEvent: AutoEvent) = viewModelScope.launch(Dispatchers.IO) {
        repository.update(autoEvent)
    }

    fun deleteAutoEvent(autoEvent: AutoEvent) = viewModelScope.launch(Dispatchers.IO) {
        repository.delete(autoEvent)
    }
}
```

CategoryViewModel.kt

```
package com.a.autoreminderapp.ui.viewmodels

import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.viewModelScope
import com.a.autoreminderapp.data.AppDatabase
import com.a.autoreminderapp.data.entities.Category
import com.a.autoreminderapp.data.repositories.CategoryRepository
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
```

```
class CategoryViewModel(application: Application) : AndroidViewModel(application) {
    val allCategories: LiveData<List<Category>>
    val repository: CategoryRepository

    init {
        val dao = AppDatabase.getDatabase(application).getCategoryDao()
        repository = CategoryRepository(dao)
        allCategories = repository.allCategories
    }

    fun addCategory(category: Category) = viewModelScope.launch(Dispatchers.IO) {
        repository.insert(category)
    }

    fun updateCategory(category: Category) = viewModelScope.launch(Dispatchers.IO) {
        repository.update(category)
    }

    fun deleteCategory(category: Category) = viewModelScope.launch(Dispatchers.IO) {
        repository.delete(category)
    }
}
```

PartViewModel.kt

```
package com.a.autoreminderapp.ui.viewmodels
```

```
import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.viewModelScope
import com.a.autoreminderapp.data.AppDatabase
import com.a.autoreminderapp.data.entities.Part
import com.a.autoreminderapp.data.repositories.PartRepository
import com.a.autoreminderapp.notifications.NotificationHelper
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class PartViewModel(application: Application, val categoryId: Int) :
    AndroidViewModel(application) {
    val allParts: LiveData<List<Part>>
    val repository: PartRepository

    init {
        val dao = AppDatabase.getDatabase(application).getPartDao()
        repository = PartRepository(dao, categoryId)
        allParts = repository.allParts
    }

    fun addPart(part: Part) = viewModelScope.launch(Dispatchers.IO) {
        repository.insert(part)
    }
```

```
    fun updatePart(part: Part) = viewModelScope.launch(Dispatchers.IO) {
        repository.update(part)
    }

    fun deletePart(part: Part) = viewModelScope.launch(Dispatchers.IO) {
        repository.delete(part)
    }

    fun updatePartsByMileage(previousMileage: Int, newMileage: Int) =
        viewModelScope.launch(Dispatchers.IO) {
            repository.updatePartsByMileage(previousMileage, newMileage, getApplication())
        }
}
```

PartViewModelFactory.kt

```
package com.a.autoreminderapp.ui.viewmodels

import android.app.Application
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider

class PartViewModelFactory(
    private val application: Application,
    private val categoryId: Int
): ViewModelProvider.NewInstanceFactory() {
```

```
    override fun <T : ViewModel> create(modelClass: Class<T>): T =
        PartViewModel(application, categoryId) as T
}
```

Executors.kt

```
package com.a.autoreminderapp.utils

import java.util.concurrent.Executors

private val IO_EXECUTOR = Executors.newSingleThreadExecutor()

fun ioThread(f : () -> Unit) {
    IO_EXECUTOR.execute(f)
}
```

ic_add.xml

```
<vector android:height="24dp" android:tint="#FFFFFF"
    android:viewportHeight="24" android:viewportWidth="24"
    android:width="24dp" xmlns:android="http://schemas.android.com/apk/res/android">
    <path android:fillColor="@android:color/white" android:pathData="M19,13h-6v6h-2v-6H5v-2h6V5h2v6h6v2z"/>
</vector>
```

ic_calendar.xml

```
<vector android:height="24dp" android:tint="#FFFFFF"
    android:viewportHeight="24" android:viewportWidth="24"
    android:width="24dp" xmlns:android="http://schemas.android.com/apk/res/android">
    <path android:fillColor="@android:color/white" android:pathData="M19,4h-1V2h-2v2H8V2H6v2H5C3.89,4 3.01,4.9 3.01,6L3,20c0,1.1 0.89,2 2,2h14c1.1,0 2,-0.9 2,-2V6C21,4.9
```

```
20.1,4 19,4zM19,20H5V10h14V20zM9,14H7v-2h2V14zM13,14h-2v-2h2V14zM17,14h-2v-2h2V14zM9,18H7v-2h2V18zM13,18h-2v-2h2V18zM17,18h-2v-2h2V18z"/>
</vector>
```

ic_categories.xml

