МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети

БАКАЛАВРСКАЯ РАБОТА

на тему

Программа мониторинга активности пользователя мобильного устройства

Студент 03.06.2020 Боровов М.С.
(подпись, дата)

Руководитель 03.06.2020 Митрохин М.А.
(подпись, дата)

Нормоконтролёр А.В. Кучин
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 03.06.2020 г. № 12)

Заведующий кафедрой М.А. Митрохин
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 22.06.20 № 10

Секретарь ГЭК Е.И. Гурин
(подпись)

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«30» апреля 2020 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Боровову Максиму Сергеевичу группы 16ВВ2 факультета вычислительной техники.

2.Тема работы «Программа мониторинга активности пользователя мобильного устройства».

Тема утверждена приказом ПГУ № 0335а/о от « 30 » апреля 2020 г.

3. Руководитель бакалаврской работы д.т.н., доцент кафедры «ВТ» Митрохин М.А

4. Задание на работу (назначение разработки, исходные данные и т.п.).
Назначение разработки - программа мониторинга активности пользователя мобильного устройства.

5. Перечень подлежащих разработке вопросов.
Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать языки программирования для приложения, выбрать способ реализации мониторинга активности.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области			
2.	Выбор архитектуры приложения, языков программирования, способа взаимодействия			
3.	Разработка мониторинга активности пользователя			
4.	Разработка визуальной части приложения			
5.	Тестирование и отладка			
6.	Оформления выпускной работы			

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы _____ 30.04.20 Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент _____ 30.04.20 Боровов М.С.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 72 страницы формата А4, 28 рисунка, 7 источника, 2 приложения.

Объект разработки – программа мониторинга активности пользователя мобильного устройства.

ПРОГРАММА МОНИТОРИНГА АКТИВНОСТИ ПОЛЬЗОВАТЕЛЯ МОБИЛЬНОГО УСТРОЙСТВА, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, JAVA, SDK, GRADLE, XML, ANDROID STUDIO.

Целью выпускной квалификационной работы является разработка мобильного приложения для мониторинга активности пользователя мобильного устройства. Разработанное приложение выполняет функцию по сбору информации с датчиков мобильного устройства. Это достигается путем обращения программы к операционной системе и сбора данных устройства в процессе его эксплуатации.

В результате проведенной работы создан проект, протестированы программные средства, удовлетворяющие поставленной задаче.

В пояснительной записке приведен обзор аналогов разрабатываемого продукта, технологии разработки, предметной области, также была проведена отладка программы.

Графическая часть содержит UML-диаграммы и схемы программы.

Содержание

РЕФЕРАТ 2
ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ 5
ВВЕДЕНИЕ 6
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 8
1.1 Описание предметной области 8
1.2 Обзор существующих решений 8
1.2.1 EXA Tools 8
1.2.2 Sensor Box for Android 9
1.2.3 Датчикер 10
1.3 Актуальность выбранной темы 11
1.4 Выбор средств разработки 12
1.4.1 Операционная система Android 13
1.4.2 Средство разработки мобильных приложений SDK 14
1.4.3 Формат исполняемого файла APK 14
1.4.4 Система автоматической сборки Gradle 15
1.4.5 Среда разработки Android 16
1.4.6 Язык программирования Java 17
1.4.7 Язык разметки XML 18
2 ОПИСАНИЕ ТРЕБОВАНИЙ К СИСТЕМЕ 19
2.1 Требования к надежности 19
2.2 Требования к пользователю 19
2.3 Требования к информационной и программной совместимости 19
3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 20
4 РЕАЛИЗАЦИЯ ПРОЕКТА 23
4.1 Процесс разработки 23
4.1.1 Реализация фонового мониторинга 23
4.1.2 Реализация вывода показаний 26
4.1.3 Реализация сохранения данных 27
4.1.4 Реализация интерфейса 29
4.2 Отладка и тестирование 31
5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 34
6 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ 34

ЗАКЛЮЧЕНИЕ 39
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 40
ПРИЛОЖЕНИЕ А.....41
ЛИСТИНГ ПРОГРАММЫ.....41
Приложение А.1 Перечень классов программы.....42
Приложение А.2 Файлы разметки 54
ПРИЛОЖЕНИЕ Б - ПРЕЗЕНТАЦИЯ 57

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

ВКР – Выпускная квалификационная работа.

ИС – Информационная система.

ПО – Программное обеспечение.

ОС – Операционная система.

ПКМ – Правая кнопка мыши.

ЛКМ – Левая кнопка мыши.

ПИ – Пользовательский интерфейс

MVVM – model view view model.

SDK – software development kit.

ВВЕДЕНИЕ

В современном мире практически любые поставленные задачи в той или иной степени поддаются автоматизации. Самым показательным примером будет использование, к примеру, роботов на конвейере в сборочных целях, вместо обычного ручного труда. Это колоссально повышает производительность и эффективность труда, а также позволяет людям сосредоточится на более важных задачах, вместо выполнения повторимых и рутинных задач.

Помимо вышеописанного, огромное количество современных и повседневных вещей переходят в так называемый «цифровой формат». Электронные книги, вместо обычных бумажных, электронные деньги, заменяющие наличные, электронные письма, как замена обычным конвертам и т.д.

По отдельности, каждая из этих тенденций позволяют увеличить эффективность труда, качество жизни, комфорт и доступность многих вещей, которые раньше не были так доступны как сейчас. Тогда как за счет конвергенции этих областей создаются по-настоящему меняющие привычный образ жизни людей.

Смартфон и интернет, наверное, самые яркие примеры соединения областей автоматизации и цифровых технологий. Они настолько прочно и стремительно вошли в нашу жизнь, что теперь могут заменять огромное количество совершенно разных вещей одним устройством.

Огромное количество датчиков, сенсоров, ежесекундно собирающих тонны информации, устройства, позволяющие сделать использование современных гаджетов более удобным. На основе собранной информации устройство имеет возможность идентифицировать пользователя и подстроиться под его нужды и требования по эксплуатации.

Все более очевидным становится то, что только развитие всех сопутствующих сфер информационных технологий поможет избегать множество проблем на пути к удобной эксплуатации гаджетов и устройств

способных удовлетворить все слои общества.

Целью данной выпускной квалификационной работы является разработка мобильного приложения, предназначенного для сбора и обработки поступающей информации с датчиков ОС Android. Сбор данных будет производиться в фоновом режиме с возможностью уведомления пользователя о текущем процессе. Также объём получаемых значений можно изучить в специальном разделе приложения.

Собранная информация, в дальнейшем, будет необходима для обучения алгоритма идентификации пользователя и обеспечения безопасности устройства. Безопасность будет обеспечиваться путём разграничения функционала и хранимой информации устройства между конечными пользователями. Идентификация должна происходить в режиме реального времени, без непосредственного вмешательства пользователя в процесс. Для выявления перечня пользователей необходимо иметь реализованный функционал идентификации клиентов текущей системы. На основе предъявленных требований, в данной выпускной квалификационной работе необходимо реализовать возможность отслеживания и накопления показаний датчиков мобильного устройства пользователя.

Для достижения поставленной цели был сформирован список задач:

- анализ предметной области;
- постановка задачи на разработку программного средства для мониторинга активности пользователя мобильного устройства;
- выбор средств и технологий разработки;
- проектирование пользовательского интерфейса;
- разработка архитектуры;
- тестирование программы.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Описание предметной области

Разрабатываемое программное обеспечение предназначено для сбора информации с датчиков и наблюдения за активностью использования мобильного устройства пользователем. Доступ к системным датчикам организуется при помощи обращения главного процесса программы к ОС, она может достаточно точно измерить запрашиваемые данные. После получения ответа от системы программный продукт организует отображение считанной информации.

Также программа реализует функционал по фоновому мониторингу активности сенсоров. Данное программное решение позволяет обрабатывать и сохранять поток входящей информации непрерывно, независимо от состояния основного приложения.

1.2 Обзор существующих решений

На сегодняшний момент существует несколько приложений, позволяющих пользователям производить мониторинг активности устройства и отслеживать показания системных датчиков. В данном разделе будут рассмотрены наиболее актуальные и популярные приложения в роли аналогов разрабатываемой программы.

Так как разрабатываемое приложение предназначено для развёртывания в операционной системе Android, то были выбраны соответствующие аналоги.

1.2.1 EXA Tools

Наиболее схожим аналогом является приложение «Датчики измерительные» от компании «EXA Tools». На рисунке 1 изображён вариант использования данного приложения в качестве аналога проектируемого приложения.

На данном изображении отображается текущее значение основных системных датчиков Android устройства, таких как акселерометр, значение

координат относительно геомагнитного поля, значение ориентации в пространстве, датчик гироскопа и освещённости.

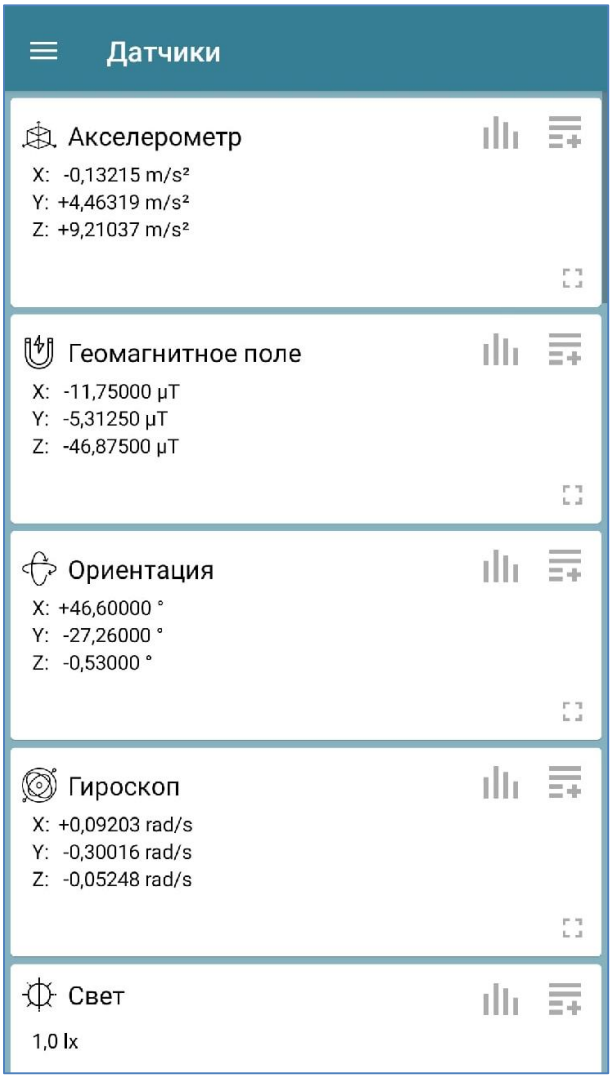


Рисунок 1 – Функционал «EXA Tools»

1.2.2 Sensor Box for Android

Следующим аналогом выступает приложение «Sensor Box for Android» от компании «HK smarter mobi tech». В данном приложении отображается текущее положение устройства тремя координатами, соответствующими трём осям в трёхмерном пространстве и имитацией шара, находящегося на ровной поверхности экрана. Модель шара реагирует на смещение в зависимости от наклона устройства.

Работу описанного аналога наглядно отображает рисунок 2.

Рисунок 2 – Функционал «Sensor Box for Android»

1.2.3 Датчикер

Приложение «Датчикер – все датчики Android» от компании «moon dance». На экран этого приложения выводится информация о текущей потребляемой электроэнергии, значения положения устройства по трём осям и прочие датчики. В данном приложении наблюдается схожий функционал по отслеживаемой информации. В сравнении с разрабатываемым приложением у приложения «Датчикер» нет возможности отслеживать показания сенсоров в фоновом режиме без участия пользователя, что выгодно выделяет разрабатываемую программу на фоне остальных аналогов данного класса.

Пример эксплуатации приложения - аналога изображён на рисунке 3.

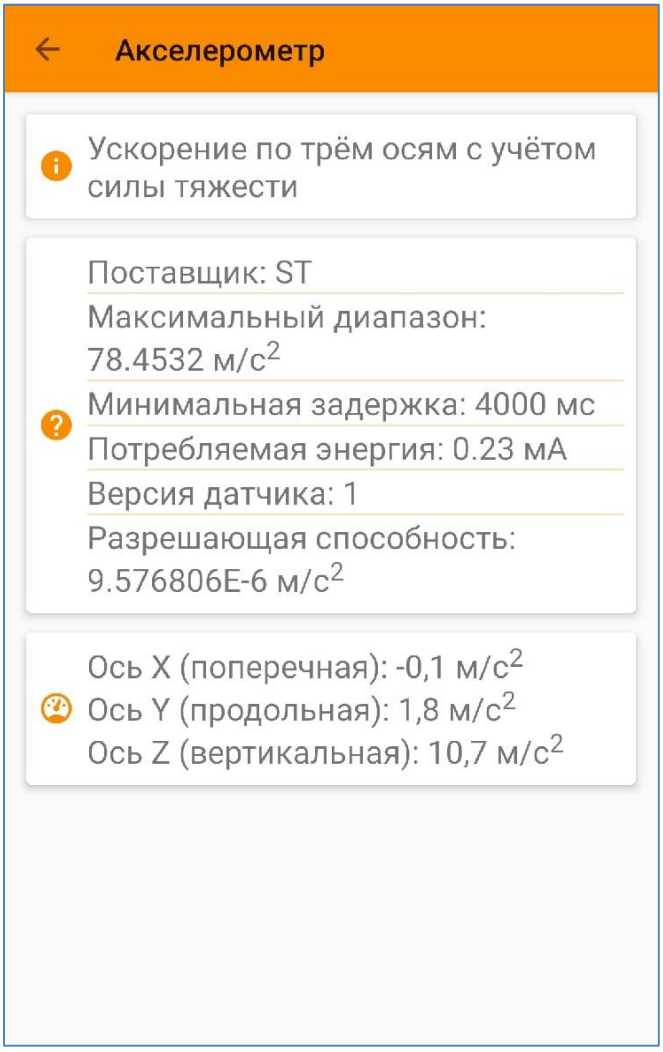


Рисунок 3 – Функционал «Датчикер»

1.3 Актуальность выбранной темы

Исходя из информации, представленной в разделе «Обзор существующих решений», мы наблюдаем, что даже самые широкие по охвату аудитории приложения обладают недостатками.

