МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»
Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

02.03.03 математическое обеспечение и администрирование
(код наименование направления подготовки, специальности)
информационных систем

Технология программирования

БАКАЛАВРСКАЯ РАБОТА

на тему «Разработка мобильного приложения для диагностики автомобиля»

Студент Е.В. Кузнецова _____
(И.О. Фамилия) (личная подпись)

Руководитель В.С. Климов _____
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой доцент, к.т.н А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____» _____ 2018 г.

Тольятти 2018

АННОТАЦИЯ

Тема: «Разработка мобильного приложения для диагностики автомобиля».

Работа выполнена студентом Тольяттинского государственного университета, института математики, физики и информационных технологий, группы МОБ-1402, Кузнецовой Екатериной Витальевной.

Объект бакалаврской работы: методы разработки мобильного приложения, реализующие диагностику электронного блока управления автомобиля.

Предмет бакалаврской работы: разработка приложения для диагностики автомобиля.

Целью бакалаврской работы является создание мобильного приложения для диагностики автомобиля.

Поставленная перед исследованием цель требует решить следующие задачи:

- рассмотрение работы текущих популярных диагностических мобильных утилит для автомобиля и установка требований для разрабатываемого приложения;
- выбор необходимых для разработки инструментов и методов
- разработка программного продукта для диагностики автомобиля соответствующего поставленным требованиям
- тестирование и анализ работы разработанного программного продукта

В первом разделе изложен анализ предметной области, используемые программы для реализации приложения, описано как работает адаптер и приведены требования к программному продукту.

Во втором разделе приведена разработка самого приложения, создание интерфейса и установление связи между устройствами.

В третьем разделе проводится тестирование и анализ работы программного продукта.

В заключении подводятся итоги разработки приложения, формируются окончательные выводы по рассматриваемой теме.

Выпускная квалификационная работа состоит из пояснительной записки на 48 странице, введения, включая 32 рисунка, список из 25 ссылок, включая 5 иностранных источников и 1 приложения.

ABSTRACT

The title of the graduation work is «Development of a Mobile Application for Car Diagnostics». This graduation work is about creating an application for Android that will be able to read data from the adapter connected to the car and to show a list of errors. With the help of this program you will be able to diagnose the car yourself. Moreover, it is possible to receive timely information about various failures and deviations from the regulatory parameters.

The graduation work may be divided into several logically connected parts which are: 1) analysis of the subject area and conceptual requirements, 2) development of program code, 3) software testing, 4) conclusion.

The aim of the work is to create an application in which a simple interface and multitasking are combined. Also you can find out all the details about the malfunction on the Internet.

The object of graduation work is development of software for receiving and processing data from the OBD2 adapter.

In conclusion, we would like to stress that all modern cars, even with the minimum configuration, are equipped with on-board computers and a lot of sensors that are not easy to deal with even for a professional. The application being developed will provide simple diagnostics of the car. The thesis consists of an explanatory note in the amount of 48 pages, graphic part containing 32 sheets, the list of 25 references including foreign sources and 1 appendices.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность бакалаврской работы заключается в том, что объем мобильного рынка постоянно растет, а современный человек все время в движении. Создаются все новые приложения для мобильных устройств, которые экономят время и упрощают многие аспекты жизни современного человека, такие как, уход за автомобилем. С каждым годом автомобили приобретают все более сложные и обширные функции. Для осуществления работы всей электроники автомобиля отвечает множество систем и датчиков, работу которых надо отслеживать. И для того что бы следить за состоянием автомобиля существуют мобильные приложения для диагностики. С помощью него можно самостоятельно, продиагностировать автомобиль на наличие неисправностей.

Задание, полученное на бакалаврскую квалификационную работу – разработать приложение по диагностике автомобиля: приложение которое, с помощью Bluetooth подключается к OBD2 адаптеру. Считывает коды ошибок из памяти ЭБУ (электронного бортового устройства), а за тем передает данные на мобильное устройство. Это помогает точно идентифицировать множество неисправностей. Приложение создается для мобильных устройств.

Целью бакалаврской работы является создание мобильного приложения для диагностики автомобиля.

Объект бакалаврской работы: методы разработки мобильного приложения, реализующие диагностику электронного блока управления автомобиля.

Предмет бакалаврской работы: разработка приложения для диагностики автомобиля.

Решаемые задачи:

- рассмотрение работы текущих популярных диагностических мобильных утилит для автомобиля и установка требований для разрабатываемого приложения;

- выбор необходимых для разработки инструментов и методов
- разработка программного продукта для диагностики автомобиля соответствующего поставленным требованиям
- тестирование и анализ работы разработанного программного продукта

В первом разделе изложен анализ предметной области, используемые программы для реализации приложения, описано как работает адаптер и приведены требования к программному продукту.

Во втором разделе приведена разработка самого приложения, создание интерфейса и установление связи между устройствами.

В третьем разделе проводится тестирование и анализ работы программного продукта.

В заключении подводятся итоги разработки приложения, формируются окончательные выводы по рассматриваемой теме.

1 ФУНКЦИАНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ДИАГНОСТИКИ АВТОМОБИЛЯ

1.1 Анализ исследуемых задач

Активное внедрение электронных схем в устройство автомобиля переросло в создание единой системы электронного управления под контролем ЭБУ (электронный блок управления). Одновременно с этим, электронными модулями управления оснастили не только двигатель внутреннего сгорания, но также другие узлы и агрегаты современного автомобиля. Например, управляющая электроника контролирует тормоза, подушку безопасности, трансмиссию, элементы ходовой части и так далее. Для управления и контроля различных систем присутствуют многочисленные датчики, которые активно взаимодействуют с модулями. Благодаря этим модулям реализована возможность быстро выявить различные неисправности и сбои, то есть выполнить компьютерную диагностику двигателя и других узлов автомобиля.

В наше время у многих есть автомобиль. Однако не каждый владелец автомобиля имеет возможность и средства, каждый раз ехать в сервисный центр, когда, например, загорелся индикатор «Check Engine» («проверьте двигатель»). Когда возникает подобное уведомление, владелец автомобиля старается сразу отправить транспортное средство в автосервис, чтобы провести компьютерную диагностику. Специалисты считывают с бортового компьютера коды ошибок, а потом по ним определяют, по какой причине появилась неполадка. Не все знают, что провести диагностику автомобиля можно самостоятельно. Для этого предназначены специализированные диагностические устройства типа OBD2. Данные устройства устанавливаются практически в каждом современном автомобиле. Они подключаются к бортовой системе управления, и сопрягаются с различными мобильными устройствами. Для сопряжения устройств используется подключение через: Bluetooth, USB, Wi-Fi. Передача данных на этих интерфейсах соответствует нескольким стандартам, ни один из которых

напрямую не совместим с мобильными устройствами. Устройство ELM327 предназначено для использования в качестве моста связи между портом бортовой диагностики (OBD) и мобильным устройством.

Компьютерная диагностика является современным способом проверки элементов и узлов, которые взаимодействуют с электронной системой управления двигателем. К тому же, возникающие неисправности в одном узле или механизме могут повлиять на работу другого, что также фиксируется во время проверки и помогает найти имеющуюся неисправность.

Для проведения полноценной компьютерной диагностики автомобиля не достаточно иметь только диагностический прибор. Также нужно иметь специальную программу, которая предназначена для работы с устройствами вида OBD2. Одним из таких приложений является разрабатываемый программный продукт. Основная задача приложения заключается в том, чтобы любой водитель без помощи диагностических станций, быстро, смог получать данные о состоянии своего автомобиля. Используя простые и понятные в обращении инструменты диагностики OBD2 адаптера, пользователь может в любое время, самостоятельно, продиагностировать автомобиль на наличие неисправностей. Существуют ситуации, при которых диагностика обязательно должна быть проведена в целях безопасности, а именно:

- в преддверии весеннего сезона
- в преддверии зимнего сезона
- перед длительным путешествием на автомобиле
- при наличии признаков неисправностей

Более того, возможно своевременно выявить неполадку и исправить критическую ошибку, опасную для автомобиля или даже человека. А так же приложение сэкономит время, так как не нужно будет ехать в сервисный центр, можно на месте, за короткий срок провести диагностику. А также, пользователю не придётся тратиться на обслуживание.

Разрабатываемое приложение подойдет даже тем водителям, которые мало знакомы с диагностикой или устройством автомобиля.

При всем многообразии, большинство автомобильных микропроцессорных систем управления, построено по единому принципу. За счет этого приложение подходит для большинства автомобилей, совместимых со стандартом OBD2.

1.2 Характеристика работы сканера и бортовой диагностической системы

Для разрабатываемого приложения будет использоваться наиболее распространённый для связи прибор, реализованный на микросхеме ELM327 (рисунок 1.1).

Рисунок 1.1 - Сканер для компьютерной диагностики ELM327

Сканер рассчитан на подключение к последовательному порту (RS232) рисунок 1.2.

