УДК 004.52 **Информационные технологии**

Горячкин Борис Сергеевич, *кандидат технических наук, доцент; Московский государственныйтехнический университет им. Н.Э. Баумана,*
E-mail: bsgor@mail.ru
Богданов Дмитрий Александрович, *магистрант, Московский государственный технический университет им. Н.Э.Баумана,*
E-mail: bogdanov.d.a.99@gmail.com
Щипицина Ксения Владимировна, *магистрант, Московский государственный технический университет им. Н.Э.Баумана,*
E-mail: ksenvaks@gmail.com

ЭРГОНОМИЧЕСКИЙ АНАЛИЗ СПОСОБОВ РАЗРАБОТКИ ИНТЕРФЕЙСА В ANDROID ПРИЛОЖЕНИИ

Аннотация: В данной статье рассматривается влияние разных способов разработки интерфейса Android приложения на установочный apk-файл. В качестве объекта исследования выбран непосредственно сам apk-файл, его структура, а также размер. Рассмотрены основные подходы к разработке интерфейса для Android приложения, предпосылки для их создания, основные отличия, а также достоинства и недостатки. Разобрано устройство установочного файла Android приложения, из чего он состоит, какие параметры влияют на его размер.

Ключевые слова: Android, XML, интерфейс, Jetpack Compose, приложение, APK-файл, Kotlin, анализ, размер.

Abstract: This article discusses the impact of different ways of developing the Android application interface on the installation apk file. The apk-file itself, its structure, and also the size was chosen as the object of study. The main approaches to the development of an interface for Android applications, the prerequisites for their

creation, the main differences, as well as advantages and disadvantages are considered. The device of the Android application installation file is disassembled, what it consists of, what parameters affect its size.

Keywords: Android, XML, interface, Jetpack Compose, application, APK – file, Kotlin, analyze, size.

Введение

Практически любое мобильное приложение имеет дисплейный интерфейс. Это своего рода мост, через который пользователь может общаться с нашей программой. Сам интерфейс может быть интуитивно понятным и легким в освоении, а может быть сложным и чересчур перегруженным. К сожалению, разработчик зачастую не в силах контролировать само качество интерфейса, под которым подразумевается грамотное расположение кнопок, полей для ввода, размер шрифта и так далее. Этими вещами обычно занимаются дизайнеры, а перед разработчиком просто ставят задачу - сверстать экран согласно макету и написать работающий код. Однако у разработчика есть пространство для выбора способа написания интерфейса. Если мы говорим о Android разработке, то интерфейс мобильного приложения можно писать двумя принципиально разными способами. Один из них это давно устоявшийся, скажем так проверенный метод. Второй – новая, активно продвигаемая компанией Google библиотека. Соответственно перед разработчиком стоит выбор, использовать проверенный годами метод или попробовать воспользоваться новой технологией, с расчетом на то, что для ее изучения нужно будет потратить определенное количество человеко-ресурсов [1]. Однако перед тем, как выбрать второй путь необходимо выяснить, стоит ли вообще изучать новую технологию. Так ли она хороша на самом деле? Нужно провести первичный анализ и понять, как она влияет на общую работоспособность приложения, скорость сборки, размер установочного файла и прочее.

В рамках данной статьи авторы погружаются в создание установочного файла Android – приложения, анализируют его и выясняют, как каждая

технология влияет на размер и его составляющие.

Разработка с использованием XML

Первый способ разработки визуальной составляющей Android – приложения подразумевает использование XML файлов. XML – это расширяемый язык разметки, который предоставляет правила для определения любых данных. В отличие от других языков программирования, XML не может выполнять вычислительные операции сам по себе [2]. Вместо этого для управления структурированными данными можно использовать любой язык программирования или программное обеспечение. XML - файл состоит из различных тегов и атрибутов, которые задают верстку всему экрану. В среде разработки, благодаря использованию специальных инструментов, можно пронаблюдать, как будет выглядеть макет на различных типах и форматах дисплеев.

Однако как упоминалось выше, сам по себе XML файл представляет только разметку, то есть то, как элементы интерфейса будут выглядеть и располагаться на экране. Для управление логикой работы экрана нам понадобиться использовать дополнительный код, который необходимо написать на языке Java или Kotlin. Только там можно устанавливать обработчики пользовательских событий, «слушать» нажатия и прочее.

