Література:

1. Про судову експертизу: Закон України від 25 лютого 1994 року № 4038-XII. URL: https://zakon.rada.gov.ua/laws/show/4038-12#Text (дата звернення: 20.06.2022).
2. Судові експертизи в процесуальному праві України : навч. посіб. / за заг. ред. О. Г. Рувіна. Київ : Видавництво Ліра-К, 2019. 424 с.

DOI https://doi.org/10.30525/978-9934-26-277-7-160

JETPACK COMPOSE: NEW APPROACHES TO ANDROID UI DEVELOPMENT

Marchenko S.
Lecturer at the Department of Computer Engineering and Information Technologies
Cherkasy State Business College
Cherkasy, Ukraine

Android API is all about constant changes and adoption of the cutting-edge technologies. Till the recent times its native UI development part was quite established with traditional imperative approach based on XML markup with UI logic in Java/Kotlin. But with growing popularity of cross-platform mobile solutions like React Native or Flutter the time has come also to rethink the process of UI development in whole.

Declarative UI building brings a new perspective to the development process. At its core it states that rebuilding of parts of your UI from scratch is more preferable than modifying them. Declarative UI-frameworks propose the idea that changes in state may trigger a rebuild of the UI. Such approach removes a whole bunch of state-related bugs.

The trend of declarative UI for mobile projects began in 2013 with React Native by Facebook. Now it is a strong controversy regarding its adoption due to many “in-house” frameworks, possible problems in scheduling of development process for the external developers, emergence of UI-frameworks made by platform vendors, including SwiftUI and Jetpack Compose [1]. Recently the latter UI-framework have reached stable version. Overall comparison of traditional XML-based and Compose UI development is shown in Table 1 [2; 3].

Basic building blocks of Jetpack Compose are Composable functions. By annotating functions as @Composable we are essentially telling the compiler that the function intents to convert data into a UI node to register (return of the function) in the composable UI "tree". Such action is called "emitting". Also such annotating changes the type of the function or expression that it is applied to, imposes some constraints or properties over it. The Compose runtime expects composable functions to comply to the mentioned properties, so it can assume certain behaviors and therefore exploit different runtime optimizations.

Table 1

XML-based vs Compose UI development

Criteria	XML-based	Jetpack Compose
Approach	Imperative	Declarative
UI representation	View hierarchy as a tree of UI widgets	Regeneration the entire screen from scratch, applying only the necessary changes
Layout design	XML + UI logic in Java/Kotlin	Fully in Kotlin
Separation of concerns	Tightly coupled layout and ViewModel, often implicitly	Follow cohesion principle due to the same language for UI design and logic
Reaction to app state changes	UI hierarchy needs to be updated to display the current data	Intelligently chooses which parts of the UI to need to be redrawn at any given time.
UI update process	Traverse the tree, change nodes. Correspondent methods change the internal state of the widget.	Reduces the overhead of navigating the UI tree. Composables can hold a state and re-run on state changes. This is called recomposition.
View configuration and customization	By XML attributes	By Compose modifiers

Any function that is annotated as @Composable gets translated by the Jetpack Compose compiler to a function that implicitly gets an instance of a Composer context passed as a parameter, and that also forwards that instance to its Composable children. The compiler will add an implicit Composable parameter to each Composable call on the tree, also with some markers to the start and end of each composable [4, p. 3–6]. A composable function can only be called within another composable. With smaller composables we get a more flexible component structure. Particularly, such approach makes UI design of lists and grids much simpler. Instead of layouting a list, single item of this list and special adapter for data population, Jetpack Compose proposes only LazyColumn for showing list items in screen. We can describe item design in item/items scope [5].

Jetpack Compose introduced many performance improvements. As an example, we can refer to [6] and acknowledge substantial decreasing of frozen frames and reduction in median page load duration. In addition, Compose improves your build time and APK size. At this point major performance issues are caused by composables that can be skipped or not during recomposition [7].

Jetpack Compose brings to Android developer community the latest approaches to UI building. Due to similarity between ordinary Kotlin functions and composables, new layouting tools fit neatly into the set of Android developer skills. Also Jetpack Compose has much smoother learning curve, though current versions of this UI framework and Android Studio IDE still have some bugs and instability.

References:

1. Steinberger P. The new shiny. On the shift from imperative to declarative UI, and what it might mean for the apps we build today and tomorrow // *Increment*. 2021. Vol. 18. P. 61–64. URL: https://increment.com/mobile/the-shift-to-declarative-ui/ (дата звернення: 19.11.2022).

2. Making the shift to Jetpack Compose. URL: https://medium.com/androiddevelopers/jetpack-compose-before-and-after-8b43ba0b7d4f (дата звернення: 19.11.2022).

3. Understanding Jetpack Compose – part 1 of 2. URL: https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050 (дата звернення: 19.11.2022).

4. Castillo J., Shikov A. Jetpack Compose Internals. Lean Publishing, 2022. 114p.

5. Say Hello to Jetpack Compose and Compare with XML. URL: https://blog.kotlin-academy.com/say-hello-to-jetpack-compose-and-compare-with-xml-6bc6053aec13 (дата звернення: 19.11.2022).

6. Comparing Jetpack Compose performance with XML. URL: https://medium.com/okcredit/comparing-jetpack-compose-performance-with-xml-9462a1282c6b (дата звернення: 19.11.2022).

7. Jetpack Compose Stability Explained. URL: https://medium.com/androiddevelopers/jetpack-compose-stability-explained-79c10db270c8 (дата звернення: 19.11.2022).

DOI https://doi.org/10.30525/978-9934-26-277-7-161

LOW-COMPLEXITY LIDAR POINT CLOUD FILTERING METHOD FOR SELF-DRIVING VEHICLES

Matvienko V. T.
Candidate of Physico-Mathematical Sciences,
Associate Professor at the Department of Complex Systems Modelling
Taras Shevchenko National University of Kyiv
Kyiv, Ukraine

Mushta I. A.
Graduate student at the Department
of Electronic Computational Equipment Design
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine

Measurements obtained from LiDAR sensor always contain noise detections. The origin of this noise is different. Most LiDAR systems suffer degradation from adverse environment conditions. The photodetector of the lidar system detects transient light from the sun and the surroundings, and this light produces noise that hinders the system's effectiveness. Also adverse weather conditions, such as snow, dust, heavy rain or fog, distort the point cloud image obtained by LiDAR sensors. So to obtain high quality LiDAR point cloud filtering methods are used. Traditional filtering algorithms are often limited to isolated outliers, cannot identify outlier groupings or, otherwise, remove a lot of useful environmental features. What`s more, some of them are too complex to have ideal real-time