Рисунок 1.2 Диагностический разъем (RS232)

Для корректной работы устройства необходимо установить рабочую скорость обмена 38400 бод. Если будет введена неправильная скорость передачи данных, то информация, которую вы отправляете или получаете, будет искажена и нечитаема. Сканер может быстро определить по содержанию, кому предназначены посылаемые команды или сообщения. Команды для сканера должны всегда начинаться с символов «AT», команды или сообщения для бортовой диагностической системы содержат любые ASCII символы. ASCII (American Standard Code for Information Interchange) - Стандартный американский код обмена информацией. ASCII - это код для представления символов английского алфавита в виде чисел.

Стандарты требуют, чтобы любая группа посылаемых на автомобиль данных, удовлетворяла OBD формату. Первый байт «режим», всегда описывает тип, характер запрашиваемой информации, в то время как, второй, третий и другие, конкретизируют данные. Эти байты называются PID номером. PID (параметр идентификации данных) - код, посылаемый блоку управления ЭБУ, однозначно определяемый параметр, который должен тот вернуть.

Адаптеры ELM327 считывают через интерфейс OBD2 в основном P-коды и U-коды, и еще некоторые другие коды.

Код неисправности по стандарту OBD состоит из буквы и четырех цифр (например, P0100). Буква определяет принадлежность кода к системе:

P - Powertrain - коды, относящиеся к силовой установке автомобиля, а точнее ко всему, что приводит автомобиль в движение (двигатель, коробка передач, гибридная установка и так далее).

C - Chassis - коды, относящиеся к системам шасси автомобиля (антиблокировочная система тормозов, система курсовой устойчивости, пневмоподвеска, усилитель рулевого управления и так далее).

B - Body - коды, относящиеся к кузовным системам, в основном находящимся внутри салона автомобиля (система управления электрооборудованием кузова, противоугонные системы, система подушек безопасности, освещение и так далее).

U - коды, относящиеся к системам обмена данными между блоками/системами управления в автомобиле.

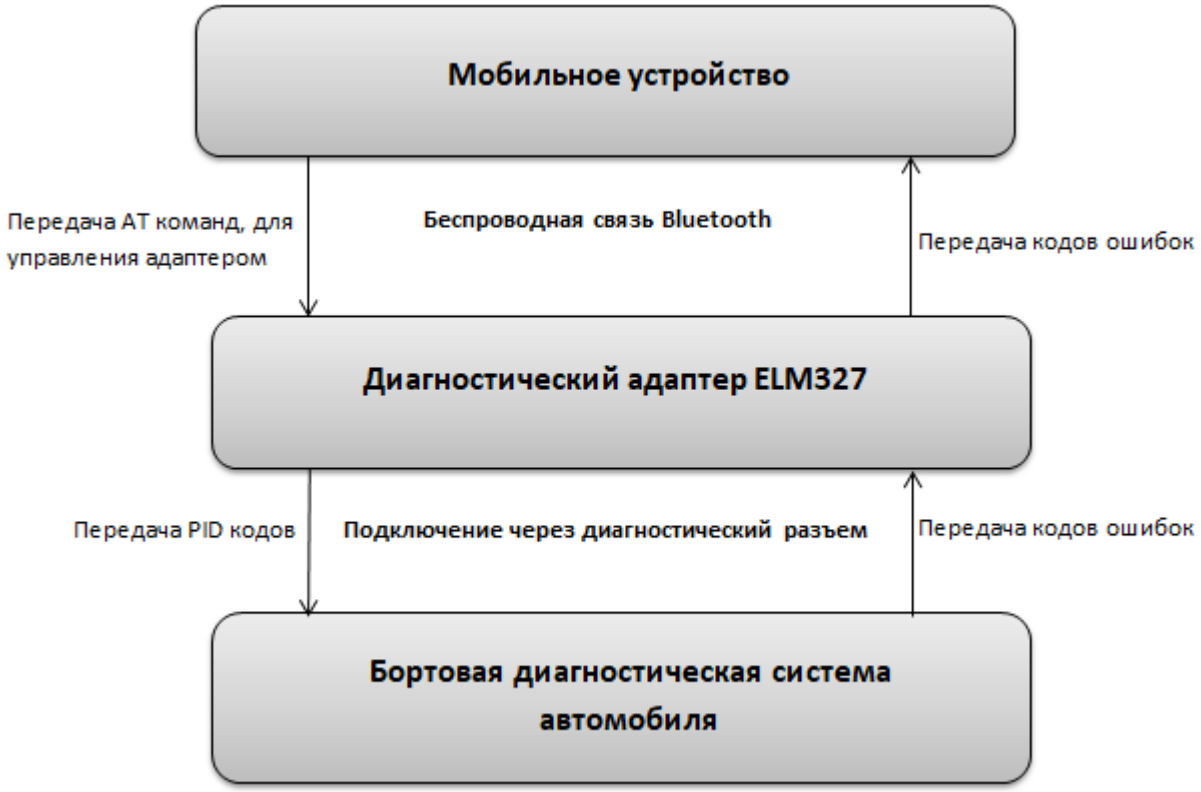

Рисунок 1.3 - Схема связи мобильного устройства и бортовой диагностической системы

1.3 Анализ известных приложений для диагностики автомобиля

Существует множество приложений на платформе Android для компьютерной диагностики автомобиля. Такие программы можно скачать в интернете или найти на электронной торговой площадке, это не составит труда. Правда большинство из них неудобны для среднестатистического пользователя, потому что отличаются сложностью применения и запутанностью. Для того что бы максимально разобраться в представленных приложениях изучим их преимущества, недостатки и особенности функционирования.

Разберем несколько самых популярных приложений для диагностики автомобиля, чтобы выявить преимущества и недостатки некоторых функций для схожих приложений.

ScanMaster Lite – приложение для диагностики автомобиля поддерживающий стандарт OBD2/EOBD. Имеет многие функции и коды ошибок в бесплатной версии, совместим со многими ELM327 чипами, удобный интерфейс на русском языке с возможностью вывода на экран данных в виде графиков. Недостатками данной программы являются: отсутствие поддержки диагностики отечественного автомобиля, для расшифровки кодов всех неисправностей требуется приобрести платную версию приложения, долго обрабатываются команды, иногда картинка графиков “ подвисает”.

OBD Авто Доктор – бесплатное приложение с поддержкой ELM327 адаптера, к тому же есть функции работы с GPS модулем. Основные преимущества программы: возможность считывать данные с автомобиля имеющих несколько электронных блоков, опция работы в GPS режиме, практичные виджеты, возможность вручную вводить команды, отображение данных в виде графиков в реальном времени. К недостаткам можно отнести следующее: долгая загрузка приложения, длительное время подключения к автомобилю, нагроможденное лишней информацией меню.

E OBD2 Facile диагностика Авто – программа производит диагностику главных параметров автомобиля с помощью ELM327 и OBD2 адаптеров. Основными плюсами приложения являются: быстрое соединение по беспроводной сети Bluetooth или Wi-Fi, стабильная работа, идентифицированные более 5000 кодов ошибок. Недостатками приложения являются: некоторые функции, связанные с удалением кодов, которые работают только в платной версии программы. Также, программа сильно энергозатратна, из-за чего мобильное устройство быстро разряжается.

Так же есть множество программ главным и решающим минусом, которого является – узконаправленность. Приложения, созданные под определенную марку автомобиля, например: ELMScan Toyota, EconTool for Nissan, FocccusScan и так далее. И большое количество полностью платных программ таких как: HobDrive, Torque Pro.

Приложения для диагностики оборудования автомобиля имеют схожие недостатки, тем самым подытожим: многие приложения не стабильно работают из-за перегруженности, некоторые долго грузятся или долго обрабатывают запросы, так же многие приложения или частично, или полностью платные. Большая часть программ узконаправленны, то есть работают только на определенных марках машин. К тому же, чтобы добраться до нужной функции почти в любом из таких приложений, приходится долго искать нужный пункт, так как многие, стараясь сделать многофункциональное приложение, добавляют не всегда нужные функции. Во многих похожих приложениях из-за огромного количества функций и плохой оптимизации, может «зависать» картинка или слишком долго обрабатываться команда. Кроме того, многие утилиты не поддерживают русский язык, что сильно затрудняет их освоение.

1.4 Требования к программному продукту

Выработка требований необходима для выявления и классификации условий, которые предъявляются к разрабатываемому мобильному приложению.

Мобильное приложение должно быть не просто красивым, а первую очередь оно должно решать проблемы пользователя. Быть практичным, удобным, интуитивно понятным и простым в восприятии. Работа приложения не должна быть осложнена ошибками, зависаниями и сбоями.

Технические требования к разрабатываемому продукту:

- осуществлять подключение к диагностическому адаптеру с помощью беспроводной связи Bluetooth
- считывать из ЭБУ коды ошибок автомобиля и выводить их на экран
- считывать из ЭБУ дополнительные данные о состоянии автомобиля в реальном времени и выводить на экран
- сохранять полученную информацию в базу данных приложения

Функциональность программного продукта состоит в том, что мобильное приложение сначала должно подключаться к OBD2 адаптеру посредством беспроводной связи Bluetooth. Если связь Bluetooth не включена на мобильном устройстве, то вывести окно с предложением включить сеть. Потом, адаптер должен считывать данные из электронного блока управления автомобиля, и передавать информацию на сопряженное устройство. При получении данных на экране мобильного устройства должна высвечивается все запрашиваемые данные.

