Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Мясников Владислав Николаевич

Разработка плагина для Android Studio по оценке энергопотребления кода

Курсовая работа

Научный руководитель:
ст. преп. С. Ю. Сартасов

Санкт-Петербург
2019

Оглавление

Введение

В XXI веке сложно себе представить мир, в котором нет мобильных устройств. Благодаря своим компактным размерам, огромным вычислительным возможностям и удобству в использовании, смартфоны, планшеты, умные часы и прочие “гаджеты” прочно вошли в нашу жизнь. Так, количество пользователей смартфонов, наиболее популярных в среде мобильных устройств, на конец 2019 года по прогнозам ожидается около 3.2 миллиардов [1], а Android продолжает лидировать по распространённости среди мобильных ОС [2].

Однако по-прежнему одной из главных проблем в мобильной индустрии остаётся расход батареи, от которого напрямую зависит “время жизни” устройства. Производители не могут обойти это стороной, так как без постоянных улучшений аппаратной и программной частей продукт не сможет долго конкурировать на рынке.

Ёмкость литиевых батарей, обычно используемых при производстве мобильных устройств, значительно увеличилась. Также улучшилась и программная составляющая: в частности, за последние пять лет Google сделал множество оптимизаций по повышению энергоэффективности Android (например, ограничения на обработку задач в фоновом режиме и внедрение алгоритмов машинного обучения [3, 4]).

Кроме аппаратной части и ОС не стоит также забывать и о приложениях. Зачастую мобильные разработчики не думают о вопросе энергопотребления написанного ими кода. Отсюда неэффективный расход батареи и, как следствие, быстрый её разряд. Нужен инструмент, который позволил бы как можно точнее оценить энергоэффективность работы приложения до его выхода на рынок. Также неплохо бы внедрить подобный инструмент в среду разработки, что повысит удобство его использования: разработчику можно будет не вносить данные вручную, а выбирать из предлагаемых вариантов в диалоговых окнах среды. Кроме того, интеграция позволит упростить установку и настройку инструмента. В данной работе предложена реализация вышеупомянутого инструмента для ОС Android в рамках проекта “Navitas Framework”.

1. Цели и задачи

Целью данной работы является разработка плагина по оценке энергопотребления кода для официальной интегрированной среды разработки под ОС Android — Android Studio. Для достижения цели были поставлены следующие задачи:

1. Провести обзор существующих решений.
2. Провести обзор технических средств по разработке плагинов для Android Studio.
3. Интегрировать “Navitas Framework” в Android Studio в виде плагина.
4. Проверить работоспособность плагина на Android-приложении.

2. Термины и определения

- Wake lock — механизм ОС Android, позволяющий приложению управлять состоянием питания устройства. Например, используя wake lock, приложение может удерживать экран или центральный процессор в активном состоянии.
- Alarm — механизм ОС Android, позволяющий выполнять операции в указанное время, даже если ваше приложение не будет запущено в этот момент.
- Job — механизм ОС Android, позволяющий выполнять действия при наступлении определённых условий.
- Power Profile — файл, содержащий информацию об энергопотреблении различных компонентов устройства за единицу времени в разных состояниях. Данный файл должен быть предоставлен производителем устройства.

3. Обзор предметной области

3.1. Обзор существующих решений

Для поиска существующих решений использовался сервис “Google Scholar” и официальные сайты: по размещению плагинов от JetBrains и по Android-разработке от Google. Найденные с помощью “Google Scholar” в июне 2019 статьи[1] были рассмотрены на предмет соответствия данной предметной области в рамках системного обзора литературы. В результате было выделено порядка 50 работ, связанных с энергопрофилированием кода на ОС Android, и только в одной из них речь шла об интеграции разработанного инструмента с Android Studio. На официальных сайтах было найдено ещё два решения. Таким образом, по результатам проведённых поисков было обнаружено три схожих решения. Но, как далее станет ясно, каждое из них имеет определённые ограничения.

3.1.1. Energy Profiler

Energy Profiler — инструмент для профилирования энергопотребления Android-приложений, интегрированный в Android Studio [5].

Energy Profiler отслеживает использование центрального процессора, сети и GPS-датчика и визуализирует потребление энергии каждым из этих компонентов. Также Energy Profiler отображает возникновение системных событий (wake locks, alarms, jobs, запросы о местоположении), оказывающих влияние на энергопотребление.

Для того, чтобы начать энергопрофилирование, нужно выполнить несколько шагов: запустить целевое приложение, перейти на вкладку “Profiler” в нижней панели инструментов и кликнуть на область с заголовком “Energy”.