Все найденные аналоги не обладают функцией фонового сбора данных активности пользователя. Это выгодно выделяет разрабатываемое приложение на фоне потенциальных конкурентов в данной сфере. Также в текущем проекте имеется широкий спектр направлений по усовершенствованию и доработке программного продукта до требуемого уровня. Расширение числа анализируемых датчиков приведёт к широкому спектру анализируемой информации. На основе собранных данных имеется возможность организовать

разграниченный доступ к хранимой информации и функционалу устройства, что, в свою очередь, существенно повысит уровень безопасности эксплуатации системы.

1.4 Выбор средств разработки

В ходе разработки были проанализированы различные средства, технологии и языки программирования.

Программные средства и технологии, использованные в данной работе:

- Операционная система Android;
- Средство разработки мобильных приложений SDK;
- Формат исполняемого архива виртуальной машиной APK;
- Система автоматической сборки Cradle;
- Среда разработки Android Studio.

Использованные языки программирования и языки разметки:

- Язык программирования Java;
- Язык разметки XML;

Выбор средств разработки исходил из функциональных возможностей, высокой производительности и бесплатного использования.

1.4.1 Операционная система Android

Android - это мобильная операционная система, основанная на модифицированной версии ядра Linux и другого программного обеспечения с открытым исходным кодом, предназначение которой является обеспечение функционалом мобильных устройств с сенсорным экраном, таких как смартфоны и планшетные устройства. Android разработан консорциумом разработчиков, известным как Open Handset Alliance, основным спонсором и коммерческим маркетологом которого является Google.

Разработка под данную операционную систему имеет ряд неоспоримых преимуществ. Исходный код Android использовался в качестве основы для

различных экосистем, в частности Google, который связан с набором проприетарного программного обеспечения под названием Google Mobile Services (GMS), который часто поставляется предварительно установленным на указанных устройствах. Это включает в себя основные приложения, такие как Gmail, в цифровой дистрибуции платформы Google Play и связанные с Google Play Services платформы разработки, и, как правило, приложения, такие как Google Chrome веб - браузер. Эти приложения лицензируются производителями устройств Android, сертифицированных в соответствии со стандартами, установленными Google. Распространение программного обеспечения обычно осуществляется через фирменные магазины приложений, такие как Google Play Store или Samsung Galaxy Store .

Android имеет растущий список сторонних приложений, которые могут быть приобретены пользователями, загрузив и установив файл APK приложения, или установив их с при помощьи магазина приложений, который позволяет конечным клиентам устанавливать, обновлять и удалять приложения с их устройств. Google Play Store - это основной магазин приложений, установленный на устройствах Android, которые отвечают требованиям совместимости Google и лицензируют программное обеспечение Google Mobile Services. Google Play Store позволяет пользователям просматривать, загружать приложения, опубликованные Google и сторонними разработчиками; по состоянию на май 2013 года в Play Store доступно более миллиона приложений для Android. По состоянию на июль 2012 года было установлено 45 миллиардов приложений. По состоянию на май 2016 года более одного миллиарда активных пользователей в месяц используют Gmail, Android, Chrome, Google Play и Карты.

Все вышеописанные факторы непосредственно виляют на выбор исходной операционной системы для разработки.

1.4.2 Средство разработки мобильных приложений SDK

Android SDK — универсальное средство разработки мобильных приложений для операционной системы Android. Главным отличием от обычных редакторов для написания кодов является наличие широких возможностей функционала, позволяющих запускать тестирование и отладку исходных кодов, оценивать работу приложения в режиме совместимости с различными версиями ОС Android и наблюдать результат в реальном времени. Охватывает большой объём мобильных устройств, среди которых выделяют: смартфоны, планшеты, телевизоры с расширенным функционалом, современные автомобили с бортовыми компьютерами на базе операционной системы Android, наручные часы и многие другие гаджеты и устройства.

В состав SDK возможно включение фрагментов устаревших версий платформы Android для случаев, когда разработчики готовы заниматься продолжением развития своих приложений для устаревших телефонных аппаратов и планшетов. Часть средств разработки поставляется в виде отдельно загружаемых дополнений.

Сформированные приложения для Android, в законченном виде, представляют собой пакеты формата «.apk».

1.4.3 Формат исполняемого файла APK

APK (англ. Android Package) — формат архивных исполняемых приложений для ОС Android. Каждое приложение Android скомпилировано и упаковано в единый файл, который включает в себя весь исходный код приложения, ресурсы, материалы, файл манифеста и подключенные библиотеки. Файл приложения может иметь любое имя, но расширение должно быть «.apk».

Файлы с описанным расширением располагаются в магазинах приложений, и загружаются с помощью данной библиотеки в смартфон либо планшетный компьютер для их использования, или устанавливаются

пользователем вручную.

Файлы этого формата не шифруются и являются подмножеством формата архива ZIP. Поэтому каждый файл с расширением «.apk» — это сжатый архив для исполнения в виртуальной машине, который может быть установлен не только на операционной системе Android.

1.4.4 Система автоматической сборки Gradle

Gradle — система автоматической сборки, построенная на принципах Ant и Maven, но предоставляющая DSL на языках Groovy и Kotlin вместо традиционной XML-образной формы представления конфигурации проекта.

Gradle был разработан для расширяемых многомодульных сборок, и поддерживает инкрементальные сборки, определяя, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска.

Основные плагины предназначены для разработки и развертывания Java, Groovy и Scala приложений, но готовятся плагины и для других языков программирования.

В отличие от Maven, основанного на концепции жизненного цикла проекта, и Ant, в котором последовательность выполнения заданий определяется взаимоотношениями зависимости, Gradle использует направленный ациклический граф для определения порядка выполнения задач.

Для приложений, состоящих из модулей, удобно использовать мультипроектный подход Gradle. Каждый модуль расположен в своей директории. Каждый модуль может включать дочерние, которые будут расположены во вложенных директориях. Описание мультипроектной структуры выполняется в settings.gradle в корневой директории проекта. В корневой директории проекта есть build.gradle который может настраивать себя и дочерние модули. Каждый данный модуль может иметь такие же два файла для настройки себя и своих дочерних модулях. Такой подход позволяет удобно запускать задачи для конкретных модулей или всего проекта в целом.

1.4.5 Среда разработки Android

Android Studio — интегрированная среда разработки (IDE) для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O.

Данная IDE находилась в свободном доступе начиная с версии 0.1, опубликованной в мае 2013, а затем перешла в стадию бета-тестирования, начиная с версии 0.8, которая была выпущена в июне 2014 года. Первая стабильная версия 1.0 была выпущена в декабре 2014 года, тогда же прекратилась поддержка плагина Android Development Tools (ADT) для Eclipse.

Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, — официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, macOS и Linux. 17 мая 2017, на ежегодной конференции Google I/O, Google анонсировал поддержку языка Kotlin, используемого в Android Studio, как официального языка программирования для платформы Android в добавление к Java и C++.

С каждой версией функционал Android Studio растет и улучшается, позволяя упрощать разработку приложений для Android. На данный момент доступно огромное количество функций, например, следующие:

- Различные виды сборок и генерация нескольких «.apk» файлов
- Статический анализатор кода, позволяющий находить проблемы производительности, несовместимости версий и другое.
- Встроенный ProGuard и утилита для подписывания приложений.
- Расширенный редактор макетов: WYSIWYG, способность работать с UI компонентами при помощи Drag-and-Drop, функция предпросмотра макета на нескольких конфигурациях экрана.
- Сборка приложений, основанная на Gradle.

1.4.6 Язык программирования Java

В современном мире Java является одним из самых известных и используемых объектно-ориентированных языков программирования. Данный ЯП разрабатывался как универсальный для решения различного горизонта задач. Java прошел огромный путь развития, в текущий момент является уже не просто ЯП, а целой платформой, которая объединяет огромное количество различных технологий, библиотек и т.д. Все это позволяет вести разработку на Java серьезных enterprise приложений, а большое количество дополнительных модулей дает возможность подобрать конкретные средства для решения поставленной задачи. Многие разработчики выбирают Java, в результате чего растет количество различных решений для возникающих вопросов, а также увеличивается объем литературы, доступной для изучения.

К особенностям Java относится:

– Одной из главных особенностей языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. Затем происходит выполнения данного кода специальной виртуальной машиной Java;

– Архитектура языка обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, из этого следует, что программам не нужно перекомпилироваться, чтобы быть используемыми на различных платформах, таких как: Windows, Linux, Mac Os и других;

– Java является языком с Си-подобным синтаксисом и близок в этом отношении к C/C++ и C#. Данная особенность позволяет разработчикам, уже имеющих опыт работы с этими языками, быстро приступить к разработке приложений на Java;

– В Java присутствует автоматический сборщик мусора, это позволяет разработчику не уделять времени на ручную очистку памяти от ранее использованных объектов, что значительно упрощает и ускоряет работу.

1.4.7 Язык разметки XML

Одной из наиболее привлекательных особенностей платформы Android является использование языка разметки XML. SDK Android поддерживает достаточно большую часть возможностей стандартной среды выполнения Java и средств разметки XML.

Java платформа долгое время поддерживает широкий спектр различных способов использования XML, причем большинство API для Java, предназначенных для XML, доступны в Android. Наглядным примером такого API может служить объектная модель документов (Document Object Model — DOM) или простой Java API для XML, которые уже давно являются частью технологии Java. Обратным примером стал новый потоковый API (Streaming API for XML – StAX), который ранее не имел поддержку в Android (при этом в состав Android входит аналогичная по требуемым возможностям библиотека дополнений). В Android также не реализован API для связывания с содержимым XML. Его безусловно можно реализовать для описанной платформы, однако он отличается некоторой тяжеловесностью, которая проявляется в том, что множество представлений разных классов часто требуется для визуализирования документов XML. В связи с этим он является менее предпочтительным при создании приложений для портативных устройств, подобных тем, на которых установлена ОС Android.

Специализация XML описывает язык и ряд вопросов, касающихся обработки и кодировки документов. Содержимое этого раздела представляет собой краткую интерпретацию описания языка в спецификации XML, адаптированное для настоящей статьи.

Нормативным считается англоязычное содержание документа, поэтому основными терминами приводятся английские оригиналы.

2 ОПИСАНИЕ ТРЕБОВАНИЙ К СИСТЕМЕ

2.1 Требования к надежности

Основными требованиями к надёжности системы являются: бесперебойное функционирование системы и защита от возможных ошибок эксплуатации.

В системе должен быть реализован контроль над организацией запросов к операционной системе. Снятие показаний с датчиков в каждой версии ОС устройства имеет перечень особенностей. Необходимо организовать обработку и перехват возможных исключений.

2.2 Требования к пользователю

Данной информационной системой может пользоваться даже не уверенный пользователь устройств смартфонов, так как у системы простой и понятный интерфейс, простота в эксплуатации и их использовании. Система предназначена для широкого спектра пользователей.

2.3 Требования к информационной и программной совместимости

Приложение для мониторинга активности должно быть реализована в среде разработки Android Studio на языке программирования Java.

Рабочие станции пользователей должны иметь:

– Установленную операционную систему Android не ниже версии 5.1;

– Сервисы Google Play.

3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

В ходе реализации Android приложения, изначально была выбрана архитектура, на основе которой будет строить приложение. Грамотно выбранная архитектура позволит в дальнейшем «безболезненно» вносить изменения, добавляя или улучшая функционал.

В данной работе будет использоваться архитектура MVVM (Model-View-ViewModel). Его концепция заключается в отделении логики представления данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента изображённых на рисунке 4.

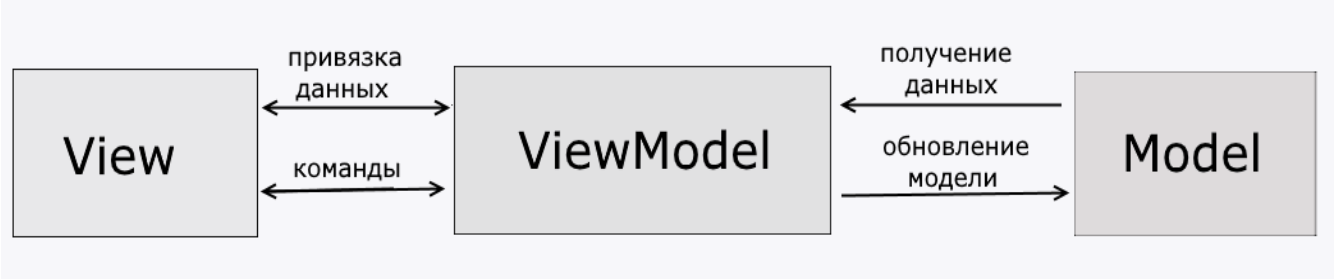


Рисунок 4 – Схема MVVM архитектуры приложения

Паттерн MVVM (Model-View-ViewModel) позволяет разраничить логику программы от части представления. Описанный паттерн представляется архитектурным, то есть он задает общую архитектуру приложения.

Текущий паттерн был представлен Джоном Госсманом (John Gossman) в 2005 году как совершенствование шаблона Presentation Model и был первоначально нацелен на разработку приложений в WPF.

Несмотря на то, что в настоящее время данный паттерн вышел за пределы WPF и применяется в самых разнообразных технологиях, в том числе при разработке под Android, iOS, тем не менее WPF является довольно яркой иллюстрацией технологии, которая раскрывает возможности данного паттерна.

MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

Model

Модель описывает используемые в программе данные. Модели могут инкапсулировать логику, непосредственно взаимосвязанную этими данными, например, логику валидации перечня свойств модели. В то же время модель не должна содержать никакой логики, связанной с визуальной составляющей данных и взаимодействием с визуальными элементами управления.

Зачастую модель воплощает интерфейс INotifyPropertyChanged или INotifyCollectionChanged, которые производят уведомления системы об изменениях перечня свойств модели. Благодаря этому улучшается прикрепление к представлению, хотя опять же прямое взаимодействие между моделью и представлением отсутствует.

