

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Андроид-приложение сбора данных о
радиообстановке ближайшего окружения**

Студент

(подпись, дата)

Сычев Матвей Дмитриевич

(ФИО полностью)

Руководитель

(подпись, дата)

Гурьянов Л.В.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой

(протокол заседания ГЭК от №)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент _____ Сычев Матвей Дмитриевич _____ гр. 22ВП2 _____ факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Андроид-приложение сбора данных о радиообстановке ближайшего
окружения

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025 г.

3. Руководитель работы _____ к.т.н., доцент кафедры «МОиПЭВМ» Гурьянов Л.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: разрабатываемое приложение предназначено для сбора
телеметрических данных о радиообстановке ближайшего окружения, их отображения
в графическом интерфейсе, и их передачи на сервер посредством протокола MQTT.

Основные функции приложения:

- сбор телеметрических данных;
- отображение собранных данных в интерфейсе пользователя;
- приведение собранных данных в структурированный формат;
- отправка структурированных данных в брокер сообщений по протоколу MQTT.

Платформа разрабатываемого приложения: Android.

Язык программирования: Kotlin.

Среда разработки: Android Studio.

ОС для разработки: Ubuntu 23.10.

Инструментальные средства: Visual Paradigm, Microsoft Office Visio, Microsoft Office

Project.

5. Перечень подлежащих разработке вопросов:

- постановка задачи и анализ требований;
- проектирование приложения;
- реализация приложения;
- контроль качества приложения;
- планирование разработки и анализ бюджета;
- оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

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

Реферат

Пояснительная записка содержит 75 листов, 27 рисунков, 20 использованных источников, 6 таблиц, 4 приложения.

СБОР ТЕЛЕМЕТРИЧЕСКИХ ДАННЫХ, СОТОВАЯ СЕТЬ, БАЗОВАЯ СТАНЦИЯ, МЕСТОПОЛОЖЕНИЕ, WI-FI, ANDROID, MQTT, JSON, KOTLIN.

Объект исследования: Android-приложение, программные средства сбора, агрегации, отображения и передачи телеметрических данных.

Цель работы: разработать Android-приложение для сбора данных о радиообстановке ближайшего окружения.

Разработка проводилась на языке программирования Kotlin в среде разработки Android Studio. В качестве сетевого протокола для передачи собранных телеметрических данных использовался MQTT. Операционной системой, использовавшейся во время разработки, являлась Ubuntu 23.10.

Результаты работы: спроектировано и разработано Android-приложение, обеспечивающее комплексный сбор данных о радиообстановке ближайшего окружения (данные о сотовой сети, Wi-Fi сетях, местоположении, устройстве), их отображение в интерфейсе пользователя и передача в брокер сообщений MQTT.

					ПГУ 09.03.04-08БР222.29 ПЗ			
Изм	Лист	№ докум.	Подп.	Дата				
Разраб.		Сычев М.Д.			«Андроид-приложение сбора данных о радиообстановке ближайшего окружения» Пояснительная записка	Лит.	Лист	Листов
Пров.		Гурьянов Л.В.					4	75
Н. контр.		Такташкин Д.В.				Группа 22ВП2		
Утв.		Козлов А.Ю.						

Содержание

Введение.....	6
1 Постановка задачи и анализ требований	8
1.1 Постановка задачи на разработку приложения.....	8
1.2 Анализ аналогов разрабатываемого приложения	10
1.3 Анализ функциональных требований.....	15
1.4 Анализ нефункциональных требований.....	19
2 Проектирование приложения.....	21
2.1 Проектирование структуры приложения.....	21
2.2 Проектирование интерфейса приложения.....	27
3 Реализация приложения	30
4 Контроль качества программного обеспечения	38
4.1 Анализ метрик.....	38
4.1.1 Вычислительная сложность.....	39
4.1.2 Цикломатическая сложность	41
4.1.3 Комплексный показатель качества кода	43
4.2 Инструментальное тестирование	44
5 Планирование разработки и оценка бюджета	47
Заключение.....	50
Список использованных источников.....	52
Приложение А (обязательное) Исходный код приложения	54
Приложение Б (рекомендуемое) Скриншоты результатов тестирования.....	67
Приложение В (рекомендуемое) Исходный код тестов.....	70
Приложение Г (рекомендуемое) Акт о внедрении.....	74

Введение

В современных условиях стремительного развития мобильных технологий и цифровой инфраструктуры смартфоны стали неотъемлемой частью повседневной жизни человека. Они позволяют людям постоянно поддерживать коммуникацию со всем миром посредством различных каналов связи: традиционной телефонной связи, мобильного интернета, а также беспроводных сетей Wi-Fi. Благодаря этому появляется возможность находиться на связи в любой момент времени, независимо от местоположения.

Исходя из этого, одной из актуальных задач является мониторинг качества связи и доступности сетевой инфраструктуры, особенно в условиях высокой нагрузки на сети, неравномерного покрытия и постоянного роста требований к надежности передачи данных. Пользователи могут сталкиваться с проблемами нестабильного соединения, низкого уровня сигнала, отсутствия доступа к сети в определенных зонах.

В связи с этим возникает потребность в разработке программных средств, способных в реальном времени собирать и агрегировать данные о состоянии мобильной сети, беспроводных точек доступа и геолокации устройства. Такие данные позволяют проводить анализ качества связи, выявлять проблемные зоны покрытия, а также оптимизировать работу сетевой инфраструктуры.

Целью данной работы является разработка мобильного приложения для платформы Android [1, 2], предназначенного для сбора телеметрических данных (об устройстве пользователя, текущем состоянии сотовой сети, Wi-Fi сетях и местоположении), их отображения в графическом интерфейсе, а также передачи на сервер посредством протокола MQTT [3].

Для достижения этой цели требуется решить ряд подзадач:

- изучить предметную область и провести анализ требований;
- спроектировать приложение;
- разработать диаграммы описания приложения;
- реализовать приложение;

- провести контроль качества приложения;
- провести планирование разработки и оценку бюджета проекта.

Успешное выполнение данных подзадач позволит создать приложение сбора данных о радиообстановке ближайшего окружения.

1 Постановка задачи и анализ требований

1.1 Постановка задачи на разработку приложения

В рамках выпускной квалификационной работы необходимо создать приложение [2] для сбора данных о радиообстановке ближайшего окружения. Оно должно предоставлять пользователю простой, понятный и наглядный интерфейс, в котором в структурированном виде будут отображаться все собранные данные, производить сбор данных без значительных задержек, работать как минимум на двух последних версиях Android [1], а также реализовывать следующие функции:

- сбор телеметрических данных о местоположении (полученные от GPS и рассчитанные на основе сети);
- сбор телеметрических данных о состоянии сотовой сети: оператор сети и сим-карты, тип радиосвязи, тип мобильной сети, сила и статус сигнала, статус подключения к базовым станциям, сила сигнала от них, типы и ID сот, ID базовых станций для LTE, коды страны и оператора базовых станций [4];
- сбор телеметрических данных о Wi-Fi сетях: статус подключения, название, MAC-адрес, стандарт и частота сети, сила сигнала и стандарты защиты сети;
- сбор телеметрических данных об устройстве пользователя: название и модель телефона, версия и ID Android, версия SDK;
- отображение собранных данных в интерфейсе пользователя;
- приведение собранных данных в структурированный формат;
- отправка структурированных данных в брокер сообщений по сетевому протоколу;
- отправка пользователю уведомлений о статусе выполнения операций.

В качестве сетевого протокола передачи сообщений необходимо использовать протокол MQTT [3]. Он является оптимальным для мобильных

устройств благодаря своей легковесности и низкому потреблению сетевых и аппаратных ресурсов. MQTT обеспечивает передачу данных с минимальным объемом служебной информации, что снижает нагрузку на интернет-соединение и уменьшает энергопотребление устройства. Также протокол стабильно работает даже при нестабильной сети и поддерживает передачу телеметрических данных в режиме реального времени.

В качестве формата упаковки данных для отправки необходимо использовать JSON [5]. Этот формат является простым, универсальным, и поддерживается большинством языков программирования. Он легко формируется, передается и обрабатывается на различных устройствах и платформах, позволяет удобно хранить структурированные данные, включая вложенные объекты и массивы, что особенно важно при передаче телеметрической информации. Также данный формат упрощает отладку и анализ передаваемых сообщений, поскольку содержимое JSON легко читается человеком.

Разработку необходимо проводить на языке программирования Kotlin [6, 7], так как он является стандартом индустрии и с 2019 года объявлен предпочтительным языком программирования для Android-разработки. В качестве среды разработки необходимо использовать Android Studio [8], так как она является официальной средой от Google, предлагающий полный спектр инструментов для разработки Android-приложений.

В качестве операционной системы при разработке приложения необходимо использовать Ubuntu, так как она является хорошо совместимой с выбранными инструментами разработки, имеет удобные средства для работы с сетью, обеспечивает простую установку и настройку MQTT-брокеров, а также предоставляет пользователю удобный и понятный интерфейс.

При выполнении всех вышеприведенных требований разрабатываемое приложение будет обеспечивать реализацию всех заявленных возможностей.

1.2 Анализ аналогов разрабатываемого приложения

В рамках разработки Android-приложения [2] для сбора данных о радиообстановке ближайшего окружения был проведен анализ существующих решений, обладающих функциональностью, частично или полностью пересекающейся с разрабатываемым. Основное назначение разрабатываемого приложения заключается в сборе телеметрических данных устройства (о сотовой сети, Wi-Fi сетях, местоположении, устройстве), их отображения в интерфейсе пользователя и передаче на удаленный сервер по протоколу MQTT.

В качестве рассматриваемых аналогов были выбраны 4 приложения.

Первым приложением является Network Cell Info Lite. Данное приложение предназначено для сбора данных о сотовой сети, предоставляющее информацию об уровне сигнала, базовых станциях и типе сети. Интерфейс одного из окон представлен на рисунке 1.



Рисунок 1 – Приложение Network Cell Info Lite

Как можем видеть, в данном окне отображаются данные о подключенной и доступных базовых станциях и их параметры.

Вторым приложением является NetMonster. Данное приложение также реализует сбор данных о сотовой сети и окружающих базовых станциях. Интерфейс представлен на рисунке 2.

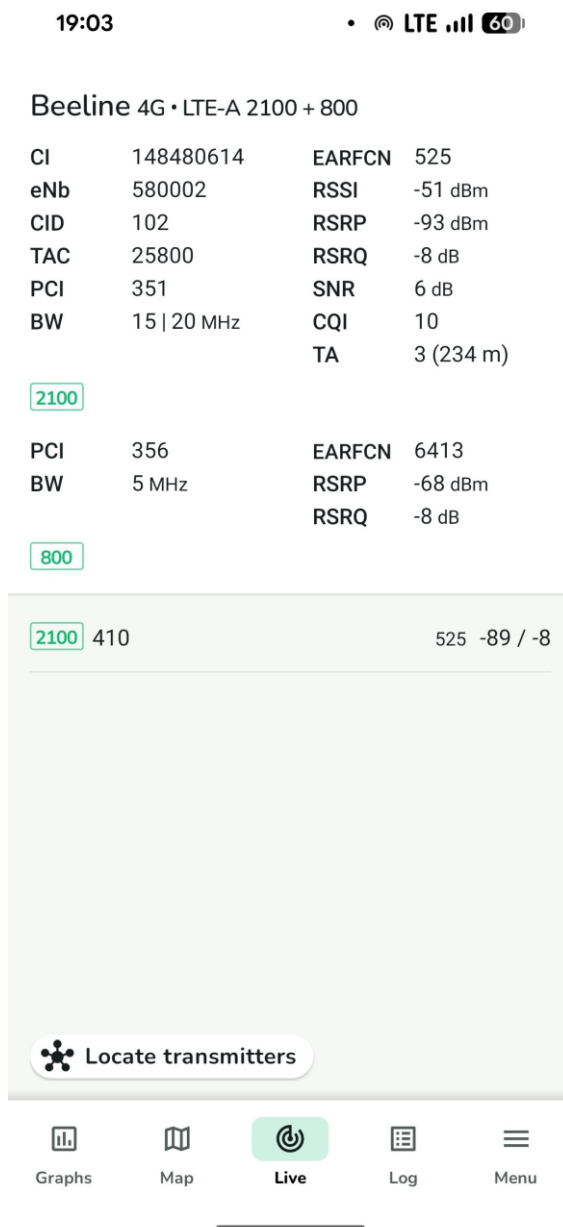


Рисунок 2 – Приложение NetMonster

Третьим приложением является WiFi Analyzer. Данное приложение, в отличие от двух предыдущих, предназначено для сканирования Wi-Fi сетей и

сбора их параметров (уровень сигнала, частота, загруженность каналов). Интерфейс данного приложения представлен на рисунке 3.



Рисунок 3 – Приложение WiFi Analyzer

Можем видеть, что в интерфейсе отображаются все доступные Wi-Fi сети, отсортированные по силе сигнала, их названия, частота и другие параметры.

Последним рассматриваемым приложением является GPS Test, в котором реализуется получение информации о геолокации устройства от разных систем навигации. Его интерфейс представлен на рисунке 4.



Рисунок 4 – Приложение GPS Test

Каждое из рассматриваемых приложений-аналогов является узкоспециализированным и не предоставляет возможность сбора данных всех типов (о сотовых сетях, Wi-Fi, местоположении и устройстве).

Для сравнения рассмотренных приложений выделен ряд критериев с установленной балльной оценкой:

- 0 – «полное отсутствие реализации»;
- 1 – «частичная реализация»;
- 2 – «полная реализация».

Далее, в таблице 1, приведено сравнение функциональных возможностей рассматриваемых приложений.

Таблица 1 – Сравнительная таблица приложений-аналогов

Параметр	Разрабатываемое приложение	Network Cell Info Lite	NetMonster	WiFi Analyzer	GPS Test
Сбор данных о сотовой сети	2	2	2	0	0
Сбор данных о Wi-Fi сетях	2	1	0	2	0
Сбор данных о местоположении	1	1	1	0	2
Сбор данных об устройстве	1	1	0	0	0
Отображение информации в интерфейсе пользователя	2	2	2	2	2
Работа в фоновом режиме	2	0	2	0	0
Итого	10	7	7	4	4

Проведённое сравнение показывает, что существующие приложения реализуют только отдельные аспекты сбора данных. Например, приложения Network Cell Info и NetMonster ориентированы только на анализ сотовой сети и получение местоположения, тогда как WiFi Analyzer предназначен только для анализа сетей Wi-Fi, а GPS Test – для работы с геолокацией.

В отличие от рассмотренных аналогов, разрабатываемое приложение позволяет производить сбор всех вышеперечисленных данных. Помимо этого, разрабатываемое приложение может продолжать работу в фоновом режиме и обновлять данные с заданной периодичностью, а также передавать их на сервер по протоколу MQTT.

Так, разрабатываемое приложение представляет собой комплексное решение, сочетающее функции сбора и передачи телеметрических данных.

В результате анализа аналогов установлено, что существующие решения не обеспечивают одновременной реализации всех функций в рамках одного

приложения. Это подтверждает актуальность разработки данного программного обеспечения.

Разрабатываемое приложение обладает более широкими функциональными возможностями за счёт интеграции различных источников данных и реализации механизма их передачи, что делает его универсальным инструментом для сбора и обработки телеметрической информации.

1.3 Анализ функциональных требований

В процессе анализа функциональных требований на основе поставленной задачи были выделены следующие сценарии работы приложения:

- запустить приложение;
- выдать разрешения;
- открыть настройки приложения;
- просмотреть текущие данные на экране;
- обновить данные вручную;
- просмотреть уведомления;
- изменить режим отправки данных;
- закрыть приложение;
- отправить данные в брокер;
- собрать телеметрические данные;
- получить данные об устройстве;
- получить данные о Wi-Fi;
- получить данные о состоянии сотовой сети;
- получить данные о местоположении;
- сформировать JSON;
- обновить данные на экране;
- отправить уведомление о статусе отправки.