Стоит отметить, что Energy Profiler измеряет расход энергии не напрямую, а использует модель для оценивания. При этом вместо единиц измерения присутствуют абстрактные значения: Light, Medium, Heavy.

[1] https://docs.google.com/spreadsheets/d/17G31QTGL-tpUSkv96W_JavoJ03bzLDy3swJIfTTLHKA

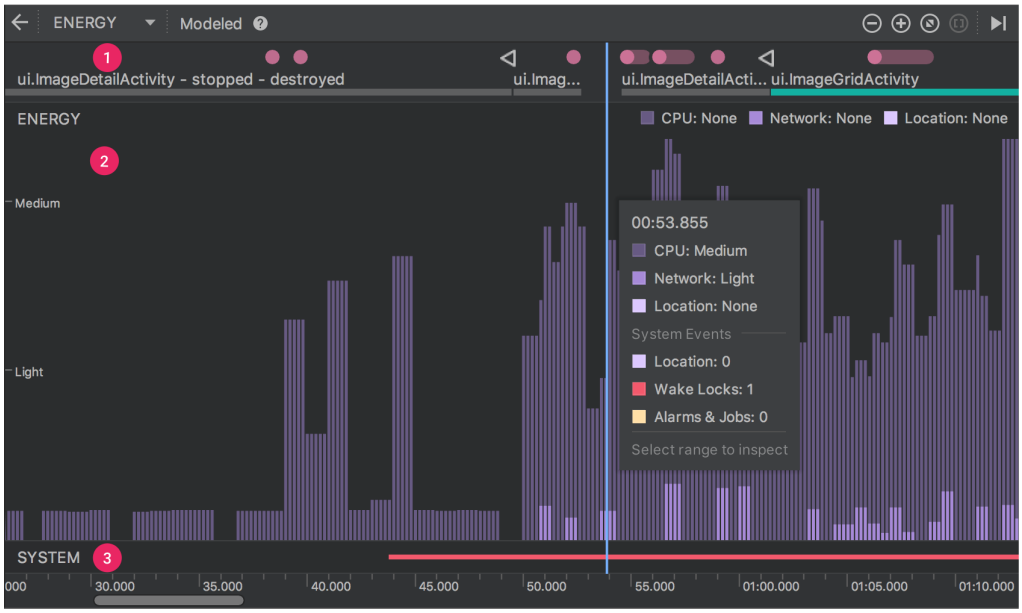

Рис. 1: Energy Profiler

Кроме того, Energy Profiler доступен при запуске приложений только на Android 8.0 и выше, что позволяет охватить около 60% от всех устройств [6].

3.1.2. Плагин AEON

AEON (Automated Android Energy-Efficiency Inspection) — плагин к Android Studio [7], который, согласно предложенному авторами описанию, предлагает следующую функциональность:

- статический анализ для обнаружения распространённых ошибок при использовании Android API (в частности, wake locks);
- оценка энергопотребления на уровне метода;
- профилирование и визуализация энергии.

Для использования данного плагина требуется установка приложения Trepn Profiler от компании “Qualcomm”, а также наличие Android-устройства с чипсетом серии Qualcomm Snapdragon. Кроме того, на весну 2020 года в доступе находится бета-версия плагина, а последние обновления в репозитории датируются сентябрём 2016 года[2].

[2]https://bitbucket.org/david_gonzalez_samudio/aeon_suite/src/master/

3.1.3. Плагин EnSights

EnSights — плагин к Android Studio, предоставляющий сведения о расходе энергии на основании изменения значений различных метрик между двумя версиями приложения [8]. Для измерений энергии используется инструмент PowerTutor [9].

На официальном сайте, где размещаются плагины для Android Studio, EnSights не найден. Это может свидетельствовать о том, что данный плагин так и не вышел за пределы исследовательской работы.

3.2. Navitas Framework

Navitas Framework[3] — проект по оценке энергопотребления Android-приложений, начавшийся в рамках летней школы “Ланит-Терком” в 2019 году. В данный момент участниками проекта являются студенты направления ”Программная инженерия” СПбГУ, а руководит проектом старший преподаватель Станислав Юрьевич Сартасов.

