|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ | |
| ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | |
| УДК 681.3 | Кафедра математического обеспечения вычислительных систем |
| **Разработка нативных мобильных приложений на платформе Android**  Эссе | |
|  | Работу выполнил студент группы ПМИ-1,2-2021 4 курса института компьютерных наук и технологий  Решетников Михаил Андреевич  «\_\_\_» \_\_\_\_\_\_\_\_2025 г. |
|
| Пермь 2025 | |

Содержание

[1. Введение 3](#_Toc197894810)

[1.1.1. Инструменты и языки разработки 3](#_Toc197894811)

[1.1.2. Языки программирования 3](#_Toc197894812)

[1.1.3. Библиотеки и фреймворки 4](#_Toc197894813)

[1.2. Архитектура и компоненты Android-приложения 4](#_Toc197894814)

[1.2.1. Основные компоненты Android-приложения 4](#_Toc197894815)

[1.3. Пользовательский интерфейс и опыт (UI/UX) 5](#_Toc197894816)

[1.3.1. Дизайнерские рекомендации и Material Design. 5](#_Toc197894817)

[1.4. Управление данными и хранение 6](#_Toc197894818)

[1.4.1. Локальное хранение 6](#_Toc197894819)

[1.4.2. Сетевые коммуникации и внешние данные 7](#_Toc197894820)

[1.5. Интеграция с оборудованием устройства 7](#_Toc197894821)

[1.5.1. Геолокация 7](#_Toc197894822)

[1.5.2. Датчики (сенсоры) 8](#_Toc197894823)

[1.5.3. Камера и мультимедиа 8](#_Toc197894824)

[1.6. Безопасность Android приложений 8](#_Toc197894825)

[1.6.1. Модель безопасности 8](#_Toc197894826)

[1.6.2. Шифрование и хранение данных 8](#_Toc197894827)

[1.6.3. Защита от уязвимостей 9](#_Toc197894828)

[1.6.4. Permissions и политика магазина 9](#_Toc197894829)

[1.7. Тестирование и отладка 9](#_Toc197894830)

[1.7.1. Непрерывная интеграция 9](#_Toc197894831)

[1.8. Развертывание и дистрибуция 10](#_Toc197894832)

[1.8.1. Google Play Store и альтернативы 10](#_Toc197894833)

[1.9. Заключение 10](#_Toc197894834)

[Список источников 12](#_Toc197894835)

1. Введение

Мобильные приложения сегодня играют ключевую роль в цифровой экономике, а платформа Android занимает лидирующие позиции по количеству пользователей по всему миру. По данным отраслевых исследований, в магазине Google Play насчитывается порядка **2,4–3 миллионов** приложений, что отражает высокую конкуренцию и спрос [1]. Android сегодня является доминирующей мобильной платформой: примерно **70–72%** смартфонов в мире работают под управлением Android (против ~28-30% на iOS). По оценкам, количество активных устройств на Android превышает **3 млрд** по всему миру, что свидетельствует о гигантской аудитории пользователей [7]. Такая массовость обуславливает высокую привлекательность платформы для разработчиков, но и накладывает определённые сложности, в частности связанные с фрагментацией версий и устройств.

В отличие от кроссплатформенных или веб-приложений, нативные Android-приложения обладают более высокой производительностью и лучшей интеграцией с функциями устройства, обеспечивая высокий уровень отклика интерфейса и доступ ко всем аппаратным ресурсам. В академической и профессиональной среде нативная Android-разработка рассматривается как оптимальный подход для создания сложных и требовательных приложений, где важны скорость работы, безопасность данных и качественный пользовательский опыт. Ниже рассмотрены основные аспекты разработки нативных Android-приложений – от инструментов и архитектуры до тестирования и развертывания – с сохранением технической глубины и ключевых особенностей каждой из тем.

Инструменты и языки разработки

Для разработки под Android используется официальная среда **Android Studio**, предоставляемая Google. Android Studio включает все необходимые инструменты: редактор кода, дизайнер интерфейсов, отладчик, эмулятор устройств и систему сборки Gradle. **Gradle** управляет процессом сборки приложения, включая компиляцию исходного кода, обработку ресурсов и подключение библиотек. Проект Android содержит файловую структуру, разделяющую исходный код (папка java/kotlin), ресурсы интерфейса (res/layout, res/drawable и др.), файл манифеста (AndroidManifest.xml) и прочие служебные файлы. Такая структура упрощает поддержку масштабных проектов и соблюдение принятых на платформе соглашений о разбиении кода и ресурсов.