Разрабатываемый программный продукт должен быть доступен бесплатно всем желающим.

Для осуществления работы приложения необходимо наличие машины, совместимой со стандартом OBD2, а также иметь адаптер ELM327 Bluetooth.

Требования к внешнему виду:

- реализация графического интерфейса (эргономичный и удобный, не требующий специальных знаний для использования)
- отображение ошибок ЭБУ автомобиля в виде текста понятного для пользователя

Благодаря простому и удобному интерфейсу, приложение будет легко в освоении. Пользователь, запустив приложение, сможет в пару кликов начать диагностику. За счет минимальных функций приложение будет работать быстро и без перегрузок. В тоже время затраты энергии будут минимальны.

Для реализации программной части приложения предлагается использовать язык программирования Java и интегрированную среду разработки Android Studio.

1.5 Обзор и выбор средств программирования

Сегодня смартфон является одним из важнейших вспомогательных устройств. Самыми популярными платформами которого являются iOS и Android, остальные менее распространены. Согласно статистике, 13% пользователей в мире предпочитают iOS, когда как 79% используют Android-девайсы. В России похожая ситуация: 11% к 77%. И около 7% пользователей выбирают Windows Phone. А оставшиеся 5 % приходится на долю остальных платформ. Android в настоящее время имеет самую большую долю среди других платформ. Именно поэтому я решила использовать платформу Android. А также эта платформа имеет открытое программное обеспечение, предназначенное для свободного доступа к исходному коду.

Платформа Android имеет огромное количество опций и может даже посоревноваться ими с операционными системами настольного компьютера. Имея многоуровневую среду на основе ядра Linux с обширными функциональными возможностями. А также Android предоставляет разработчику богатые коммуникационные возможности с Bluetooth, который позволяет девайсу осуществлять беспроводной обмен данными с другими устройствами Bluetooth. Структура приложения обеспечивает доступ к функциям Bluetooth через API Android Bluetooth. Эти API-интерфейсы позволяют приложениям отдаленно подключаться к другим устройствам Bluetooth, позволяя использовать функции «точка-точка» и «точка-много

точка». Используя Bluetooth API, приложение Android может выполнять следующие действия:

- Запрос локального адаптера Bluetooth для сопряженных устройств Bluetooth
- Подключение к другим устройствам через обнаружение служб
- Передача данных на другие устройства и обратно
- Управление несколькими подключениями

Связь девайса с адаптером будет осуществляться с помощью беспроводной связи Bluetooth. У Bluetooth сканеров скорость считывания чуть ниже чем у сканеров на базе USB, но отсутствие проводов и возможность подключения через смартфоны и другие устройства, перекрывают этот недостаток.

Мобильные приложения всегда отставали от стационарных компьютеров способами хранения данных. Android решает эту проблему благодаря наличию компактной встраиваемой базе данных с открытым исходным кодом SQLite. На рисунке 1.4 изображена упрощенная схема уровней программного обеспечения Android.

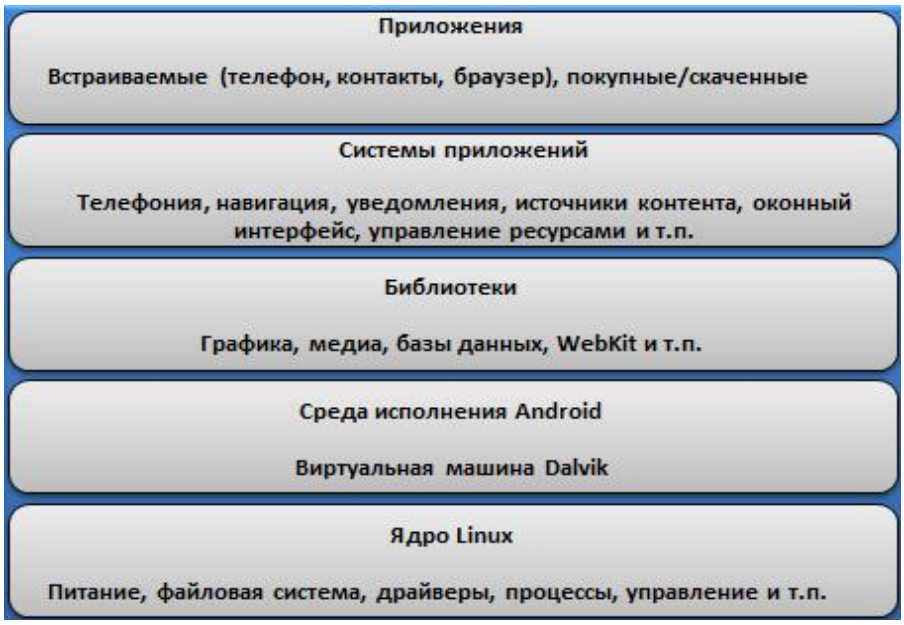

Рисунок 1.4 - Иерархия программного обеспечения платформы Android

Java является наиболее широко используемым языком заработки приложений. Согласно сайту VersionEye [16], который отслеживает открытые библиотеки исходного программного обеспечения, разработчики завершили большинство проектов при помощи языка программирования Java.

Язык Java является объектно-ориентированным, предоставляет разработчикам доступ к мощным библиотекам классов, ускоряющих разработку приложений. Java является наиболее подходящим языком для разработки мобильных приложений, так как он работает на всех платформах, включая все известные Android. А также операционная система Android написана на Java. Ключевые особенности языка:

- Объектно-ориентированный язык
- Работает на всех платформах
- Поддержка API-интерфейсов
- Язык легок в освоении и чтении
- Сотни библиотек с открытыми исходными текстами

Для создания программ на языке Java необходимо специальное программное обеспечение. Мною были выбраны следующие программы для реализации приложения:

1. SDK (Software Development Kit)
2. JDK (Java Development Kit)
3. ADB (Android Debug Bridge)
4. Android Studio

Самый простой способ приступить к разработке приложений для Android – это загрузить SDK Android. SDK – это программное обеспечение, позволяющие разработчикам создавать приложения для платформы Android. Оно включает в себя примеры проектов с исходным кодом, инструментами разработки, эмулятором и необходимыми библиотеками для создания

приложения для Android. Приложения пишутся с использованием языка программирования Java и выполняются в Dalvik Virtual Machine, пользовательской виртуальной машине, внутри нее запускается каждое приложение, которая работает в свою очередь в пределах управляемого ядром Linux процесса, как показано на рисунке 1.5.

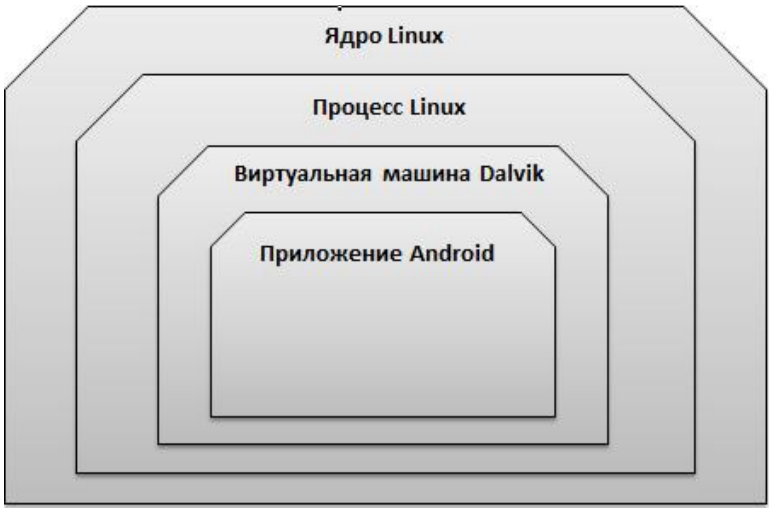

Рисунок 1.5 - Архитектура Android

JDK –это среда разработки программ для написания Java апплета и приложений. Она состоит из среды выполнения, которая находится «на вершине» операционно-системного слоя, а также инструментов и программ, которыми разработчики должны компилировать, отлаживать и запускать. приложения, написанные на языке Java.

ADB – это универсальный инструмент командной строки, который позволяет вам общаться с устройством (эмулятором или подключенным устройством Android). ADB входит в пакет Android-SDK Platform-Tools. Команда ADB облегчает различные действия устройства, такие как установка и отладка приложений, и обеспечивает доступ к оболочке Unix, которую вы можете использовать для запуска различных команд на устройстве. Это клиент-серверная программа, которая включает в себя три компонента:

- Клиент, который отправляет команды. Он работает на вашей машине разработки. Вы можете вызвать клиента из терминала командной строки, выпустив команду adb.
- Демон (adbd), который запускает команды на устройстве. Он выполняется как фоновый процесс на каждом устройстве.
- Сервер, который управляет связью между клиентом и демоном. Он работает как фоновый процесс на вашей машине разработки.