За время летней школы был создан плагин для Gradle, под названием “NaviProf”, выполняющий инструментовку APK, запуск UI-тестов на устройстве и выгрузку собранных логов на компьютер. А осенью, после успешной презентации проделанных работ, стали видны дальнейшие планы по развитию проекта:

- улучшение качества профилирования Gradle-плагина (например, добавление в логи информации о таких компонентах, как яркость экрана, Wi-Fi и др.);
- создание плагина для Android Studio с целью:
  - интеграции процесса профилирования “NaviProf” в процесс разработки Android-приложений;
  - проведения анализа собранных логов;
  - визуализации результатов профилирования.

[3] https://github.com/Stanislav-Sartasov/Navitas-Framework

3.3. Обзор технических средств

Android Studio построена на базе IntelliJ IDEA — среды разработки от компании JetBrains. Поэтому для работы требуется IntelliJ Platform SDK (Software Development Kit) и документация к нему [10]. Далее предлагается краткий обзор ключевых аспектов, касающихся разработки плагина в данной работе.

3.3.1. Конфигурационный файл

Конфигурационный файл *plugin.xml* — файл, в котором объявляются различные компоненты плагина (в частности, регистрируются действия), а также хранится дополнительная информация (название плагина и его описание, имена разработчиков и др.).

3.3.2. Система действий

IntelliJ Platform предоставляет концепцию действий. Действием является класс, унаследованный от класса *AnAction*. Его метод, под названием *actionPerformed*, выполняется при вызове действия. Благодаря данной концепции, плагин может добавлять собственные элементы в меню и на панели инструментов. Действия объединяются в группы, причём группы могут быть вложенными. Группа действий может образовывать меню или панель инструментов, а подгруппы внутри группы — подменю.

В плагине “Navitas Profiler” действия используются для открытия конфигурационного окна, запуска и остановки профилирования, а также для навигации.

3.3.3. PSI

Program Structure Interface (PSI) — слой, отвечающий за парсинг файлов и представление кода в виде иерархической модели. Разные языки программирования имеют разные PSI-модели.

В плагине “Navitas Profiler” API данного слоя используется для модификации файлов сборки *build.gradle* пользовательских проектов.

3.3.4. Пользовательский интерфейс

IntelliJ Platform включает огромное количество собственных Swing-компонентов. Далее будут рассмотрены основные из них, используемые в работе.

- Окна инструментов — окна внутри IDE для отображения информации, могут иметь свои панели инструментов и отображать несколько вкладок. Располагаются вдоль левого, правого и нижнего края главного окна IDE.
- *DialogWrapper* — базовый класс для создания модальных и некоторых немодальных окон, умеющий отображать текст ошибки в случае некорректных данных ввода и управлять диалогом при помощи “горячих” клавиш.
- *WizardDialog* — класс, наследуемый от *DialogWrapper* и позволяющий создавать многостраничные диалоги.
- Всплывающие окна — полумодальные окна, не имеющие кнопки закрытия и исчезающие автоматически при потере фокуса.
- Уведомления — немодальные окна, рекомендуемые к использованию вместо модальных окон при уведомлении пользователя об ошибках и в других ситуациях, когда может потребоваться его внимание.
- *JBTable*, *JBLabel*, *JBList*, *Tree* и многие другие.

“Navitas Profiler” имеет собственное окно инструментов. *WizardDialog* отлично подходит для окна конфигурации, а *JBTable и Tree* — для отображения информации об энергопотреблении.

4. Плагин Navitas Profiler

4.1. Архитектура

Плагин написан на языке Kotlin, с применением таких архитектурных шаблонов, как *Model-View-ViewModel (MVVM)* и *Repository*. Для подписки на события используется библиотека *RxJava 2*.