Для демонстрации выделенных сценариев была составлена диаграмма вариантов использования [9, 10] (рисунок 5).

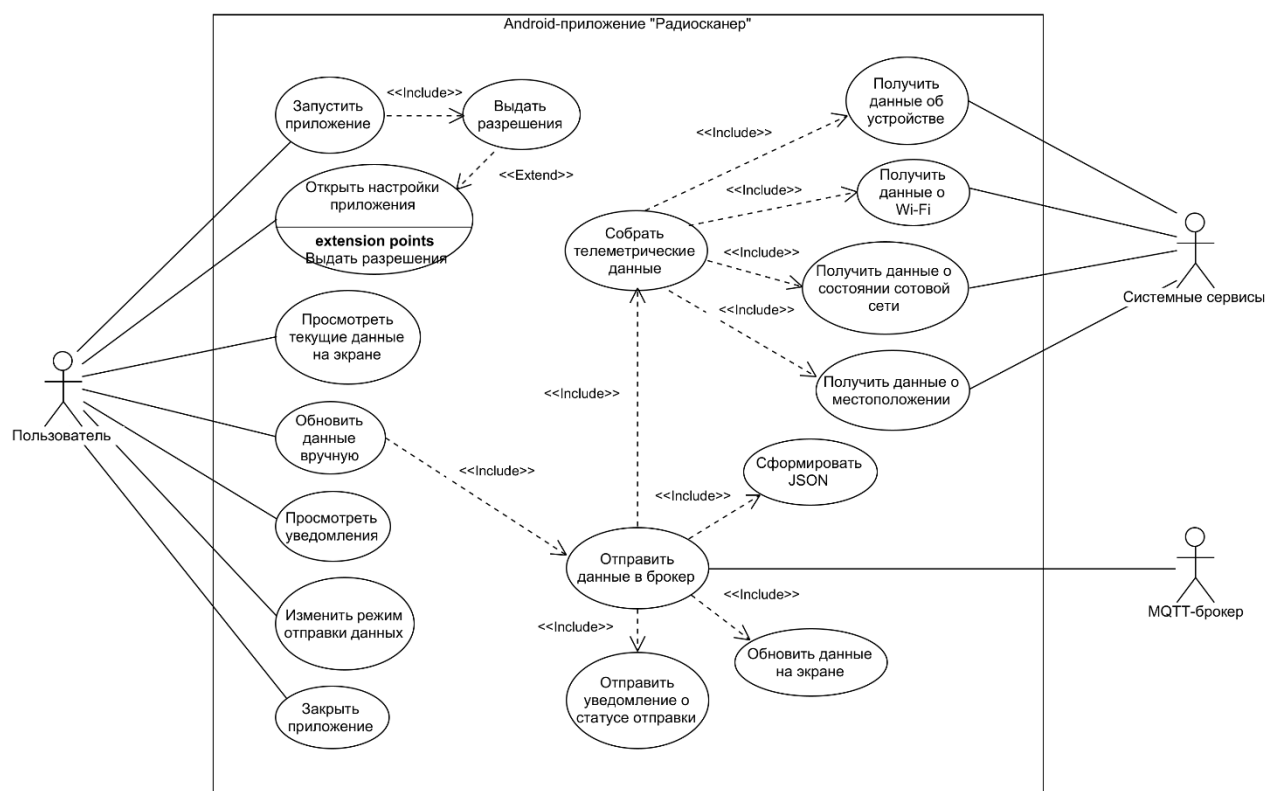


Рисунок 5 – Диаграмма вариантов использования

Ниже будет приведено описание некоторых наиболее важных сценариев вариантов использования (далее ВИ) разрабатываемого приложения. Для начала рассмотрим ВИ «Выдать разрешения» (рисунок 6) и «Просмотреть текущие данные на экране» (рисунок 7).

Прецедент: выдать разрешения
ID: 1
Краткое описание: после запуска приложения пользователь выдает разрешения для его корректной работы.
Действующие лица: пользователь, приложение
Предусловие: приложение запущено
Основной поток: <ol style="list-style-type: none"> 1. Приложение запрашивает у пользователя разрешение на доступ к данным о местоположении устройства. Пользователь предоставляет разрешение «только во время использования приложения». 2. Приложение запрашивает у пользователя разрешение на совершение звонков и управление ими. Пользователь предоставляет разрешение. 3. Приложение открывает настройки в разделе «Разрешение «Местоположение»». Пользователь предоставляет разрешение «в любом режиме».
Постусловие: приложение получило все необходимые разрешения для корректной работы.

Рисунок 6 – Описание ВИ «Выдать разрешения»

Прецедент: просмотреть текущие данные на экране
ID: 2
Краткое описание: пользователь просматривает на экране актуальные на момент обновления данные о радиообстановке ближайшего окружения
Действующие лица: пользователь, приложение
Предусловие: приложение запущено, разрешения выданы, данные получены
Основной поток: <ol style="list-style-type: none"> 1. Пользователь просматривает данные, которые были получены в момент их последнего запроса и отображены в соответствующих элементах интерфейса.
Постусловие: пользователь просмотрел текущие данные на экране

Рисунок 7 – Описание ВИ «Просмотреть текущие данные на экране»

Описанные выше варианты использования выполнялись при участии пользователя. Далее будут рассмотрены ВИ, которые выполняются в фоновом режиме без его участия. Такими вариантами использования являются «Собрать телеметрические данные» (рисунок 8) и «Сформировать JSON» (рисунок 9).

Прецедент: собрать телеметрические данные
ID: 3
Краткое описание: получение актуальной информации об устройстве, параметрах сети Wi-Fi, состоянии сотовой сети и местоположении
Действующие лица: приложение, системные сервисы
Предусловие: приложение запущено, разрешения выданы
Основной поток: <ol style="list-style-type: none"> 1. Раз в определенный промежуток времени или по запросу пользователя инициируется процесс сбора телеметрических данных. 2. Приложение обращается к системным сервисам Android. 3. Выполняется получение данных об устройстве. 4. Выполняется получение данных о сетях Wi-Fi. 5. Выполняется получение данных о состоянии сотовой сети. 6. Выполняется получение данных о местоположении устройства. 7. Все полученные данные записываются в единую переменную для дальнейшего использования в приложении.
Постусловие: новые данные о радиообстановке ближайшего окружения получены.

Рисунок 8 – Описание ВИ «Собрать телеметрические данные»

Прецедент: сформировать JSON
ID: 4
Краткое описание: преобразование собранных телеметрических данных в структурированный JSON-объект для последующей передачи через MQTT-протокол
Действующие лица: приложение
Предусловие: приложение запущено, разрешения выданы, телеметрические данные собраны
Основной поток: <ol style="list-style-type: none"> 1. После получения новых телеметрических данных инициируется формирование JSON. 2. Создается объект JSON. 3. В JSON добавляются данные об устройстве. 4. В JSON добавляются данные о местоположении устройства. 5. Если переключатель режима отправки данных «вкл», в JSON добавляются данные о Wi-Fi-сети. 6. Если переключатель режима отправки данных «вкл», в JSON добавляются данные о состоянии сотовой сети.
Постусловие: на основе новых полученных телеметрических данных сформирован JSON

Рисунок 9 – Описание ВИ «Сформировать JSON»

В отличие от двух предыдущих сценариев, результат выполнения которых был неочевиден для пользователя, в последнем он может явно наблюдать результат работы приложения – получение новых данных о радиообстановке ближайшего окружения (рисунок 10).

Прецедент: обновить данные на экране
ID: 5
Краткое описание: после запроса и получения новых данных они заменяют старые на экране
Действующие лица: приложение
Предусловие: приложение запущено, разрешения выданы, телеметрические данные собраны
Основной поток: <ol style="list-style-type: none"> 1. После получения новых данных компонент пользовательского интерфейса получает соответствующий сигнал. 2. Компонент пользовательского интерфейса получает новые данные и обновляет содержимое соответствующих текстовых полей.
Постусловие: данные на экране обновлены

Рисунок 10 – Описание ВИ «Обновить данные на экране»

Также, пользователь имеет возможность в любой момент самостоятельно инициировать процесс обновления телеметрических данных посредством нажатия на соответствующую кнопку (рисунок 11).

Прецедент: обновить данные вручную
ID: 6
Краткое описание: пользователь самостоятельно инициирует процесс обновления телеметрических данных
Действующие лица: пользователь, приложение
Предусловие: приложение запущено, разрешения выданы
Основной поток: <ol style="list-style-type: none"> 1. Пользователь нажимает на кнопку обновления данных. 2. Приложение запускает процесс сбора данных, формирования из них JSON и отправку в MQTT-брокер, а также отправку сигнала компоненту пользовательского интерфейса о том, что данные были обновлены.
Постусловие: инициирован процесс обновления телеметрических данных

Рисунок 11 – Описание ВИ «Обновить данные на экране»

Спроектированные варианты использования будут реализованы в ходе разработки приложения.

1.4 Анализ нефункциональных требований

Разрабатываемое приложение должно соответствовать ряду нефункциональных требований для обеспечения его нормальной работы:

- удобство использования: приложение должно быть интуитивно понятным и удобным для пользователя, а также иметь простой и наглядный интерфейс, в котором в структурированном виде будут отображаться все собранные данные [11];
- производительность: приложение должно обеспечивать сбор, обработку и передачу данных без значительных задержек и без влияния на производительность устройства;
- совместимость: приложение должно корректно работать на устройствах с различными версиями Android (как минимум с двумя последними).

Соблюдение вышеперечисленных нефункциональных требований позволит обеспечить стабильную и эффективную работу разрабатываемого приложения.

Удобный и наглядный интерфейс упростит взаимодействие пользователя с приложением и позволяет быстро воспринимать собранные телеметрические данные. Требование к производительности обеспечит своевременный и быстрый сбор, обработку и передачу информации. Совместимость с актуальными версиями Android, в свою очередь, повысит универсальность приложения и обеспечивает возможность его корректного использования на современных мобильных устройствах.

2 Проектирование приложения

2.1 Проектирование структуры приложения

На основе результатов проведенного анализа требований и предметной области была предложена структура мобильного приложения для сбора данных о радиообстановке ближайшего окружения, состоящая из следующих функциональных блоков:

- сбор данных (о сотовой сети, Wi-Fi сетях, местоположении, устройстве);
- сбор телеметрии и формирование JSON;
- хранение результата последнего сбора данных;
- хранение состояния приложения;
- пользовательский интерфейс;
- передача телеметрических данных;
- формирование и отправка уведомлений.

Структура приложения представлена на рисунке 12.

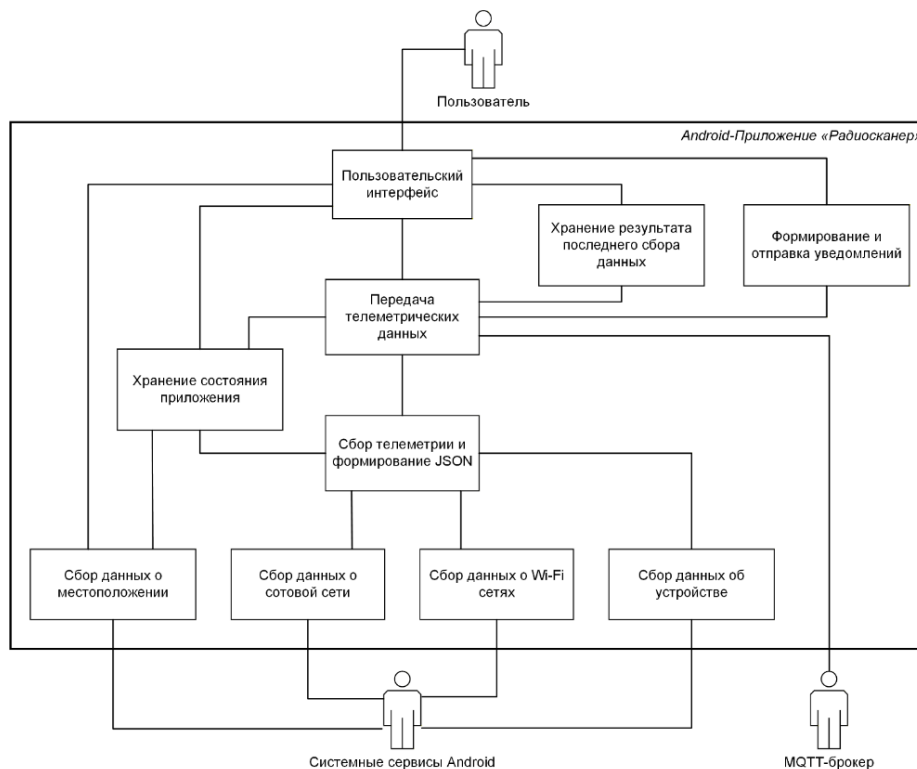


Рисунок 12 – Структура приложения «Радиосканер»

На схеме программная система представлена в виде набора взаимосвязанных функциональных блоков.

Взаимодействие с системой инициируется пользователем через пользовательский интерфейс. Он отвечает за отображение актуальных на момент обновления телеметрических данных, обработку пользовательских действий, а также запуск фоновых сервисов, к которым относятся передача телеметрических данных и сбор данных о местоположении.

Фоновый сбор данных о местоположении предназначен для получения координат устройства с использованием системных средств геолокации. Полученные данные сохраняются в блок хранения состояния приложения и передаются в блок пользовательского интерфейса посредством широковещательных сообщений.

Блок передачи телеметрических данных инициирует периодический сбор и передачу телеметрических данных в MQTT-брокер [3]. Раз в определенный промежуток времени или по запросу пользователя происходит обращение к блоку сбора телеметрии и формирования JSON [5] для выполнения соответствующих операций. Данные, не упакованные в формат JSON, блок передачи телеметрических данных сохраняет в блок хранения результата последнего сбора данных для последующего доступа к ним из блока пользовательского интерфейса, а данные в формате JSON он публикует в MQTT-брокер, с которым предварительно было установлено соединение. При этом если соединение не было успешно установлено, данные сохраняются в локальную переменную до следующей успешной попытки подключения, после которой в MQTT-брокер будут отправлены не только актуальные данные, но и те, которые были накоплены за время отсутствия соединения. После выполненного обновления блок передачи телеметрических данных сохраняет в блок хранения состояния приложения время обновления, а также отправляет широковещательное сообщение об обновлении в блок пользовательского интерфейса.

За доступ к данным устройства отвечают сборщики данных. Они инкапсулируют взаимодействие с системными API Android и обеспечивают получение информации о сотовой связи, Wi-Fi сетях, устройстве.

Блок сбора телеметрии и формирования JSON отвечает за опрос сборщиков данных и формирование структурированного JSON-объекта. В процессе своей работы он обращается к блоку хранения состояния приложения для получения координат, статуса отправки данных и идентификатора текущей сессии.

Блок уведомлений реализует механизм создания и отображения системных уведомлений.

Для наглядного представления взаимодействия между блоками приложения и внешними сущностями в процессе сбора телеметрических данных, их отправки в MQTT-брокер и отображении в пользовательском интерфейсе, была составлена диаграмма коммуникаций [9, 10] (рисунок 13).