Android Studio – это официальная интегрированная среда разработки (IDE) для создания приложений для Android на основе IntelleJ IDEA. Этот сложный программный комплекс включает в себя текстовый редактор с подсветкой синтаксиса и подсказками, компилятор или интерпретатор, интегрированный отладчик и средства для автоматизации сборки. В дополнение к мощным редакторам и инструментам разработчика, Android Studio имеет большое количество возможностей, которые повышают производительность при создании приложений для Android таких как:

- Гибкая система сборки
- Быстрый и многофункциональный эмулятор
- Единая среда, в которой возможно разрабатывать приложения для всех Android-устройств
- Шаблоны кода и интеграция Github, чтобы помогающие создавать общие функции приложения и импортировать пример кода
- Обширные инструменты и средства тестирования

2 РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ДИАГНОСТИКИ АВТОМОБИЛЯ

2.1 Архитектура разрабатываемого мобильного приложения

Для разработки приложения необходимо построение архитектуры для того что бы код приложения был модульным и легко читаемым. Что бы построить архитектуру приложения, можно разбить проект на 3 слоя, каждый из которых имеет свою цель и может работать независимо от остальных (Рисунок 2.1). Рассмотрим каждый слой подробнее.

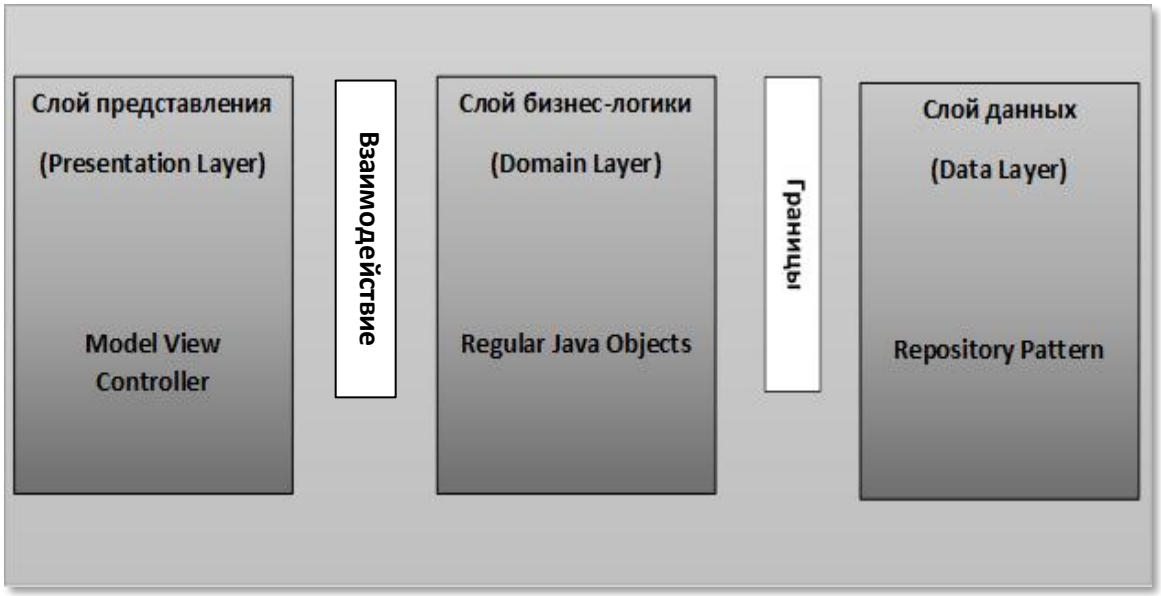

Рисунок 2.1 Схема слоев приложения

Слой представления (Presentation Layer) состоит из различных компонентов Android-флеймворка таких как: Fragments, Activities, ViewGroups и др. Для реализации слоя представления выбран паттерн Model View Controller (MVC).

Когда пользователь взаимодействует с View (например нажимает кнопку), View передает эту информацию в Controller. Controller в свою очередь обрабатывает это событие в соответствии со своей логикой и изменяет Model (Рисунок 2.2).

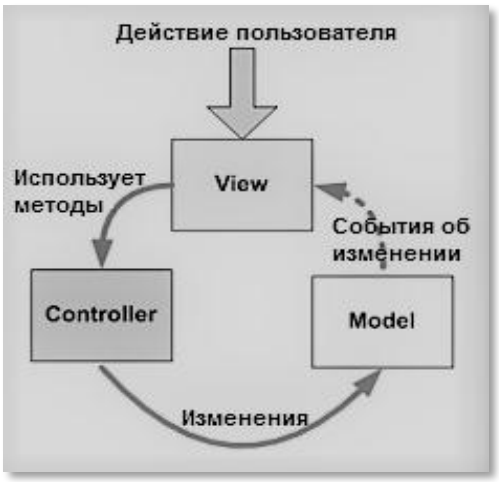

Рисунок 2.2 Схема паттерна MVC

Слой бизнес-логики (Domain Layer) вся логика и реализация взаимодействий (use cases – методы использования) находится в этом слое. Здесь взаимодействуют операторы для объединения, фильтрации и преобразования полученных данных от вспомогательных классов.

Слой данных (Data Layer) вмещает в себе все данные, необходимые для приложения. Которые поставляются в слой бизнес-логики посредством паттерна хранилища (Repository Pattern) использующего реализацию UserRepository. Паттерн выбирает различные источники в зависимости от определенных условий (Рисунок 2.3).

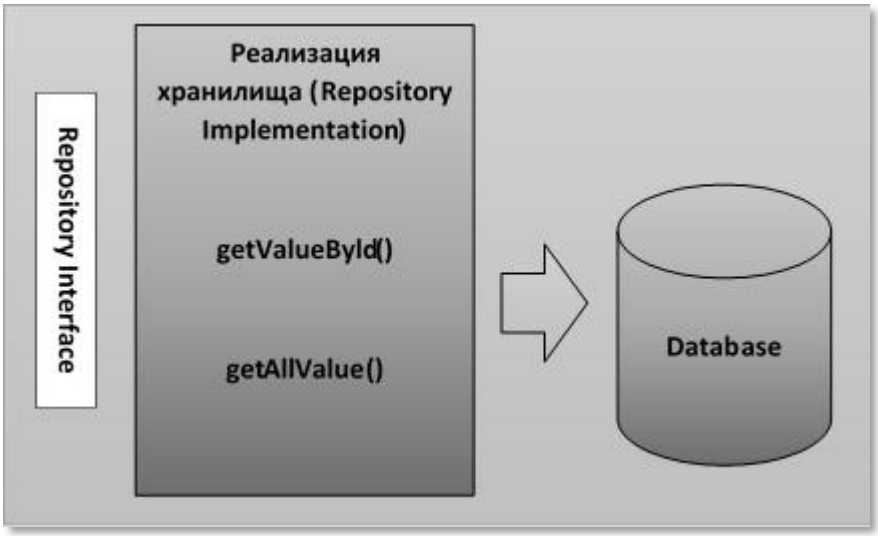

Рисунок 2.3 Схема работы с данными в приложении

2.2 Методы разработки программного продукта

Java предоставляет ряд модификаторов, используемых не для доступа, к реализации многих других функциональных возможностей. И вот не которые из них, которые использовались в разработке приложения:

Ключевое слово static используется для создания переменных, которые будут существовать независимо от каких-либо экземпляров, созданных для класса. Только одна копия переменной static в Java существует вне зависимости от количества экземпляров класса. Статические переменные также известны как переменные класса. Так же локальные переменные в Java не могут быть объявлены статическими (рисунок 2.12).

Переменная final может быть инициализирована только один раз. Ссылочная переменная, объявленная как final, никогда не может быть назначена для обозначения другого объекта. Однако данные внутри объекта могут быть изменены. Таким образом, состояние объекта может быть изменено, но не ссылки. С переменными в Java модификатор final часто используется со static, чтобы сделать константой переменную класса (рисунок 2.4).

```
private static final String TAG = TroubleCodesActivity.class.getName();
private static final int NO_BLUETOOTH_DEVICE_SELECTED = 0;
private static final int CANNOT_CONNECT_TO_DEVICE = 1;
private static final int NO_DATA = 3;
private static final int DATA_OK = 4;
private static final int CLEAR_DTC = 5;
private static final int OBD_COMMAND_FAILURE = 10;
```

Рисунок 2.4 - Часть кода с переменной final и stasic

Метод onCreate генерируется автоматически при создании проекта приложения и вызывается системой после запуска класса активности (рисунок 2.5). Метод onCreate обычно инициализирует переменные экземпляра Activity и компоненты GUI. Этот метод должен быть предельно упрощен, чтобы приложение загружалось быстро. Фактически в случае, если загрузка приложения занимает более пяти секунд, операционная система

отображает диалоговое окно ANR (Application Not Responding, приложение не отвечает).

```
//выззывается при первом создании акстивности
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // вызов врсии суперкласса
```