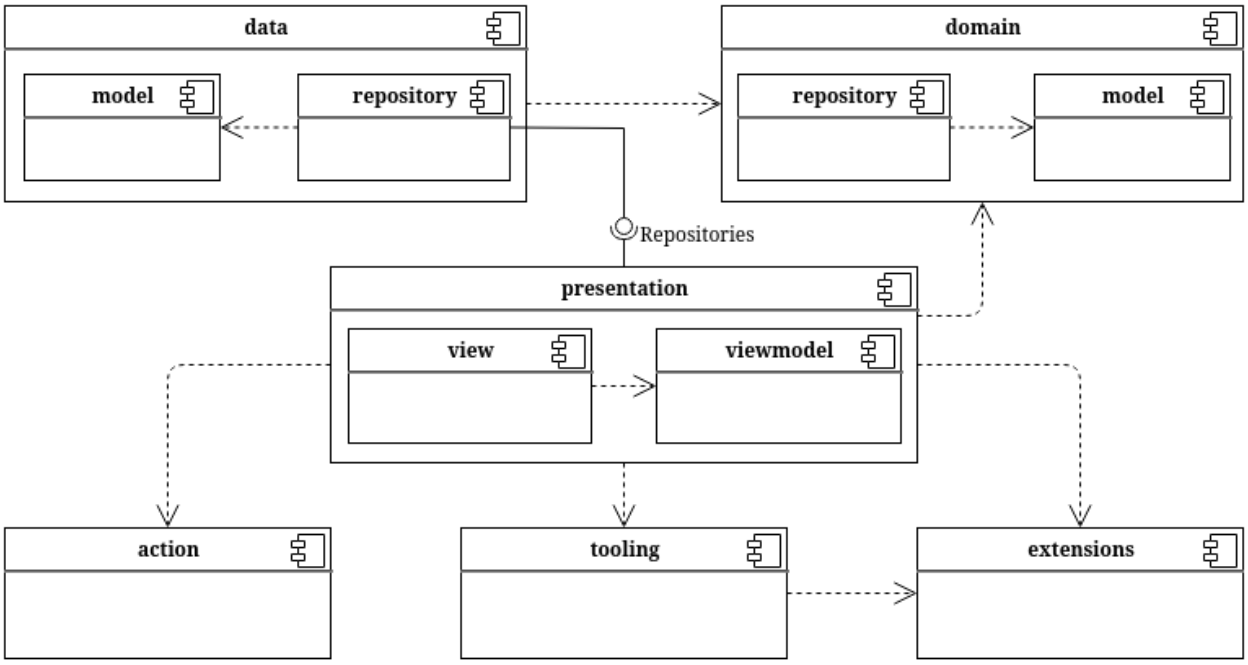

Рис. 2: Диаграмма компонентов

Плагин имеет трёхслойную архитектуру:

- В слое *domain* хранятся модели данных, используемые в бизнес-логике плагина. Обычно такие данные отображаются в пользовательском интерфейсе. Также здесь объявлены интерфейсы репозиториев.

- В слое *data* хранятся модели данных, не относящиеся к бизнес-логике напрямую, но при этом используемые в работе плагина. К примеру, из таких данных могут впоследствии создаваться данные для *domain*-слоя. Так, например, модели собранных логов с устройства из *data*-слоя в процессе работы анализатора преобразу-

ются в *domain*-модели с информацией об энергопотреблении. Также здесь реализованы интерфейсы репозиториев.

- В слое *presentation* лежат классы, связанные с пользовательским интерфейсом. Во *view* содержатся UI-компоненты, распределённые по функциональности на пакеты (например, задание конфигурации для профилирования и просмотр отчётов по результатам профилирования — это разные функциональности). Во *viewmodel* хранятся модели представления [11].

Также были выделены следующие компоненты (отдельно от слоёв):

- *action* — для действий, добавляемых в окно инструментов “Navitas Profiler”;
- *extensions* — для функций-расширений Kotlin;
- *tooling* — для различных классов-инструментов, таких как парсер JSON, анализатор собранных с устройства логов и др.

4.2. Работа с плагином

Процесс работы с плагином состоит из четырёх этапов. Вначале пользователь задаёт необходимую конфигурацию и запускает профилирование. Затем, по его завершении, собранная с устройства информация выгружается на компьютер и анализируется. После этого пользователь видит отчёт по результатам профилирования.

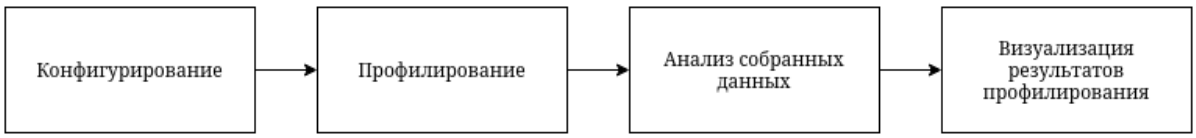

Рис. 3: Процесс работы с плагином

4.2.1. Конфигурирование

Пользователь указывает Android-модуль и тесты из этого модуля, которые будут запущены для профилирования. Также он указывает

Power Profile (данная функциональность — задача другого члена команды).

4.2.2. Профилирование

Для выполнения профилирования используется Gradle-задача из "NaviProf" под названием *defaultProfile*. Ей в параметры передаётся конфигурация, заданная пользователем, а запуск осуществляется благодаря методу *runTask* класса *ExternalSystemUtil*.