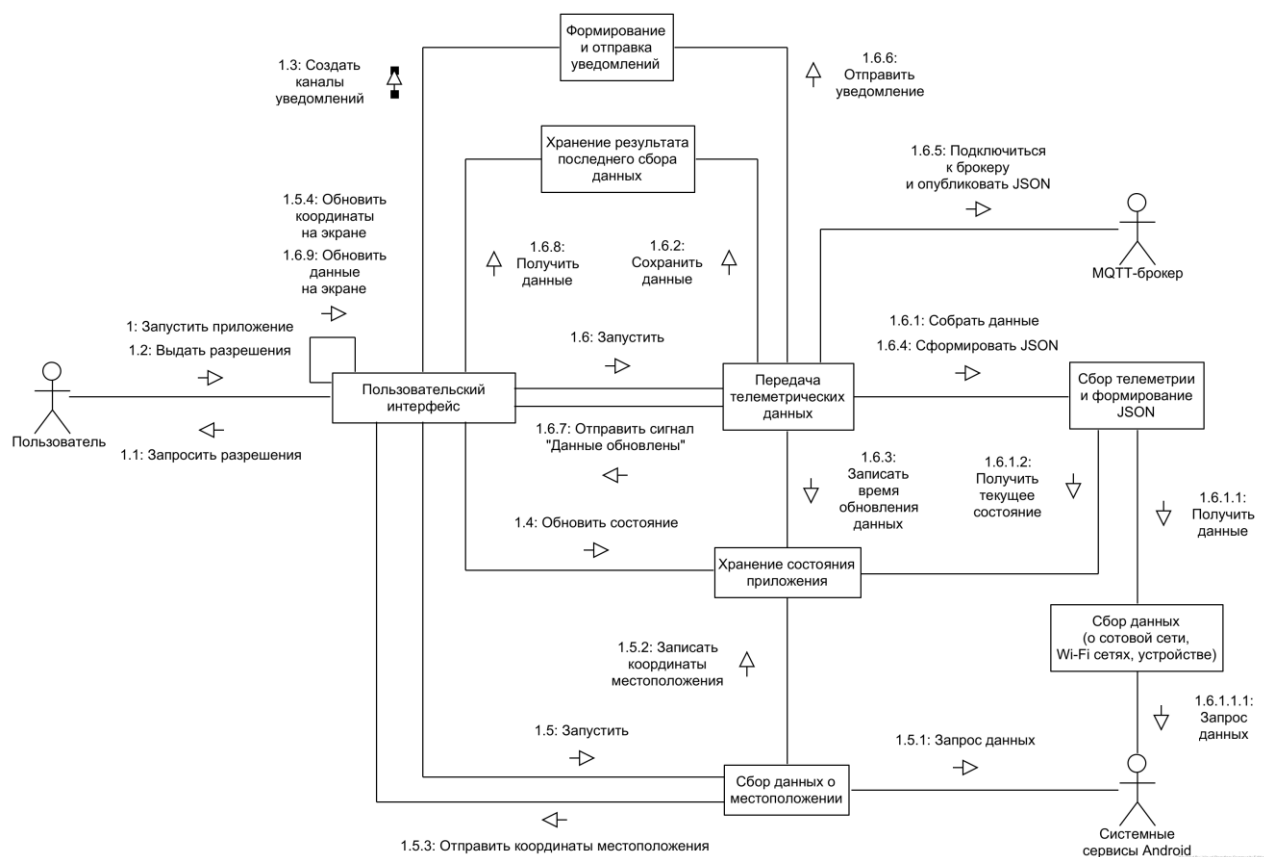


Рисунок 13 – Диаграмма коммуникаций

Взаимодействие начинается с действий пользователя, который запускает приложение, а затем предоставляет необходимые разрешения для корректной работы.

После запуска приложения выполняется ряд действий: создаются каналы уведомлений, в блок хранения состояния приложения записывается статус переключателя режима отправки данных, а также номер текущей сессии. Далее запускается сбор данных о местоположении и передача телеметрических данных, которые продолжают свою работу в фоновом режиме.

Сбор данных о местоположении действует независимо от других функций, так как получение координат носит асинхронный характер, требует фонового выполнения и может быть недоступно в момент запуска обновления остальных телеметрических данных. В связи с этим данное действие вынесено в отдельный блок, который регулярно получает данные о местоположении, отправляет их в пользовательский интерфейс и сохраняет в блок хранения состояния приложения для дальнейшего формирования JSON. Такой подход снижает нагрузку на устройство и не заставляет системные сервисы геолокации заново включаться при каждом запросе обновления данных.

Передача телеметрических данных также работает в фоновом режиме и координирует сбор, отправку и сохранение телеметрических данных. Изначально запрос о сборе данных отправляется в блок сбора телеметрии и формирования JSON, который опрашивает сборщики данных (о сотовой сети, Wi-Fi сетях, устройстве) и забирает данные о состоянии приложения (координаты, статус переключателя режима отправки данных, номер сессии). Затем весь набор полученных данных сохраняется в блок хранения результата последнего сбора данных для дальнейшего отображения в пользовательском интерфейсе, а в блок хранения состояния приложения записывается время обновления данных. После этого происходит повторное обращение к блоку сбора телеметрии и формирования JSON для формирования JSON, который далее отправляется в MQTT-брокер после успешного подключения. Для отправки уведомления со статусом отправки данных в брокер происходит обращение в

Таблица 2 – Описание классов спроектированного мобильного приложения

Класс	Назначение	Зона ответственности	Основные атрибуты	Основные методы
MainActivity	Реализует пользовательский интерфейс приложения и запуск фоновых сервисов	Пользовательский интерфейс	gsm, wifi, notifications, infoCollector, appStateStore	onCreate(), onStart(), onStop(), renderTelemetry(), requestManualRefresh()
LocationService	Получает координаты устройства в фоновом режиме	Сбор данных о местоположении	fusedLocationClient, locationCallback, locationManager	startLocationUpdates(), updateLocation(), getLastKnownLocation()
MqttService	Координирует сбор телеметрии, ее сохранение, отправку на MQTT-брокер в формате JSON	Передача телеметрических данных	mqtt, infoCollector, serviceScope, publishJob, refreshMutex	onStartCommand(), refreshTelemetryOnce(), publishJsonToMqtt()
Mqtt	Выполняет подключение и публикацию данных в MQTT	Передача телеметрических данных	client, options	connectToMqttBroker(), publishJsonObject()
InfoCollector	Собирает телеметрические данные от сборщиков и формирует единый JSON	Сбор телеметрии и формирование JSON	gsm, wifi, device, appStateStore	collectSnapshot(), snapshotToJson()
AppStateStore	Хранит текущие данные о состоянии приложения	Хранение состояния приложения	prefs	setTrackId(), getTrackId(), setGpsLatitude(), getGpsLatitude()
TelemetryRepository	Хранит последний снимок телеметрии	Хранение результата последнего сбора данных	lastSnapshot	—
TelemetrySnapshot	Хранит последний сохраненный полный набор телеметрических данных		deviceInfo, gsm, signal, cellTowers, wifiAccessPoints	—
Gsm	Получает данные о GSM-сети и вышках	Сбор данных о сотовой сети	context, telephonyManager	getGsmInfo(), getSignalInfo(), getAllCellTowersInfo()
TowerInfo	Хранит данные о базовой станции		cellId, lac, tac, earfcn, bandwidth	—

Продолжение таблицы 2

Класс	Назначение	Зона ответственности	Основные атрибуты	Основные методы
SignalInfo	Хранит данные об уровне сигнала	Сбор данных о сотовой сети	signalType, signalStrengthDbm, signalStatus	—
GsmInfo	Хранит данные о GSM-сети		networkOperatorName, simOperatorName, radioType	—
Wifi	Получает данные о Wi-Fi сетях	Сбор данных о Wi-Fi сетях	context, wifiManager	getWifiAccessPointsInfo()
WifiAccessPointInfo	Хранит данные о Wi-Fi сетях	Сбор данных о Wi-Fi сетях	ssid, bssid, frequency, signalStrength	—
Device	Получает данные об устройстве	Сбор данных об устройстве	context	getDeviceInfo()
DeviceInfo	Хранит данные об устройстве		brand, model, androidVersion	—
Notifications	Создает и отображает системные уведомления	Формирование и отправка уведомлений	notificationManager, context	createNotificationChannels(), sendStateNotification()
AppBroadcasts	Содержит константы broadcast-сообщений	Хранение констант для broadcast-сообщений	ACTION_LOCATION_UPDATED, ACTION_TELEMETRY_REFRESHED	—

Так, взаимодействие классов, каждый из которых имеет свою зону ответственности, реализует полноценный функционал приложения.

2.2 Проектирование интерфейса приложения

Для приложения для сбора данных о радиообстановке ближайшего окружения необходимо спроектировать прототип интерфейса в соответствии со сценариями использования.

Интерфейс должен наглядно демонстрировать следующие актуальные телеметрические данные:

- о местоположении (от GPS и полученные на основе сети);
- о сотовой сети и сигнале;
- о базовых станциях;
- о Wi-Fi сетях.

Также в интерфейсе должны присутствовать кнопки для ручной инициализации обновления данных и выхода из приложения, время последнего обновления данных, а также переключатель режима отправки данных.

Прототип интерфейса представлен на рисунке 15.



Рисунок 15 – Прототип интерфейса приложения

Большую часть интерфейса занимают текстовые области, в которых будут отображаться собранные телеметрические данные:

- 3, 4 – координаты местоположения, полученные от GPS и рассчитанные на основе сети;
- 5, 6 – данные о сотовой сети и о типе и силе сигнала;
- 7 – данные о подключенной базовой станции;
- 8 – данные об окружающих Wi-Fi сетях.

В верхней части экрана, под номерами 1 и 2, отображается информация о приложении и логотип компании-изготовителя.

В нижней части экрана располагаются элементы для взаимодействия пользователя с приложением:

- 9 – кнопка для инициализации обновления данных пользователем вручную;
- 10 – переключатель режима отправки данных;
- 11 – кнопка для остановки работы приложения и выхода.

Таким образом, спроектированный прототип интерфейса приложения является простым, интуитивно понятным и информативным, а также предоставляет возможность для реализации всех поставленных функциональных требований.

3 Реализация приложения

В ходе выполнения бакалаврской работы было разработано андроид-приложение сбора данных о радиообстановке ближайшего окружения, код которого приведен в приложении А.

Для отображения взаимодействия частей программы была составлена диаграмма компонентов [9, 10] (рисунок 16).

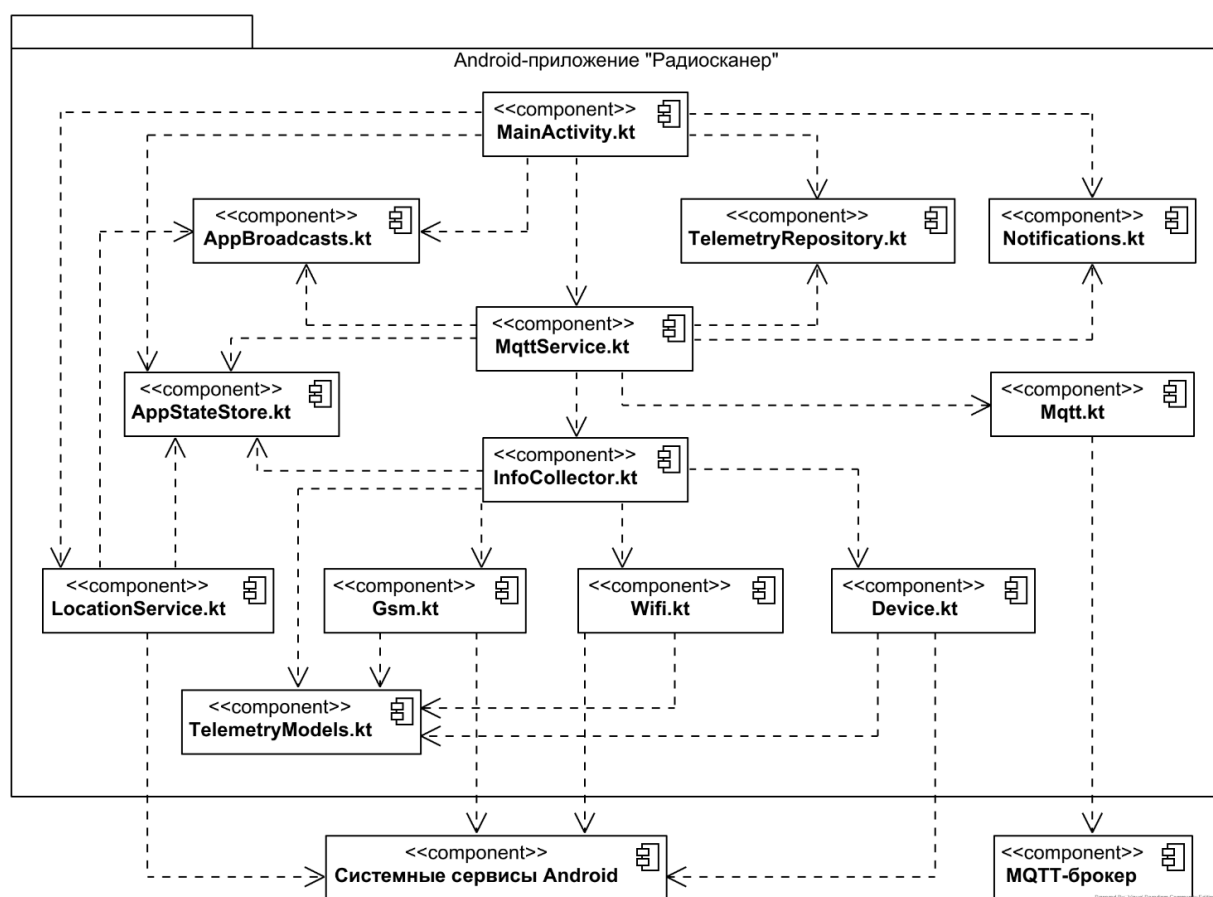


Рисунок 16 – Диаграмма компонентов

Ниже, в таблице 3, представлено краткое описание всех компонентов системы.

Таблица 3 – Описание компонентов мобильного приложения

Компонент	Описание
MainActivity.kt	Основной компонент пользовательского интерфейса приложения. Отвечает за отображение телеметрических данных, обработку действий пользователя, запуск сервисов и обновление интерфейса.

Продолжение таблицы 3

Компонент	Описание
LocationService.kt	Фоновый сервис, выполняющий получение координат устройства с использованием системных средств геолокации Android.
MqttService.kt	Фоновый сервис, обеспечивающий периодический сбор телеметрии, формирование JSON и отправку данных на MQTT-брокер.
InfoCollector.kt	Компонент бизнес-логики, агрегирующий данные из различных источников и формирующий объект телеметрии и JSON-представление данных.
AppStateStore.kt	Компонент хранения состояния приложения, реализующий сохранение и получение текущих параметров работы приложения.
TelemetryRepository.kt	Компонент, предназначенный для хранения последнего актуального снимка телеметрических данных.
AppBroadcasts.kt	Компонент, содержащий константы широковещательных сообщений (broadcast), используемых для взаимодействия между компонентами приложения.
Gsm.kt	Компонент получения данных о сотовой сети, уровне сигнала и базовых станциях.
Wifi.kt	Компонент получения информации о доступных Wi-Fi сетях и параметрах беспроводного соединения.
Device.kt	Компонент получения информации об устройстве, включая модель, производителя и версию операционной системы.
Notifications.kt	Компонент создания и отображения системных уведомлений Android.
Mqtt.kt	Компонент, реализующий подключение к MQTT-брокеру и публикацию телеметрических данных.
TelemetryModels.kt	Набор моделей данных, используемых для структурированного хранения телеметрической информации приложения.
Системные сервисы Android	Внешний платформенный компонент, предоставляющий доступ к геолокации, телекоммуникационным данным и Wi-Fi интерфейсам устройства.
MQTT-брокер	Внешняя система, принимающая и обрабатывающая телеметрические сообщения, отправляемые мобильным приложением.