View

View или представление определяет интерфейс пользователя, через который он производит взаимодействие с программой. Применительно к WPF представление - это код в xaml, который определяет визуальную часть в виде кнопок, текстовых полей и прочих визуальных элементов.

Хотя окно (класс Window) в WPF может инкапсулировать как интерфейс в xaml, так и привязанный к нему код, однако в идеале код не должен содержать какой-то логики, кроме разве что конструктора, который вызывает метод InitializeComponent и выполняет начальную инициализацию визуальной области. Вся же основная логика разработанного приложения вытесняется в функционал ViewModel.

В некоторых случаях в исходном файле прикреплённого кода все же может размещаться некоторая логика, которую трудно реализовать в пределах паттерна MVVM.

Представление не обрабатывает события за редким исключением, а выполняет действия в основном посредством команд.

ViewModel

ViewModel или модель представления связывает модель и представление через механизм привязки данных. Если в модели изменяются значения

свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

ViewModel также содержит логику по получению данных из модели, которые потом транслируются в часть визуализации. И также VewModel определяет логику по обновлению данных в модели.

Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с ViewModel посредством команд.

Итогом применения паттерна MVVM является функциональное разделение программы на три части, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать.

Что же это дает нам в конечном итоге?

- гибкость разработки;
- тестирование;
- разграничение логики.

При использовании описанного подхода, приложение будет иметь разграниченный функционал по файлам и классам. Каждый из подобных модулей будет нести ответственность за исполнение заложенного в нём функционала.

Таким образом, визуальная часть приложения будет реализована в специализированных файлах, а выполнение подключения и работа с операционной системой в отдельных классах.

На основе выбранной архитектуры были разработаны специализированные классы для каждого из окна приложения. Переход между данными областями реализован при помощи меню, хранящимся в классе, созданном для обработки переходов по его пунктам и обработки визуальной части выбора пунктов меню.

Каждый из множества классов обладает перечнем переменных и методов, которые, в свою очередь, были созданы для разграничения функционала и сохранения состояний выполнения программы. На рисунке 5 отображена диаграмма классов разрабатываемого приложения.

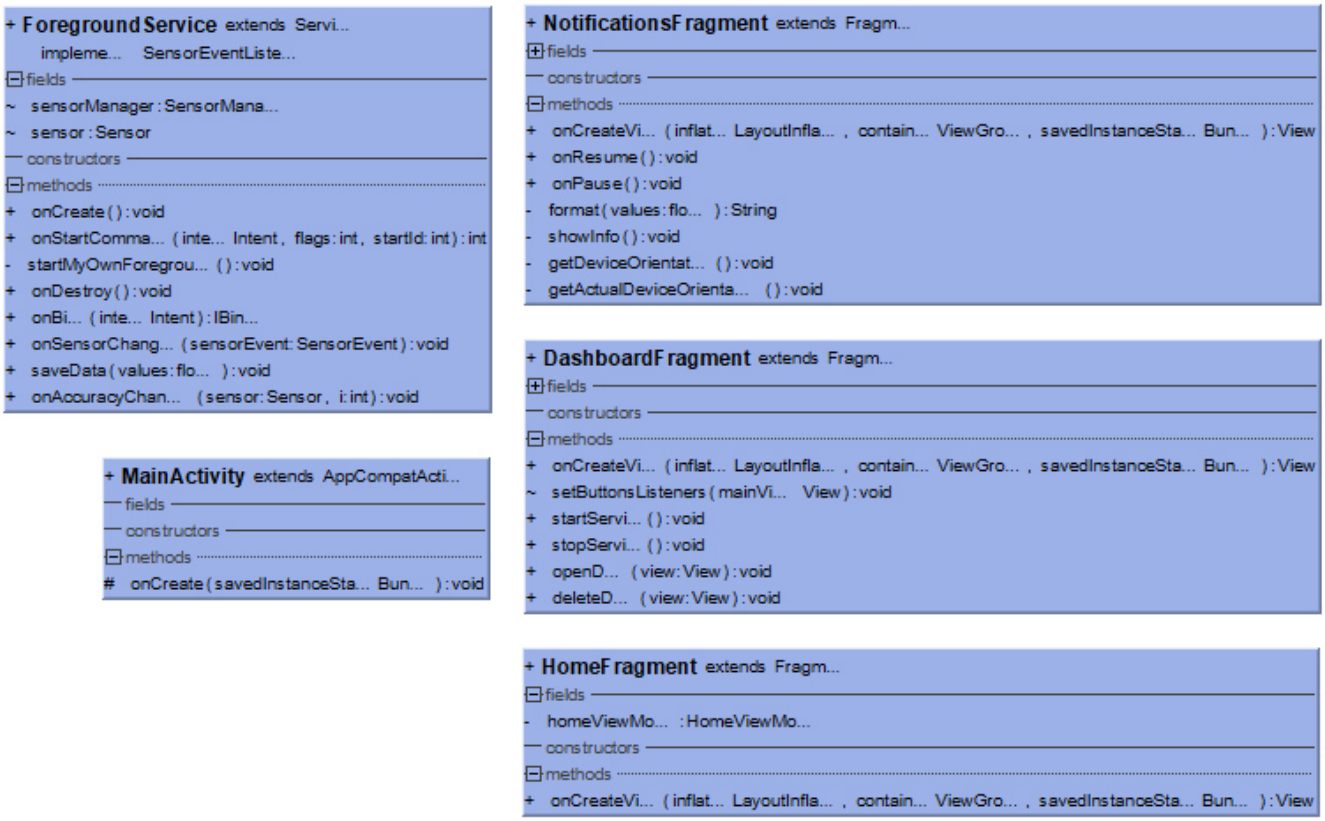


Рисунок 5 – Диаграмма классов

Класс «ForegrounService» был создан для выполнения задачи по сбору показаний датчиков в фоновом режиме. Данный класс реализует бесперебойную запись показаний сенсоров после закрытия приложения и до остановки текущего процесса.

Класс «MainActivity» является собирательным классом, запуск приложения начинается с выполнения этого класса, после инициируется запуск главного окна.

Класс «HomeFragment» отвечает за обработку и вывод визуальной части окна «главная».

Класс «NotificationsFragment» создан для обработки поступающих команд из окна «Службы». Также описанный класс исполняет функционал по

управлению фоновой службой и сбору опказаний сенсоров.

Исполнение класса «DashboadFragment» позволяет пользователю отслеживать текущие значения датчиков. Обновление значений происходит каждые 0.4 секунды, такой промежуток времени необходим для соблюдения баланса между используемыми ресурсами устройства и актуальностью данных.

4 РЕАЛИЗАЦИЯ ПРОЕКТА

4.1 Процесс разработки

4.1.1 Реализация фонового мониторинга датчиков

Основополагающим функционалом в разрабатываемом приложении является сбор показаний с датчиков в фоновом режиме.

Для выполнения поставленной задачи был разработан отдельных класс «ForegroundService.java». Указанный класс организует отправку запросов для регистрации фоновой службы и получения показаний с отслеживаемых сенсоров.

Исходя из требований google, для осуществления подобных решений требуется унаследовать класс «Service» и реализовать интерфейс «SensorEventListener». Конечным результатом является строго типизированная конструкция класса с необходимым перечнем методов для выполнения поставленной задачи.

Элементы интерфейса «Service» и «SensorEventListener» наглядно изображены на рисунке 6 и рисунке 7 соответственно.

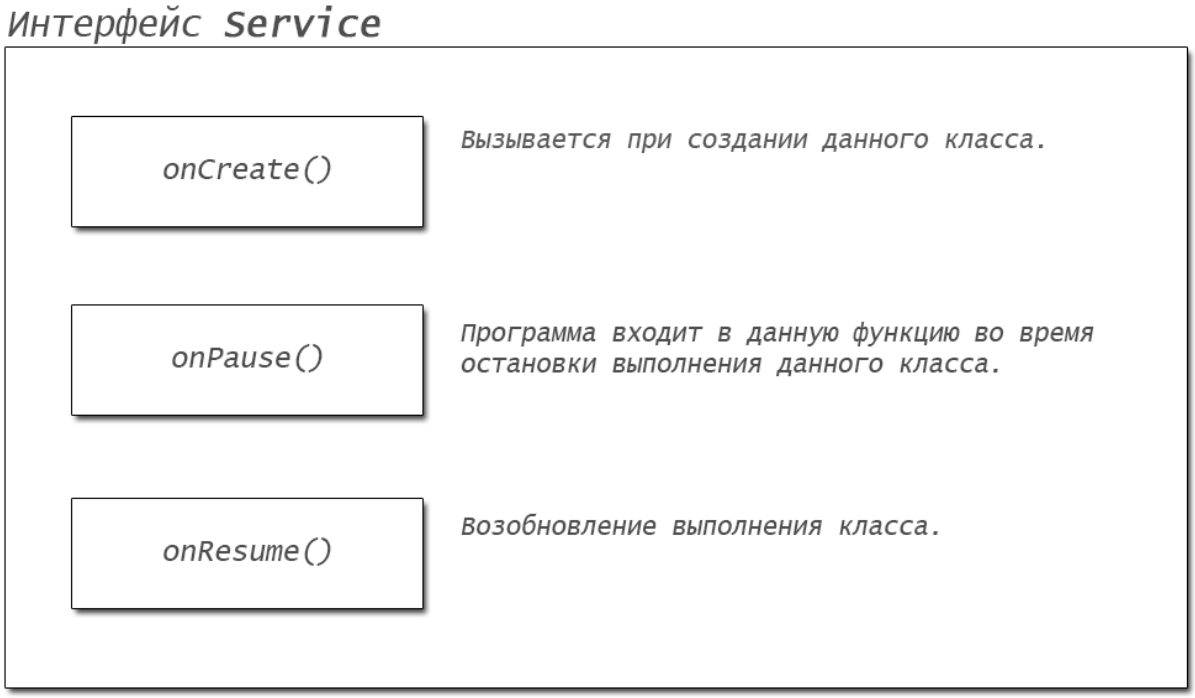


Рисунок 6 – Схема интерфейса «Service»

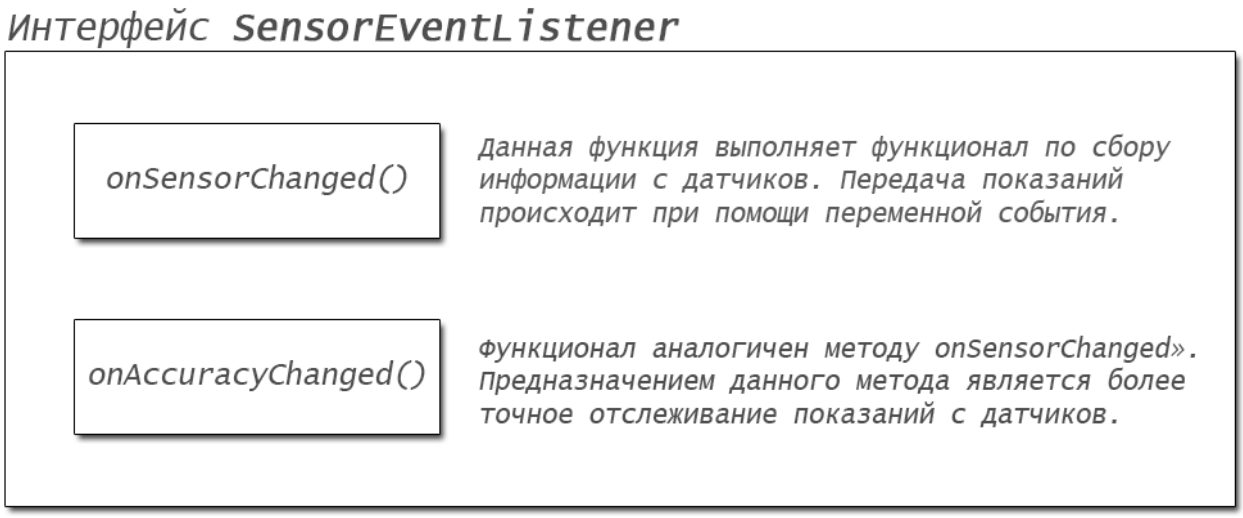


Рисунок 7 – Схема интерфейса «SensorEventListener»

Реализация представленных интерфейсов позволяет грамотно организовать доступ фонового сервиса к операционной системе.

Ещё одним обязательным требованием для мониторинга активности пользователя в фоновом режиме выступает уведомление пользователя о текущем процессе. Для выполнения данного требования была поставлена задача по разработке системного уведомления (notification). Реализацию данного критерия полно отображает следующий фрагмент кода:

```
String NOTIFICATION_CHANNEL_ID = "com.example.ident";
String channelName = "My Background Service";
NotificationChannel chan = new NotificationChannel(NOTIFICATION_CHANNEL_ID,
        channelName, NotificationManager.IMPORTANCE_NONE);
chan.setLightColor(Color.BLUE);
chan.setLockscreenVisibility(Notification.VISIBILITY_PRIVATE);
NotificationManager manager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
assert manager != null;
manager.createNotificationChannel(chan);

NotificationCompat.Builder notificationBuilder =
        new NotificationCompat.Builder(this, NOTIFICATION_CHANNEL_ID);
Notification notification = notificationBuilder.setOngoing(true)
        .setSmallIcon(R.drawable.arrow)
        .setContentTitle("Accelerometer service")
        .setPriority(NotificationManager.IMPORTANCE_MIN)
        .setCategory(Notification.CATEGORY_SERVICE)
        .build();

Toast.makeText(this, "Служба по сбору данных запущена.",
Toast.LENGTH_SHORT).show();
startForeground(2, notification);
```

На данной области кода происходит создание уведомления для оповещения о процессе сбора информации с устройства пользователя.

Экземпляр класса NotificationChannel устанавливает перечень настроек для реализации уведомления. NotificationManager осуществляет управление развёрнутым уведомлением. Переменная Notification является собирательной по отношению к данным вышеупомянутых классов. Она комбинирует ранее указанные параметры и информацию для запуска уведомления. Функция по запуску текущей фоновой службы и уведомления «startForeground()» является отправной точной выполнения фоновой службы.