Данный метод хорош только тем, что раньше интерфейс Android-приложений можно было писать только с помощью него. То есть любой опытный специалист хорошо владеет этой технологией и будущему работодателю не придется тратить ресурсы на его обучение. Но этот подход имеет и ряд недостатков. К примеру, использование XML файлов сильно размазывает логику работы нашего приложения по различным модулям. Особенностью всех Android приложений является то, что все XML файлы не лежат рядом с кодом. Они лежат в отдельном месте в проекте. Это иногда сильно усложняет разработку, заставляет программиста «бегать» по разным модулям в поиске нужного файла. Например, даже для создания самого простейшего экрана с одной кнопкой, необходимо создавать как минимум два файла, один XML и

один Kotlin или Java файл. Ниже приведена общая схема работы описанного выше подхода (рис. 1).

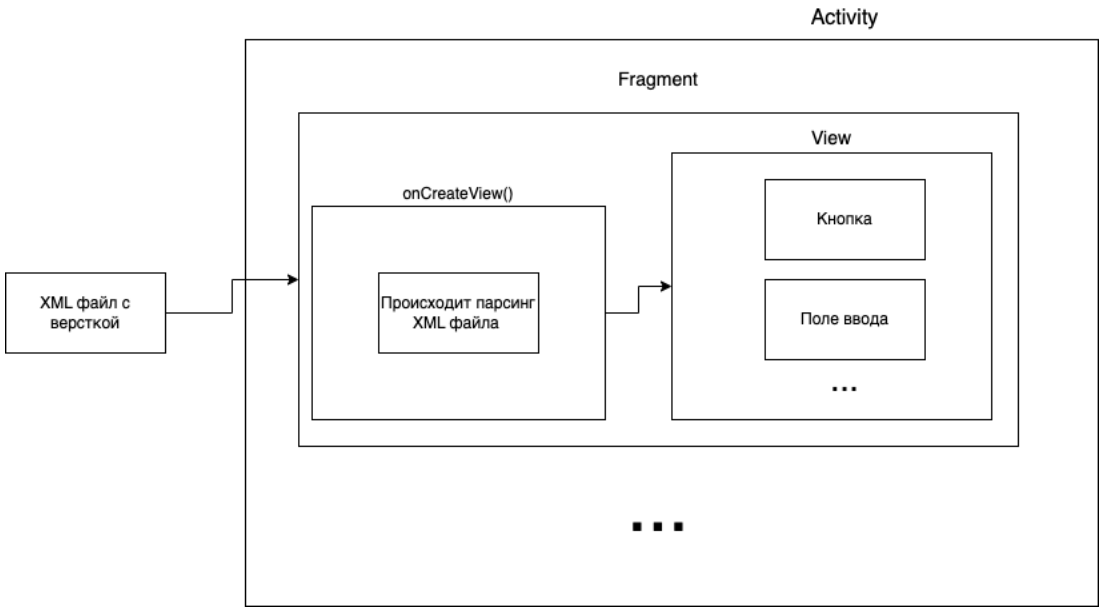

Рис 1. Схема работы первого подхода с XML

Разработка через библиотеку Jetpack Compose

Трендом мобильной разработки за последние несколько лет стал декларативный UI. Такое решение уже давно успешно применяется в веб и кроссплатформенных решениях и, наконец, добралось и до нативной разработки. На iOS существует SwiftUI (представленный на WWDC 2019), а на Android – Jetpack Compose (представленный месяцем ранее на Google I/O 2019). В ней вообще не используются XML файлы, а написание кода происходит в функциональном стиле, а не в объектно-ориентированном [3].

Как и упоминалось выше, одним из главных преимуществ нового подхода является отказ от использования XML файлов, то есть больше не нужно переключаться между классами и xml-файлами – вся работа с UI происходит в одном Kotlin-файле. При использовании этого подхода код становится очень компактным, помещается в один файл и вообще не содержит дополнительных классов.

Общую схему работы можно представить в следующем виде (рис. 2).

