Лабораторийн ажил 4

Unit 4: Navigation and Architecture

Гүйцэтгэсэн: О.Саруулгэрэл /22B1NUM1637/

Шалгасан: Г.Ууганбаяр

2025 он

Улаанбаатар хот

PATHWAY 1: ARCHITECTURE

Activity Lifecycle (Амьдралын мөчлөг)

Android үйлдлийн систем дээрх Activity lifecycle буюу амьдралын мөчлөг гэдэг нь нэг Activity үүсэхээс эхлэн устах хүртэл дамждаг үе шатууд болон тэдгээртэй холбоотой callback аргууд юм. Kotlin хэл дээр эдгээрийг override (дахин тодорхойлж) ашиглан, Activity-ийн төлвийг зөв удирдаж, системийн нөөцийн үр дүнтэй хэрэглээг зохицуулдаг.

Lifecycle-ийн үндсэн үе шатууд болон аргууд:

1. **onCreate()**: Activity анх үүсэх үед нэг удаа дуудна.
2. **onStart()**: Activity хэрэглэгчид харагдахын өмнөх үед дуудна.
3. **onResume()**: Activity урд талд харагдаж, харилцах боломжтой үед дуудна.
4. **onPause()**: Activity-аас түр гарахад дуудна.
5. **onStop()**: Activity харагдахаа больж арын дэвсгэрт шилжих үед дуудна.
6. **onRestart()**: onStop() төлөвөөс дахин эхлэх үед дуудна.
7. **onDestroy()**: Activity бүр мөсөн устах үед дуудна.

Үе шат дамжих дараалал:

- Activity үүсэх үед: onCreate() → onStart() → onResume()
- Activity өөр Activity-д дарагдах үед: onPause() → onStop()
- Activity руу буцаж ирэх үед: onRestart() → onStart() → onResume()
- Activity устах үед: onPause() → onStop() → onDestroy()

Log Class

Kotlin хэл дээр Log class нь Android програмын ажиллагааг ажиглах, алдаа илрүүлэх, тэмдэглэл үүсгэх зорилготой ба Logcat-д тэмдэглэлийг харуулдаг.

Log-ийн үндсэн функцууд:

- Log.v() — Verbose (дэлгэрэнгүй)
- Log.d() — Debug (дебаг)
- Log.i() — Info (мэдээлэл)
- Log.w() — Warning (анхааруулга)
- Log.e() — Error (алдаа)

Параметрүүд:

- **TAG**: Логийг ялгах богино тэмдэглэгээ (классын нэрийг ихэвчлэн ашиглана).
- **msg**: Логт гаргах богино мэдээлэл.

Ашиглалтын жишээ:

```
private val TAG = "MainActivity"
Log.d(TAG, "onCreate Called")
```

UI State ба ViewModel

UI State

UI нь хэрэглэгчийн дэлгэцэнд харагдах элементүүд бөгөөд UI state нь апп-ын өгөгдөл дээр суурилсан дэлгэцийн төлөв юм. UI state өөрчлөгдөх үед UI автоматаар шинэчлэгддэг.

remember болон StateFlow

StateFlow нь UI-д state-ийг харуулахад ашиглагддаг дата урсгал юм. MutableStateFlow нь UI-д өөрчлөлт орж ирэх үед шууд дамжуулдаг.

ViewModel

ViewModel нь UI-д хэрэгтэй мэдээллийг хадгалж, UI-д мэдээлэл дамжуулах үүрэгтэй. Activity-г устгах үед ViewModel доторх мэдээлэл хадгалагдаж үлддэг тул програмын өгөгдлийн тогтвортой байдлыг хангана.

Unidirectional Data Flow (Нэг чиглэлтэй өгөгдлийн урсгал)

Нэг чиглэлт өгөгдлийн урсгал нь state (төлөв) UI руу дамжиж, хэрэглэгчийн үйлдэл (events) буцаад state-г удирдах хэсэг рүү дамждаг архитектурын зарчим юм.

Үндсэн зарчмууд:

- ViewModel нь UI-ийн ашиглах мэдээллийг хадгална.
- UI нь ViewModel-ээс мэдээллийг авч харуулна.
- UI хэрэглэгчийн үйлдлийг ViewModel-д мэдэгдэнэ.
- ViewModel хэрэглэгчийн үйлдлийг боловсруулж state-г өөрчилнө.
- State өөрчлөгдөх үед UI автоматаар шинэчлэгдэнэ.