Сразу после запуска фоновой службы мониторинга сенсоров устройства, программа отслеживает изменения показаний датчиков и сохраняет их в файл. Описанный функционал демонстрирует следующий участок кода:

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    if(sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
        saveData(sensorEvent.values);
}

public void saveData(float values[]) {
    FileOutputStream fos = null;
    try {
        fos = openFileOutput("sensor.txt", MODE_APPEND);
        fos.write((values[0]+" "+values[1]+" "+values[2]+"\n").getBytes());
    }
    catch(IOException ex) {
        Log.e("service", Objects.requireNonNull(ex.getMessage()));
    }
    finally {
        try {
            if (fos != null)
                fos.close();
        }
        catch (IOException ex) {
            Log.e("service", Objects.requireNonNull(ex.getMessage()));
        }
    }
}
```

Метод onSensorChanged() выполняет мониторинг датчиков и передачу полученных значений методу saveData(). Последний в свою очередь сохраняет принятые показания в файл «sensor.txt» директории запущенного приложения.

4.1.2 Реализация мониторинга датчиков и вывода показаний

В ходе проектирования приложения, было принято решение о создании раздела для отслеживания показаний сенсоров в реальном времени. В связи с поставленной задачей, в конечном продукте реализован мониторинг активности мобильного устройства пользователя в реальном времени.

Для описанных нужд был создан класс «NotificationsFragment.java», который, в свою очередь, реализует сбор данных и отображение полученных значений через пользовательский интерфейс. Следующий участок кода демонстрирует сбор показаний для мониторинга активности.