```
<vector android:height="24dp" android:tint="#FFFFFF"
    android:viewportHeight="24" android:viewportWidth="24"
    android:width="24dp" xmlns:android="http://schemas.android.com/apk/res/android">
    <path android:fillColor="@android:color/white" android:pathData="M12,2l-5.5,9h11z"/>
    <path android:fillColor="@android:color/white" android:pathData="M17.5,17.5m-4.5,0a4.5,4.5 0,1 1,9 0a4.5,4.5 0,1 1,-9 0"/>
    <path android:fillColor="@android:color/white" android:pathData="M3,13.5h8v8H3z"/>
</vector>
```

ic_launcher_background.xml

```
<?xml version="1.0" encoding="utf-8"?>
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="108"
    android:viewportHeight="108">
    <path
        android:fillColor="#3DDC84"
        android:pathData="M0,0h108v108h-108z" />
    <path
        android:fillColor="#00000000"
        android:pathData="M9,0L9,108"
        android:strokeWidth="0.8"
```

```
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,0L19,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M29,0L29,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M39,0L39,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M49,0L49,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M59,0L59,108"
    android:strokeWidth="0.8"
```

```
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M69,0L69,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M79,0L79,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M89,0L89,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M99,0L99,108"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,9L108,9"
    android:strokeWidth="0.8"
```

```
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,19L108,19"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,29L108,29"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,39L108,39"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,49L108,49"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,59L108,59"
    android:strokeWidth="0.8"
```

```
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,69L108,69"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,79L108,79"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,89L108,89"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M0,99L108,99"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,29L89,29"
    android:strokeWidth="0.8"
```

```
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,39L89,39"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,49L89,49"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,59L89,59"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,69L89,69"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M19,79L89,79"
    android:strokeWidth="0.8"
```

```
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M29,19L29,89"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M39,19L39,89"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M49,19L49,89"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M59,19L59,89"
    android:strokeWidth="0.8"
    android:strokeColor="#33FFFFFF" />
<path
    android:fillColor="#00000000"
    android:pathData="M69,19L69,89"
    android:strokeWidth="0.8"
```

```
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M79,19L79,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
</vector>
```

ic_launcher_foreground.xml

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="24"
    android:viewportHeight="24"
    android:tint="#DB0007">
  <group android:scaleX="0.58"
      android:scaleY="0.58"
      android:translateX="5.04"
      android:translateY="5.04">
    <path
        android:fillColor="@android:color/white"
        android:pathData="M16.22,12c0.68,0 1.22,-0.54 1.22,-1.22c0,-0.67 -0.54,-1.22 -1.22,-
1.22S15,10.11 15,10.78C15,11.46 15.55,12 16.22,12zM6.56,10.78c0,0.67 0.54,1.22
1.22,1.22S9,11.46 9,10.78c0,-0.67 -0.54,-1.22 -1.22,-1.22S6.56,10.11
6.56,10.78zM7.61,4L6.28,8h11.43l-1.33,-4H7.61zM16.28,3c0,0 0.54,0.01 0.92,0.54c0.02,0.02
0.03,0.04 0.05,0.07c0.07,0.11 0.14,0.24 0.19,0.4C17.66,4.66 19,8.69 19,8.69v6.5c0,0.45 -
0.35,0.81 -0.78,0.81h-0.44C17.35,16 17,15.64 17,15.19V14H7v1.19C7,15.64 6.65,16
6.22,16H5.78C5.35,16 5,15.64 5,15.19v-6.5c0,0 1.34,-4.02 1.55,-4.69c0.05,-0.16 0.12,-0.28
```

0.19,-0.4C6.77,3.58 6.78,3.56 6.8,3.54C7.18,3.01 7.72,3 7.72,3H16.28zM4,17.01h16V19h-7v3h-2v-3H4V17.01z"/>

</group>

</vector>

ic_repair.xml

<vector android:height="24dp" android:tint="#DB0007"

android:viewportHeight="24" android:viewportWidth="24"