Так, на языке Kotlin [7] было реализовано андроид-приложение сбора данных о радиообстановке ближайшего окружения, состоящее из нескольких компонентов, выполняющих определенные функции в рамках своей зоны ответственности.

Интерфейс приложения был реализован в среде Android Studio, позволяющей работать с макетом как в графическом виде, так и в виде кода. Окно разработки интерфейса в Android Studio [8] представлено на рисунке 17.

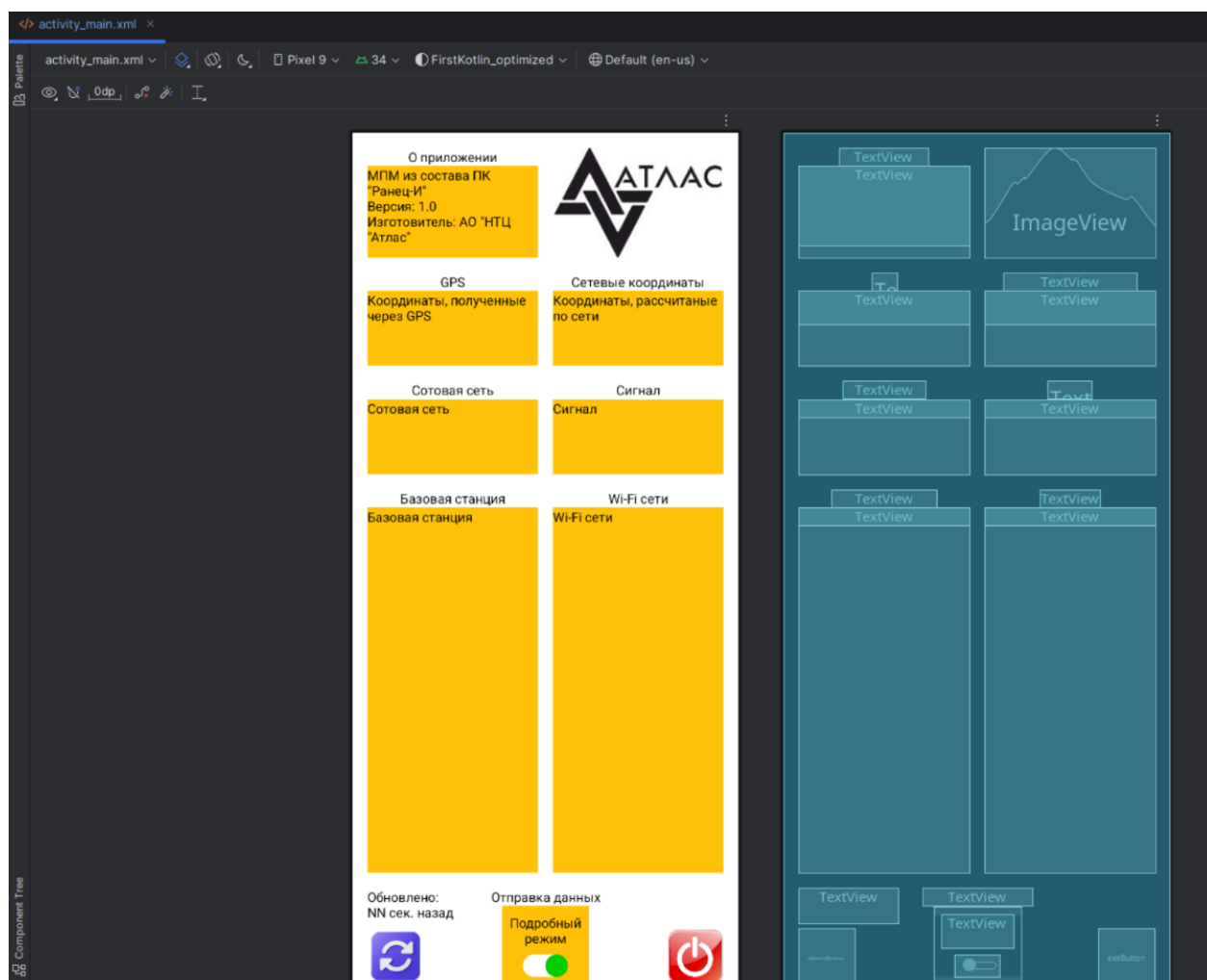


Рисунок 17 – Окно разработки интерфейса в среде Android Studio

Для реализации интерфейса использовались следующие элементы:

- ScrollView [12] – контейнеры для отображения текстовой информации переменного объёма с возможностью прокрутки содержимого. Используются для вывода данных о координатах GPS, сетевых координатах, данных о сотовой сети, сигнале, базовых станциях и Wi-Fi сетях;
- TextView – используются для отображения текстовой информации, заголовков разделов и значений телеметрических данных;
- ImageView – используется для отображения логотипа;
- ImageButton – кнопки для инициализации ручного обновления данных и выхода из приложения;
- Switch – переключатель автоматического режима передачи данных.

Вид интерфейса пользователя после запуска приложения и корректного сбора всех данных представлен на рисунке 18.



Рисунок 18 – Интерфейс пользователя в ходе работы приложения

Таким образом, можно видеть, что в пользовательском интерфейсе в соответствующих блоках отображаются все актуальные собранные телеметрические данные:

- координаты местоположения;
- данные о сотовой сети;
- данные о сигнале;

- данные о базовых станциях;
- данные о Wi-Fi сетях;
- время последнего обновления.

Также в интерфейсе имеются кнопки для ручного обновления данных и выхода из приложения, а также переключатель режима отправки данных.

Реализованный интерфейс является простым, информативным и наглядным для конечного пользователя.

Также в приложении были реализованы уведомления, сообщающие:

- о фоновой работе приложения;
- о статусе и времени обновления телеметрии;
- о состоянии подключения и отправки данных в MQTT-брокер.

Демонстрация работы уведомлений представлена на рисунках 19 и 20.

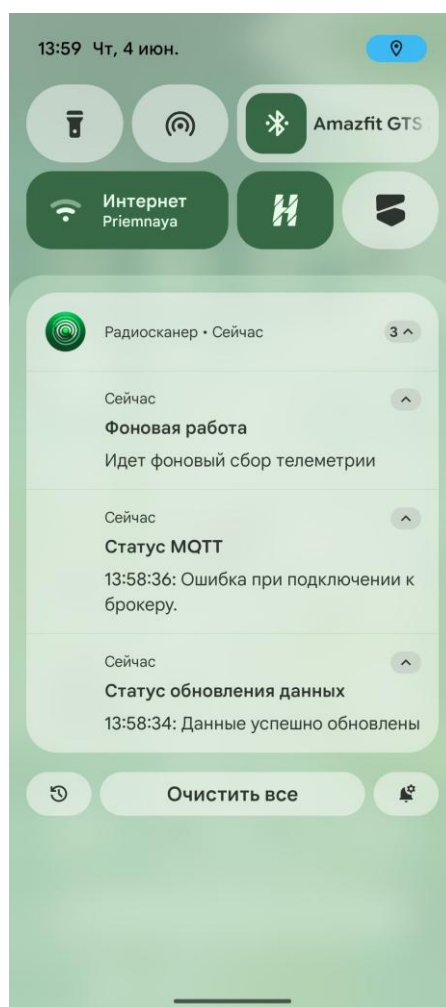


Рисунок 19 – Уведомление об ошибке подключения к брокеру

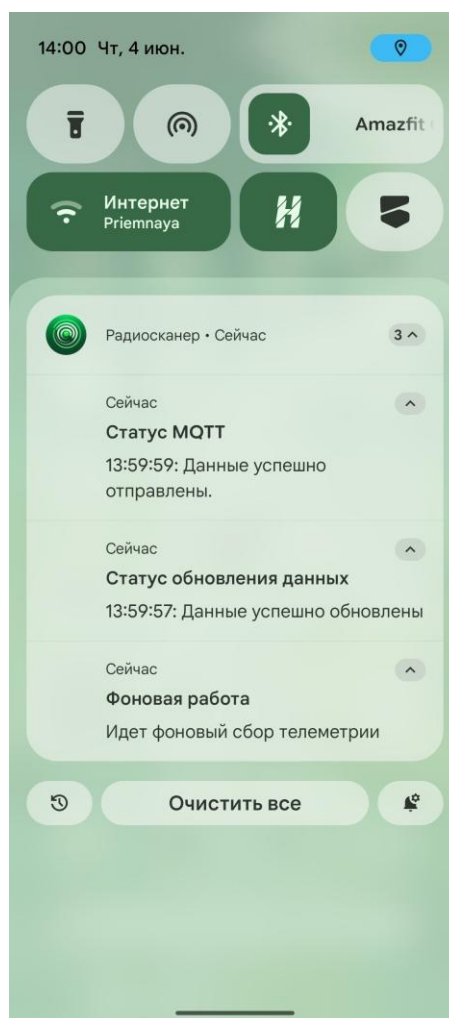


Рисунок 20 – Уведомление об успешной отправке данных в брокер

Можем видеть, что в обоих случаях приложение работает в фоновом режиме и успешно обновляет данные. Однако, в первом случае возникла ошибка при подключении к брокеру, следовательно, данные не были отправлены. Во втором же случае подключение было установлено и обновленные телеметрические данные были успешно отправлены в нужный топик.

При этом стоит отметить, что при неудачной попытке подключения к брокеру собранные данные не удаляются, а сохраняются в локальной переменной внутри приложения. После восстановления соединения в брокер будут отправлены не только актуальные телеметрические данные, но и накопленные ранее в следствие отсутствия подключения. Если количество накопленных сообщений превысит лимит, то самое неактуальное сообщение будет удалено. Также буфер сообщений очищается при выходе или перезапуске приложения.

Результат отправки телеметрических данных в MQTT-брокер в подробном режиме представлен на рисунке 21.

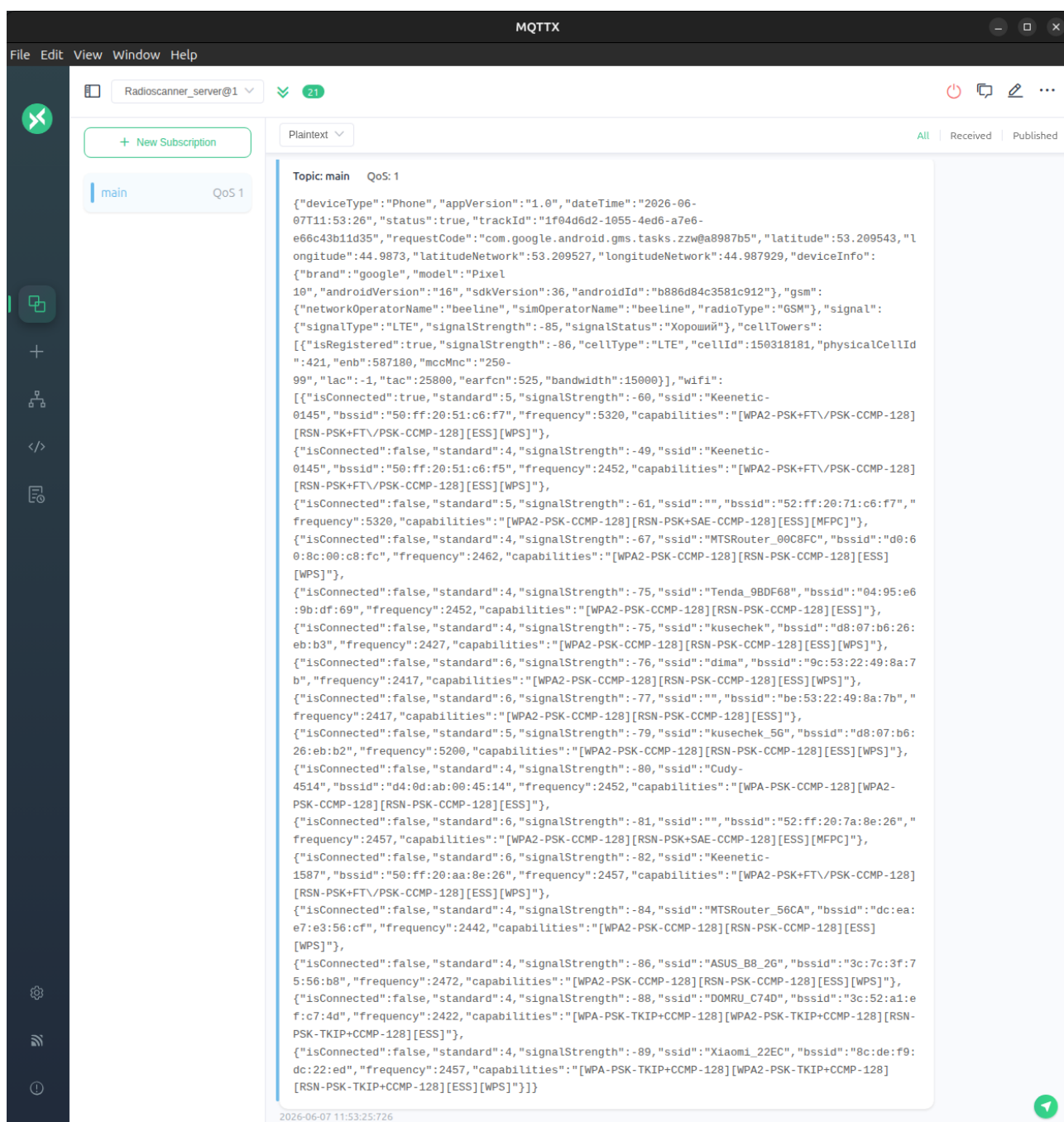


Рисунок 21 – Отправка данных в брокер в подробном режиме

Можем видеть все собранные телеметрические данные: служебные данные, дата и время, сведения об устройстве, сведения о сигнале, состоянии сотовой сети, о базовой станции, данные обо всех Wi-Fi сетях в окружении.

Ниже, на рисунке 22, представлен результат отправки данных в непомощном режиме.