```
@Override
public void onSensorChanged(SensorEvent event) {
    switch (event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            for (int i=0; i < 3; i++){
                valuesAccel[i] = event.values[i];
            }
            break;
        case Sensor.TYPE_MAGNETIC_FIELD:
            for (int i=0; i < 3; i++){
                valuesMagnet[i] = event.values[i];
            }
            break;
        case Sensor.TYPE_LIGHT:
            for (int i=0; i < 3; i++){
                valuesLight[i] = event.values[i];
            }
        default:
            break;
    }
}
```

В приведённом методе onSensorChanged() происходит процесс сбора данных при помощи переменной event. В теле функции располагается оператор switch, при помощи которого организуется распределение поступающей информации с датчиков в соответствующие переменные для дальнейшего хранения и эксплуатации.

Для более полного описания протекающего процесса по сбору показаний с датчиков была спроектирована диаграмма взаимодействия программы и ОС устройства, рисунок 8.

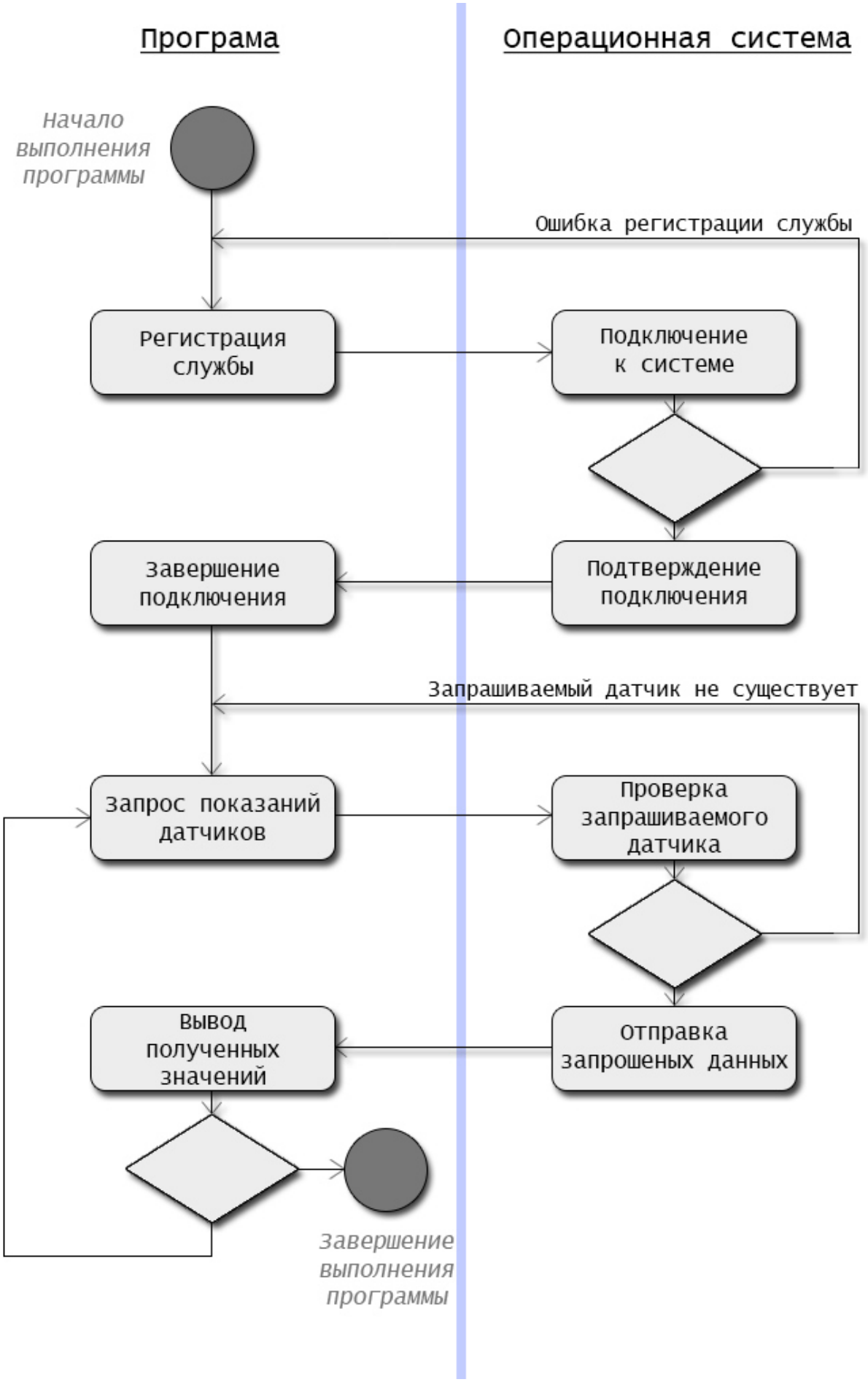


Рисунок 8 – UML диаграмма взаимодействия программы и ОС

4.1.3 Реализация сохранения данных

В процессе вычисления активности пользователя устройства генерируется большое количество информации. При сборе данных требуется реализация сохранения поступающего потока данных. Полученные показания накапливаются в файле «sensor.txt» и сохраняются в строковом формате.

В качестве наиболее важных показаний датчиков для сохранения были выбраны данные ориентации устройства в пространстве. В дальнейшем, на основе собранных данных программа может идентифицировать пользователя по манере использования устройства. Функционал идентификации пользователя позволяет разграничить сферы эксплуатации и использование хранимых данных устройства для обеспечения безопасности.

Сохранение данных реализовано при помощи класса «FileOutputStream». Записи в файл производятся на основе изменений состояния сенсоров. В момент изменения показаний датчиков устройства срабатывает условие о записи информации о текущем положении.

На рисунке 9 изображена диаграмма сохранения показаний сенсоров.

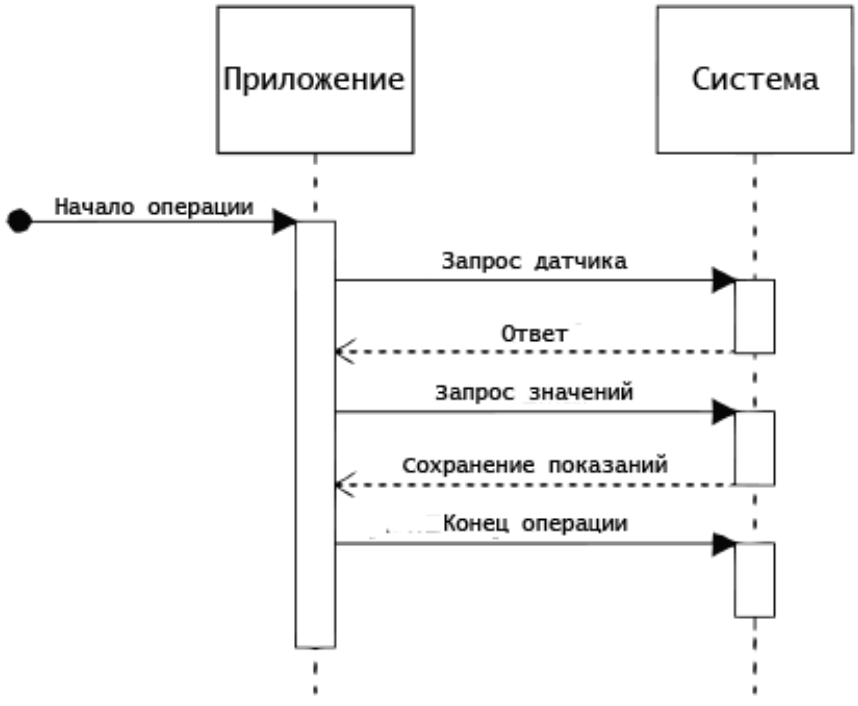


Рисунок 9 – Схема сохранение показаний основного датчика

4.1.4 Реализация интерфейса

В процессе разработки приложения были спроектированы следующие макеты экранов:

- главная;
- службы;
- датчики.

Распределение функционала приложения на отдельные экраны поможет лучше разграничить выполнение поставленных задач при разработке и при использовании программы.

Макет экрана «главная» является приветственным окном и открывается сразу после запуска приложения, рисунок 10.

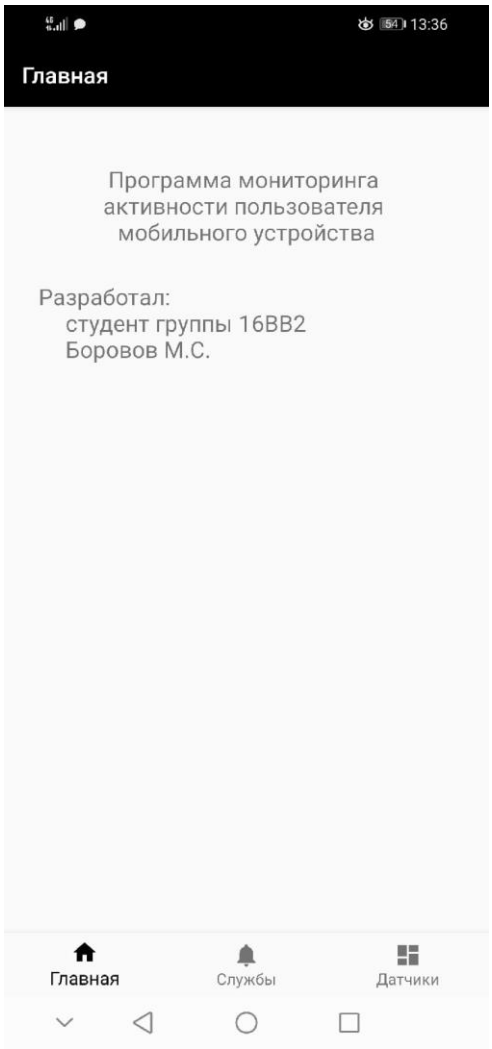


Рисунок 10 – Стартовое окно приложения

Задачей экрана «службы» является организация пользовательского интерфейса для управления фоновой службой, рисунок 11. В данном интерфейсе расположены кнопки для управления выполнением фоновой службы и для работы с полученной информацией.

Рисунок 11 – Окно управления службами

Визуальная часть экрана «датчики» содержит в себе список с показаниями сенсоров, рисунок 12. Визуальная составляющая разделена на строки с описанием каждого датчика и его основных характеристик.

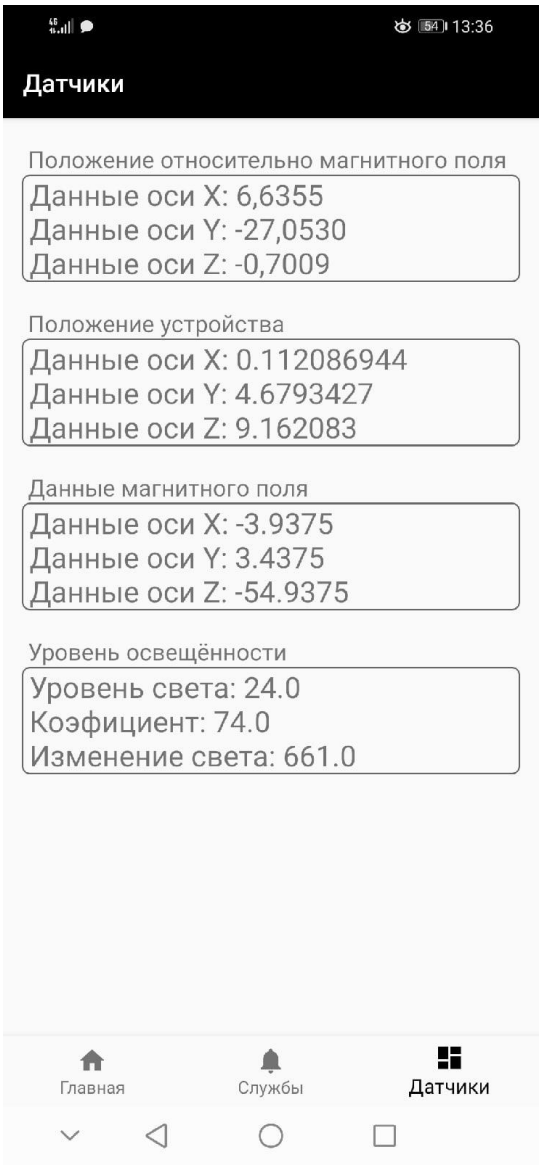


Рисунок 12 – Экран с показаниями датчиков

Наглядным отображением вариантов использования интерфейса приложения и иго содержимого выступает диаграмма активности пользователя, рисунок 13.

Рисунок 13 – Диаграмма вариантов использования приложения

4.2 Отладка и тестирование

Весь спектр видов тестирования программных продуктов, в зависимости от преследуемых целей, можно условно разделить на следующие группы:

- функциональные;
- нефункциональные;
- связанные с изменениями.

Функциональные виды тестирования

Функциональные тесты основываются на функциях и особенностях, а также взаимосвязи с иными системами, и могут быть воплощены на всех уровнях тестирования: компонентном или модульном, интеграционном, системном и приемочном. Функциональные виды тестирования рассматривают внешнее поведение рассматривание системы.

Нефункциональные виды тестирования

Нефункциональное тестирование описывает тесты, востребованные для индикации характеристик программного обеспечения, которые могут быть измерены самыми разнообразными значениями. В целом, это тестирование того, "Как" система работает.

Связанные с изменениями виды тестирования

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть пере тестировано для подтверждения того факта, что проблема была действительно решена.

Данная программа написана и отлажена с использованием среды программирования Android Studio.

Отладка проводилась с помощью таких инструментов, как точка останова, выполнение кода по шагам, анализа содержимого переменных.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с синтаксисом конструируемых в программе системных запросов.

Основной областью возникновения исключений, до процесса тестирования, являлся функционал по сбору показаний сенсоров. При помощи точки остановы была произведена отладка и устранение возникающих ошибок.

Отслеживание транспортировки данных датчиков устройства наглядно отображено на рисунке 14.

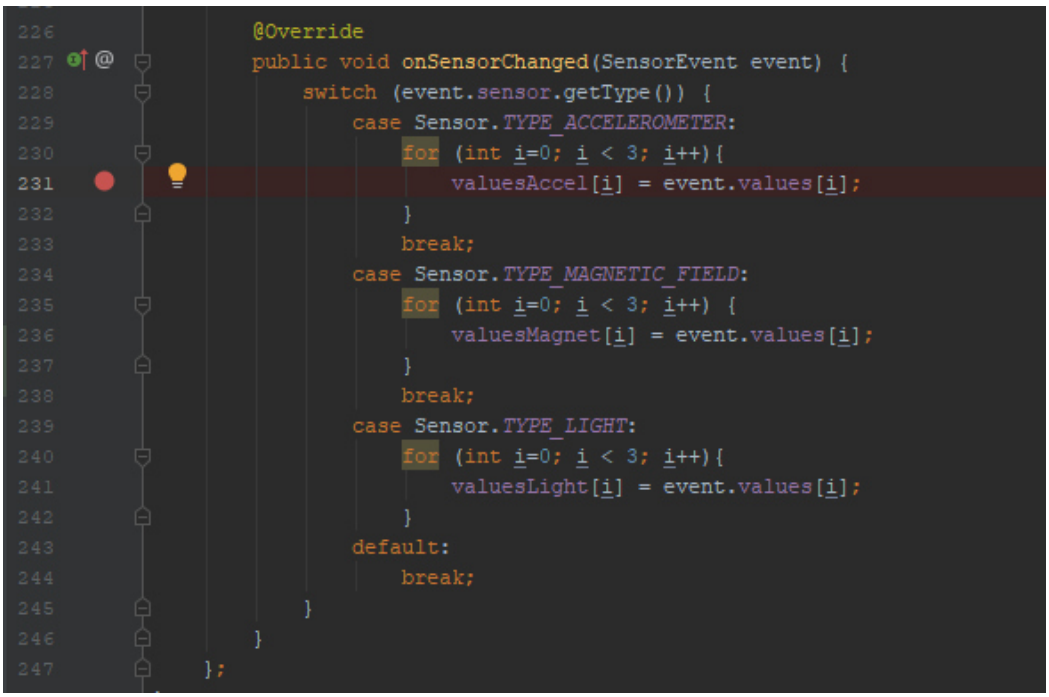


Рисунок 14 – Точка остановы

На приведённом изображении производится процесс отладки. Точка остановы расположена на моменте передачи полученных значений сенсоров вспомогательным переменным программы. Исходные значения расположены в переменной event класса SensorEvent. На рисунке 15 отображено тело экземпляра данного класса.

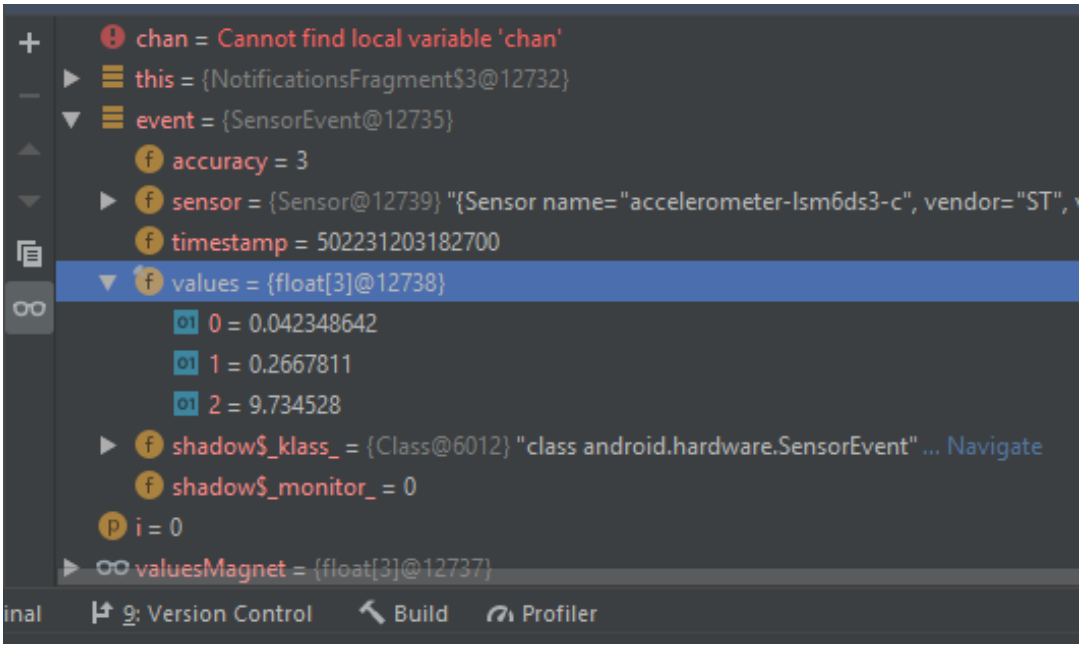


Рисунок 15 – Отладка показаний датчиков

На рисунке 16 отображён процесс передачи показаний из системной переменной в переменную программы и дальнейшее сохранение значений.

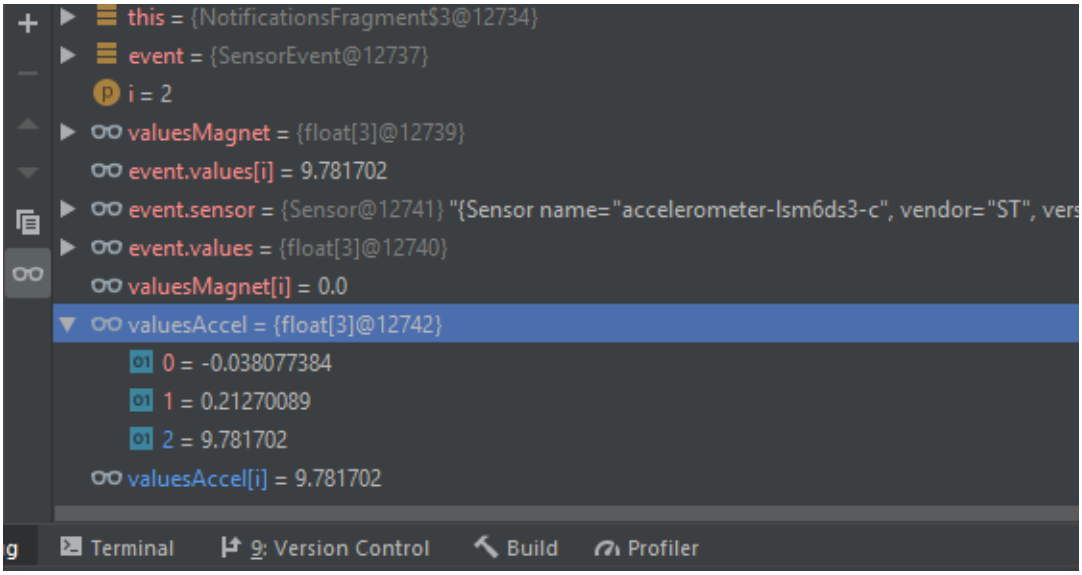


Рисунок 16 – Сохранение данных в программе

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Работа приложения начинается с открытия окна приветствия. Данное окно служит для ознакомления с данным приложением и не содержит функционала по выполнению поставленных задач мониторинга активности, рисунок 17.

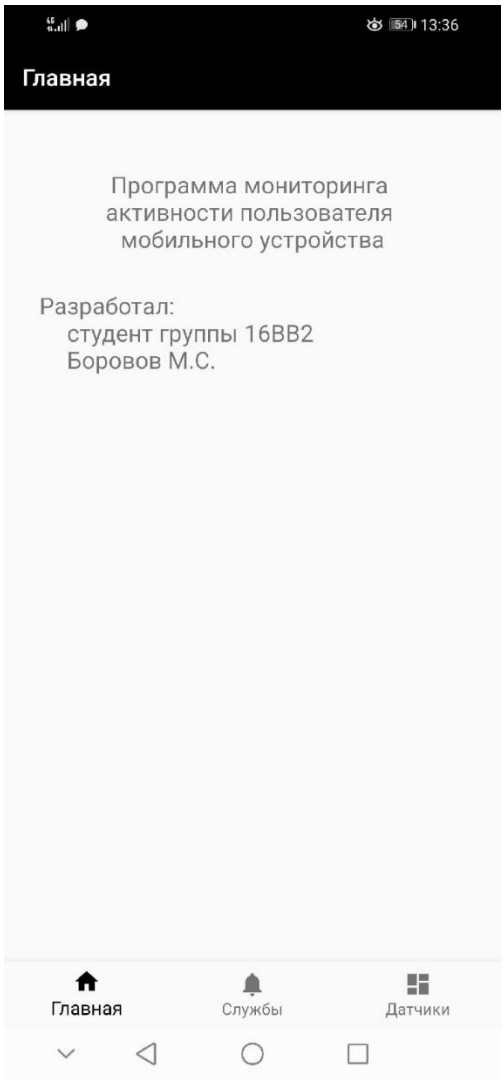


Рисунок 17 – Окно приветствия

Пользователю для выбора предоставляется меню переключения рабочих экранов. Данная реализация позволяет удобно и быстро перемещаться по экранам приложения, рисунок 18.

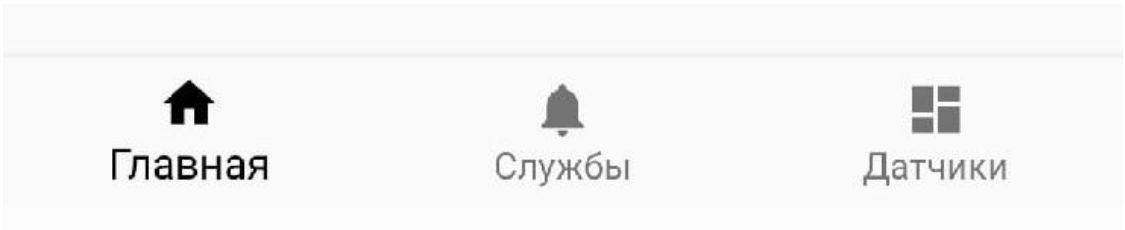


Рисунок 18 – Меню

Осуществить запуск фоновой службы или завершить её выполнение можно в окне служб. В данном окне расположены управляющие кнопки по работе со службой и для работы с накопившимися показаниями сенсоров. Пример запуска фоновой службы отображён на рисунке 19.

Рисунок 19 – Старт службы сбора информации

После нажатия на кнопку «Запустить», фоновая служба начнёт свою работу по мониторингу активности пользователя. Также создастся уведомление о текущем процессе, рисунок 20.

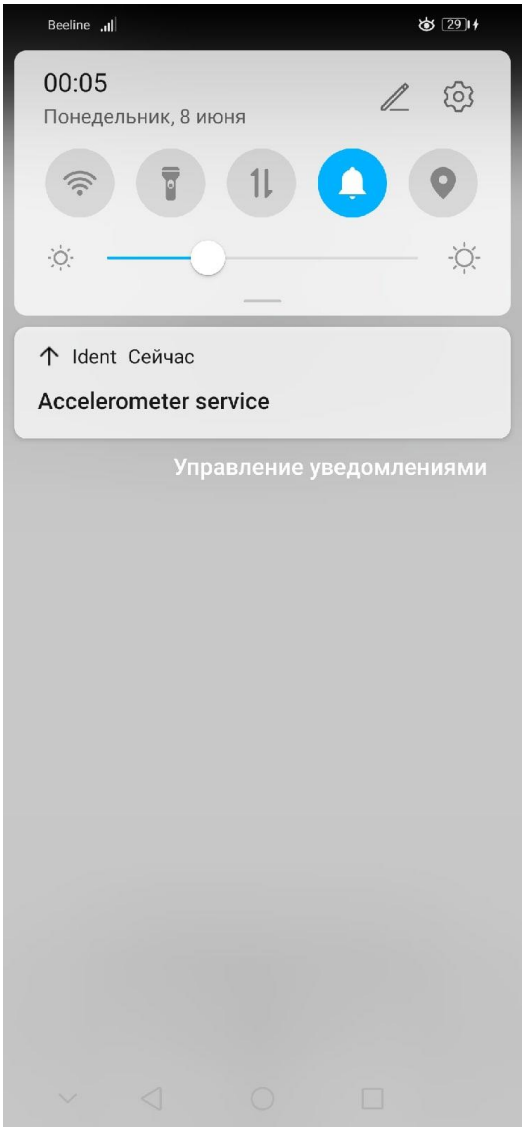


Рисунок 20 – Уведомление о выполнении мониторинга

На данном рисунке отображено уведомление от разработанного приложения «Ident». Уведомление содержит сведение об отслеживаемом датчике устройства, в данном изображении это сенсор акселерометра.

В данном разделе по управлению службами присутствует функционал по просмотру полученных показаний с датчика фонового мониторинга, рисунок 21.

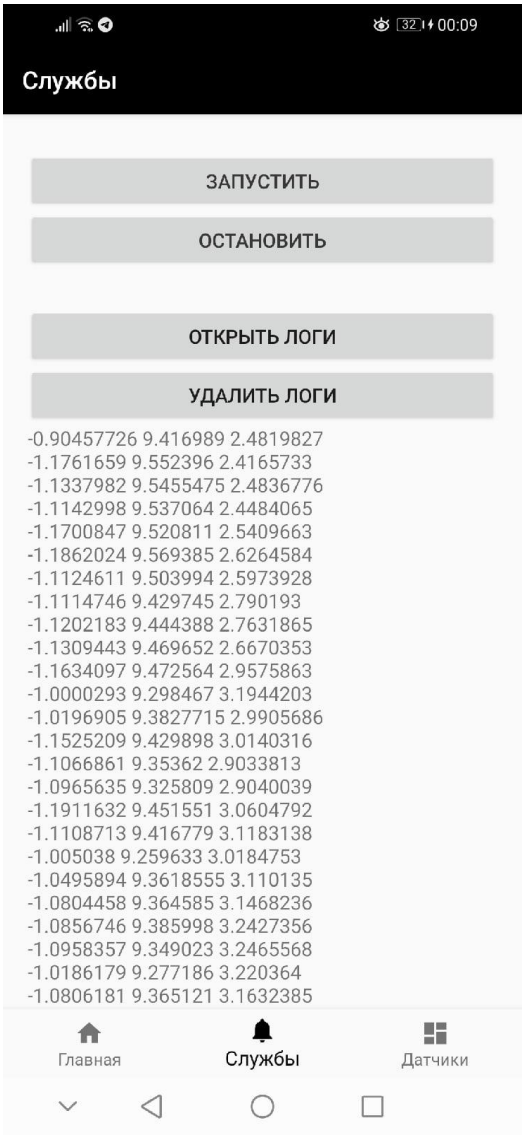


Рисунок 21 – Отображение полученных показаний

Также, неотъемлемым функционалом для выполнения поставленной задачи мониторинга активности пользователя мобильного устройств является отслеживание показаний сенсоров в реальном времени. Для таких нужд был спроектирован и разработан отдельных экран с соответствующим функционалом. Пример эксплуатации описанного функционала отображён на рисунке 22.

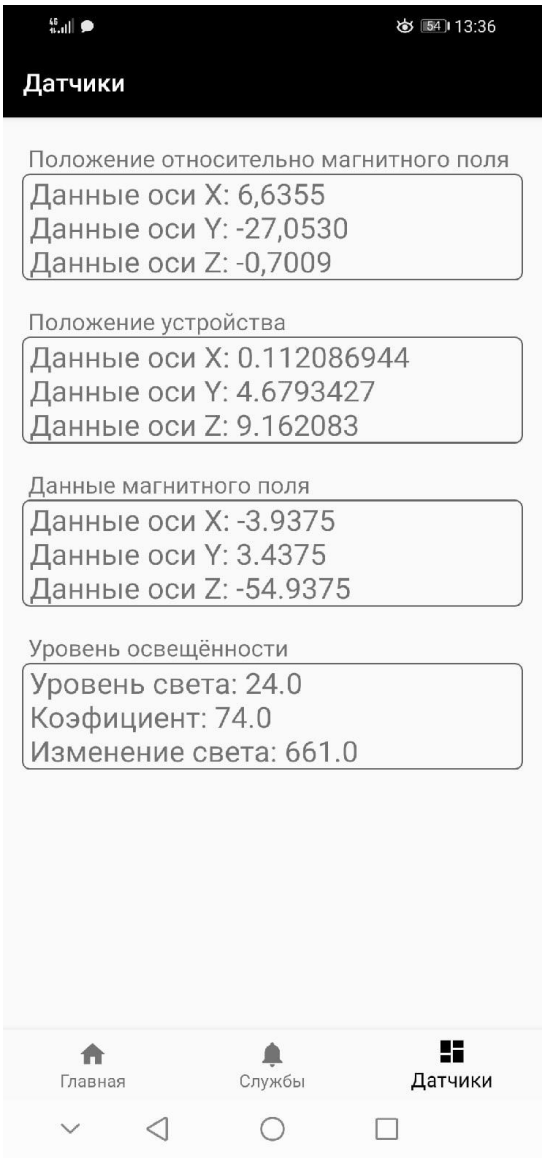


Рисунок 22– Показания датчиков

6 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

В последние годы большое внимание уделяется улучшению условий труда пользователей электронно-вычислительных машин и видеодисплейных терминалов, несмотря на то, что качество и безопасность самых ПЭВМ и ВДТ постоянно улучшаются. В развитых странах, в том числе в Германии, Швеции, США, вопрос об опасности работы за дисплеями поднялся до уровня национальной проблемы, а в Германии работа за дисплеями входит в список сорока наиболее опасных и вредных профессий.

Главным фактором, влияющим на производительность труда людей, работающих с электронно-вычислительными машинами и видеодисплейных терминалов, являются комфортные и безопасные условия их труда.

Условия труда пользователя, работающего с персональным компьютером, определяются:

- особенностями организации рабочего места;
- условиями производственной среды (освещением, микроклиматом, шумом, электромагнитными полями, визуальными эргономическими параметрами дисплея и т. д.);
- характеристиками информационного взаимодействия человека и персональных электронно-вычислительных машин.

При выполнении работ на персональном компьютере (ПК) согласно ГОСТу 12.0.003-74 “ССБТ. Опасные и вредные производственные факторы. Классификация” могут иметь место следующие факторы:

- повышенная температура поверхностей ПК;
- повышенная или пониженная температура воздуха рабочей зоны;
- выделение в воздух рабочей зоны ряда химических веществ;
- повышенная или пониженная влажность воздуха;
- повышенный или пониженный уровень аэроионов;

- повышенное значение напряжения в электрической цепи, замыкание;
- повышенный уровень статического электричества;
- повышенный уровень электромагнитных излучений;
- повышенная напряженность электрического поля;
- отсутствие или недостаток естественного света;
- недостаточная искусственная освещенность рабочей зоны;
- повышенная яркость света;
- повышенная контрастность;
- прямая и отраженная блёскость;
- зрительное напряжение;
- монотонность трудового процесса;
- нервно-эмоциональные перегрузки.

Работа на персональном компьютере сопровождается постоянным напряжением функций зрительного аппарата. Одной из основных особенностей является иной принцип чтения информации, чем при обычном чтении. При обычном чтении текст на бумаге, расположенный горизонтально на столе, считывается работником с наклоненной головой при падении светового потока на текст. При работе на ПК оператор производит считывание текста, практически не наклоняя голову, глаза смотрят прямо или почти прямо вперед, текст формируется по другую сторону экрана, поэтому пользователь не считывает отраженный текст, а смотрит непосредственно на источник света, что вынуждает глаза и орган зрения в целом работать в несвойственном ему стрессовом режиме длительное время, в связи с этим, у пользователя данной системы возникают.

Всемирная организация здравоохранения ввела понятие «компьютерный зрительный синдром», типичным перечнем симптомов, которого являются жжение в глазах, ощущение инородного объекта или песка под веками, боли

в области глазниц и лба, затуманивание зрения, замедленная перефокусировка с ближних объектов на дальние.

Повышенные статические и динамические нагрузки на организм у пользователей ПК приводят к симптомам похожим на боли в спине, шейном отделе позвоночника и руках. Из всего перечня недугов, обусловленных работой на компьютере, чаще встречаются те, которые непосредственно связаны с использованием клавиатуры. В период выполнения операций ввода значений количество незначительных однотипных манипуляций кистью и пальцами рук за смену может превысить 60 тысяч, что в соответствии с классификацией труда относится к категории вредных и опасных. Поскольку каждое нажатие на клавишу связано с сокращением мышц, сухожилия непрерывно скользят вдоль костей и соприкасаются с соединительными материалами, вследствие чего развиваются болезненные воспалительные процессы. Воспалительные процессы тканей сухожилий получили общее название «травма повторяющихся нагрузок».

Большинство работающих рано или поздно начинают предъявлять жалобы на боли в шее и спине. Эти недомогания накапливаются постепенно и получили название «синдром длительных статических нагрузок».

Другой причиной возникновения СДСН может быть длительное пребывание в положении «сидя», которое приводит к сильному перенапряжению мышц спины и ног, в результате чего возникают боли и неприятные ощущения в нижней части спины. Основной причиной перенапряжения мышц спины и ног являются нерациональная высота рабочей поверхности стола и сидения, отсутствие опорного механизма и опоры для локтей, непрвильное размещение монитора, клавиатуры и документов, отсутствие подставки для ног.

Для симптомного уменьшения неприятных ощущений или боли, возникающих у пользователей ПК, необходимы частые перерывы в работе и эргономические усовершенствования, в том числе оборудование рабочего места так, чтобыустранять неудобные позы и длительные напряжения.

ЗАКЛЮЧЕНИЕ

Результатом работы является приложение для мониторинга активности пользователя с интуитивно понятным пользовательским интерфейсом. Приложение позволяет работать с операционной системой и системными датчиками. Данный проект полностью удовлетворяет поставленной цели.

Данная информационная система позволяет пользователю беспрепятственно вести учет активности пользователя. Показания составляются на основе полученных значений с датчиков устройства.

Отладка и тестирование состоит из проверки целостности передачи данных в пользовательский интерфейс и обработки полученных значений.

Работа над проектом позволила расширить знания и улучшить навыки создания подобных приложений для ОС Android.

Программа может быть модернизирована в дальнейшем, в соответствии с потребностями пользовательского сообщества.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Герберт Шилдг. Java руководство для начинающих.
2. Шилдт Герберт- Java 8 Полное руководство.
3. Официальная документация по Android - разработке [Электронный ресурс] – https://developer.android.com
4. Дейтел П., Дейтел Х., Уолд А. Android для разработчиков.
5. Мартин Р. Чистый Код: создание, анализ и рефакторинг.
6. Гамма Э., Хелм., Джонсон Р., Влиссидес., Приемы объектно-ориентированного проектирования.
7. Medium – Android Developers [Электронный ресурс] – https://medium.com/androiddevelopers

ПРИЛОЖЕНИЕ А
ЛИСТИНГ ПРОГРАММЫ

Приложение А.1 Перечень классов программы

Файл «DashboardFragment.java»