android:width="24dp" xmlns:android="http://schemas.android.com/apk/res/android">

<path android:fillColor="@android:color/white" android:pathData="M16.22,12c0.68,0 1.22,-0.54 1.22,-1.22c0,-0.67 -0.54,-1.22 -1.22,-1.22S15,10.11 15,10.78C15,11.46 15.55,12 16.22,12zM6.56,10.78c0,0.67 0.54,1.22 1.22,1.22S9,11.46 9,10.78c0,-0.67 -0.54,-1.22 -1.22,-1.22S6.56,10.11 6.56,10.78zM7.61,4L6.28,8h11.43l-1.33,-4H7.61zM16.28,3c0,0 0.54,0.01 0.92,0.54c0.02,0.02 0.03,0.04 0.05,0.07c0.07,0.11 0.14,0.24 0.19,0.4C17.66,4.66 19,8.69 19,8.69v6.5c0,0.45 -0.35,0.81 -0.78,0.81h-0.44C17.35,16 17,15.64 17,15.19V14H7v1.19C7,15.64 6.65,16 6.22,16H5.78C5.35,16 5,15.64 5,15.19v-6.5c0,0 1.34,-4.02 1.55,-4.69c0.05,-0.16 0.12,-0.28 0.19,-0.4C6.77,3.58 6.78,3.56 6.8,3.54C7.18,3.01 7.72,3 7.72,3H16.28zM4,17.01h16V19h-7v3h-2v-3H4V17.01z"/>

</vector>

ic_settings.xml

<vector android:height="24dp" android:tint="#FFFFFF"

android:viewportHeight="24" android:viewportWidth="24"

android:width="24dp" xmlns:android="http://schemas.android.com/apk/res/android">

<path android:fillColor="@android:color/white" android:pathData="M19.14,12.94c0.04,-0.3 0.06,-0.61 0.06,-0.94c0,-0.32 -0.02,-0.64 -0.07,-0.94l2.03,-1.58c0.18,-0.14 0.23,-0.41 0.12,-0.61l-1.92,-3.32c-0.12,-0.22 -0.37,-0.29 -0.59,-0.22l-2.39,0.96c-0.5,-0.38 -1.03,-0.7 -1.62,-0.94L14.4,2.81c-0.04,-0.24 -0.24,-0.41 -0.48,-0.41h-3.84c-0.24,0 -0.43,0.17 -0.47,0.41L9.25,5.35C8.66,5.59 8.12,5.92 7.63,6.29L5.24,5.33c-0.22,-0.08 -0.47,0 -0.59,0.22L2.74,8.87C2.62,9.08 2.66,9.34 2.86,9.48l2.03,1.58C4.84,11.36 4.8,11.69 4.8,12s0.02,0.64 0.07,0.94l-2.03,1.58c-0.18,0.14 -0.23,0.41 -0.12,0.61l1.92,3.32c0.12,0.22 0.37,0.29 0.59,0.22l2.39,-0.96c0.5,0.38 1.03,0.7 1.62,0.94l0.36,2.54c0.05,0.24 0.24,0.41

0.48,0.41h3.84c0.24,0 0.44,-0.17 0.47,-0.41l0.36,-2.54c0.59,-0.24 1.13,-0.56 1.62,-0.94l2.39,0.96c0.22,0.08 0.47,0 0.59,-0.22l1.92,-3.32c0.12,-0.22 0.07,-0.47 -0.12,-0.61L19.14,12.94zM12,15.6c-1.98,0 -3.6,-1.62 -3.6,-3.6s1.62,-3.6 3.6,-3.6s3.6,1.62 3.6,3.6S13.98,15.6 12,15.6z"/>

</vector>

ic_launcher.xml

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@color/ic_launcher_background"/>
    <foreground android:drawable="@drawable/ic_launcher_foreground"/>
</adaptive-icon>
```

arrays.xml

```
<resources>
    <!-- Reply Preference -->
    <string-array name="reply_entries">
        <item>Reply</item>
        <item>Reply to all</item>
    </string-array>

    <string-array name="reply_values">
        <item>reply</item>
        <item>reply_all</item>
    </string-array>
</resources>
```

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
    <color name="red_1">#DB0007</color>
    <color name="red_2">#D10209</color>
    <color name="red_3">#910106</color>
    <color name="red_4">#EA0F16</color>
    <color name="red_5">#FF0910</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

ic_launcher_background.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="ic_launcher_background">#FFFFFF</color>
</resources>
```

strings.xml

```
<resources>
    <string name="app_name">Приложение для своевременного обслуживания авто</string>
</resources>
```

themes.xml

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.AutoReminderApp"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/red_2</item>
```

```
        <item name="colorPrimaryVariant">@color/red_3</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/red_4</item>
        <item name="colorSecondaryVariant">@color/red_5</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

themes.xml(night)

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.AutoReminderApp"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/red_1</item>
        <item name="colorPrimaryVariant">@color/red_3</item>
        <item name="colorOnPrimary">@color/black</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/red_4</item>
        <item name="colorSecondaryVariant">@color/red_4</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
```

```
        <item name="android:statusBarColor">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```