Рисунок 2.5 - Часть кода с методом OnCreate

Java поддерживает несколько потоков для выполнения. Это может привести к тому, что два или более потока получат доступ к одному и тому же полю или объекту. Чтобы выполнять все параллельные потоки в программе синхронно используется synchronized. Синхронизация позволяет избежать ошибок согласованности памяти, вызванные из-за непоследовательного доступа к общей памяти. Synchronized используется в классах определяя синхронизированные методы или блоки (рисунок 2.6).

```
synchronized (this) {
    Log.d(TAG, "Запуск службы..");

    final BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();
    dev = btAdapter.getRemoteDevice(params[0]);

    Log.d(TAG, "Остановка сканирования Bluetooth.");
    btAdapter.cancelDiscovery();

    Log.d(TAG, "Запуск соединения OBD..");
```

Рисунок 2.6 - Часть кода с синхронизацией

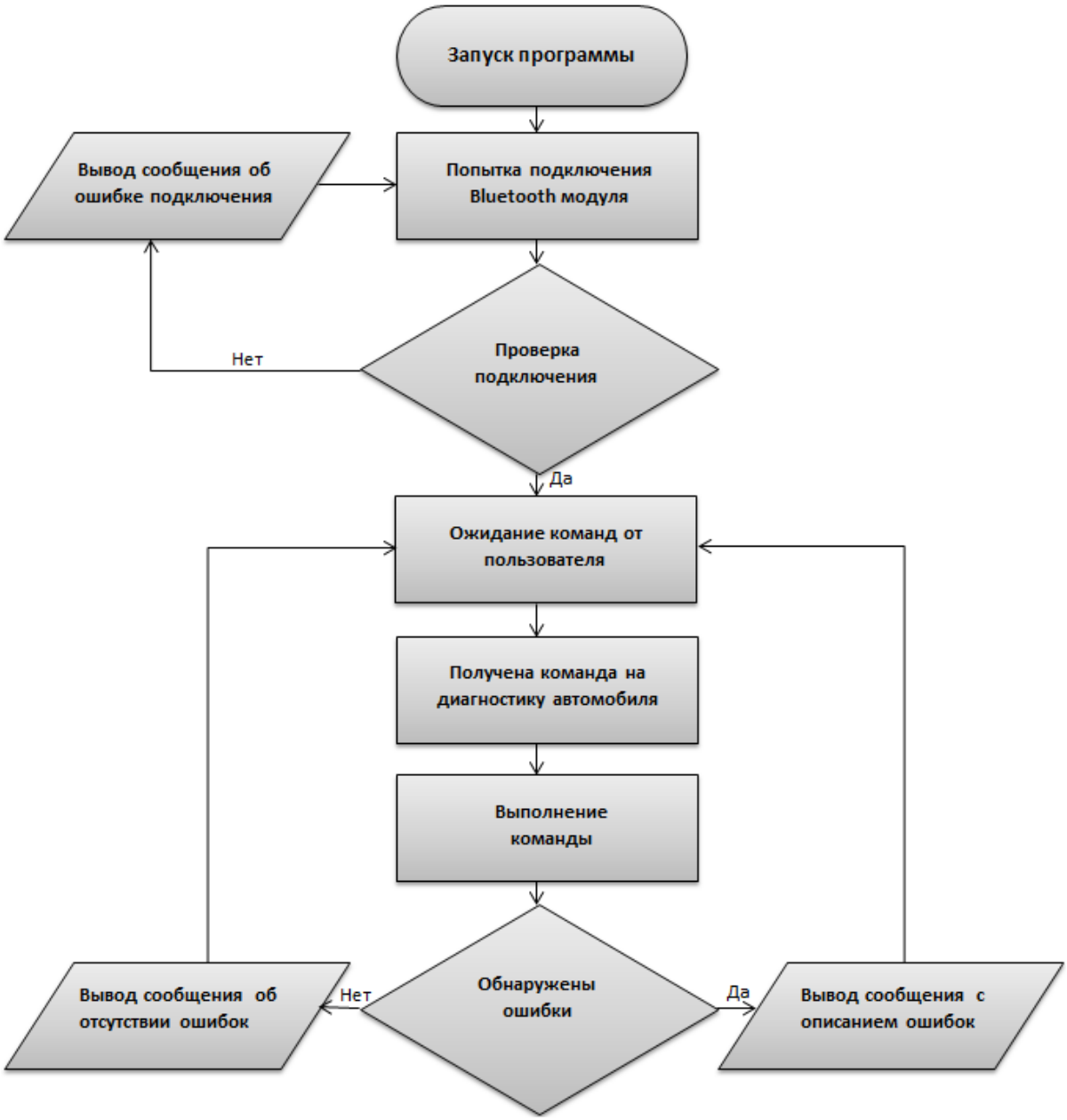

Рисунок 2.7 - Блок схема работы мобильного приложения

2.3 Разработка интерфейса

Для создания интерфейса был использован язык разметки XML (Extensible Markup Language). Он позволяет структурировать информацию разного типа, может использоваться в любом приложении, которому необходима структурированная информация. Так что XML – документ

является универсальным форматом для обмена информацией внутри большой программы и отдельными ее компонентами.

Так как я работаю в Android Studio, в нем можно открыть файл activity_main.xml и внизу при помощи кнопки Design переключиться к графическому представлению интерфейса в виде смартфона.

При любых изменениях в режиме дизайнера все действия будут синхронизироваться с файлом activity_main.xml

Перед тем как преступить к созданию интерфейса, стоит выделить несколько основных положений: интерфейс должен быть удобным и минималистичным, а его структура понятна пользователю (Рисунок 2.8).

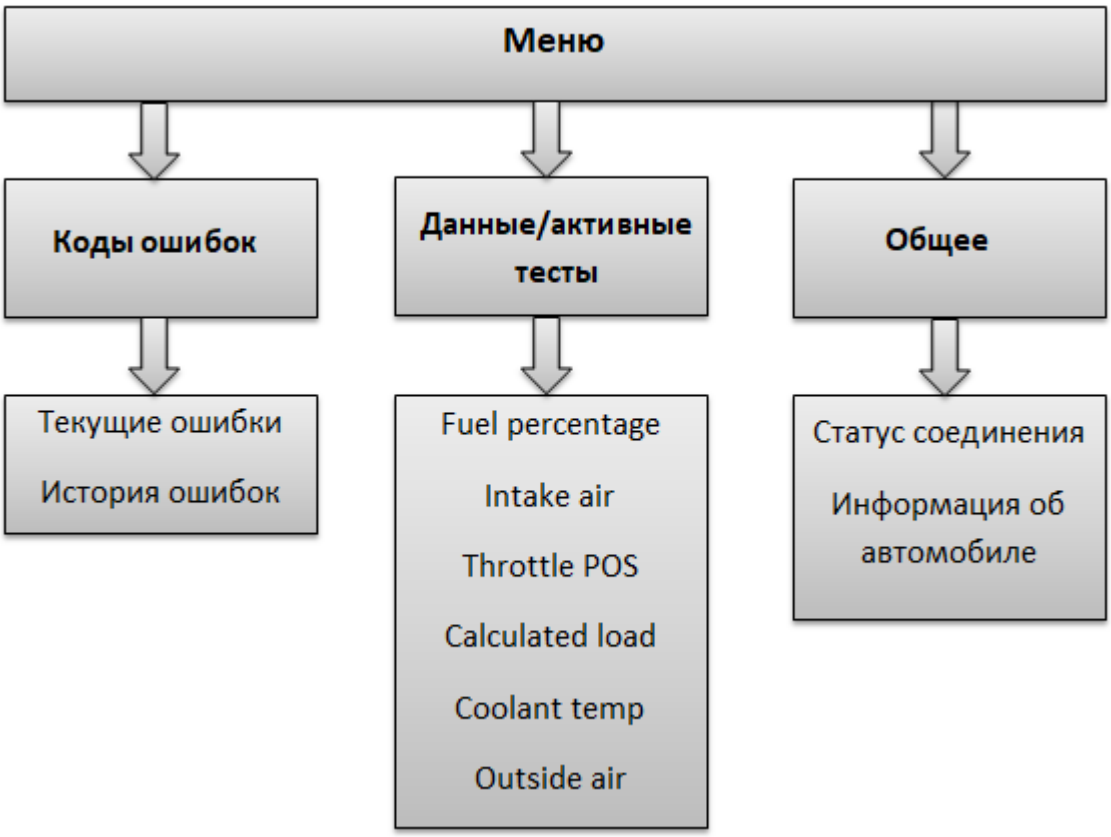

Рисунок 2.8 - Схема структуры интерфейса

Графический интерфейс должен представлять собой иерархию объектов android.view.View и android.view.ViewGroup (рисунок 2.9). В каждом объекте ViewGroup содержатся дочерние объекты View. Простые объекты View это элементы взаимодействия с программой такие как: кнопки, виджеты текстовые поля.