```
package com.example.ident.ui.dashboard;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.Nullable;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;
import com.example.ident.ForegroundService;
import com.example.ident.R;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import static android.content.Context.MODE_PRIVATE;

public class DashboardFragment extends Fragment {

    private DashboardViewModel dashboardViewModel;
    String FILE_NAME = "sensor.txt";

    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        dashboardViewModel =
ViewModelProviders.of(this).get(DashboardViewModel.class);
        View root = inflater.inflate(R.layout.fragment_dashboard, container,
false);

        final TextView textView = root.findViewById(R.id.textTitleSensors);
        dashboardViewModel.getText().observe(this, new Observer<String>() {
            @Override
            public void onChanged(@Nullable String s) {
                textView.setText(s);
            }
        });

        setButtonsListeners(root);

        return root;
    }

    void setButtonsListeners(final View mainView) {
        Button butStart = mainView.findViewById(R.id.butStart);
        butStart.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View view)
            {
                startService();
            }
        });

        Button butStop = mainView.findViewById(R.id.butStop);
        butStop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view)
            {
                stopService();
            }
        });

        Button butDataOpen = mainView.findViewById(R.id.butDataOpen);
        butDataOpen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view)
            {
                openData(mainView);
            }
        });

        Button butDataDelete = mainView.findViewById(R.id.butDataDelete);
        butDataDelete.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view)
            {
                deleteData(mainView);
            }
        });
    }

    public void startService() {
        Intent serviceIntent = new Intent(getActivity(),
ForegroundService.class);
        serviceIntent.putExtra("name", "value");
        getActivity().startService(serviceIntent);
    }

    public void stopService() {
        Intent serviceIntent = new Intent(getActivity(),
ForegroundService.class);
        getActivity().stopService(serviceIntent);
    }

    public void openData(View view) {
        FileInputStream fin = null;
        TextView textView = view.findViewById(R.id.textData);
        try {
            fin = getContext().openFileInput(FILE_NAME);
            byte[] bytes = new byte[fin.available()];
            fin.read(bytes);
            String text = new String (bytes);
```

```
            if (textView != null) textView.setText(text);
        }
        catch(IOException ex) {
            Log.e("tag", ex.getMessage());
            Toast.makeText(getContext(), ex.getMessage(),
Toast.LENGTH_SHORT).show();
        }
        finally {
            try {
                if(fin!=null)
                    fin.close();
            }
            catch(IOException ex) {
                Toast.makeText(getContext(), ex.getMessage(),
Toast.LENGTH_SHORT).show();
            }
        }
    }

    public void deleteData(View view) {
        FileOutputStream fos = null;
        TextView textView = view.findViewById(R.id.textData);
        try {
            fos = getContext().openFileOutput(FILE_NAME, MODE_PRIVATE);
            fos.write("".getBytes());
            if (textView != null) textView.setText("");
        }
        catch(IOException ex) {
            Toast.makeText(getContext(), ex.getMessage(),
Toast.LENGTH_SHORT).show();
        }
        finally {
            try {
                if (fos != null)
                    fos.close();
            }
            catch (IOException ex) {
                Toast.makeText(getContext(), ex.getMessage(),
Toast.LENGTH_SHORT).show();
            }
        }
    }

}
```

Файл «HomeFragment.java»

```
package com.example.ident.ui.home;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.Nullable;
import androidx.annotation.NonNull;
```

```
import androidx.fragment.app.Fragment;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;

import com.example.ident.R;

public class HomeFragment extends Fragment {

    private HomeViewModel homeViewModel;