Однако, прежде чем запустить эту задачу, сначала нужно подключить "NaviProf" к целевому проекту. Это достигается за счёт модификации файлов *build.gradle*, через использование определённых методов класса *PsiFile*, позволяющих менять структуру PSI-модели.

4.2.3. Проведение анализа

На вход анализатору поступает JSON, содержащий собранные с устройства логи, а также Power Profile, выбранный пользователем на этапе конфигурирования. На выходе — информация об энергопотреблении стека вызовов (только пользовательских методов, без Android API и сторонних библиотек) для каждого теста, с разделением по процессам и потокам.

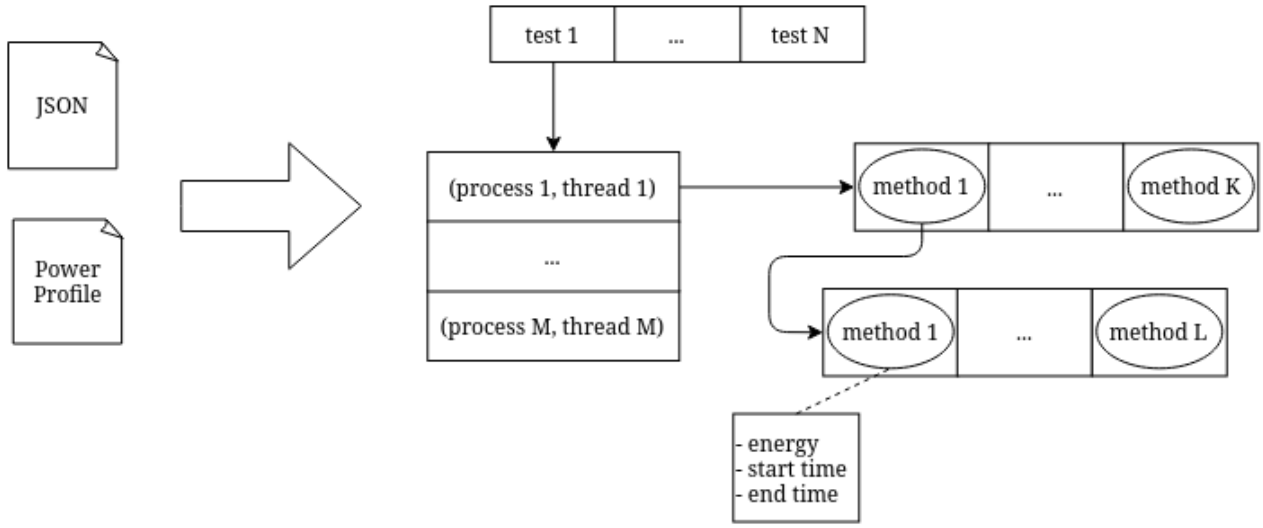

Рис. 4: Схема преобразования собранных данных

Оценка энергопотребления CPU на протяжении вызова метода вычисляется по следующей формуле:

$$E_{CPU} = \sum_{i \in cores} \sum_{j \in frequencies_i} (endTime_{ij} - startTime_{ij}) * C_{ij} \quad (1)$$

где $startTime_{ij}$ и $endTime_{ij}$ — время пребывания ядра i на частоте j в момент входа и выхода из метода соответственно, C_{ij} — коэффициент из *Power Profile*.

4.2.4. Отображение результатов

Информация об энергопотреблении всех тестов и каждого из них по отдельности представляется как в текстовом формате, так и в графическом (работа над графиками — задача другого члена команды).