****Языки программирования****

Исторически основным языком разработки Android-приложений является Java. Google выбрал Java в качестве первого официального языка платформы, и значительная часть стандартной документации и примеров была изначально ориентирована на Java. В 2017 году Google объявил **Kotlin** вторым официальным языком для Android. Kotlin полностью совместим с Java, что позволяет постепенно внедрять его в существующие проекты. При этом приложения на Kotlin компилируются в тот же байт-код JVM/ART, что и Java-приложения, не уступая им в производительности или размере бинарных файлов. Kotlin отличается более лаконичным синтаксисом и сниженным количеством шаблонного кода, благодаря чему исходный код получается компактным и легче читается. В современном Android-разработке Kotlin быстро стал доминирующим языком для создания нового кода, хотя значительное количество проектов продолжает использовать и Java.

****Библиотеки и фреймворки****

Экосистема Android предлагает обширный набор библиотек, упрощающих разработку. Официальные библиотеки **Android Jetpack [3]** включают компоненты для работы с базами данных, визуальными, фоновыми задачами и многими другими аспектами. Использование этих библиотек способствует соблюдению лучших практик и совместимости с различными версиями Android. В Android Studio интегрированы менеджеры зависимостей (Gradle/Maven), позволяющие легко подключать внешние библиотеки. Разработчики также используют системы контроля версий и системы непрерывной интеграции для автоматической сборки и проверки приложений.

* 1. Архитектура и компоненты Android-приложения

Android-приложения исторически строились на паттерне MVC, однако Android-специфичная реализация эволюционировала к MVVM (Model-View-ViewModel) как рекомендуемому образцу. Google публикует официальное руководство **Guide to App Architecture** [2], где пропагандирует разделение ответственности и минимизацию логики во View-слоях.

В современной архитектуре обычно выделяют по крайней мере два слоя: **UI-слой** и **слой данных** (бизнес-логика, управление данными). Часто добавляется промежуточный **доменный слой** с бизнес-правилами и use-case, что соответствует принципам **Clean Architecture** (чистая архитектура, предложенная Робертом Мартином) – разделение на слои презентации, домена и данных с зависимостями направленными внутрь.

В типичной Android-реализации презентационный слой реализован через паттерн MVVM: **ViewModel** выступает в роли «презентера», хранящего состояние UI и логику взаимодействия, **View** (Activity/Fragment/Compose) подписывается на данные из ViewModel и отображает их. ViewModel обращается к репозиториям слоя данных, которые инкапсулируют доступ к источникам данных – локальным или удалённым (веб-API). Репозитории, в свою очередь, могут оперировать сущностями доменного слоя при наличии такового. Главное – каждый слой отвечает за свою задачу и не знает деталей реализации других. Такая модульность облегчает тестирование, масштабирование приложения и поддержку.

****Основные компоненты**** Android-приложения

Android-приложение состоит из нескольких типов компонентов, каждый из которых отвечает за свою задачу в приложении. Выделяются следующие ключевые компоненты:

* **Активности (Activity)** – компоненты интерфейса пользователя, представляющие отдельные экраны приложения. Активность обычно содержит элементы UI (кнопки, списки, поля ввода и т.д.) и управляет их поведением. Пользователь взаимодействует с приложением преимущественно через активности. В приложении может быть несколько экранов-активностей, между которыми осуществляется навигация.
* **Службы (Service)** – компоненты для фоновых операций, не требующие непосредственного взаимодействия с пользователем. Сервис выполняется в фоновом режиме, даже если приложение не активно на экране. Примеры использования служб: воспроизведение музыки в фоне, загрузка данных из сети, отслеживание геопозиции в реальном времени. Служба может запускаться как самим приложением, так и системой (например, расписанием или в ответ на какое-то событие).
* **Приемники широковещательных сообщений (Broadcast Receivers)** – компоненты, позволяющие приложению реагировать на глобальные события или события системы. Android рассылает сообщения при различных событиях: изменение состояния питания (подключение зарядки), завершение загрузки устройства, получение SMS и т.д. Приложение может зарегистрировать приемник, чтобы выполнить код при наступлении такого события. Приемники не имеют UI и обычно работают очень кратковременно, выполняя только нужные действия.
* **Поставщики контента (Content Providers)** – компоненты, обеспечивающие доступ к данным приложения другим приложениям или частям системы. Content Provider управляет общим хранилищем данных (например, базой данных SQLite) и по запросу предоставляет данные другим приложениям через универсальный интерфейс.