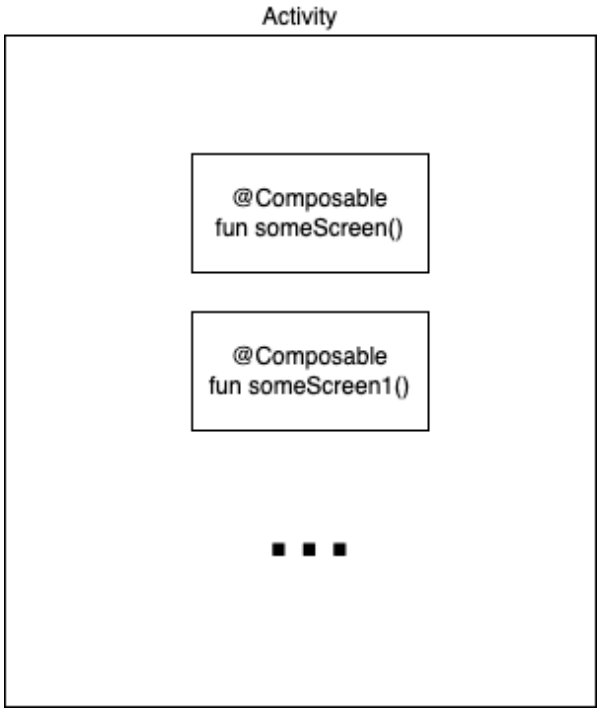

Рис 2. Схема работы подхода с Jetpack Compose

Средства, инструменты и объект анализа

Средства

Для проведения анализа будем использовать исходный код и установочные файлы приложения Tivi, которые находятся в открытом доступе [4].

Инструменты

В качестве инструментов для проведения исследований будем использовать среду разработки Android Studio и встроенный анализатор apk файлов.

Объект

Объектом нашего исследования будет APK – файл. Это специальный тип файлов, необходимых для установки приложения на устройство под управлением операционной системы Android. Одного такого файла обычно достаточно, чтобы установить целое и работающее приложение [5].

Рассмотрим структуру APK – файла подробнее

APK файл состоит из основных частей:

1. classes.dex - один или несколько файлов, которые представляют собой код приложения, скомпилированный в Dalvik-байткод.

2. res - папка, в которой лежат андроид-ресурсы в Binary XML формате.

3. assets - папка, содержащая ассеты приложения (специальный раздел, куда можно вложить различные файлы с некомпилируемыми разрешениями, например txt, jpg, mp4 и так далее).

4. resources.arsc - файл, в котором хранится таблица маппинга id ресурсов в соответствующие файлы.

5. папка META-INF содержит следующие файлы:

a. MANIFEST.MF – манифест-файл, который содержит SHA-хэши всех файлов в APK-пакете

b. Сертификат приложения

c. Файлы с дополнительной метаинформацией

6. AndroidManifest.xml – манифест Андроид-приложения. Этот файл хранится в скомпилированном Binary XML формате.

Сравнительный анализ установочных файлов

Рассмотрим два установочных файла. Первый был получен при использовании XML – файлов, второй при разработке через библиотеку Jetpack Compose [6].

XML - 4,2 Мб

Jetpack Compose – 2,6 Мб

Разница – 1, 6 Мб (~ 1,6 раз)

Для того, чтобы понять, почему произошли такие существенные изменения в конечном размере установочного файла, необходимо рассмотреть подробнее каждую часть APK- файла, где непосредственно произошли эти самые изменения.

Папка с ресурсами

Папка «res» хранит в себе все файлы разметки, которые используется в приложении. Сравним размер этой папки с использованием двух разных подходов.

XML – 343 Кб

Jetpack Compose – 63 Кб

Разница – уменьшение на 280 Кб (в 5,5 раз)

Как было упомянуто ранее, подход с использованием библиотеки Jetpack Compose практически исключает использование XML файлов, что сильно уменьшает размер этой папки. В свою очередь папка «res» подразделяется еще на несколько подпапок, которые мы рассмотрим далее, чтобы понять за счет чего происходит уменьшение размера более чем в 5 раз.

Папка «layout»

В данной папке находятся «макеты» различных экранов и их составляющие. При использовании первого подхода размер данной папки составляет 75,5 Кб. При использовании второго подхода, данная папка вовсе отсутствует. Это объясняется тем, что использование библиотеки Jetpack Compose освобождает разработчика от создания специальных отдельных файлов разметки, так как все описание интерфейса и логики работы экрана происходит в файлах Kotlin.