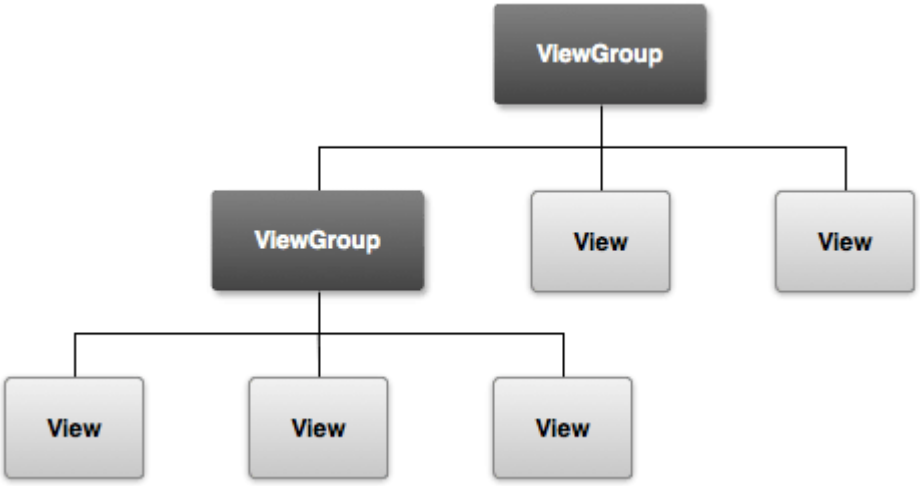

Рисунок 2.9 - Схема иерархии объектов визуальных элементов

При создании разметки в XML стоит помнить о том, что каждый файл разметки должен содержать корневой элемент представляющий объект View или ViewGroup.

Класс ArrayAdapter это адаптер, который связывает массив данных с элементами TextView. Источником данных является массив объектов. ArrayAdapter у каждого объекта вызывает метод toStrong() приводя к строковому виду и получившуюся строку устанавливает в элемент TextView. Пример разметки на рисунке 2.10.

```
    <?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/activity_main"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

<ListView
        android:id="@+id/countriesList"
                android:layout_width="match_parent"
                android:layout_height="match_parent">
</ListView>

</RelativeLayout>
```

Рисунок 2.10 - Файл разметки activity_main.xml

Через ArrayAdapter свяжем ListView с некоторыми данными, как показано на рисунке 2.11.

```
package com.example.eugene.listapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    String[] countries = {"Общие, Коды ошибок, Данные/активные тесты"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // получаем элемент ListView
        ListView countriesList = (ListView) findViewById(R.id.countriesList);

        // создаем адаптер
        ArrayAdapter<String> adapter = new ArrayAdapter(this,
                android.R.layout.simple_list_item_1, countries);

        // устанавливаем для списка адаптер
        countriesList.setAdapter(adapter);
    }
}
```

Рисунок 2.11 - Код activity

Сначала получаем по ID элемент ListView, потом создаётся адаптер. Адаптер зарабатывался при помощи конструктора ArrayAdapter<String> (this, android.R.layout.simple_list_item_1, countries), где:

- this – объект activity
- android.R.layout.simple_list_item_1- файл разметки списка
- countries – массив или список данных

При включении приложения запускается интерфейс программы, далее пользователь попадает на основной экран «Общее». Он состоит из нескольких пунктов:

- статус соединения (установлено ли соединение с электронным блоком управления)
- информация об автомобиле (модель, двигатель, количество текущих и ранее сохраненных кодов ошибок)

Нажав на пункт «Статус соединения» можно вручную запустить поиск ближайших Bluetooth устройств, для спряжения, а затем выбрать нужное устройство.

На основном экране также имеется кнопка «Подключиться к ECU» осуществляющая соединение мобильного устройства и диагностического адаптера (рисунок 2.12).

Рисунок 2.12 - Основной экран программы

Нажав на кнопку вызова меню (кнопка слева вверху), слева выдвинется боковая панель с пунктами меню (рисунок 2.13):

- Общие
- Коды ошибок
- Данные/активные тесты

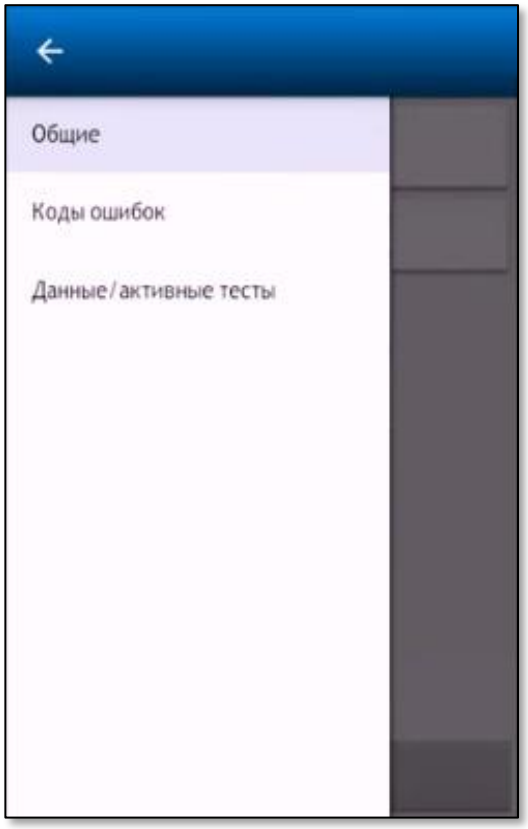

Рисунок 2.13 - Боковая панель с меню программы

После выбора пункта «Коды ошибок» программа автоматически запускает сканирование ЭБУ автомобиля (мобильное устройство должно быть подключено к диагностическому адаптеру для выполнения команды), а затем выдает коды ошибок на экран.

При выборе пункта «Данные/активные тесты» автоматически запускается диагностика нескольких параметров сразу, полученная с диагностического адаптера, информация, выдается в виде списка.

При необходимости можно стереть все полученные ранее данные с помощью кнопки «Очистить список».

2.4 Организация настроек и передача данных через Bluetooth

Платформа Android предоставляет разработчику богатые коммуникационные возможности. Для работы с Bluetooth в состав Android входит мощный API, позволяющий легко производить сканирование окружающего пространства на предмет наличия готовых к соединению устройств, передачу данных между устройствами и многое другое.

Работа с Bluetooth состоит из четырех этапов: установка настроек bluetooth адаптера, поиск доступных для соединения устройств, установка соединения, передача данных.

Bluetooth API располагается в пакете android.bluetooth. В его состав входит несколько классов:

BluetoothAdapter- отвечает за работу с установленным в мобильном устройстве Bluetooth модулем. Экземпляр этого класса есть в любой программе, использующей bluetooth. В состав этого класса входят методы, позволяющие производить поиск доступных устройств, запрашивать список подключенных устройств, создавать экземпляр класса BluetoothDevice на основании известного MAC адреса и создавать BluetoothServerSocket для ожидания запроса на соединение от других устройств.

BluetoothDevice- класс, ассоциирующийся с удаленным Bluetooth устройством. Экземпляр этого класса используется для соединения через BluetoothSocket или для запроса информации об удаленном устройстве (имя, адресс, класс, состояние).

BluetoothSocket- интерфейс для Bluetooth socket, аналогичный TCP сокетам. Это точка соединения, позволяющая обмениваться данными с удаленным устройством через InputStream и OutputStream.

Схема работы приложения:

- подключение к ELM327 адаптеру через Bluetooth
- инициализировать ELM327 адаптер при помощи AT команд
- постоянно получать необходимые данные с автомобиля с помощью отправки соответствующих PID (параметр идентификации данных) кодов

Для работы с Bluetooth модулем, необходимо подключить пакет API (рисунок 2.14).

```
import android.bluetooth.*;
```

Рисунок 2.14 - Код подключения пакета API

Далее необходимо разрешить приложению использовать Bluetooth модуль, для этого добавим следующую строку в манифест программы (рисунок 2.15).

```
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Рисунок 2.15 - Строка кода разрешающая использование Bluetooth модуля.

Создаем класс BluetoothAdapter (рисунок 2.16).

```
BluetoothAdapter bluetooth= BluetoothAdapter.getDefaultAdapter();
```

Рисунок 2.16 - Класс BluetoothAdapter.

Для поиска устройств, путем сканирования радиодиапазона на наличие устройств доступных для сопряжения, используем метод startDiscovery(). Который проходит в отдельном потоке, и возвращает true если удалость запустить сканирование. Для получения информации о найденных устройствах устройство должно зарегистрировать BroadcastReceiver для интента ACTION_FOUND, который вызывается для каждого найденного устройства. Интент содержит дополнительные поля EXTRA_DEVICE и EXTRA_CLASS, которые содержат объекты BluetoothDevice и BluetoothClass соответственно. (пример на рисунке 2.17).

Для сопряжения с удаленным устройством необходимо получить информацию о нем, которая содержится в объекте BluetoothDevice. Этот объект используется для инициализации сопряжения (пример на рисунке 2.18). Что бы не замедлять сопряжение устройств, мы должны убедится, что поиск устройств для сопряжения прекращено, для этого используем метод cancelDiscovery().

```
// Создаем BroadcastReceiver для ACTION_FOUND
private final BroadcastReceiver mReceiver=new BroadcastReceiver(){
public void onReceive(Context context, Intent intent){
        String action= intent.getAction();
// Когда найдено новое устройство
        if(BluetoothDevice.ACTION_FOUND.equals(action)){
// Получаем объект BluetoothDevice из интента
        BluetoothDevice device= intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        mArrayAdapter.add(device.getName()+"\n"+ device.getAddress());
        }
        }
        };
// Регистрируем BroadcastReceiver
        IntentFilter filter=new IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(mReceiver, filter);
```

Рисунок 2.17 - Часть кода осуществляющий поиск устройств.