Перечисленные компоненты взаимодействуют друг с другом с помощью механизма **Intent** – специального объекта-сообщения, посредством которого можно запускать активности, службы или отправлять/принимать широковещательные уведомления. Intent-ы позволяют как внутри приложения переходить между экранами, так и вызывать компоненты других приложений.

* 1. Пользовательский интерфейс и опыт (UI/UX)

Интерфейс пользователя в Android-приложениях традиционно описывается с помощью XML-разметки и создается из **View-элементов**. Разметка определяет иерархию элементов интерфейса (виджетов), таких как кнопки, текстовые поля, списки, изображения, располагая их внутри контейнеров-**Layout**-ов (линейных, табличных, с относительным позиционированием. Разметки хранятся в ресурсах (res/layout) и могут быть переиспользуемыми для разных экранов и конфигураций. При запуске активности система загружает соответствующий XML-файл и превращает его в объекты View, которыми приложение может управлять программно.

Дизайнерские рекомендации и Material Design.

Для обеспечения единообразного пользовательского опыта в столь разнородной среде Google разработала систему дизайн-гайдов **Material Design**. Это набор принципов визуального и интерактивного дизайна, охватывающий цвета, типографику, анимации, поведение интерфейсов и пр. Material Design был представлен в 2014 году и стал стандартом для Android-приложений. Следование этим рекомендациям позволяет приложениям выглядеть и вести себя последовательно с ожиданиями пользователей Android. Официальная спецификация Material Design публикуется на сайте [7] и описывает лучшие практики UI/UX для Android. Например, рекомендуются такие паттерны, как плавающая кнопка действия для основной действия, **app bar** с навигацией и поиском, **navigation drawer** (выдвижное меню) для разделов приложения, использование стандартных компонентов (карт, списков, диалогов) из библиотеки Material Components. При создании нового проекта Android Studio по умолчанию применяет Material Design-тему к приложению, что задаёт базовый современный стиль.

За прошедшие годы Material Design эволюционировал. Актуальная версия – **Material Design 3**, представленная вместе с Android 12. Ключевая идея Material Design – персонализация и динамический дизайн: система автоматически подбирает цветовую схему интерфейса приложений на основе обоев пользователя, что делает внешний вид более индивидуальным. Кроме цветовых тем, Material Design уделяет внимание адаптивности под различные размеры экранов и новым форм-факторам (например, складные устройства). Для разработчиков это означает, что следует поддерживать динамическое тематическое оформление и проверять, как приложение выглядит в различных темах. В целом, соблюдение гайдлайнов Material Design повышает качество интерфейса: приложения выглядят профессионально, знакомо пользователю Android и хорошо масштабируются на разных устройствах.

Важно уделять внимание **UX (опыту пользователя)**: логичная навигация, единый стиль оформления, быстрое время отклика интерфейса. Android предоставляет множество готовых решений для улучшения UX – системы навигации между экранами фрагменты для многоразовых частей интерфейса, поддержка жестов и анимаций. Разработчик должен следить за тем, чтобы приложение реагировало на пользовательские действия без задержек, не блокировало главный поток длительными операциями. Для этого дорогостоящие задачи (загрузки из сети, вычисления) выполняются асинхронно – либо с помощью колбеков/хендлеров, либо с использованием современных средств вроде Kotlin Coroutines, которые упрощают написание асинхронного кода.

* 1. Управление данными и хранение

Практически любое приложение оперирует данными, и Android предоставляет разнообразные механизмы для их хранения. Выбор способа хранения зависит от характера данных и требований к их объему, приватности и устойчивости к закрытию приложения.