    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        homeViewModel = ViewModelProviders.of(this).get(HomeViewModel.class);
        View root = inflater.inflate(R.layout.fragment_home, container, false);

        final TextView textView = new TextView(getContext());
        homeViewModel.getText().observe(this, new Observer<String>() {
            @Override
            public void onChanged(@Nullable String s) {
                textView.setText(s);
            }
        });
        return root;
    }
}
```

Файл «NotificationsFragment.java»

```
package com.example.ident.ui.notifications;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.Display;
import android.view.LayoutInflater;
import android.view.Surface;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.TextView;

import androidx.annotation.Nullable;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;

import com.example.ident.R;

import java.util.Timer;
import java.util.TimerTask;

import static android.content.Context.SENSOR_SERVICE;
```

```
public class NotificationsFragment extends Fragment {

    // Переменные подключения к датчикам
    private TextView tvText;
    private TextView tvLight;
    private TextView tvMagnet;
    private TextView tvSysAccel;
    private SensorManager sensorManager;
    private Sensor sensorAccel;
    private Sensor sensorMagnet;
    private Sensor sensorLight;

    // Переменные датчиков
    private float[] valuesAccel = new float[3];
    private float[] valuesMagnet = new float[3];
    private float[] valuesResult = new float[3];
    private float[] valuesResult2 = new float[3];
    private float[] valuesLight = new float[3];

    // Переменные вывода
    private StringBuilder sbAccel = new StringBuilder();
    private StringBuilder sbLight = new StringBuilder();
    private StringBuilder sbMagnet = new StringBuilder();
    private StringBuilder sbSysAccel = new StringBuilder();
    private Timer timer;
    private int rotation;


    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        NotificationsViewModel notificationsViewModel =
ViewModelProviders.of(this).get(NotificationsViewModel.class);
        View root =
                inflater.inflate(R.layout.fragment_notifications, container,
false);

        final TextView textView = root.findViewById(R.id.text_orientation);
        notificationsViewModel.getText().observe(this, new Observer<String>() {
            @Override
            public void onChanged(@Nullable String s) {
                textView.setText(s);
            }
        });

        // Регистрация переменных
        tvText = root.findViewById(R.id.text_orientation);
        tvMagnet = root.findViewById(R.id.text_magnet);
        tvLight = root.findViewById(R.id.text_light);
        tvSysAccel = root.findViewById(R.id.text_sysaccel);
        sensorManager = (SensorManager)
getActivity().getSystemService(SENSOR_SERVICE);

        // Виды датчиков
```

```
        sensorAccel = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorMagnet =
sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
        sensorLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);

        return root;
    }


    @Override
    public void onResume() {
        super.onResume();
        // Регистрация датчиков
        sensorManager.registerListener(listener, sensorAccel,
SensorManager.SENSOR_DELAY_NORMAL);
        sensorManager.registerListener(listener, sensorMagnet,
SensorManager.SENSOR_DELAY_NORMAL);
        sensorManager.registerListener(listener, sensorLight,
SensorManager.SENSOR_DELAY_NORMAL);

        timer = new Timer();
        TimerTask task = new TimerTask() {
            @Override
            public void run() {
                getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        getDeviceOrientation();
                        getActualDeviceOrientation();

                        showInfo();
                    }
                });
            }
        };
        timer.schedule(task, 0, 400);

        WindowManager windowManager = ((WindowManager)
getActivity().getSystemService(Context.WINDOW_SERVICE));
        Display display = windowManager.getDefaultDisplay();
        rotation = display.getRotation();
    }


    @Override
    public void onPause() {
        super.onPause();
        sensorManager.unregisterListener(listener);
        timer.cancel();
    }


    private String format(float[] values) {
        return String.format(" Данные оси X: %1$.4f\n" +
                " Данные оси Y: %2$.4f\n" +
```

```
                " Данные оси Z: %3$.4f",
                values[0], values[1], values[2]);
    }


    private void showInfo() {
        // Положение с учётом магнитного поля
        sbAccel.setLength(0);
        sbAccel.append(format(valuesResult2));
        tvText.setText(sbAccel.toString());

        // Положение устройства
        sbSysAccel.setLength(0);
        sbSysAccel.append(" Данные оси X: ");
        sbSysAccel.append(valuesAccel[0]).append("\n");
        sbSysAccel.append(" Данные оси Y: ");
        sbSysAccel.append(valuesAccel[1]).append("\n");
        sbSysAccel.append(" Данные оси Z: ");
        sbSysAccel.append(valuesAccel[2]);

        tvSysAccel.setText(sbSysAccel.toString());

        // Магнитное поле
        sbMagnet.setLength(0);
        sbMagnet.append(" Данные оси X: ");
        sbMagnet.append(valuesMagnet[0]).append("\n");
        sbMagnet.append(" Данные оси Y: ");
        sbMagnet.append(valuesMagnet[1]).append("\n");
        sbMagnet.append(" Данные оси Z: ");
        sbMagnet.append(valuesMagnet[2]);

        tvMagnet.setText(sbMagnet.toString());

        // Освещение
        sbLight.setLength(0);
        sbLight.append(" Уровень света: ");
        sbLight.append(valuesLight[0]).append("\n");
        sbLight.append(" Коэфициент: ");
        sbLight.append(valuesLight[1]).append("\n");
        sbLight.append(" Изменение света: ");
        sbLight.append(valuesLight[2]);

        tvLight.setText(sbLight.toString());
    }


    private void getDeviceOrientation() {
        float[] r = new float[9];
        SensorManager.getRotationMatrix(r, null, valuesAccel, valuesMagnet);
        SensorManager.getOrientation(r, valuesResult);

        // Перевод из радиан в градусы
        valuesResult[0] = (float) Math.toDegrees(valuesResult[0]);
        valuesResult[1] = (float) Math.toDegrees(valuesResult[1]);
        valuesResult[2] = (float) Math.toDegrees(valuesResult[2]);
    }
```

```
    private void getActualDeviceOrientation() {
        float[] inR = new float[9];
        float[] outR = new float[9];

        SensorManager.getRotationMatrix(inR, null, valuesAccel, valuesMagnet);
        int x_axis = SensorManager.AXIS_X;
        int y_axis = SensorManager.AXIS_Y;

        switch (rotation) {
            case (Surface.ROTATION_90):
                x_axis = SensorManager.AXIS_Y;
                y_axis = SensorManager.AXIS_MINUS_X;
                break;
            case (Surface.ROTATION_180):
                y_axis = SensorManager.AXIS_MINUS_Y;
                break;
            case (Surface.ROTATION_270):
                x_axis = SensorManager.AXIS_MINUS_Y;
                y_axis = SensorManager.AXIS_X;
                break;
            default:
                break;
        }

        SensorManager.remapCoordinateSystem(inR, x_axis, y_axis, outR);
        SensorManager.getOrientation(outR, valuesResult2);
        valuesResult2[0] = (float) Math.toDegrees(valuesResult2[0]);
        valuesResult2[1] = (float) Math.toDegrees(valuesResult2[1]);
        valuesResult2[2] = (float) Math.toDegrees(valuesResult2[2]);
    }


    private SensorEventListener listener = new SensorEventListener() {

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
        }

        @Override
        public void onSensorChanged(SensorEvent event) {
            switch (event.sensor.getType()) {
                case Sensor.TYPE_ACCELEROMETER:
                    for (int i=0; i < 3; i++){
                        valuesAccel[i] = event.values[i];
                    }
                    break;
                case Sensor.TYPE_MAGNETIC_FIELD:
                    for (int i=0; i < 3; i++) {
                        valuesMagnet[i] = event.values[i];
                    }
                    break;
                case Sensor.TYPE_LIGHT:
                    for (int i=0; i < 3; i++) {
```

```
                        valuesLight[i] = event.values[i];
                    }
                default:
                    break;
            }
        }
    };
}
```

Файл «ForegroundService.java»

```
package com.example.ident;
import android.inputmethodservice.InputMethodService;
import android.inputmethodservice.Keyboard;
import android.inputmethodservice.KeyboardView;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.inputmethod.InputConnection;
import android.widget.Toast;

import java.io.FileOutputStream;
import java.io.IOException;

public class KeyboardService extends InputMethodService implements
KeyboardView.OnKeyboardActionListener {

    KeyboardView kv;
    Keyboard keyboard, keyboard_sym;
    String TAG = "EnterKey";
    String appName;
    boolean caps = false;
    final static String FILE_NAME = "content.txt";

    @Override
    public View onCreateInputView() {
        kv = (KeyboardView)getLayoutInflater().inflate(R.layout.keyboard, null);
        keyboard = new Keyboard(this, R.xml.qwerty);
        // keyboard_sym = new Keyboard(this, R.xml.symbol);
        kv.setKeyboard(keyboard);
        kv.setOnKeyboardActionListener(this);
        Log.i(TAG,"KeyboardService is started");

        appName = getCurrentInputEditorInfo().packageName;
        Toast.makeText(this, appName , Toast.LENGTH_SHORT).show();
        return kv;
    }

    @Override
    public void onKey(int primaryCode, int[] keyCodes) {
        InputConnection ic = getCurrentInputConnection();
        String ch = Character.toString ((char) primaryCode);
        String character = primaryCode == -1?"caps":ch;

        saveText(character);
```

```
        Log.i(TAG,"Clicked " + primaryCode + " " + character);

        switch(primaryCode) {
            case Keyboard.KEYCODE_DELETE :
                ic.deleteSurroundingText(1, 0);
                break;
            case Keyboard.KEYCODE_SHIFT:
                caps = !caps;
                keyboard.setShifted(caps);
                kv.invalidateAllKeys();
                break;
            case Keyboard.KEYCODE_DONE:
                Log.i("SimpleKeyboard"," Hello " + KeyEvent.ACTION_DOWN + " " +
KeyEvent.KEYCODE_ENTER);
                ic.sendKeyEvent(new KeyEvent(KeyEvent.ACTION_DOWN,
KeyEvent.KEYCODE_ENTER));
                break;
            case 1000:
                // kv.setKeyboard(keyboard_sym);
                    break;
            case 1001:
                // kv.setKeyboard(keyboard);
                break;
            default:
                char code = (char)primaryCode;

                if(Character.isLetter(code) && caps){
                    code = Character.toUpperCase(code);
                }
                ic.commitText(String.valueOf(code),1);
        }
    }

    @Override
    public void onWindowHidden() {

    }

    public void saveText(String character) {
        FileOutputStream fos = null;
        try {
            fos = openFileOutput(FILE_NAME, MODE_APPEND);
            fos.write(character.getBytes());
        }
        catch(IOException ex) {
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
        finally {
            try {
                if (fos != null)
                    fos.close();
            }
            catch (IOException ex) {
                Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
            }
```

```
        }
    }

    @Override
    public void onPress(int primaryCode) {
        Log.e("SimpleKeyboard","onPress " + primaryCode);
    }

    @Override
    public void onRelease(int primaryCode) {
    }

    @Override
    public void onText(CharSequence text) {
        Log.e("SimpleKeyboard","onText " + text);
    }

    @Override
    public void swipeDown() {
    }

    @Override
    public void swipeLeft() {
    }

    @Override
    public void swipeRight() {
    }

    @Override
    public void swipeUp() {
    }

}
```

Файл «MainAcrivity.java»

```
package com.example.ident;
import android.os.Bundle;
import com.google.android.material.bottomnavigation.BottomNavigationView;

import androidx.appcompat.app.AppCompatActivity;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        BottomNavigationView navView = findViewById(R.id.nav_view);
        // Passing each menu ID as a set of Ids because each
```

```
        // menu should be considered as top level destinations.
        AppBarConfiguration appBarConfiguration = new
AppBarConfiguration.Builder(
                R.id.navigation_home, R.id.navigation_dashboard,
R.id.navigation_notifications)
                .build();
        NavController navController = Navigation.findNavController(this,
R.id.nav_host_fragment);
        NavigationUI.setupActionBarWithNavController(this, navController,
appBarConfiguration);
        NavigationUI.setupWithNavController(navView, navController);
    }

}
```

Приложение А.2 Файлы разметки

Файл «Fragment_dashboard.xml»

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp">

    <Button
        android:id="@+id/butStart"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/colorWindowBackground"
        android:layout_marginTop="30dp"
        android:text="Запустить" />

    <Button
        android:id="@+id/butStop"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/colorWindowBackground"
        android:text="Остановить" />

    <Button
        android:id="@+id/butDataOpen"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:text="Открыть логи" />

    <Button
        android:id="@+id/butDataDelete"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Удалить логи" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/textData"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="" />

    </ScrollView>

</LinearLayout>
```

Файл «Fragment_home.xml»

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_marginLeft="30dp"
    android:layout_marginRight="30dp">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAlignment="center"
        android:textSize="20dp"
        android:paddingTop="50dp"
        android:text="Программа мониторинга \nактивности пользователя
\nмобильного устройства" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:paddingTop="30dp"
        android:textAlignment="textStart"
        android:text="Разработал: \n\tстудент группы 16ВВ2 \n\tБоровов М.С." />