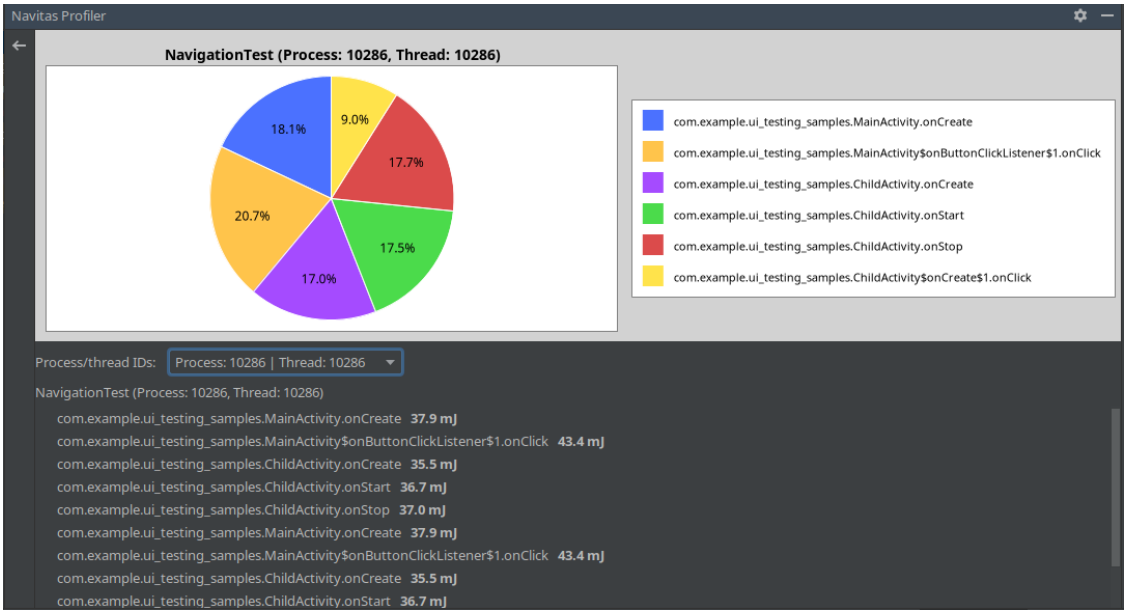

Рис. 5: Пример отчёта об энергопотреблении теста

4.3. Ограничения

1. Не поддерживается инструментовка файлов *build.gradle*, написанных на Kotlin.

2. “NaviProf” поддерживает один формат собираемых логов, однако для некоторых Android-устройств этот формат может быть иным.

5. Проверка работоспособности

Правильность работы плагина в ходе его разработки проверялась на специально созданном Android-приложении. Во время проверки тестировались различные сценарии использования плагина:

- корректное задание конфигурации (например, нельзя выбрать модуль без единого теста);
- выполнение модификации файлов *build.gradle*;
- правильность переходов между экранами внутри окна инструментов “Navitas Profiler”;
- отображение соответствующей выбранному тесту информации об энергопотреблении;
- достоверность отображаемого стека вызовов для конкретного теста.

Заключение

В результате работы были выполнены следующие задачи:

1. Проведён обзор существующих решений.
2. Проведён обзор технических средств по разработке плагинов для Android Studio.
3. Разработан плагин для Android Studio по оценке энергопотребления кода и проведена интеграция с Gradle-плагином по профилированию “NaviProf”.
4. Работа плагина проверена на тестовом Android-приложении.

У данной работы имеются дальнейшие перспективы по её развитию: добавление новых компонентов в анализ (Wi-Fi, Bluetooth и др.) и новых видов отчётов (например, трассы энергопотребления), а также возможности сопоставления результатов профилирования с предыдущими запусками.

Список литературы

[1] Number of smartphone users worldwide from 2016 to 2021. — URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (online; accessed: 22.11.2019).

[2] Mobile operating systems' market share worldwide from January 2012 to July 2019. — URL: https://cutt.ly/Ve6r1RW (online; accessed: 22.11.2019).

[3] Background optimizations. — URL: https://developer.android.com/topic/performance/background-optimization.html (online; accessed: 17.11.2019).

[4] App Standby Buckets. — URL: https://developer.android.com/topic/performance/appstandby (online; accessed: 17.11.2019).

[5] Inspect energy use with Energy Profiler. — URL: https://developer.android.com/studio/profile/energy-profiler (online; accessed: 17.11.2019).

[6] Google kills Android distribution numbers on the web, but we've got you covered. — URL: https://9to5google.com/2020/04/10/google-kills-android-distribution-numbers-web (online; accessed: 01.05.2020).

[7] AEON: Automated Android Energy-Efficiency Inspection. — URL: https://plugins.jetbrains.com/plugin/7444-aeon-automated-android-energy-efficiency-inspection/ (online; accessed: 17.11.2019).

[8] Hareem Sahar. Abdul Bangash. Mirza Beg. Towards Energy Aware Object-Oriented Development of Android Applications. — Sustainable Computing: Informatics and Systems, 2018. — URL: https://doi.org/10.1016/j.suscom.2018.10.005.

[9] PowerTutor: A Power Monitor for Android-Based Mobile Platforms. — URL: http://ziyang.eecs.umich.edu/projects/powertutor/ (online; accessed: 24.11.2019).

[10] IntelliJ Platform SDK. — URL: http://www.jetbrains.org/intellij/sdk/docs/welcome.html (online; accessed: 24.11.2019).

[11] Model–view–viewmodel. — URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel (online; accessed: 19.05.2020).