****Локальное хранение****

Для небольших объемов простых данных (настройки, флаги, параметры пользователя) используются **SharedPreferences** – хранилище пар «ключ-значение», позволяющее сохранять примитивные типы (строки, числа, булевы значения) в XML-файл, доступный только приложению. Это удобно для сохранения настроек конфигурации, состояний UI (например, последний открытый раздел) и другой информации, которую нужно помнить между запусками приложения.

Для структурированных данных Android поддерживает встроенную **SQLite**–базу данных. SQLite – легковесная реляционная СУБД, размещаемая локально в памяти устройства. Приложение может создавать и использовать любые таблицы и запросы SQL для управления своими данными. Однако прямое использование SQLite через API android.database.sqlite требует написания существенного объема шаблонного кода: необходимо самостоятельно определять схемы, писать SQL-запросы, обрабатывать курсоры результатов и т. д.

Для упрощения работы Google предлагает библиотеку **Room (Jetpack)** – надстройку над SQLite, которая обеспечивает объектно-реляционное отображение (ORM). Используя Room, разработчик описывает сущности (Entity) в виде классов данных и предоставляет интерфейсы DAO для операций, а библиотека генерирует весь необходимый код SQL под капотом.

Кроме баз данных, приложение может хранить данные во **внутренней памяти** устройства (Internal Storage) – это файлы, доступ к которым имеет только данное приложение. Они подходят для хранения приватных файлов, кэша, документов. Также возможна запись на **внешнее хранилище** (SD-карта, общая файловая система), но для этого требуются особые разрешения и учет того, что такие файлы могут быть прочитаны пользователем или другими приложениями.

****Сетевые коммуникации и внешние данные****

Часто локального хранения недостаточно, и приложение взаимодействует с внешними серверами или облачными сервисами. Android не накладывает ограничений на сетевое взаимодействие: доступны стандартные библиотеки для HTTP(S)-запросов (WebSocket-соединений, работы с RESTful и SOAP API и пр.). Следует осуществлять сетевые операции в фоновом потоке (например, через Kotlin coroutines/ExecutorService – в новых), чтобы не блокировать UI. Полученные с сервера данные (JSON/XML) обычно сразу сохраняются в локальные структуры или базу данных для кэширования и офлайн-доступа.

* 1. Интеграция с оборудованием устройства

Одно из преимуществ нативной разработки – полный доступ к аппаратным возможностям устройства через системные API. Android предоставляет разработчикам прямые интерфейсы для работы с датчиками и модулями смартфона.

****Геолокация****

Для определения местоположения используются модули GPS, а также данные сетей Wi-Fi и сотовой связи (AGPS). Android имеет класс LocationManager и сервисы Google Play Services, которые позволяют запрашивать координаты устройства с заданной точностью и частотой. Приложение может получать обновления местоположения в реальном времени, например, для навигации или фитнес-трекеров.

****Датчики (сенсоры)****

Современные устройства снабжены множеством сенсоров: акселерометр (ускорение по осям), гироскоп (угловая скорость вращения), магнитометр (компас), датчики освещенности, приближения, барометр, термометр и др. Android объединяет работу с ними через **SensorManager** и классы Sensor [4]. Разработчик может получить список доступных сенсоров на устройстве и зарегистрировать слушателей (SensorEventListener) для получения данных с определенной частотой. Сырые данные сенсоров позволяют создавать разнообразные функции: отслеживание жестов, шагомеры, определение ориентации устройства в пространстве, реакции на наклоны/повороты (например, для игр).

****Камера и мультимедиа****

Android дает возможность интегрироваться с камерой устройства либо опосредованно, либо напрямую через API. Прямой доступ более сложен, но позволяет создавать кастомные камеры внутри приложения, настраивать параметры съемки, обрабатывать поток с камеры в реальном времени (например, для дополненной реальности). Аналогично, имеются API для записи аудио (AudioRecord) и воспроизведения звука (AudioTrack, MediaPlayer), а также для работы с видео.