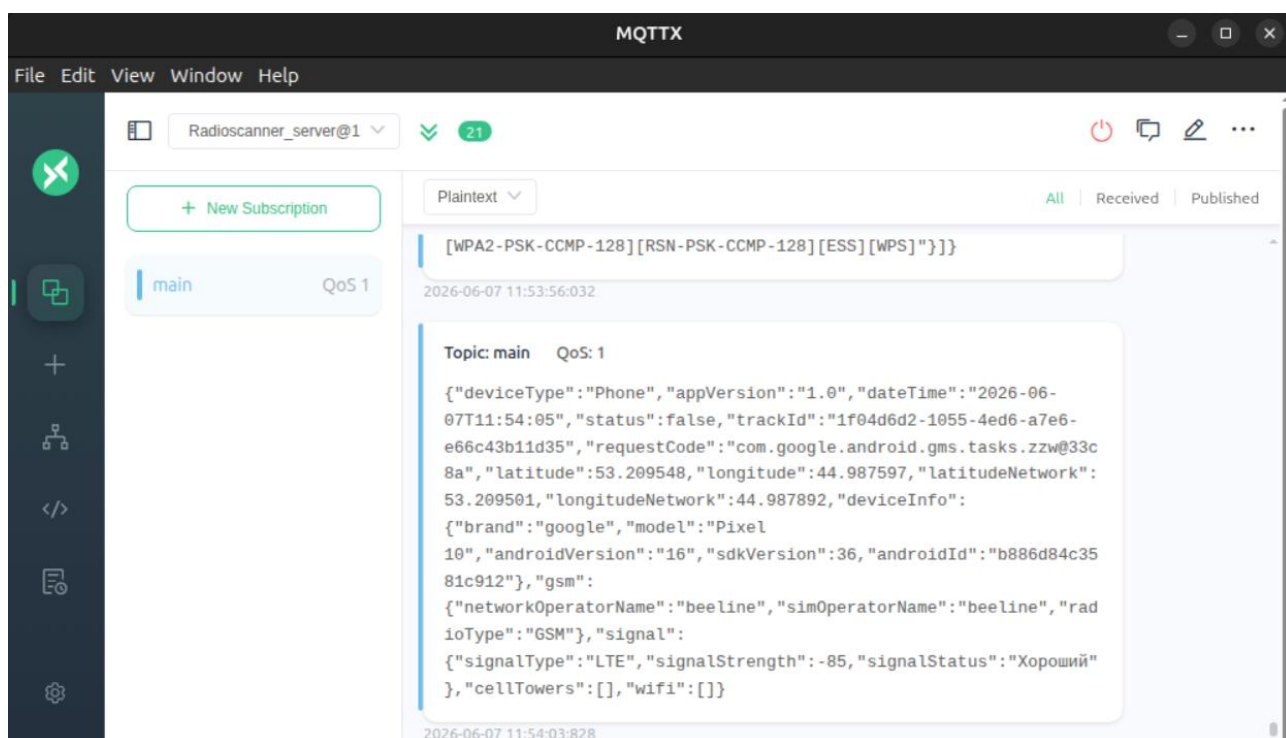


Рисунок 22 – Отправка данных в брокер в непомощном режиме

Можем видеть результат отправки, когда переключатель «Помощный режим» выключен. В этом случае в сообщение не включаются данные о базовой станции и окружающих Wi-Fi сетях.

Таким образом, андроид-приложение сбора данных о радиообстановке ближайшего окружения было успешно реализовано: был написан код приложения в соответствии со спроектированной структурой, реализован графический интерфейс пользователя, а также продемонстрирована функция отправки данных в MQTT-брокер.

4 Контроль качества программного обеспечения

4.1 Анализ метрик

На примере компонента Wifi.kt проведем расчет часто используемой метрики под названием «Комплексный показатель качества кода», или Maintainability Index [13], которая представлена следующей формулой:

$$MI = \text{MAX}(0, \frac{171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(\text{LoC})}{171} * 100), (1)$$

где HV (Halstead Volume – вычислительная сложность) [14] – чем больше операторов, тем больше значение этой метрики;

CC (Cyclomatic Complexity – цикломатическая сложность) [15] – показывает структурную сложность кода, т. е. количество различных ветвей в коде;

LoC (Lines of Code) – количество строк кода [16].

Код компонента, для которого будет рассчитана данная метрика, представлен в листинге 1.

Листинг 1 – Код компонента Wifi

```
class Wifi(private val context: Context) {
    private val wifiManager by lazy {
        context.getSystemService(Context.WIFI_SERVICE) as
        WifiManager
    }

    fun hasWifiStatePermission(): Boolean {
        return ContextCompat.checkSelfPermission(
            context,
            ACCESS_WIFI_STATE
        ) == PackageManager.PERMISSION_GRANTED
    }

    @RequiresApi(Build.VERSION_CODES.R)
    fun getWifiAccessPointsInfo(): List<WifiAccessPointInfo> {
        if (!hasWifiStatePermission()) return emptyList()

        return try {
            val wifiInfo = wifiManager.connectionInfo
            val currentSsid =
wifiInfo.ssid.removeSurrounding("\"")
            val currentBssid = wifiInfo.bssid
```

```

val scanResults = if (hasWifiStatePermission()) {
    wifiManager.scanResults
} else {
    emptyList()
}

scanResults
    .asSequence()
    .map { scanResult ->
        WifiAccessPointInfo(
            isConnected = scanResult.SSID ==
currentSSID &&
                scanResult.BSSID == currentBssid,
            standard =
                scanResult.wifiStandard,
            signalStrength = scanResult.level,
            ssid = scanResult.SSID,
            bssid = scanResult.BSSID,
            frequency = scanResult.frequency,
            capabilities = scanResult.capabilities
        )
    }
    .sortedWith(
        compareByDescending<WifiAccessPointInfo> {
it.isConnected }
            .thenByDescending { it.signalStrength }
    )
    .toList()
} catch (_: SecurityException) {
    emptyList()
}
}
}

```

Для начала рассчитаем параметр вычислительной сложности кода.

4.1.1 Вычислительная сложность

Данные для расчета словаря и длины программы сведены в таблице 4.

Таблица 4 – Данные для расчета словаря и длины

Операторы	№	Количество	Операнды	№	Количество
.	1	26	scanResult	1	9
(	2	19	context	2	3
)	3	19	wifiManager	3	3
{	4	11	hasWifiStatePermission	4	3
}	5	11	WifiAccessPointInfo	5	3
=	6	11	emptyList	6	3

Продолжение таблицы 4

Операторы	№	Количество	Операнды	№	Количество
,	7	7	wifiInfo	7	3
val	8	6	scanResults	8	3
:	9	4	Context	9	2
return	10	3	currentSsid	10	2
==	11	3	ssid	11	2
private	12	2	currentBssid	12	2
Fun	13	2	bssid	13	2
<	14	2	isConnected	14	2
>	15	2	SSID	15	2
if	16	2	BSSID	16	2
class	17	1	signalStrength	17	2
by	18	1	frequency	18	2
lazy	19	1	capabilities	19	2
as	20	1	it	20	2
@	21	1	Wifi	21	1
!	22	1	getSystemService	22	1
try	23	1	WIFI_SERVICE	23	1
else	24	1	WifiManager	24	1
->	25	1	Boolean	25	1
&&	26	1	ContextCompat	26	1
catch	27	1	checkSelfPermission	27	1
			ACCESS_WIFI_STATE	28	1
			PackageManager	29	1
			PERMISSION_GRANTED	30	1
			RequiresApi	31	1
			Build	32	1
			VERSION_CODES	33	1
			R	34	1
			getWifiAccessPointsInfo	35	1
			List	36	1
			connectionInfo	37	1
			removeSurrounding	38	1
			«\»»»	39	1
			asSequence	40	1
			map	41	1
			standard	42	1
			wifiStandard	43	1
			level	44	1
			sortedWith	45	1
			compareByDescending	46	1
			thenByDescending	47	1
			toList	48	1
			_	49	1
			SecurityException	50	1
Всего	η₁ = 27	N₁ = 141	Всего	η₂ = 50	N₂ = 84

На основе полученных значений можем рассчитать словарь и длину программы.

Словарь программы рассчитывается по формуле:

$$\eta = \eta_1 + \eta_2 = 27 + 50 = 77,$$

где η_1 – число простых (отдельных) операторов и операций в программе;

η_2 – число простых (отдельных) операндов (констант и переменных) в программе.

Длина программы рассчитывается по формуле:

$$N = N_1 + N_2 = 141 + 84 = 225,$$

где N_1 – общее число простых (отдельных) операторов и операций в программе;

N_2 – общее число простых (отдельных) операндов в программе.

Теперь подставим полученные значения словаря и длины программы в формулу вычислительной сложности [14]:

$$HV = N * \log_2 \eta = 225 * 6.2668 = 1410.03.$$

Так, вычислительная сложность для компонента Wifi.kt составляет 1410.03. Данное значение свидетельствует о его довольно компактной структуре и относительно небольшой сложности.

Далее переходим к вычислению цикломатической сложности.

4.1.2 Цикломатическая сложность

Для расчета цикломатической сложности компонента Wifi.kt необходимо построить управляющий граф. Он представляет собой ориентированный граф, вершины которого соответствуют отдельным участкам программы, а дуги отражают возможные переходы управления между ними в процессе выполнения. Такой граф позволяет наглядно представить структуру алгоритма и определить количество независимых путей выполнения программы.

Управляющий граф представлен на рисунке 23.

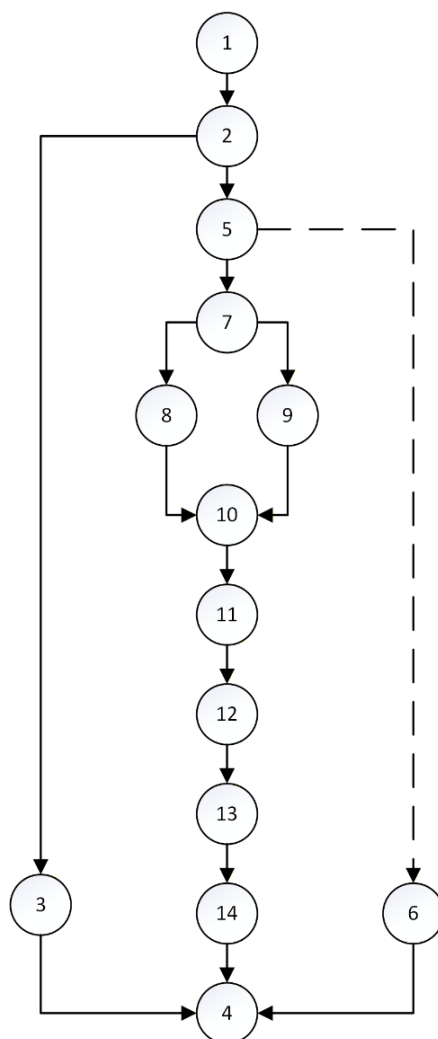


Рисунок 23 – Управляющий граф компонента Wifi

Расшифровка вершин приведена в таблице 5.

Таблица 5 – Расшифровка вершин управляющего графа компонента Wifi

№	Описание действия
1	Начало метода <code>getWifiAccessPointsInfo()</code>
2	Проверка <code>!hasWifiStatePermission()</code>
3	Возврат <code>emptyList()</code> при отсутствии разрешения
4	Конец метода
5	Начало блока <code>try</code> , получение <code>wifiInfo</code> , <code>currentSsid</code> , <code>currentBssid</code>
6	Обработка <code>SecurityException</code>
7	Проверка разрешения перед получением <code>scanResults</code>
8	Получение <code>wifiManager.scanResults</code>
9	Присваивание <code>emptyList()</code>
10	Преобразование списка в последовательность <code>asSequence()</code>
11	Формирование объектов <code>WifiAccessPointInfo</code> через <code>map</code>
12	Сортировка через <code>sortedWith()</code>
13	Преобразование результата в список <code>toList()</code>
14	Возврат списка точек доступа

Далее подставляем количество дуг и вершин в формулу для расчета цикломатической сложности:

$$CC = e - v + 2 * p = 16 - 14 + 2 * 1 = 4,$$

где e – число дуг ориентированного графа программы;

v – число вершин ориентированного графа программы;

p – число компонент связности ориентированного графа программы.

Так, цикломатическая сложность [15] равна четырем, что является показателем низкой структурной сложности кода. Невысокое значение метрики свидетельствует о хорошей читаемости кода, простоте тестирования и сопровождения компонента.

4.1.3 Комплексный показатель качества кода

Получив последнее значение – количество строк кода (LoC), равное 63, можем вычислить комплексный показатель качества кода [13], используя формулу (1):

$$MI = \text{MAX}(0, \frac{171 - 5.2 * \ln(1410,03) - 0.23 * 4 - 16.2 * \ln(63)}{171} * 100) = 38.17.$$

Таким образом, проанализированный компонент обладает приемлемым уровнем сопровождаемости для специализированного компонента мобильного приложения, предназначенного для работы с системными средствами Android и получения информации о доступных Wi-Fi сетях. Значение показателя обусловлено использованием API Android, обработкой разрешений безопасности, а также логикой формирования и сортировки данных о точках доступа.

При этом компонент имеет небольшую цикломатическую сложность и компактную структуру. Код разделен на отдельные методы, отвечающие за проверку разрешений и получение информации о Wi-Fi сетях, что повышает его читаемость и упрощает дальнейшую модификацию.

4.2 Инструментальное тестирование

Инструментальное тестирование – это метод проверки программного обеспечения (чаще всего мобильных приложений) непосредственно на реальном устройстве или эмуляторе, требующий специальной среды исполнения [18, 19].

Такие тесты работают с настоящими компонентами системы:

- экранами приложения (Activity), фоновыми службами (Service);
- разрешениями пользователя;
- системными сервисами (геолокация, связь, Wi-Fi).

Инструментальное тестирование было выбрано по следующим причинам:

- позволяет тестировать работу с реальными возможностями устройства;
- проверяет взаимодействие компонентов приложения;
- учитывает влияние разрешений и системной среды;
- обеспечивает проверку поведения приложения в условиях, близких к реальным.

Описание проведенных тестов приведено в таблице 6.

Таблица 6 – Таблица проведенных инструментальных тестов

№	Название теста	Назначение и краткое описание	Начальные условия	Ожидаемый результат	Фактический результат	Итог выполнения
1	MainActivityLaunchTest	Проверка корректного запуска главного экрана приложения и наличия основных элементов интерфейса.	Приложению выданы основные разрешения: доступ к телефону, геолокации и Wi-Fi.	MainActivity успешно запускается, основные TextView, переключатель, кнопка обновления и кнопка выхода присутствуют на экране.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.1.	Тест успешно пройден
2	NoPhonePermissionTest	Проверка обработки отсутствия разрешения на доступ к данным телефона.	Разрешение READ_PHONE_STATE не выдано.	В блоках сотовой сети, сигнала и базовых станций отображаются сообщения об отсутствии доступа к данным телефона.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.2.	Тест успешно пройден

Продолжение таблицы 6

№	Название теста	Назначение и краткое описание	Начальные условия	Ожидаемый результат	Фактический результат	Итог выполнения
3	LocationNoPermissionShowsErrorTest	Проверка отображения сообщения при отсутствии разрешений на геолокацию.	Разрешения ACCESS_FINE_LOCATION и/или ACCESS_COARSE_LOCATION не выданы.	В полях GPS-координат и координат по сети отображается сообщение об отсутствии доступа к геолокации.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.3.	Тест успешно пройден
4	WifiEmptyListShowsMessageTest	Проверка поведения интерфейса при отсутствии доступных Wi-Fi сетей.	В TelemetryRepository передается снимок телеметрии с пустым списком wifiAccessPoints.	В блоке Wi-Fi отображается сообщение: «Нет доступных Wi-Fi сетей».	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.4.	Тест успешно пройден
5	LocationBroadcastUpdatesUITest	Проверка обновления интерфейса после получения broadcast с координатами.	MainActivity запущена, приложению выданы разрешения на геолокацию. В тесте отправляется ACTION_LOCATION_UPDATE с GPS- и network-координатами.	Поля GPS и координат по сети обновляются значениями, переданными в broadcast.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.5.	Тест успешно пройден
6	SwitchChangesAutoModeStateTest	Проверка влияния переключателя режима работы на сохраненное состояние приложения.	MainActivity запущена, переключатель автоматического режима доступен.	При изменении положения переключателя значение autoModeEnabled в AppStateStore изменяется соответственно.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.6.	Тест успешно пройден
7	MqttConnectionAndPublishTest	Проверка возможности подключения приложения к MQTT-брокеру и отправки сообщения в MQTT-топик. Тест создаёт MQTT-подписчика, подключается к брокеру через компонент Mqtt, публикует тестовое JSON-сообщение и проверяет его получение подписчиком.	MQTT-брокер запущен и доступен по указанному адресу. Устройство имеет сетевой доступ к брокеру. Пользователь авторизован на брокере с корректными учётными данными.	Подключение к MQTT-брокеру успешно установлено. Тестовое сообщение успешно опубликовано в заданный топик и получено подписчиком.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.7.	Тест успешно пройден

Окончание таблицы 6

№	Название теста	Назначение и краткое описание	Начальные условия	Ожидаемый результат	Фактический результат	Итог выполнения
8	RefreshButtonUpdatesUiAfterTelemetryBroadcastTest	Проверка сценария ручного обновления данных: нажатие кнопки обновления и последующее обновление UI после broadcast.	MainActivity запущена. Приложению выданы все необходимые разрешения.	Интерфейс обновляется данными из нового снимка телеметрии. Также обновляется время обновления.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.8.	Тест успешно пройден
9	ExitButtonFinishesActivityTest	Проверка работы кнопки выхода из приложения.	MainActivity запущена, кнопка выхода доступна.	После нажатия на кнопку выхода активность завершается.	Наблюдаемый результат соответствует ожидаемому. Скриншот теста приведен на рисунке Б.9.	Тест успешно пройден

В ходе инструментального тестирования было проведено 8 тестов, охватывающих основные функциональные возможности приложения, включая работу с пользовательским интерфейсом, обработку разрешений, взаимодействие компонентов, а также формирование и передачу данных. Код некоторых тестов приведен в приложении В.