```
public ConnectThread(BluetoothDevice device){
        BluetoothSocket tmp=null;
        mmDevice= device;

// получаем BluetoothSocket чтобы соединиться с BluetoothDevice
        try{
        tmp= device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch(IOException e){}
        mmSocket= tmp;
        }

public void run(){
// Отменяем сканирование, поскольку оно тормозит соединение
        mBluetoothAdapter.cancelDiscovery();

        try{

        mmSocket.connect();
        } catch(IOException connectException){
// Невозможно соединиться. Закрываем сокет и выходим.
        try{
        mmSocket.close();
        } catch(IOException closeException){}
        return;
        }

// управлчем соединением (в отдельном потоке)

        manageConnectedSocket(mmSocket);
        }

/** отмена ожидания сокета */
public void cancel(){
        try{
        mmSocket.close();
        } catch(IOException e){}
        }
        }
```

Рисунок 2.18 - Часть кода инициализирующее соединение.

После успешного сопряжения устройств можно реализовать передачу данных при помощи объекта BluetoothSocket. Объекты InputStream и OutputStream отвечают за управление передачей через сокет. Получаем их

при помощи методов getInputStream() и getOutputStream(). Для чтения и записи данных нужно использовать отдельные потоки, потому что методы read(byte[]) и write(byte[]), которые мы используем, блокируют друг друга (пример на рисунке 2.19).

```
public ConnectedThread(BluetoothSocket socket){
        mmSocket= socket;
        InputStream tmpIn=null;
        OutputStream tmpOut=null;

// Получить входящий и исходящий потоки данных
        try{
        tmpIn= socket.getInputStream();
        tmpOut= socket.getOutputStream();
        } catch(IOException e){}

        mmInStream= tmpIn;
        mmOutStream= tmpOut;
        }

public void run(){
        byte[] buffer=new byte[1024];
        int bytes;

        while(true){
        try{

        bytes= mmInStream.read(buffer);
        mHandler.obtainMessage(MESSAGE_READ, bytes,-1, buffer)
        .sendToTarget();
        } catch(IOException e){
        break;
        }
        }
        }
/* Вызываем этот метод из главной деятельности, чтобы отправить данные
удаленному устройству */
public void write(byte[] bytes){
        try{
        mmOutStream.write(bytes);
        } catch(IOException e){}
        }

/* Вызываем этот метод из главной деятельности,
чтобы разорвать соединение */
public void cancel(){
        try{
        mmSocket.close();
        } catch(IOException e){}
        }
```

Рисунок 2.19 - Часть кода для реализации передачи данных.

2.5 Реализация хранения данных

Для работы с полученными данными используется библиотека Room, предназначенная для работы с базой данных SQLite. Room имеет три основных компонента: Entity, Dao и Database.

Аннотация Entity помечает объект, который будет храниться в базе данных. Создадим класс CarValue который будет представляет собой активные данные автомобиля (Рисунок 2.20)

```
@Entity
public class CarValue {

    @PrimaryKey
    public long idvalue;

    public float load;

    public int fuel;
```

Рисунок 2.20 Часть кода с аннотацией Entity

Объекты класса CarValue при работе с базой данных будут использоваться. Класс CarValue используется для создания таблицы в базе. Имя класса соответствует имени таблицы. Поля таблицы создаются в соответствии с полями класса. PrimaryKey помечает поле, которое будет ключом в таблице.

Описывать методы для работы с базой данных, мы будем в объекте Dao. А также необходимо создать методы для получения списка данных автомобиля. А также для таких операций как: удаление, изменение, добавление данных. Пример их описания на Рисунке 2.21.

```
@Dao
public interface CarValueDao {

    @Query("SELECT * FROM carvalue")
    List<CarValue> getAll();

    @Query("SELECT * FROM carvalue WHERE idvalue = :idvalue")
    CarValue getById(long idvalue);

    @Insert
    void insert(CarValue carvalue);

    @Update
    void update(CarValue carvalue);

    @Delete
    void delete(CarValue carvalue);

}
```

Рисунок 2.21 Часть кода с аннотацией Dao

Методы getAll и getByld вызывают весь список данных автомобиля или конкретную величину по idvalue. Для получения данных нам нужно прописать SQL запросы в аннотации Query.

Аннотацией Database помечается основной класс, взаимодействующий с базой данных. (Рисунок 2.22)

```
@Database(activedata = {CarValue.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract CarValueDao carvalueDao();
}
```

Рисунок 2.22 Часть кода с аннотацией Database

В параметрах Database обозначаем версию базы и какие Entity будут использоваться. Таблица будет создаваться для каждого класса Entity из списка activedata. Создание Database изображено на Рисунке 2.23.

```
AppDatabase db =  Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class, "database").build();
```

Рисунок 2.23 Часть кода создания объекта Database

Используем класс Application для создания и хранения AppDatabase (Рисунок 2.24).

```
public class App extends Application {

    public static App instance;

    private AppDatabase database;

    @Override
    public void onCreate() {
        super.onCreate();
        instance = this;
        database = Room.databaseBuilder(this, AppDatabase.class, "database")
                .build();
    }

