ORES

Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004

ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА БАЗЕ ФРЕЙМВОРКА JETPACK COMPOSE

Попов Анатолий Анатольевич,
доцент техн. наук, доцент кафедры системотехники,
Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева», г. Красноярск,
e-mail: tolynbms@yandex.ru

Юрьев Андрей Дмитриевич,
студент 4 курса института информатики и телекоммуникаций,
Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева», г. Красноярск,
e-mail: a.yurev.stud@mail.ru

Кушнер Эмиль Игоревич,
студент 4 курса института информатики и телекоммуникаций,
Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева», г. Красноярск,
e-mail: emil.kushner.stud@mail.ru

Аннотация

В данной статье рассматривается фреймворк Jetpack Compose, реализованный на основе языка программирования Kotlin, его особенности и сравнение декларативного подхода разработки интерфейса Android приложений с императивным подходом.

Ключевые слова: android, kotlin, jetpack compose, разработка, интерфейс.

FEATURES OF DESIGNING THE USER INTERFACE OF MOBILE APPLICATIONS BASED ON THE JETPACK COMPOSE FRAMEWORK

Popov A. A.,
candidate of Technical Sciences,
Siberian State University of Science and
Technology named after Academician M.F. Reshetnev, Krasnoyarsk,
e-mail: tolynbms@yandex.ru

Yurev A. D.,
4th year student, Institute of Informatics and Telecommunications
Siberian State University of Science and

Technology named after Academician M.F. Reshetnev, Krasnoyarsk,
e-mail: a.yurev.stud@mail.ru

Kushner E. I.,
4th year student, Institute of Informatics and Telecommunications
Siberian State University of Science and
Technology named after Academician M.F. Reshetnev, Krasnoyarsk,
e-mail: emil.kushner.stud@mail.ru

ABSTRACT

This article discusses the Jetpack Compose framework, implemented based on the Kotlin programming language, its features, and a comparison of the declarative approach to developing the Android application interface with the imperative approach.

Keywords: android, kotlin, jetpack compose, development, interface.

Введение

На текущий момент разработка мобильных приложений занимает важное место в современной индустрии, так как у каждого второго человека на планете имеется смартфон. В самой разработке обязательной частью является создание интерфейса – проводника между приложением и пользователем. При создании больших проектов имеет место выбор подхода и фреймворков для разработки, так как они будут влиять на временные и денежные затраты, а также на качество продукта.

Цель исследования

Анализ фреймворка Jetpack Compose для Kotlin и сравнение императивного и декларативного подходов разработки интерфейса.

Материалы исследования

Jetpack Compose – это фреймворк на основе Kotlin, предназначенный для создания дизайна пользовательского интерфейса для Android приложений. Он был представлен компанией Google в 2021 году.

Создание интерфейса при помощи Jetpack Compose считается полностью декларативным программированием, означает то, что можно описать свой пользовательский интерфейс, используя набор компонуемых элементов. Последние десять лет использовался традиционный способ императивного проектирования пользовательского интерфейса [1]. Данная парадигма программирования позволяет разработчику сосредоточиться на том, что именно он должен получить в интерфейсе на основе получаемых данных.

Особенности проектирования на Jetpack Compose

Создание пользовательского интерфейса с помощью Compose реализуется определением набора составных функций, которые принимают данные и создают элементы пользовательского интерфейса. Основной единицей при проектировании интерфейса является функция с аннотацией @Composable. Таким образом компилятор понимает, какие функции предназначены для преобразования данных в элемент UI. В

качестве параметров функции могут подаваться любые данные, которые будут использоваться для отображения. В тело подпрограммы помещается контент, который необходимо вывести в данном информационном блоке. Функции Compose ничего не возвращают, так как они не создают виджеты интерфейса, а только описывают нужное для разработчика состояние экрана. Вывод спроектированного дизайна на основной экран происходит при помощи компонента setContent, в который помещают composable-функции. В декларативном подходе элементы интерфейса не представляются как объекты. Интерфейс обновляется, вызывая одну и ту же функцию, но с разными аргументами [2].

Основными элементами являются объекты Text, Button, Canvas, Image, TextField, CheckBox, RadioButton, Selectable, TopAppBar и BottomAppBar, Snackbar, Switch и другие компоненты. Чтобы правильно расположить данные элементы внутри одного блока используются такие контейнеры, как Box, Row, Column, Surface, Grid, а также Lazy-контейнеры и Flow-контейнеры.

Рассмотрим простой пример Composable функции с именем Sample, который выводит иконку звонка и текст с надписью «Call me», расположенные на цвете teal_200. Интерфейс, который необходимо получить показан на рисунке 1.