Все проведённые тесты завершились успешно: фактические результаты во всех случаях полностью соответствуют ожидаемым.

Проведённое тестирование подтвердило:

- корректную обработку разрешений пользователя;
- правильную работу пользовательского интерфейса (запуск приложения, обработка нажатий, отображение данных);
- корректное функционирование сервисов, в частности LocationService, и передачу данных через broadcast-сообщения.

Таким образом, инструментальное тестирование показало, что приложение стабильно функционирует в условиях, максимально приближенных к реальной эксплуатации, корректно обрабатывает как штатные, так и нештатные ситуации и обеспечивает ожидаемое поведение всех основных компонентов.

5 Планирование разработки и оценка бюджета

Для отражения основных этапов разработки андроид-приложения сбора данных о радиообстановке ближайшего окружения в соответствии с заданными требованиями и технологиями была составлена диаграмма Ганта [20] плана разработки в среде Microsoft Office Project (рисунок 24).

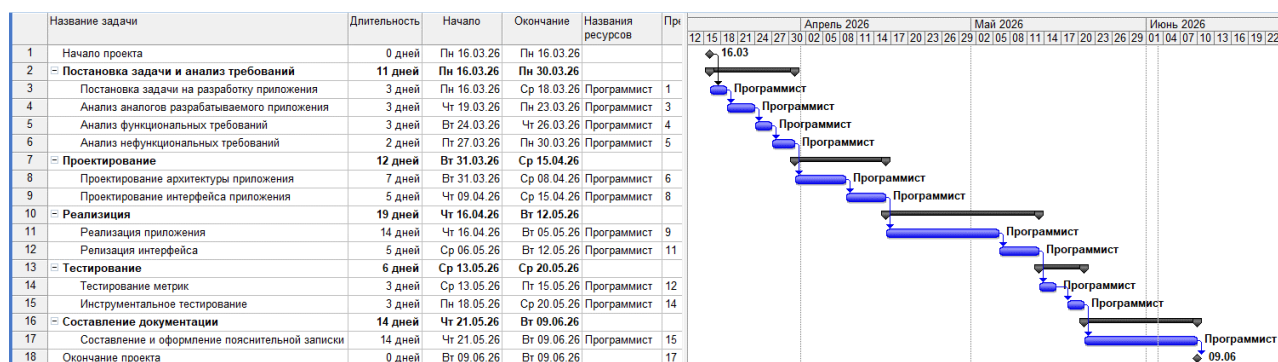


Рисунок 24 – Диаграмма Ганта плана разработки

На диаграмме представлены все этапы разработки приложения:

- постановка задачи и анализ требований;
- проектирование;
- реализация;
- тестирование;
- составление документации.

В разработке приложения участвует только один человек – программист.

Для него установлен стандартный рабочий календарный график и ставка 200 рублей в час (рисунок 25).

	Название ресурса	Тип	Краткое название	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Начисление	Базовый календарь
1	Программист	Трудовой	Прог	100%	р.200,00/ч	р.250,00/ч	Пропорциональное	Стандартный

Рисунок 25 – Трудовой ресурс

Ниже представлена сводка по затратам времени на реализацию задач программистом (рисунок 26).

	а	Название задачи	Трудозатраты	Длительность	Начало	Окончание
1		Начало проекта	0 ч	0 дней	Пн 16.03.26	Пн 16.03.26
2		▢ Постановка задачи и анализ требований	88 ч	11 дней	Пн 16.03.26	Пн 30.03.26
3		▢ Постановка задачи на разработку приложения	24 ч	3 дней	Пн 16.03.26	Ср 18.03.26
		Программист	24 ч		Пн 16.03.26	Ср 18.03.26
4		▢ Анализ аналогов разрабатываемого приложения	24 ч	3 дней	Чт 19.03.26	Пн 23.03.26
		Программист	24 ч		Чт 19.03.26	Пн 23.03.26
5		▢ Анализ функциональных требований	24 ч	3 дней	Вт 24.03.26	Чт 26.03.26
		Программист	24 ч		Вт 24.03.26	Чт 26.03.26
6		▢ Анализ нефункциональных требований	16 ч	2 дней	Пт 27.03.26	Пн 30.03.26
		Программист	16 ч		Пт 27.03.26	Пн 30.03.26
7		▢ Проектирование	96 ч	12 дней	Вт 31.03.26	Ср 15.04.26
8		▢ Проектирование архитектуры приложения	56 ч	7 дней	Вт 31.03.26	Ср 08.04.26
		Программист	56 ч		Вт 31.03.26	Ср 08.04.26
9		▢ Проектирование интерфейса приложения	40 ч	5 дней	Чт 09.04.26	Ср 15.04.26
		Программист	40 ч		Чт 09.04.26	Ср 15.04.26
10		▢ Реализация	152 ч	19 дней	Чт 16.04.26	Вт 12.05.26
11		▢ Реализация приложения	112 ч	14 дней	Чт 16.04.26	Вт 05.05.26
		Программист	112 ч		Чт 16.04.26	Вт 05.05.26
12		▢ Релизация интерфейса	40 ч	5 дней	Ср 06.05.26	Вт 12.05.26
		Программист	40 ч		Ср 06.05.26	Вт 12.05.26
13		▢ Тестирование	48 ч	6 дней	Ср 13.05.26	Ср 20.05.26
14		▢ Тестирование метрик	24 ч	3 дней	Ср 13.05.26	Пт 15.05.26
		Программист	24 ч		Ср 13.05.26	Пт 15.05.26
15		▢ Инструментальное тестирование	24 ч	3 дней	Пн 18.05.26	Ср 20.05.26
		Программист	24 ч		Пн 18.05.26	Ср 20.05.26
16		▢ Составление документации	112 ч	14 дней	Чт 21.05.26	Вт 09.06.26
17		▢ Составление и оформление пояснительной записки	112 ч	14 дней	Чт 21.05.26	Вт 09.06.26
		Программист	112 ч		Чт 21.05.26	Вт 09.06.26
18		Окончание проекта	0 ч	0 дней	Вт 09.06.26	Вт 09.06.26

Рисунок 26 – Использование задач в проекте

Можно заметить, что самой длинной задачей будет являться реализация программы, включающая реализацию самого приложения, а также пользовательского интерфейса, и займет 19 дней или 152 часа.

Второй по длительности задачей будет являться составление документации, и займет 14 дней или 112 часов.

При этом самой короткой по времени задачей будет являться тестирование программного продукта, включающее в себя тестирование метрик (вычислительная сложность, цикломатическая сложность, комплексный показатель качества кода), а также инструментальное тестирование, и займет всего 6 дней или 48 часов.

Итоговая сводка по проекту представлена на рисунке 27.

Даты			
Начало:	Пн 16.03.26	Окончание:	Вт 09.06.26
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	62 дней	Оставшаяся:	62 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	62 дней	Процент завершени	0%
Трудозатраты			
Текущий план:	496 ч	Оставшиеся:	496 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	496 ч	Процент завершени	0%
Затраты			
Текущий план:	р.99 200,00	Оставшиеся:	р.99 200,00
Базовый план:	р.0,00	Фактические:	р.0,00
Отклонение:	р.99 200,00		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	18	Трудовые ресурсы:	1
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дос:	0
Завершенные задачи:	0	Материальные ресурсы:	0
Итого задач:	18	Итого ресурсов:	1

Рисунок 27 – Статистика проекта

Подводя итог, можем видеть, что разработка андроид-приложения сбора данных о радиообстановке ближайшего окружения в одиночку одним программистом, с учетом стандартного рабочего календаря, займет 62 рабочих дня, что соответствует 496 часам. Бюджет разработки составит почти 100 тысяч рублей.

Заключение

В рамках выпускной квалификационной работы было разработано андроид-приложение для сбора данных о радиообстановке ближайшего окружения.

Была поставлена задача на разработку, проведен обзор приложений-аналогов, проанализированы функциональные и нефункциональные требования, а также составлена диаграмма вариантов использования, отражающая основные сценарии работы приложения.

На стадии проектирования была разработана схема структуры приложения, диаграмма коммуникаций, на которой были отражены основные цепочки команд, с помощью которых части системы будут взаимодействовать друг с другом, и диаграмма классов приложения. Также был спроектирован графический интерфейс пользователя.

На стадии реализации был написан код приложения на языке программирования Kotlin, составлена диаграмма компонентов. Также для приложения был разработан интерфейс, который отвечает требованиям о простоте понимания, наглядности и информативности. На нем отображаются все собранные телеметрические данные о радиообстановке ближайшего радиоокружения, а также имеются соответствующие кнопки для взаимодействия пользователя с приложением.

Был проведен контроль качества программного обеспечения, в ходе которого были проверены различные метрики – вычислительная сложность, цикломатическая сложность и комплексный показатель качества. Также было проведено инструментальное тестирование приложения. По итогам тестирования был сделан вывод, что приложение работает корректно.

В завершение работы было проведено планирование разработки и оценка бюджета в среде Microsoft Office Project. По итогам было установлено, что разработка приложения составит 62 рабочих дня или 496 часов, а бюджет разработки составит чуть меньше ста тысяч рублей.

Таким образом, все поставленные задачи были выполнены успешно и в результате было спроектировано и разработано андроид-приложение для сбора данных о радиообстановке ближайшего окружения.

Актуальность результатов разработки, выполненной в рамках данной выпускной квалификационной работы, подтверждает факт их использования на реальном предприятии, о чем свидетельствует акт о внедрении, представленный на рисунке Г.1.

Список использованных источников

1. Документация Android : Android. [Электронный ресурс]. URL: <https://source.android.com/docs>
2. Харди Б., Филлипс Б., Стюарт К., Марсиано К. Android. Программирование для профессионалов. 2-е изд. – СПб.: Питер, 2016. – 640 с.
3. MQTT Version 5.0 Specification : OASIS Standard. [Электронный ресурс]. URL: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>
4. Rappaport, T. S. Wireless Communications: Principles and Practice / T. S. Rappaport. – 2nd ed. – Upper Saddle River : Prentice Hall, 2002. – 736 p.
5. The JavaScript Object Notation (JSON) Data Interchange Format : RFC 8259. [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc8259>
6. Джемеров Д. Kotlin в действии / Д. Джемеров, С. Исакова. – СПб. : Питер, 2018. – 544 с.
7. Документация Kotlin : Kotlin. [Электронный ресурс]. URL: <https://kotlinlang.org/docs/home.html>
8. Обзор Android Studio : Android. [Электронный ресурс]. URL: <https://developer.android.com/studio/intro>
9. Арлоу Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. / Д. Арлоу, И. Нейштадт. – СПб: Символ-Плюс, 2007. – 624 с.
10. Фаулер М. UML. Основы. – М.: Символ-Плюс, 2003. – 192 с.
11. Тидвелл Д. Разработка интерфейсов. Паттерны проектирования. 2-е изд. / Д. Тидвелл, Ч. Брюэр, Э. Валенсия – М. [и др.]: Питер, 2011. – 474 с.
12. ScrollView : Android. [Электронный ресурс]. URL: <https://developer.android.com/reference/android/widget/ScrollView>
13. Maintainability Index Report : IBM Documentation. [Электронный ресурс]. URL: <https://www.ibm.com/docs/en/addi/6.1.4?topic=reports-maintainability-index-report>

14. Halstead M. H. Elements of Software Science / M. H. Halstead. – New York : Elsevier North-Holland, 1977. – 127 p.
15. McCabe T. J. A Complexity Measure // IEEE Transactions on Software Engineering. – 1976. – Vol. SE-2, № 4. – P. 308–320.
16. Программный код и его метрики : Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/>
17. Семахин А. М. Методы верификации и оценки качества программного обеспечения: учебное пособие. – Курган : Изд-во КГУ, 2018. – 150 с.
18. Android Instrumented Tests: testing types and how they work : Medium. [Электронный ресурс]. URL: <https://natanximenes.medium.com/android-instrumented-tests-testing-types-and-how-they-work-8dc6cdff1022>
19. Создание инструментальных тестов : Android. [Электронный ресурс]. URL: <https://developer.android.com/training/testing/instrumented-tests?hl=ru>
20. Кларк У. Графики Ганта. Учет и планирование работы. – Москва, Ленинград: Мосполиграф, 1931. – 124 с.

Приложение А
(обязательное)
Исходный код приложения

Листинг A.1 – MqttService.kt

```

@RequiresApi (Build.VERSION_CODES.R)
override fun onStartCommand(intent: Intent?, flags: Int, startId:
Int): Int {
    createNotificationChannel()
    startForeground(
        notificationId,
        createNotification(),
        FOREGROUND_SERVICE_TYPE_DATA_SYNC
    )

    when (intent?.action) {
        ACTION_REFRESH_NOW -> {
            serviceScope.launch {
                refreshTelemetryOnce()
            }
        }

        else -> {
            if (publishJob?.isActive != true) {
                publishJob = serviceScope.launch {
                    while (isActive) {
                        refreshTelemetryOnce()
                        delay(30_000)
                    }
                }
            }
        }
    }

    return START_STICKY
}

@RequiresApi (Build.VERSION_CODES.R)
private suspend fun refreshTelemetryOnce() {
    refreshMutex.withLock {
        val snapshot = infoCollector.collectSnapshot()
        TelemetryRepository.lastSnapshot = snapshot
        notifications.SendInfoUpdatedNotification()

        appStateStore.setLastTelemetryUpdateTimestamp(System.currentTimeMillis())

        val jsonObject = infoCollector.snapshotToJson(snapshot)
        publishJsonToMqtt(jsonObject)

        sendBroadcast(Intent(AppBroadcasts.ACTION_TELEMETRY_REFRESHED))
    }
}

```

```

private fun publishJsonToMqtt(jsonObject: JSONObject) {
    val topic = "main"

    if (!mqtt.connectToMqttBroker()) {
        addPendingMessage(jsonObject)
        notifications.sendMqttStateNotification("Ошибка при
подключении к брокеру.")
        return
    }

    val messagesToSend = getPendingMessagesWithCurrent(jsonObject)

    for (message in messagesToSend) {
        val isSent = mqtt.publishJsonObject(topic, message)

        if (!isSent) {
            clearPendingMessages()

            messagesToSend.dropWhile { it != message }.forEach {
failedMessage ->
                addPendingMessage(failedMessage)
            }

            notifications.sendMqttStateNotification (
                "Ошибка отправки данных в брокер."
            )
            return
        }
    }

    clearPendingMessages()
    notifications.sendMqttStateNotification("Данные успешно
отправлены.")
}

companion object {
    const val ACTION_REFRESH_NOW =
"com.example.firstkotlin.action.REFRESH_NOW"

    private const val MAX_PENDING_MESSAGES = 20
    private val pendingMessages = ArrayDeque<JSONObject>()

    @Synchronized
    private fun addPendingMessage(jsonObject: JSONObject) {
        if (pendingMessages.size >= MAX_PENDING_MESSAGES) {
            pendingMessages.removeFirst()
        }
        pendingMessages.addLast(JSONObject(jsonObject.toString()))
    }

    @Synchronized