* 1. Безопасность ****Android**** приложений

Безопасность – критически важный аспект разработки, учитывая, что мобильные приложения оперируют персональными данными и могут быть мишенью для злоумышленников. Android как платформа обладает встроенными механизмами безопасности, но разработчик также несет ответственность за безопасное поведение своего приложения.

****Модель безопасности****

Как упоминалось, Android изолирует приложения друг от друга на системном уровне. Каждый апп по сути – отдельный пользователь Linux, и прямой доступ к файлам и данным других приложений невозможен без специальных привилегий. **Принцип наименьших привилегий** реализован в системе: приложение получает по умолчанию только базовый набор прав и должно явно запрашивать у пользователя доступ к чувствительным ресурсам. Пользователь может отказаться предоставить разрешение, и приложение должно корректно обработать этот случай (ограничить функциональность или повторно запросить при необходимости, пояснив причину).

****Шифрование и хранение данных****

Данные, сохраняемые приложением, следует защищать. Если приложение хранит конфиденциальную информацию (пароли, токены сессий, личные данные), рекомендуется использовать шифрование. Android предоставляет API для криптографии, а также **Android KeyStore** для хранения ключей. С помощью KeyStore можно генерировать и сохранять ключи шифрования так, что они будут защищены аппаратно и недоступны извне.

****Защита от уязвимостей****

Разработчик должен следовать безопасным практикам программирования, чтобы предотвратить распространенные уязвимости. К примеру, необходимо осторожно работать с вводом пользователя, чтобы избежать SQL-инъекций (использовать параметризованные запросы в SQLite/Room), проверять достоверность данных из внешних источников, не оставлять в коде «бекдоры» или пароли. В веб-взаимодействии следует применять только безопасные соединения (HTTPS) и удостоверяться в подлинности сертификатов. При обмене данными между приложениями через Intent или Content Provider, следует явно указывать флаги строгой выборки адресатов (чтобы посторонние приложения не перехватили интент, предназначенный конкретному компоненту) и проверять permissions у вызывающей стороны при получении запросов через content provider. Такой список уязвимостей известен и выложен в качестве помощника в устранении – OWASP Mobile Security [5].

****Permissions и политика магазина****

Google Play также вносит свой вклад в безопасность: перед публикацией приложения проходит автоматическую проверку Play Protect на наличие вредоносного поведения. Кроме того, политика маркета требует от разработчиков запрашивать только обоснованные разрешения и предоставлять пользователям прозрачность в отношении использования данных. Нарушение этих правил может привести к удалению приложения из маркета. В целом, сочетание системных средств (песочница, permissions, шифрование) и внимательного отношения разработчика к безопасности обеспечивает надежную защиту. Тем не менее, история показывает, что уязвимости случаются даже в популярных приложениях, поэтому тестирование на безопасность (пен-тесты, код-ревью с фокусом на безопасность) является желательным этапом в цикле разработки.

* 1. Тестирование и отладка

Тестирование приложения – необходимое условие обеспечения его качества и надежности перед выпуском. Android-разработка располагает развитой инфраструктурой для различных видов тестов. На этапе разработки программисты активно используют эмуляторы Android (встроенные в Android Studio Android Virtual Devices), позволяющие проверить работу приложения на разных версиях ОС и форм-факторах без наличия каждого физического устройства. Также применяются реальные устройства для проверки специфичных функций (например, датчиков, камеры) в реальных условиях.

****Непрерывная интеграция****

В профессиональной разработке тестирование интегрировано в процессы CI/CD. Каждый коммит кода может запускать сборку приложения и прогон всех автотестов на сервере. Если какой-то тест падает, команда немедленно узнает об этом, что позволяет быстро исправить проблему. Такой подход предотвращает внедрение регрессий и поддерживает высокое качество кода на протяжении всего жизненного цикла проекта.

Отладка приложения также поддерживается на высоком уровне: Android Studio позволяет в режиме реального времени просматривать значения переменных, устанавливать точки останова, отслеживать выполнение потоков.