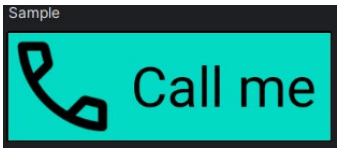

Рисунок 1 – Планируемый интерфейс

Чтобы элементы не накладывались друг на друга, их необходимо поместить в контейнер строки Row, на который добавляется свойства нажатия кнопки (без события) и цвета фона. Затем располагается сначала иконка, для которой выбрано изображение из рисунков, загруженный в проект, и задан размер изображения в 50 dp, при этом описание иконки отсутствует, так как является декоративным элементом. После задаётся текст, имеющий поля с текстовой переменной со значением «Call me», кеглем текста в 26 sp, и модификатором отступа в 7 dp (для выравнивания по вертикали). Листинг функции предоставлен на рисунке 2.

```
@Preview(showBackground = true)
@Composable
fun Sample(){
    Row (modifier = Modifier
        .clickable { }
        .background(colorResource(id = R.color.teal_200))){ this: RowScope
        Icon(
            painter = painterResource(id = androidx.core.R.drawable.ic_call_answer),
            contentDescription = null,
            modifier = Modifier
                .size(50.dp))
        Text(
            text = "Call me",
            fontSize= 26.sp,
            modifier = Modifier
                .padding(7.dp)
        )
    }
}
```

Рисунок 2 – Листинг функции Sample

Пользователь взаимодействует с интерфейсом приложения, и интерфейс вызывает события onCLick. Эти события информируют логику приложения о взаимодействии, которая может изменить состояние приложения. Если это произошло, вызываются составные функции с новыми данными. Происходит рекомпозиция виджетов.

Составные функции могут вызываться достаточно часто. Они должны быть быстрыми, не провоцируя зависания. Компонуемые функции не имеют четкой последовательности выполнения и могут выполняться параллельно. Рекомпозиция пропускает максимально возможное количество компонуемых функций. Она оптимистична и может быть отменена. Это означает, что Compose ожидает завершения рекомпозиции, прежде чем данные снова изменятся. Иначе, Compose может отменить ее и запустить снова с новыми параметрами.

Сравнение старого и нового подхода

Основными преимуществами данной библиотеки считаются простота в использовании, совмещение с императивным подходом, быстрая и бесперебойная работа.

Графический пользовательский интерфейс в Android исторически представляет собой иерархию в виде дерева виджетов. Из-за частого изменения состояния приложения иерархию интерфейса необходимо постоянно обновлять. Происходит это проходом по дереву и изменением необходимых узлов с помощью специальных функций. Такой подход увеличивает шанс возникновения ошибки. Это происходит потому, что обновление UI – это ответственность разработчика. А чем сложнее UI в приложении, тем выше вероятность того, что разработчик допустит какую-либо ошибку.

Декларативный подход позволяет упростить создание и обновление пользовательского интерфейса[3]. В его основе лежит неизменяемый UI и его зависимость от состояния приложения. И когда состояние приложения изменится Jetpack Compose перекомпонует UI так, чтобы он соответствовал текущему состоянию. Проблема такого подхода заключается в дополнительных затратах с точки зрения времени, вычислительной мощности и использования батареи, но платформа Compose может разумно перекомпановать только те элементы, которые изменились.

Заключение

Задача данной статьи была показать особенности декларативного подхода проектирования интерфейса с помощью фреймворка Jetpack Compose. В ходе изучения материала были выявлены достоинства и недостатки декларативного подхода по отношению к императивному.

Использование Jetpack Compose заметно упрощает разработку пользовательских интерфейсов Android приложений и делает ее приятным и эффективным занятием, так как разработчику не приходится прописывать весь алгоритм реагирования интерфейса на состояние приложения.

Список литературы:

1. Jetpack Compose Introduction URL: https://www.jetpackcompose.net/jetpack-compose-introduction (дата обращения: 16.08.2024).
2. Мыслить в композиции | Jetpack Compose | Android Developers. URL: https://developer.android.com/develop/ui/compose/mental-model?hl=ru (дата обращения 16.08.2024). – Текст. Изображение: электронные.
3. Jetpack Compose: руководство для начинающих | AppMaster URL: https://appmaster.io/ru/blog/jetpack-compose-rukovodstvo-dlia-nachinaiushchikh (дата обращения 17.08.2024).

References:

1. Jetpack Compose Introduction URL: https://www.jetpackcompose.net/jetpack-compose-introduction (дата обращения: 16.08.2024).

2. Thinking in Compose | Jetpack Compose | Android Developers. URL: https://developer.android.com/develop/ui/compose/mental-model?hl=ru (дата обращения 16.08.2024).
3. Jetpack Compose: A beginner’s guide | AppMaster URL: https://appmaster.io/ru/blog/jetpack-compose-rukovodstvo-dlia-nachinaiushchikh (дата обращения 17.08.2024).