```

```

        private fun getPendingMessagesWithCurrent(currentJsonObject:
JSONObject): List<JSONObject> {
            val result = pendingMessages.map {
                JSONObject(it.toString()) }.toMutableList()
            result.add(JSONObject(currentJsonObject.toString()))
            return result
        }

        @Synchronized
        fun clearPendingMessages() {
            pendingMessages.clear()
        }
    }
}

```

Листинг A.2 – Mqtt.kt

```

fun connectToMqttBroker(): Boolean {
    return try {
        if (::client.isInitialized && client.isConnected) {
            return true
        }

        client = MqttClient("tcp://10.10.100.204:1883", "sychev",
MemoryPersistence())
        options = MqttConnectOptions().apply {
            isCleanSession = false
            userName = "user"
            password = "123456".toCharArray()
            connectionTimeout = 1
        }

        client.connect(options)
        client.isConnected
    } catch (e: MqttException) {
        e.printStackTrace()
        false
    }
}

fun publishJsonObject(topic: String, jsonObject: JSONObject):
Boolean {
    return try {
        if (!::client.isInitialized || !client.isConnected) {
            return false
        }

        val message =
MqttMessage(jsonObject.toString().toByteArray())
        client.publish(topic, message)
        true
    } catch (e: MqttException) {

```

```

        e.printStackTrace()
        false
    }
}

```

Листинг A.3 – MainActivity.kt

```

@RequiresApi(Build.VERSION_CODES.R)
private val permissionsLauncher = registerForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { permissions ->
    val allGranted = requiredPermissions.all { permissions[it] ==
true || hasPermission(it) }

    if (allGranted) {
        requestBackgroundLocationIfNeeded()
        startCoreServicesIfNeeded()
        renderInitialTelemetry()
    } else {
        when {
            shouldShowAnyMainPermissionRationale() -> {
                showMainPermissionsRationaleDialog()
            }

            hasAnyDeniedMainPermission() -> {
                showOpenSettingsDialog(
                    title = "Нужны разрешения",
                    message = "Для корректной работы приложения
нужны разрешения на геолокацию, доступ к данным телефона и Wi-Fi.
Вы можете выдать их в настройках приложения."
                )
                renderInitialTelemetry()
            }

            else -> {
                renderInitialTelemetry()
            }
        }
    }
}

@RequiresApi(Build.VERSION_CODES.R)
@SuppressLint("MissingInflatedId", "SetTextI18n")
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()

    setContentView(R.layout.activity_main)
    setupViewInsets()
    initViews()
}

```

```

gsm = Gsm(this)
wifi = Wifi(this)
notifications = Notifications(this)
device = Device(this)
appStateStore = AppStateStore(this)
infoCollector = InfoCollector(
    context = this,
    gsm = gsm,
    wifi = wifi,
    device = device,
    appStateStore = appStateStore
)

ensureTrackId()

switch1.isChecked = appStateStore.isAutoModeEnabled()
appStateStore.setAutoModeEnabled(switch1.isChecked)

switch1.setOnCheckedChangeListener { _, isChecked ->
    appStateStore.setAutoModeEnabled(isChecked)
}

refreshButton.setOnClickListener {
    requestManualRefresh()
}

exitButton.setOnClickListener {
    exitApp()
}

createNotificationChannels()
renderLocationFromStore()
renderInitialTelemetry()
updateLastUpdatedText()

registerAppReceivers()
ensurePermissionsAndStart()
}

private fun startCoreServicesIfNeeded() {
    if (servicesStarted) return

    if (hasPermission(ACCESS_FINE_LOCATION) &&
        hasPermission(ACCESS_COARSE_LOCATION)) {
        startForegroundService(Intent(this,
            LocationService::class.java))
    }

    startForegroundService(Intent(this,
        MqttForegroundService::class.java))
    servicesStarted = true
}

```

```

private fun renderTowersInfo(snapshot: TelemetrySnapshot) {
    towerTextView.text = if (!snapshot.hasPhoneStatePermission) {
        "Нет доступа к данным базовых станций.\nРазрешите доступ к
        телефону в настройках."
    } else {
        val hasLocationPermission =
        hasPermission(ACCESS_FINE_LOCATION)
        if (!hasLocationPermission) {
            "Нет доступа к данным базовых станций.\nРазрешите
            доступ к местоположению в настройках."
        } else {
            if (snapshot.cellTowers.isEmpty()) {
                "Нет доступных базовых станций."
            } else {
                buildString {
                    snapshot.cellTowers.forEach { tower ->
                        appendLine(" — Подключено: ${if
(tower.isRegistered) "да" else "нет"}")
                        if (tower.signalStrength != null)
appendLine(" — Сила сигнала: ${tower.signalStrength} dBm")
                        if (tower.cellType != "null") appendLine("
— Тип соты: ${tower.cellType}")
                        if (tower.cellId != -1) appendLine(" — ID
соты: ${tower.cellId}")
                        if (tower.physicalCellId != -1)
appendLine(" — Физический ID соты: ${tower.physicalCellId}")
                        if (tower.enb != -1) appendLine(" — ID
базовой станции для LTE: ${tower.enb}")
                        if (tower.mccMnc != null) appendLine(" —
Код страны и оператора: ${tower.mccMnc}")
                        if (tower.lac != -1) appendLine(" — Код
зоны местоположения: ${tower.lac}")
                        if (tower.tac != -1) appendLine(" — Код
зоны отслеживания: ${tower.tac}")
                        if (tower.earfcn != -1) appendLine(" —
Номер радиочастотного канала: ${tower.earfcn}")
                        if (tower.bandwidth != -1) appendLine(" —
Ширина канала: ${tower.bandwidth} kHz")
                        appendLine()
                    }
                }
            }
        }
    }
}

```

Листинг A.4 – Wifi.kt

```

@RequiresApi (Build.VERSION_CODES.R)
fun getWifiAccessPointsInfo(): List<WifiAccessPointInfo> {

```

```

if (!hasWifiStatePermission()) return emptyList()

return try {
    val wifiInfo = wifiManager.connectionInfo
    val currentSsid = wifiInfo.ssid.removeSurrounding("\"")
    val currentBssid = wifiInfo.bssid

    val scanResults = if (hasWifiStatePermission()) {
        wifiManager.scanResults
    } else {
        emptyList()
    }

    scanResults
        .asSequence()
        .map { scanResult ->
            WifiAccessPointInfo(
                isConnected = scanResult.SSID == currentSsid
                &&
                    scanResult.BSSID == currentBssid,
                standard =
                    scanResult.wifiStandard,
                signalStrength = scanResult.level,
                ssid = scanResult.SSID,
                bssid = scanResult.BSSID,
                frequency = scanResult.frequency,
                capabilities = scanResult.capabilities
            )
        }
        .sortedWith(
            compareByDescending<WifiAccessPointInfo> {
it.isConnected }
                .thenByDescending { it.signalStrength }
            )
        .toList()
    } catch (_: SecurityException) {
        emptyList()
    }
}

```

Листинг A.5 – Gsm.kt

```

fun getSignalInfo(): SignalInfo? {
    if (!hasPhoneStatePermission()) return null

    return try {
        val signalType = when (telephonyManager.dataNetworkType) {
            TelephonyManager.NETWORK_TYPE_EDGE -> "EDGE"
            TelephonyManager.NETWORK_TYPE_HSPA -> "H"
            TelephonyManager.NETWORK_TYPE_HSPAP -> "H+"
            TelephonyManager.NETWORK_TYPE_LTE -> "LTE"

```

```

        TelephonyManager.NETWORK_TYPE_IWLAN -> "IWLAN"
        else -> "Неизвестно"
    }

    val signalStrengthDb = telephonyManager.signalStrength
        ?.cellSignalStrengths
        ?.firstOrNull()
        ?.dbm

    val signalStatus = when (signalStrengthDb) {
        in -50 downTo -79 -> "Отличный"
        in -80 downTo -89 -> "Хороший"
        in -90 downTo -99 -> "Средний"
        in -100 downTo -109 -> "Плохой"
        in -110 downTo -120 -> "Очень плохой"
        else -> "Неизвестно"
    }

    SignalInfo(
        signalType = signalType,
        signalStrengthDbm = signalStrengthDb,
        signalStatus = signalStatus
    )
} catch (_: SecurityException) {
    null
}
}

@RequiresApi(Build.VERSION_CODES.R)
fun getAllCellTowersInfo(): List<TowerInfo> {
    if (!hasPhoneStatePermission()) return emptyList()

    return try {
        telephonyManager.allCellInfo
            .asSequence()
            .mapNotNull { cellInfo ->
                if (!cellInfo.isRegistered) return@mapNotNull null

                val cellIdentity = getCellIdentity(cellInfo) ?:
return@mapNotNull null
                val signalStrength =
cellInfo.cellSignalStrength.dbm

                if (signalStrength >= 0) return@mapNotNull null

                val cellId = getCellId(cellIdentity)
                val towerInfo = TowerInfo(
                    isRegistered = cellInfo.isRegistered,
                    cellType = getCellType(cellInfo),
                    cellId = cellId,
                    physicalCellId =
getPhysicalCellId(cellIdentity),

```

```

        enb = getEnb(cellIdentity),
        mccMnc = getMccMnc(cellIdentity),
        lac = getLac(cellIdentity),
        tac = getTac(cellIdentity),
        earfcn = getEarfcn(cellIdentity),
        bandwidth = getBandwidth(cellIdentity),
        signalStrength = signalStrength
    )

    towerInfo
}
.sortedWith(
    compareByDescending<TowerInfo> { it.isRegistered }
        .thenByDescending { it.signalStrength ? :
Int.MIN_VALUE }
)
.toList()
} catch (_: SecurityException) {
    emptyList()
}
}

```

Листинг A.6 – Device.kt

```

@SuppressLint("HardwareIds")
fun getDeviceInfo(): DeviceInfo {
    return DeviceInfo(
        brand = Build.BRAND,
        model = Build.MODEL,
        androidVersion = Build.VERSION.RELEASE,
        sdkVersion = Build.VERSION.SDK_INT,
        androidId = Settings.Secure.getString(
            context.contentResolver,
            Settings.Secure.ANDROID_ID
        )
    )
}

```

Листинг A.7 – InfoCollector.kt

```

@RequiresApi(Build.VERSION_CODES.R)
fun collectSnapshot(): TelemetrySnapshot {
    val appVersion = context.packageManager
        .getPackageInfo(context.packageName, 0)
        .versionName

    val dateFormat = SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss",
        Locale.getDefault())
    val currentDate = dateFormat.format(Date())
}

```

```

return TelemetrySnapshot(
    deviceType = "Phone",
    appVersion = appVersion,
    dateTime = currentDate,
    status = appStateStore.isAutoModeEnabled(),
    trackId = appStateStore.getTrackId(),
    requestCode = appStateStore.getRequestCode(),
    latitude = appStateStore.getGpsLatitude(),
    longitude = appStateStore.getGpsLongitude(),
    latitudeNetwork = appStateStore.getNetworkLatitude(),
    longitudeNetwork = appStateStore.getNetworkLongitude(),
    hasPhoneStatePermission = gsm.hasPhoneStatePermission(),
    hasWifiStatePermission = wifi.hasWifiStatePermission(),
    deviceInfo = device.getDeviceInfo(),
    gsm = gsm.getGsmInfo(),
    signal = gsm.getSignalInfo(),
    cellTowers = gsm.getAllCellTowersInfo(),
    wifiAccessPoints = wifi.getWifiAccessPointsInfo()
)
}

fun snapshotToJson(snapshot: TelemetrySnapshot): JSONObject {
    val jsonObject = JSONObject()

    jsonObject.put("deviceType", snapshot.deviceType)
    jsonObject.put("appVersion", snapshot.appVersion)
    jsonObject.put("dateTime", snapshot.dateTime)
    jsonObject.put("status", snapshot.status)
    jsonObject.put("trackId", snapshot.trackId)
    jsonObject.put("requestCode", snapshot.requestCode)

    jsonObject.put("latitude", snapshot.latitude)
    jsonObject.put("longitude", snapshot.longitude)
    jsonObject.put("latitudeGlonass", snapshot.latitudeNetwork)
    jsonObject.put("longitudeGlonass", snapshot.longitudeNetwork)

    jsonObject.put("deviceInfo",
deviceInfoToJson(snapshot.deviceInfo))
        jsonObject.put("gsm", snapshot.gsm?.let { gsmInfoToJson(it) })
        jsonObject.put("signal", snapshot.signal?.let {
signalInfoToJson(it) })

        if (snapshot.status) {
            jsonObject.put("cellTowers",
towersToJson(snapshot.cellTowers))
            jsonObject.put("wifi",
wifiToJson(snapshot.wifiAccessPoints))
        } else {
            jsonObject.put("cellTowers", JSONArray())
            jsonObject.put("wifi", JSONArray())
        }
    }
}

```

```

        return jsonObject
    }

```

Листинг A.8 – LocationService.kt

```

private fun updateLocation() {
    var requestCode = ""
    if (checkLocationPermission())
        requestCode =
fusedLocationClient.locationAvailability.toString()

    val gpsLocation =
getLastKnownLocation(LocationManager.GPS_PROVIDER)
    val networkLocation =
getLastKnownLocation(LocationManager.NETWORK_PROVIDER)

    val gpsLatitude = gpsLocation?.let { round(it.latitude) }
    val gpsLongitude = gpsLocation?.let { round(it.longitude) }
    val networkLatitude = networkLocation?.let {
round(it.latitude) }
    val networkLongitude = networkLocation?.let {
round(it.longitude) }
    val gpsAccuracy = gpsLocation?.accuracy
    val networkAccuracy = networkLocation?.accuracy

    val intent = Intent(AppBroadcasts.ACTION_LOCATION_UPDATED)
    intent.putExtra("requestCode", requestCode)
    intent.putExtra("gpsLatitude", gpsLatitude)
    intent.putExtra("gpsLongitude", gpsLongitude)
    intent.putExtra("networkLatitude", networkLatitude)
    intent.putExtra("networkLongitude", networkLongitude)
    intent.putExtra("gpsAccuracy", gpsAccuracy)
    intent.putExtra("networkAccuracy", networkAccuracy)
    sendBroadcast(intent)
}

```

Листинг A.9 – Notifications.kt

```

fun createForegroundNotification(): Notification {
    return NotificationCompat.Builder(this, channelId)
        .setContentTitle("Фоновая работа")
        .setContentText("Идет фоновый сбор телеметрии")
        .setSmallIcon(R.drawable.ic_launcher_background)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .build()
}

fun sendMqttStateNotification(state: String) {
    val notificationId = 1
}