* 1. Развертывание и дистрибуция

После разработки и тестирования наступает этап развертывания приложения и его доставки конечным пользователям. Android-приложение компилируется в единый пакет – **APK (Android Package)**, представляющий собой архив с байт-кодом, ресурсами и манифестом, подписанный цифровой подписью разработчика. Подпись удостоверяет целостность и издателя приложения: при каждом обновлении система проверяет, совпадает ли сертификат, иначе обновление не будет установлено. В 2018 году Google ввел новый формат **Android App Bundle (AAB)** – формат публикации, из которого автоматически генерируются оптимизированные APK для разных конфигураций устройств при загрузке с Google Play. App Bundle позволяет снизить размер загружаемых файлов, устанавливая на устройство только необходимые ресурсы (например, нужный язык, специфичные для данного разрешения экрана изображения и т.д.).

****Google Play Store и альтернативы****

Является основным каналом распространения Android-приложений. Для публикации разработчику необходимо зарегистрироваться в качестве издателя, подготовить страницу приложения (иконка, описание, скриншоты, категория, возрастной рейтинг и пр.) и загрузить APK/AAB через консоль разработчика. Перед появлением в магазине приложение проходит проверку на соответствие правилам (отсутствие вредоносного кода, запрещенного контента, соблюдение требований к приватности). После одобрения приложение становится доступно для пользователей по всему миру.

Помимо официального магазина, Android позволяет альтернативные способы дистрибуции. Приложение можно распространять через сторонние магазины (Samsung Galaxy Store, Huawei AppGallery, отечественные магазины типа RuStore и др.), или даже напрямую – выкладывая APK-файл на сайте для скачивания.

Наконец, после публикации начинается этап сопровождения приложения: сбор метрик и логов (через встроенные механизмы аналитики Firebase Analytics или внешние платформы), отслеживание сбоев (Crashlytics) и оперативный выпуск исправлений.

* 1. Заключение

Разработка нативных мобильных приложений для Android сочетает в себе множество технологий и навыков – от владения языками программирования и понимания архитектуры приложения до умения оптимизировать работу с данными и обеспечивать безопасность. Нативный подход обеспечивает разработчика всеми возможностями платформы: прямым доступом к оборудованию, высокопроизводительным выполнением кода, гибкостью в реализации пользовательского интерфейса по стандартам платформы. Благодаря этому нативные Android-приложения способны предоставить пользователям лучший опыт использования устройств, интегрируясь в экосистему Google. Тем не менее, успешный Android-разработчик должен следовать лучшим практикам проектирования (разделение ответственности между компонентами, паттерны проектирования вроде MVVM/MVP для поддержки читаемого и тестируемого кода), учитывать ограничения мобильной среды (питание, память, прерываемость работы), а также постоянно обновлять свои знания в стремительно развивающейся области. Современные инструменты – такие как Jetpack Compose для UI, библиотеки Jetpack для архитектуры и данных, облачные сервисы для тестирования и аналитики – позволяют создавать все более качественные и сложные приложения с меньшими затратами времени.

## Список источников

1. **Appbooster.** Краткие итоги 2023-го года для рынка мобильных приложений // Sostav.ru [Электронный ресурс]. – URL: https://www.sostav.ru/blogs/268746/43070 (дата обращения: 10.05.2025).
2. **Application fundamentals – App Architecture** [Электронный ресурс]. – URL: https://developer.android.com/guide/components/fundamentals?hl=ru (дата обращения: 10.05.2025).
3. **Jetpack Compose – Modern toolkit for building native UI** [Электронный ресурс]. – URL: https://developer.android.com/compose (дата обращения: 10.05.2025).
4. **Sensors Overview – Sensors and Location** [Электронный ресурс]. – URL: https://developer.android.com/develop/sensors-and-location/sensors/sensors\_overview?hl=ru (дата обращения: 10.05.2025).
5. **OWASP Mobile Application Security** – [Электронный ресурс]. – URL: https://mas.owasp.org/ (дата обращения: 10.05.2025).
6. **Material Design 3** - Google's latest open-source design system – [Электронный ресурс]. – URL: https://m3.material.io/ (дата обращения: 10.05.2025).
7. **Mobile Operating System Market Share Worldwide | Statcounter Global Stats** – [Электронный ресурс]. – URL: https://gs.statcounter.com/os-market-share/mobile/worldwide#:~:text=Mobile%20Operating%20Systems%20Percentage%20Market,23