Папка «drawable»

Папка с ресурсами-изображениями. Поддерживает форматы JPG, GIF, PNG (самый предпочтительный) и др. Каждое изображение является отдельным файлом и получает собственный идентификатор, который формируется по имени файла без расширения [7].

В первом случае вес данной папки составляет 40,7 Кб. Во втором случае все импортирование изображений происходит через код, поэтому данная папка вовсе отсутствует в установочном файле.

Папка «color»

Папка, в которой хранятся цвета, используемые по всему приложению. При использовании первого подхода каждый цвет задается в виде xml-файла. Однако при использовании второго подхода все необходимые цвета задаются в отдельном Kotlin файле, к которому можно также получить доступ из любой части проекта. Таким образом мы добиваемся уменьшения размера общей папки с ресурсами еще на 20,5 Кб.

Папка «anim»

В данной папке лежат анимации, используемые в приложении [8]. При разработке первым способом размер составляет 12,6 КБ. При разработке вторым способом, анимации для различных объектов интерфейса указываются непосредственно в коде, и соответственно данная папка отсутствует в результирующем APK – файле.

Исходя из всего вышеперечисленного можно с уверенностью говорить о том, что применение декларативного UI освобождает от использование файлов с xml-ресурсами, что в свою очередь ведет к значительному уменьшению (~ 5 раз) папки с ресурсами приложения.

Dex - файл

Рассмотрим файл classes.dex, который занимает большую часть всего установочного файла.

Файл с расширением .dex содержит код для исполнения в виртуальной машине Android Runtime. Каждый APK имеет отдельный файл classes.dex, который ссылается на все классы или методы, используемые в приложении. По сути, любой класс, используемый в нашем коде, будут преобразован в байты, то есть в файл dex, который можно запустить как приложение Android [9].

Далее рассмотрим этот файл детальнее, чтобы понять какие изменения в нем произошли в зависимости от использования одного или другого методов.

Рассмотрим модуль showdetails. На рисунке ниже представлен набор его классов при использовании первого подхода (рис. 3).

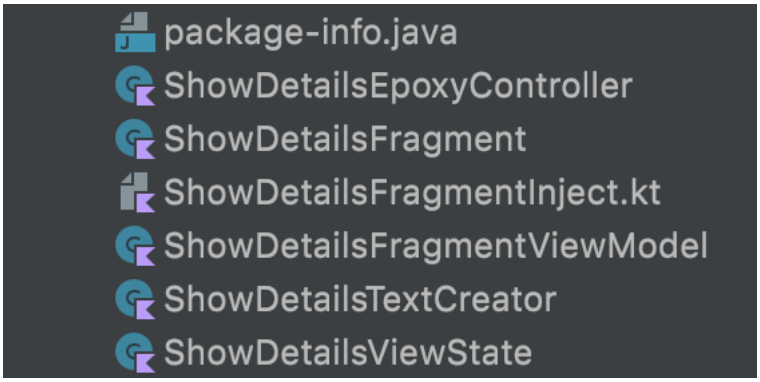

Рис 3. Структура модуля details при первом методе

Если не брать в расчет файл package-info.java (он служебный), то для функционирования этого экрана необходимо использовать 6 файлов. Посмотрим, что будет содержать в себе этот модуль при использовании второго подхода (рис. 4).

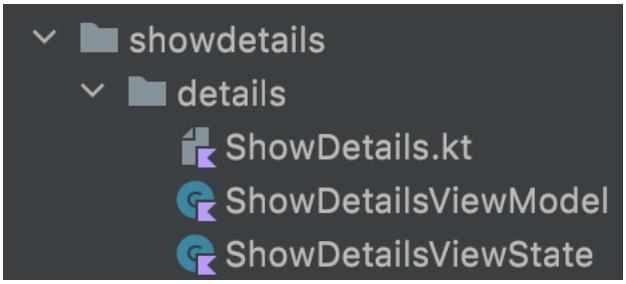

Рис 4. Структура модуля details при втором методе

Всего 3 файла. С использованием Jetpack Compose количество необходимых классов уменьшилось в два раза.