</LinearLayout>
```

Файл «Fragment_notifications.xml»

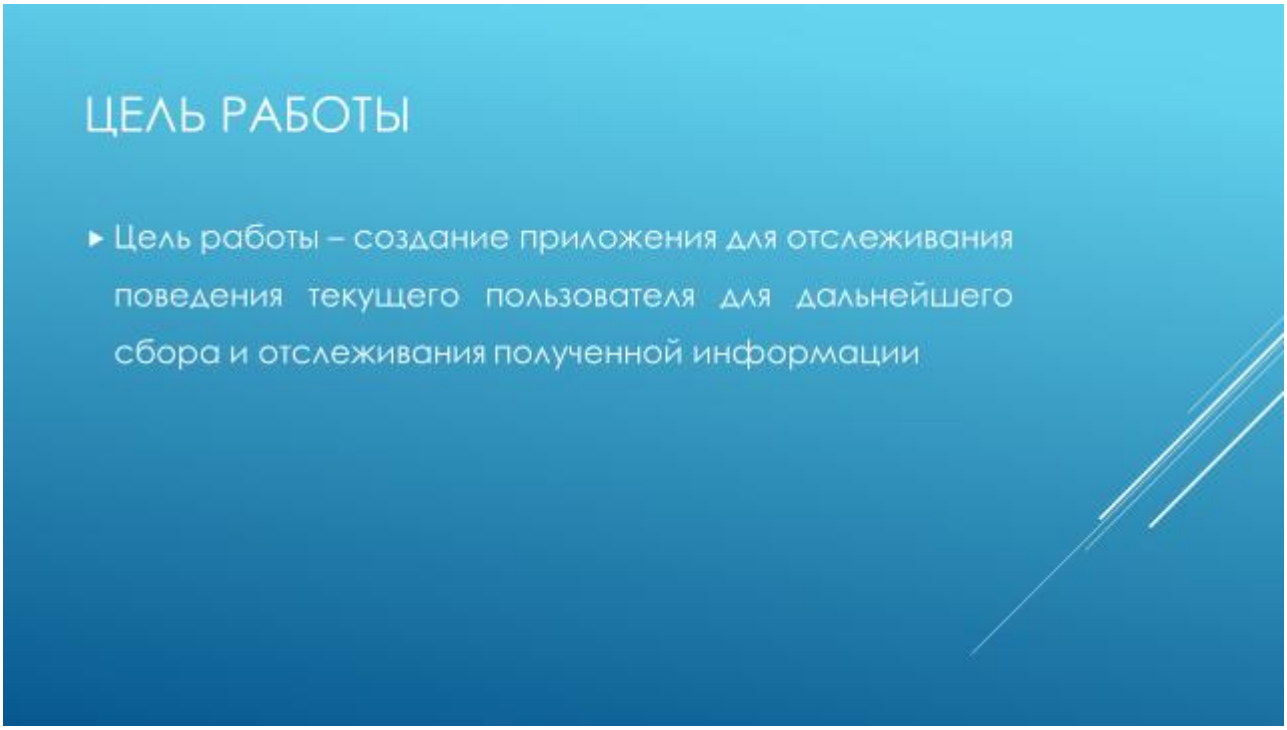
```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_marginStart="15dp"
    android:layout_marginEnd="15dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:textSize="16sp"
            android:text=" Положение относительно магнитного поля" />

        <TextView
            android:id="@+id/text_orientation"
            android:layout_width="match_parent"
```

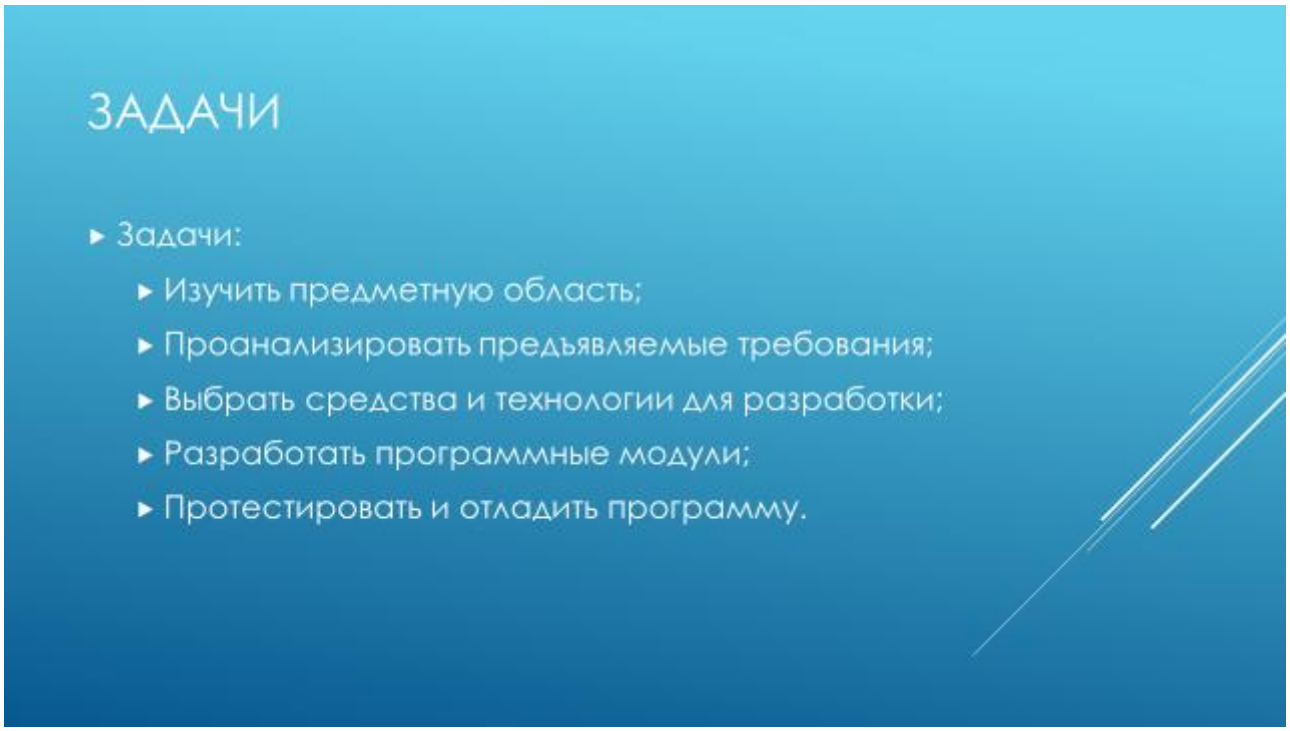
```
            android:layout_height="wrap_content"
            android:background="@drawable/stroke"
            android:textAlignment="textStart"
            android:textSize="20sp" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:textSize="16sp"
            android:text=" Положение устройства" />

        <TextView
            android:id="@+id/text_sysaccel"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@drawable/stroke"
            android:textAlignment="textStart"
            android:textSize="20sp" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:textSize="16sp"
            android:text=" Данные магнитного поля" />

        <TextView
            android:id="@+id/text_magnet"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@drawable/stroke"
            android:textAlignment="textStart"
            android:textSize="20sp" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:textSize="16sp"
            android:text=" Уровень освещённости" />

        <TextView
            android:id="@+id/text_light"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@drawable/stroke"
            android:textAlignment="textStart"
            android:textSize="20sp" />

    </LinearLayout>

</ScrollView>
```

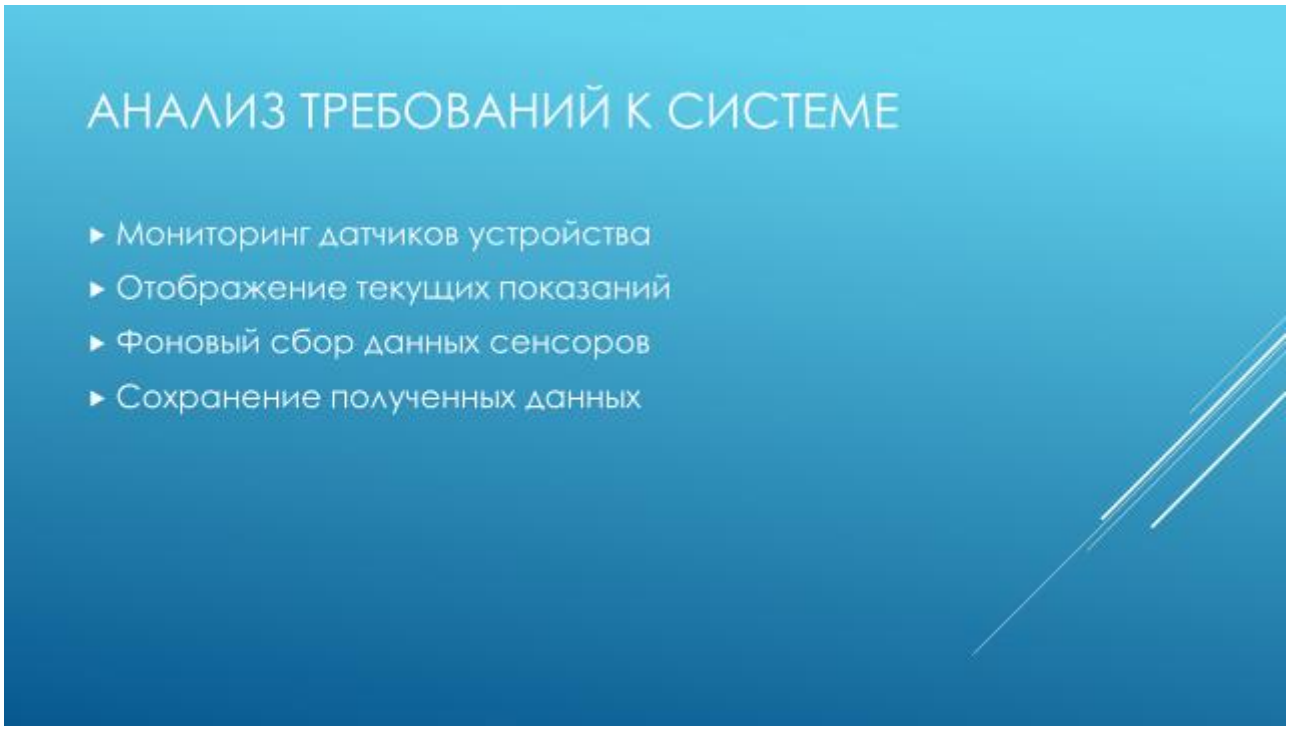
ПРИЛОЖЕНИЕ Б

ПРЕЗЕНТАЦИЯ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ
КАФЕДРА «ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»
ПРОГРАММА МОНИТОРИНГА АКТИВНОСТИ ПОЛЬЗОВАТЕЛЯ МОБИЛЬНОГО УСТРОЙСТВА
Выполнил:
студент группы 16ВВ2
Боровов М.С.
Руководитель:
д.т.н., Митрохин М.А.
Пенза 2020


ЦЕЛЬ РАБОТЫ
Цель работы – создание приложения для отслеживания поведения текущего пользователя для дальнейшего сбора и отслеживания полученной информации


ЗАДАЧИ
Задачи:
Изучить предметную область;
Проанализировать предъявляемые требования;
Выбрать средства и технологии для разработки;
Разработать программные модули;
Протестировать и отладить программу.


АНАЛИЗ ТРЕБОВАНИЙ К СИСТЕМЕ
Мониторинг датчиков устройства
Отображение текущих показаний
Фоновый сбор данных сенсоров
Сохранение полученных данных

ВЫБОР ЯЗЫКОВ, СРЕДСТВ И ТЕХНОЛОГИЙ

- Языки
  - Java
  - XML
- Средства и технологии
  - ОС Android
  - Android Studio
  - Gradle
  - SDK

ЭКРАНЫ ПРИЛОЖЕНИЯ

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ

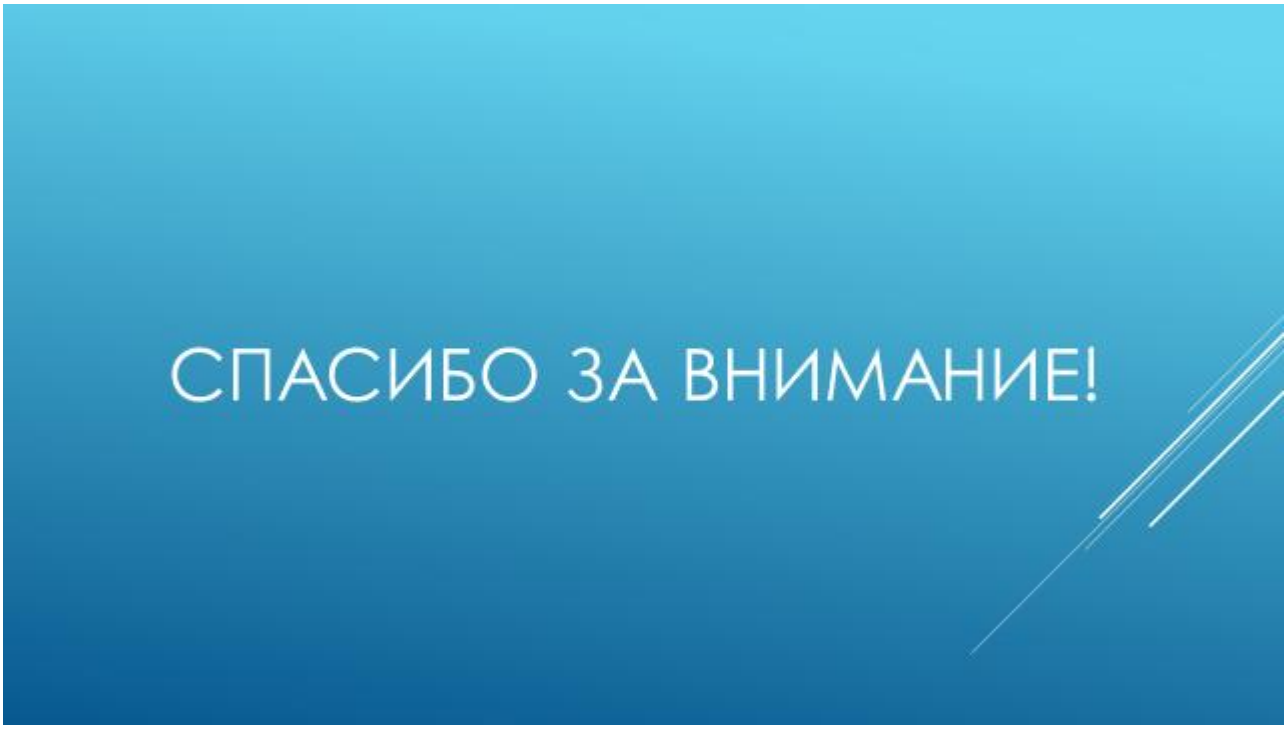
ТЕСТИРОВАНИЕ

- Тестирование отдельных модулей
- Комплексное тестирование взаимодействия модулей
- Исправление обнаруженных ошибок
- Повторная отладка

ДИАГРАММА КЛАССОВ

ЗАКЛЮЧЕНИЕ

- Результатом проделанной работы является:
  - Приложение с доступным и понятным интерфейсом
  - Возможность мониторинга показаний датчиков в реальном времени
  - Функционал для фонового сбора показаний сенсоров
  - Полностью сформированное и автономное приложение, удовлетворяющее поставленной задаче


СПАСИБО ЗА ВНИМАНИЕ!