```

```

        val currentTime = SimpleDateFormat("HH:mm:ss",
Locale.getDefault()).format(Date())
        val notificationContent = "$currentTime: $state"

        val notificationBuilder = NotificationCompat.Builder(context,
STATE_INFO_CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_launcher_background)
            .setContentTitle("Статус MQTT")
            .setContentText(notificationContent)
            .setPriority(NotificationCompat.PRIORITY_MAX)

        if (ContextCompat.checkSelfPermission(
            context,
            Manifest.permission.POST_NOTIFICATIONS
        ) == PackageManager.PERMISSION_GRANTED
        ) {
            notificationManager.notify(notificationId,
notificationBuilder.build())
        }
    }

fun sendInfoUpdatedNotification() {
    val notificationId = 2
    val currentTime = SimpleDateFormat("HH:mm:ss",
Locale.getDefault()).format(Date())
    val notificationContent = "$currentTime: Данные успешно
обновлены"

    val notificationBuilder = NotificationCompat.Builder(context,
STATE_INFO_CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_launcher_background)
        .setContentTitle("Статус обновления данных")
        .setContentText(notificationContent)
        .setPriority(NotificationCompat.PRIORITY_MAX)

    if (ContextCompat.checkSelfPermission(
        context,
        Manifest.permission.POST_NOTIFICATIONS
    ) == PackageManager.PERMISSION_GRANTED
    ) {
        notificationManager.notify(notificationId,
notificationBuilder.build())
    }
}

```

Приложение Б
(рекомендуемое)
Скриншоты результатов тестирования

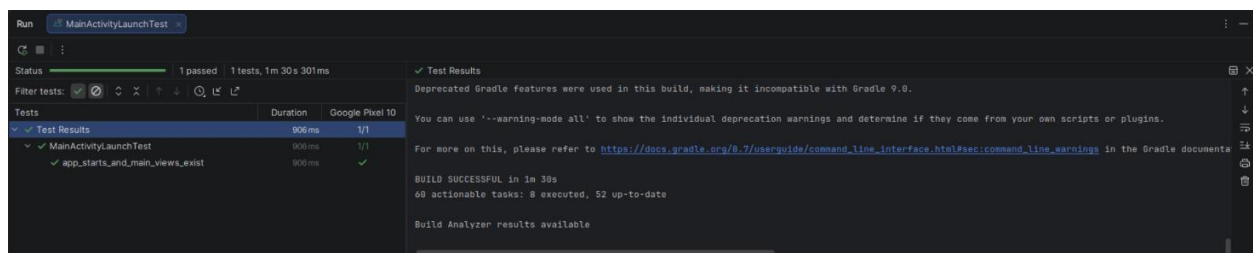


Рисунок Б.1 – Скриншот выполнения теста MainActivityLaunchTest

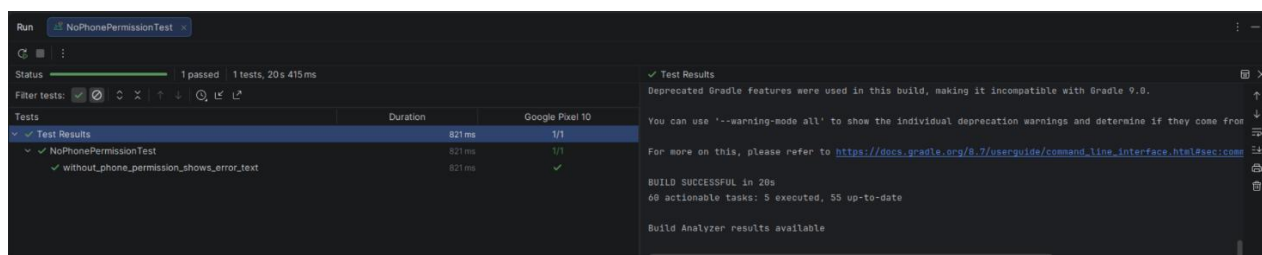


Рисунок Б.2 – Скриншот выполнения теста NoPhonePermissionTest

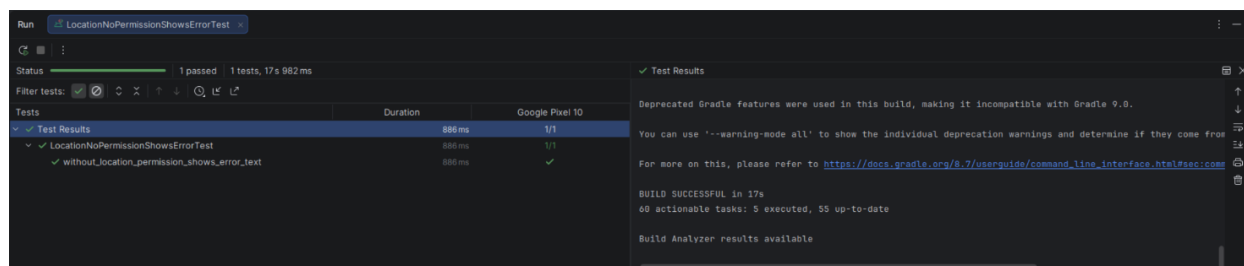


Рисунок Б.3 – Скриншот выполнения теста LocationNoPermissionShowsErrorTest

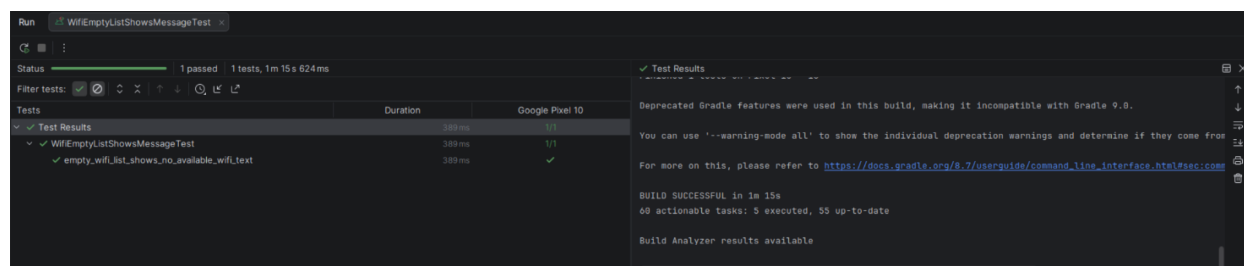


Рисунок Б.4 – Скриншот выполнения теста WifiEmptyListShowsMessageTest

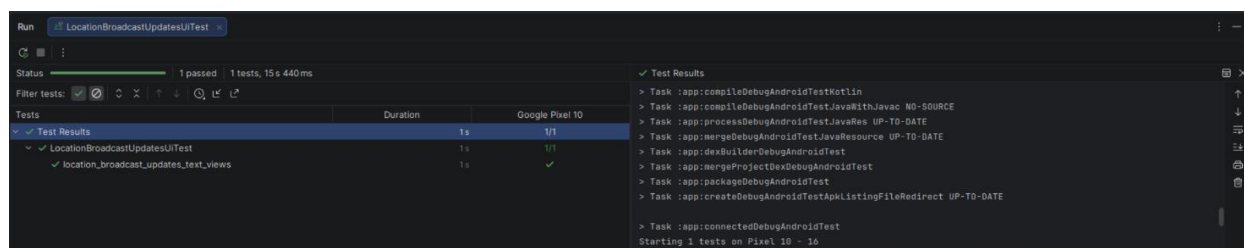


Рисунок Б.5 – Скриншот выполнения теста LocationBroadcastUpdatesUiTest

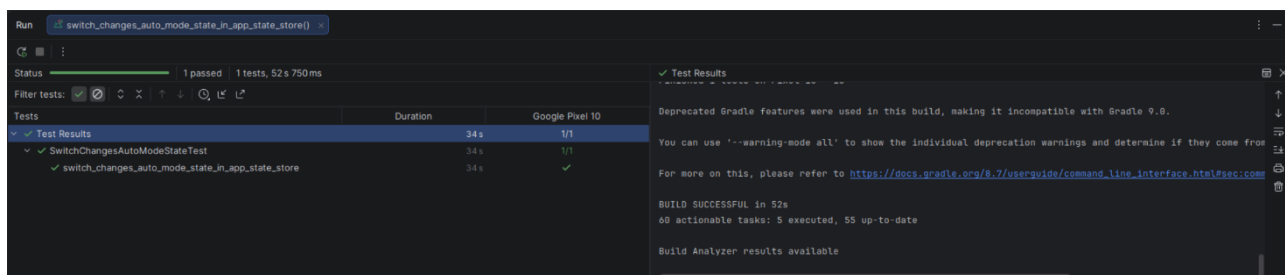


Рисунок Б.6 – Скриншот выполнения теста SwitchChangesAutoModeStateTest

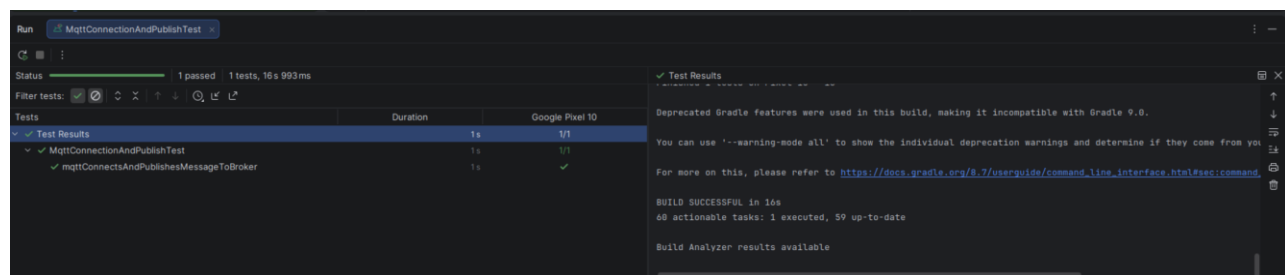


Рисунок Б.7 – Скриншот выполнения теста MqttConnectionAndPublishTest

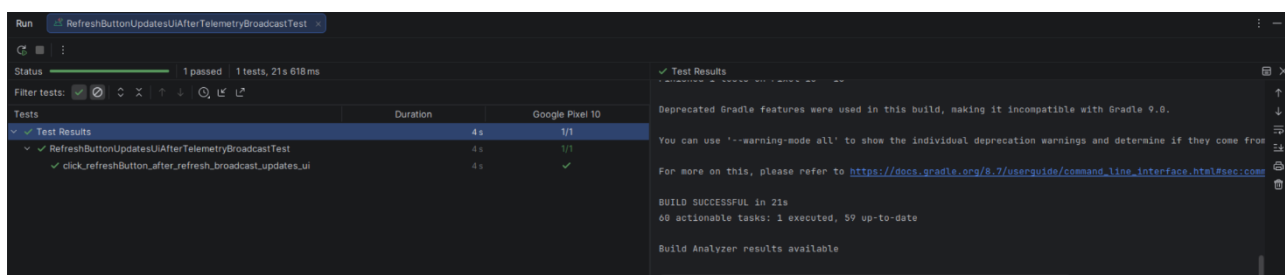


Рисунок Б.8 – Скриншот выполнения теста RefreshButtonUpdatesUiAfterTelemetryBroadcastTest

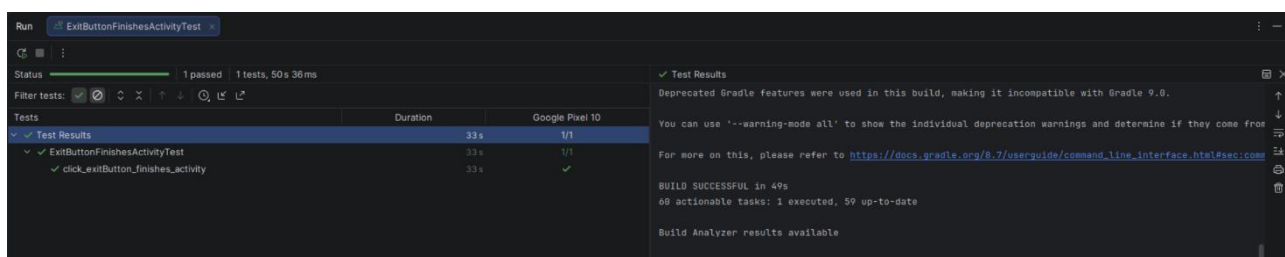


Рисунок Б.9 – Скриншот выполнения теста ExitButtonFinishesActivityTest

Приложение В
(рекомендуемое)
Исходный код тестов

Листинг В.1 – LocationNoPermissionShowsErrorTest.kt

```

@RunWith(AndroidJUnit4::class)
class LocationNoPermissionShowsErrorTest {

    @Test
    fun without_location_permission_shows_error_text() {

        val context =

ApplicationProvider.getApplicationContext<android.content.Context>
()

        val fineGranted =
            ContextCompat.checkSelfPermission(
                context,
                Manifest.permission.ACCESS_FINE_LOCATION
            ) == PackageManager.PERMISSION_GRANTED

        val coarseGranted =
            ContextCompat.checkSelfPermission(
                context,
                Manifest.permission.ACCESS_COARSE_LOCATION
            ) == PackageManager.PERMISSION_GRANTED

        assertTrue(!fineGranted || !coarseGranted)

        val scenario =
            ActivityScenario.launch(MainActivity::class.java)

        scenario.onActivity { activity ->

            val gpsText =
                activity.findViewById<TextView>(R.id.gpsTextView)
                    .text
                    .toString()

            val networkText =

activity.findViewById<TextView>(R.id.networkCoordsTextView)
                    .text
                    .toString()

            assertTrue(
                gpsText.contains("Нет доступа к геолокации")
            )

            assertTrue(
                networkText.contains("Нет доступа к геолокации")
            )

        }
    }
}

```

Листинг В.2 – MqttConnectionAndPublishTest.kt

```

@RunWith(AndroidJUnit4::class)
class MqttConnectionAndPublishTest {

    private val brokerUrl = "tcp://192.168.1.86:1883"
    private val topic = "main"
    private val username = "admin"
    private val password = "3212".toCharArray()

    @Test
    fun mqttConnectsAndPublishesMessageToBroker() {
        val testMessageId = UUID.randomUUID().toString()
        val latch = CountDownLatch(1)

        var receivedPayload: String? = null

        val subscriber = MqttClient(
            brokerUrl,
            "test-subscriber-${UUID.randomUUID()}",
            MemoryPersistence()
        )

        val subscriberOptions = MqttConnectOptions().apply {
            isCleanSession = true
            userName = username
            password = this@MqttConnectionAndPublishTest.password
            connectionTimeout = 5
        }

        subscriber.connect(subscriberOptions)

        subscriber.subscribe(topic) { _, message ->
            val payload = String(message.payload)
            if (payload.contains(testMessageId)) {
                receivedPayload = payload
                latch.countDown()
            }
        }

        val publisher = Mqtt()
        val connected = publisher.connectToMqttBroker()

        assertTrue("Не удалось подключиться к MQTT-брокеру",
            connected)

        val jsonObject = JSONObject().apply {
            put("testMessageId", testMessageId)
            put("source", "instrumented_test")
            put("message", "MQTT publish test")
        }
    }
}

```

```
        val published = publisher.publishJsonObject(topic,
jsonObject)

        assertTrue("Сообщение не было опубликовано в MQTT-топик",
published)

        val messageReceived = latch.await(10, TimeUnit.SECONDS)

        assertTrue("Сообщение не было получено подписчиком из
MQTT-брокера", messageReceived)
        assertTrue(
            "Полученное сообщение не содержит testMessageId",
            receivedPayload?.contains(testMessageId) == true
        )

        subscriber.disconnect()
        subscriber.close()
    }
}
```

Приложение Г
(рекомендуемое)
Акт о внедрении

АКТ О ВНЕДРЕНИИ РЕЗУЛЬТАТОВ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Настоящий акт составлен об использовании на предприятии Пензенского филиала АО «Научно-технический центр «Атлас» результатов выпускной квалификационной работы на тему «Андроид-приложение сбора данных о радиообстановке ближайшего окружения» студента группы 22ВП2 Пензенского государственного университета Сычева Матвея Дмитриевича.

Начальник

Специализированного отдела №5

Специализированной научно-технической службы

ПФ АО «НТЦ «Атлас»



/ Чекин Р.Н.

Рисунок Г.1 – Акт о внедрении