    public static App getInstance() {
        return instance;
    }

    public AppDatabase getDatabase() {
        return database;
    }
}
```

Рисунок 2.24 Часть кода AppDatabase

3 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ДИАГНОСТИКИ АВТОМОБИЛЯ

3.1Метод тестирования

Для того что бы проверить работоспособность приложения, необходимо провести испытание программного продукта, то есть протестировать его.

Для тестирования данного мобильного приложения было выбрано системное тестирование. Системное тестирование – это испытание программного продукта, с целью проверки соответствия системы исходным требованиям и относится оно к сфере тестирования черного ящика (Black Box Testing). Метод черного ящика рассматривает систему как «черный ящик», знание внутренней структуры и кода не используется. Основное внимание уделяется тестированию на функциональность системы в целом.

Эти методы наиболее подходят для тестирования моего программного продукта по ряду причин:

- приложение должно соответствовать выдвинутым ранее требованиям
- приложение должно выполнять свои функции, а знание внутренней работы системы не важно
- нам необходимо протестировать все программные компоненты

Начнем тестирование с проверки запуска программы, и базовых функций. Потом проверка подключения приложения к адаптеру и получение данных.

Тестируемые функции:

1. Корректный запуск программы.
2. Корректная работа переходов между окнами программы, а также возврата на предыдущую станицу.
3. Функция поиска и сопряжения устройств по сети Bluetooth.
4. Функция поиска кодов ошибок автомобиля.

5. Функция диагностики дополнительных параметров.

6. Испытание работоспособности всех функциональных кнопок приложения.

3.2 Выполнение тестирования и вывод

В ходе разработки приложения для диагностики автомобиля было выполнено:

- создан интерфейс программного продукта
- реализовано соединение мобильного устройства и диагностического адаптера по беспроводной сети Bluetooth
- создано приложение, которое посредством связи с диагностическим адаптером получает необходимые данные из ЭБУ автомобиля

Для проверки работоспособности приложения необходимо иметь диагностический адаптер ELM327 и автомобиль, поддерживающий его протоколы.

Сначала подключим диагностический адаптер (ELM327) к последовательному порту (RS232) автомобиля. Далее будем работать только с приложением.

После запуска приложения необходимо подключиться к ЭБУ автомобиля для считывания данных. Это осуществляется путем связи мобильного приложения и диагностического адаптера с помощью сети Bluetooth. Для этого необходимо выбрать пункт «Статус соединения», а затем нажать кнопку поиск устройств, работа сканирования радиодиапазона на наличие доступных устройств показана на рисунке 3.1.

Рисунок 3.1 - Поиск устройств для сопряжения

На рисунке 3.1 видно, что найдено диагностическое устройство (OBD2). Что бы осуществить связь между мобильным устройством и нужным адаптером, достаточно выбрать его в списке. Выбрав нужное устройство, приложение спрягается с ним и готово к работе.

Для диагностики автомобиля на предмет ошибок в ЭБУ, необходимо в меню выбрать пункт «Коды ошибок», программа сразу же начнет поиск кодов ошибок. Если найдены ошибки, они выводится на экран с пометкой «Текущие», как показано на рисунке 3.2. Если ошибки не найдены, выведется сообщение «Ошибок нет». Все сканируемые ранее ошибки остаются в списке даже при закрытии приложения. Ошибки, выявленные ранее, помечаются надписью: «Прошлые».

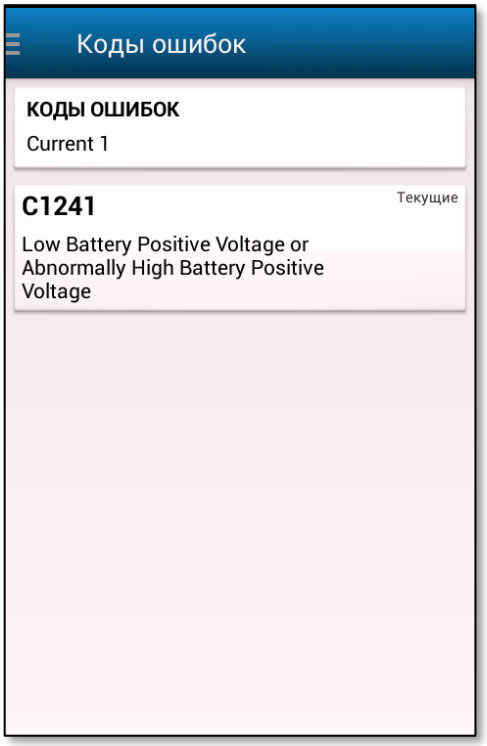

Рисунок 3.2 - Коды ошибок

На рисунке 3.2 видно, что при сканировании была найдена ошибка, «Low Battery Positive Voltage or Abnormally High Battery Positive Voltage» (низкое или аномально высокое напряжение в аккумуляторе).

В разделе «Данные/активные тесты» программа проводит диагностику дополнительных параметров, таких как уровень топлива, температура внешнего воздуха, температура всасываемого воздуха, положение дроссельной заслонки, нагрузка на двигатель, температура охлаждающей жидкости и тд. (рисунке 3.3).

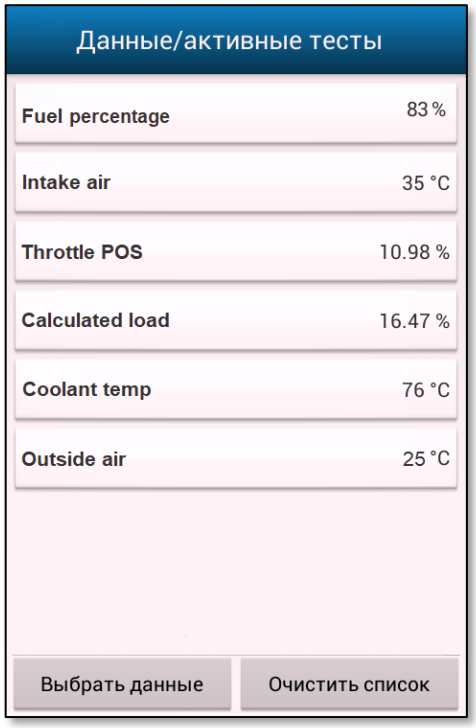

Рисунок 3.3 - Данные/активные тесты

Данные на рисунке 3.3 отображаются корректно, ошибок не выявлено.

Из полученных данных следует: все проверяемые функции работают исправно. Приложение осуществляет сопряжение с диагностическим адаптером. При подключении программного продукта к автомобилю отображаются все запрашиваемые данные.

Все заявленные модули программного продукта функционирует корректно. Приложение ошибок не выдает, все окна приложения функционируют.

ЗАКЛЮЧЕНИЕ

Темой бакалаврской работы является: «Разработка мобильного приложения для диагностики автомобиля».

В процессе выполнения бакалаврской работы была выделена актуальность исследуемой темы, определены объект, предмет исследования, цели и задачи работы.

При разработке программы были решены следующие задачи:

1. Были изучены методы разработки мобильного приложения, реализующие диагностику электронного блока управления автомобиля.
2. Были рассмотрены работы текущих популярных диагностических мобильных утилит для автомобиля и установка требований для разрабатываемого приложения;
3. Были выбраны необходимые инструменты для разработки
4. Был разработан программный продукт для диагностики автомобиля, соответствующего поставленным требованиям
5. Были выполнены тестирование и анализ работы разработанного программного продукта
6. На языке Java была разработана и протестирована программа для диагностики автомобиля на платформе Android. Приложение, которое посредством сети Bluetooth подключается к OBD2 адаптеру. Считывает данные из памяти ЭБУ и выводит их на экран мобильного устройства.

Для дальнейших продвижений в разработке данного программного продукта, необходимо адаптировать его под другие платформы, для расширения круга пользователей данного приложения.

Реализация поставленных задач в бакалаврской работе привела к созданию удобного мобильного приложения. Программный продукт со своей задачей справляется и работает стабильно. Главная особенность разработанной программы в ее простоте, так как основную функцию – поиск кодов ошибок, пользователь может совершить всего в пару кликов. Для улучшения качества диагностики помимо поиска кодов программой

осуществляется вывод дополнительных параметров, такие как уровень топлива, температура охлаждающей жидкости, температура всасываемого воздуха и так далее.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Йенер, М. Java EE. Паттерны проектирования для профессионалов / М. Йенер, А. Фидом. – СПб.: Питер, 2016. – 240 с.
2. Харди, Б. Android. Программирование для профессионалов / Б. Харди, Б. Филлипс, К. Стюарт, К. Марсикано. - СПб.: Питер, 2016. – 640.
3. Макграт, М. Создание приложений на Android для начинающих / М. Макграт. – М.: Эксмо, 2016. – 192 с.
4. Дейтел, П. Android для разработчиков / П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано. - СПб.: Питер, 2015. – 384 с.
5. Дарвин, Я. Ф. Android. Сборник рецептов. Задачи и решения для разработчиков приложений / Я. Ф. Дарвин. – М.: Вильямс, 2017. 768 с.
6. Ёранссон А. Эффективное использование потоков в операционной системе Android. Технологии асинхронной обработки данных / А. Ёранссон. – М.: ДМК Пресс, 2017. 304 с.
7. Лонг, Ф. Руководство для программиста на Java: 75 рекомендаций по написанию надёжных и защищённых программ / Лонг Ф. – М.: Вильямс, 2014. – 256 с.
8. Гонсалвес, Э. Изучаем Java EE 7 / Э. Гонсалвес. - СПб.:Питер, 2016.– 640.
9. Клифтон, Ян. Проектирование пользовательского интерфейса в Android / Я. Клифтон. – М.:ДМК Пресс, 2017. 452 с.
10. Голощапов, А. Google Android: системные компоненты и сетевые коммуникации / М.: БХВ-Петербург, 2014. 384 с.
11. Машнин, Т. С. Google App Engine Java и Google Web Toolkit: разработка Web-приложений / Т. С. Машнин. - М.: БХВ-Петербург, 2013. – 352 с.
12. Сеттер, Р.В. Изучаем Java на примерах и задачах / Р.В. Сеттер. - СПб.: Наука и техника, 2016. – 240 с.

13. Вязовик, Н. А. Программирование на Java / Н. А. Вязовик. - 2-е изд., - М.: Интуит, 2016. – 600 с.

14. Daum, B. System Architecture with XML / Berthold Daum, Udo Merten, 2016. – 458 p.

15. Griffiths, D. Head First Android Development / Dawn Griffiths, David Griffiths. - O'Reilly Media, 2015. – 734 p.

16. Delessio, C. Hello, Android Application Development in 24 Hours, Sams Teach Yourself / Carmen Delessio. - Sams Publishing, 2013. – 448 p.

17. Chan, J. Java: Learn Java in One Day and Learn It Well. Java for Beginners with Hands-on Project / Jamie Chan. - LCF Publishing. 2016. –236 p.

18. Oaks, S. Java Performance: The Definitive Guide: Getting the Most Out of Your Code 1st Edition / Scott Oaks. - O'Reilly Media, 2014. – 426 p.

19. Jinseong K. Efficient Protection of Android Applications through User Authentication Using Peripheral Devices, 2018.

20. Lindholm T., Yellin F., Bracha G., Buckley A. Inside Java :The Java Virtual Machine, 2015.

ПРИЛОЖЕНИЕ А
Часть кода TroubleCodesActivity.java

```
public boolean handleMessage(Message msg) {
        Log.d(TAG, "Message received on handler");
        switch (msg.what) {
          case NO_BLUETOOTH_DEVICE_SELECTED:
            makeToast(getString(R.string.text_bluetooth_nodevice));
            finish();
            break;
          case CANNOT_CONNECT_TO_DEVICE:
            makeToast(getString(R.string.text_bluetooth_error_connecting));
            finish();
            break;

 case OBD_COMMAND_FAILURE:
            makeToast(getString(R.string.text_obd_command_failure));
            finish();
            break;
          case OBD_COMMAND_FAILURE_IO:
            makeToast(getString(R.string.text_obd_command_failure)  + " IO");
            finish();
            break;
          case OBD_COMMAND_FAILURE_IE:
            makeToast(getString(R.string.text_obd_command_failure)  + " IE");
            finish();
            break;
          case OBD_COMMAND_FAILURE_MIS:
            makeToast(getString(R.string.text_obd_command_failure)  + " MIS");
            finish();
            break;
          case OBD_COMMAND_FAILURE_UTC:
            makeToast(getString(R.string.text_obd_command_failure)  + "
UTC");
            finish();
            break;
          case OBD_COMMAND_FAILURE_NODATA:
            makeToastLong(getString(R.string.text_noerrors));
            //finish();
            break;

          case NO_DATA:
            makeToast(getString(R.string.text_dtc_no_data));
            ///finish();
            break;
```

```
            case DATA_OK:
                dataOk((String) msg.obj);
                break;

        }
        return false;
}
```