Также посчитаем количество строк кода в каждом классе

В первом случае следующие классы занимают:

- ShowDetailsEpoxyController – 338 строк
- ShowDetailsFragment – 242 строки
- ShowDetailsFragmentInject.kt – 33 строки
- ShowDetailsFragmentViewModel – 204 строки
- ShowDetailsTextCreator – 127 строк
- ShowDetailsViewState – 43 строки

Всего 987 строк

С использованием второго метода:

- ShowDetails.kt – 1136 строк
- ShowDetailsViewModel – 197 строк
- ShowDetailsViewState – 43 строки

Всего 1376 строк

Также посмотрим на размер модуля в скомпилированном виде.

При первом подходе –32 Кб

При втором подходе – 23 Кб

При втором подходе количество строк в файлах увеличилось, а общий размер модуля уменьшился. Все дело в том, что, при использовании Jetpack Compose используется функциональный подход, который уменьшает количество необходимых файлов, но в свою очередь увеличивает их размер. Однако если смотреть на результирующий размер модуля после компиляции, выигрывает все равно второй подход с меньшим количеством файлов.

Из этого можно сделать вывод о том, что при использовании второго подхода, происходит уменьшение количества необходимых Kotlin-файлов в 2 раза.

Подключаемые библиотеки

Также стоит рассмотреть подключаемые к проекту библиотеки. Одна из основных библиотек, необходимых для работы с разными версиями Android – это библиотека поддержки AndroidX. Она предоставляет дополнительные классы и функции, недоступные в стандартном API Framework, для упрощения разработки и поддержки на большем количестве устройств.

Размер Android библиотек, подключаемых при использовании первого подхода составляет:

- AndroidX – 655 Кб
- Android – 100 Кб

Размер Android библиотек, подключаемых при использовании второго подхода составляет:

- AndroidX – 109 Кб
- Android – 59 Кб

Тут тоже стоит отметить только положительные изменения. При использовании второго способа сокращается размер импортируемых библиотек более чем в 4 раза. Объяснить это можно тем, что подход с XML – файлами используется с самого начала становления всей Android – разработки, соответственно на данный момент необходимо использовать большее количество компонентов библиотеки AndroidX для поддержания

работоспособности приложения на разных версиях операционных систем в сравнении со вторым подходом, который отличается своей универсальностью при работе с разными версиями Android API.

Заключение

В данной работе были рассмотрены два подхода к созданию интерфейса Android приложения. Производилось сравнение двух установочных файлов, полученных разными методами. В результате анализа доказано, что при использовании библиотеки Jetpack Compose размер установочного файла уменьшается более чем в 1,5 раза. Полученная разница довольна существенная и вынуждает перейти от подхода с использованием XML файлов разметки, к использованию новой библиотеки, активно продвигаемой компанией Google.

Библиографический список:

1. Разработка интерфейса в Android приложениях [Электронный ресурс] – Режим доступа: https://developer.android.com/develop/ui (дата обращения: 10.12.2022).
2. Работа с XML в Android [Электронный ресурс] – Режим доступа: https://developer.android.com/develop/ui/views/layout/declaring-layout (дата обращения: 12.12.2022).
3. Jetpack Compose [Электронный ресурс] – Режим доступа: https://developer.android.com/jetpack/compose (дата обращения: 14.12.2022).
4. Проект Tivi [Электронный ресурс] – Режим доступа: https://github.com/chrisbanes/tivi (дата обращения: 16.12.2022).
5. Голощапов А.Л. Google Android: системные компоненты и сетевые коммуникации. - СПб.: БХВ-Петербург, 2012. - 384 с.
6. Jetpack Compose before and after [Электронный ресурс] – Режим доступа: https://medium.com/androiddevelopers/jetpack-compose-before-and-after8b43ba0b7d4f (дата обращения: 20.12.2022).
7. Фелкер Д. Android: разработка приложений. - М.: Диалектика, 2012. - 336 с.

8. Хашими С., Коматинени С., Маклин Д. Разработка приложений для Android. - СПб.: Питер, 2011. - 736 с.

9. Майер Р. Android 4. Программирование приложений для планшетных компьютеров и смартфонов. - М.: Эксмо, 2013. - 816 с.