Quiz

1. Which method is first called when the app no longer has focus?

    a. onPause()
    b. onStart()
    c. onCreate()
    d. onStop()

2. After ___, the app is no longer visible on screen.

    a. onPause()
    b. onStart()
    c. onCreate()
    d. onStop()

3. Use ___ to write a debug message. This method takes two arguments: the log tag and the log message.

a. Log.i()
b. Log.d()
c. Log.e()
d. Log.w()

4. To save a value that needs to survive a configuration change, declare its variables with ___.

   a. MutableState{}
   b. rememberSaveable{}
   c. remember{}
   d. State Hoisting

5. The separation of concerns design principle states that the app should be divided into classes, each with separate responsibilities.

   a. True
   b. False

6. The UI is what the user sees, while the UI state is what the app says they should see.

   a. True
   b. False

7. According to the recommended app architecture, each application should have at least the following two layers:

   a. The domain layer and the data layer
   b. The UI layer and the data layer
   c. Repository layer and the UI layer
   d. The domain layer and the UI layer

8. StateFlow is a data-holder observable flow that emits the current and new state updates.

   a. True
   b. False

9. Which of the following configurations should be added to the build.gradle file to add dependencies for the unit test source code?

   a. implementation
   b. testImplementation
   c. debugImplementation
   d. androidTestImplementation

10. Unit tests are executed on an Android device or emulator.

    a. True
    b. False

PATHWAY 2: NAVIGATION

Navigation Component

Navigation Component нь дэлгэцүүдийн хооронд шилжилт хийх үйлдлийг хялбаршуулсан Jetpack Compose болон Android-ийн нэг хэсэг юм. Энэ нь аппликэйшнд олон дэлгэц (screen) байхад хооронд нь холбож, навигацийг хялбар болгоно.

NavController

NavController нь аппликейшний доторх навигацыг удирдах гол элемент юм. Энэ нь дэлгэц хооронд шилжилт хийх, буцах үйлдлийг гүйцэтгэх зэрэг бүх навигацийн командыг удирддаг.

Жишээ:

```
navController.navigate("detail/$itemId")
navController.popBackStack()
```

NavGraph

NavGraph нь аппликэйшний дэлгэцүүдийг хооронд нь холбосон зураглал юм. Энэ зураглал нь дэлгэцүүд болон тэдгээрийн хоорондох шилжилтүүдийг тодорхойлдог. Ингэснээр дэлгэц хоорондын логик, бүтэц тодорхой, ойлгомжтой болдог.

NavHost

NavHost нь аппликэйшний дэлгэцүүдийг тодорхойлох болон тэдгээрийн хооронд чиглэл тогтооход ашиглагддаг үндсэн composable юм.

Жишээ:

```
@Composable
fun MyNavHost(navController: NavHostController) {
    NavHost(navController = navController, startDestination = "home") {
        composable("home") { HomeScreen(navController) }
        composable("detail/{itemId}") { backStackEntry ->
            val itemId = backStackEntry.arguments?.getString("itemId")
            DetailScreen(itemId)
        }
    }
}
```

Back Stack

Back stack-ийг ашиглан өмнөх дэлгэц рүү буцах эсвэл тодорхой дэлгэцүүдийг устгах боломжтой.

Жишээ:

```
navController.popBackStack() // Өмнөх дэлгэц рүү буцах
navController.popBackStack("home", inclusive = false) // home дэлгэц рүү буцах
```

Intent

Intent-ийг ашиглан бусад аппликэйшн руу өгөгдөл дамжуулах боломжтой.

Жишээ:

```
val context = LocalContext.current
val shareIntent = Intent(Intent.ACTION_SEND).apply {
   type = "text/plain"
   putExtra(Intent.EXTRA_TEXT, "Өгөгдөл хуваалцах текст")
}
context.startActivity(Intent.createChooser(shareIntent, "Хуваалцах"))
```

Quiz

1. A route is defined with a(n) ___ data type.
    a. @Composable function
    b. NavHost.Route
    c. String
    d. NavRoute
2. With a NavHost, you must explicitly specify a starting screen.
    a. True
    b. False
3. It’s considered best practice to not pass a NavHostController to individual composables.
    a. True
    b. False
4. ___ is a composable that manages which screen is displayed based on a given route.
    a. NavController
    b. NavHostController
    c. NavHost
    d. ComposableNavigator

5. The composable() function called in a NavHost takes which two parameters?

    a. Destination content and a route
    b. A route and composable content
    c. A path and a composable
    d. Composable content and an intent.

6. You can change the currently displayed route using the ___ method.

    a. update()
    b. composable()
    c. transition()
    d. navigate()

7. The ___ method removes one or more screens from the backstack.

    a. popToStartDestination()
    b. popBackStack()
    c. popComposable()
    d. popToBackStack()

8. In a multi-screen app, navigating to a new screen puts it on the bottom of the backstack.

    a. True
    b. False

9. Intent ___ contain additional data passed to an Intent.

    a. arguments
    b. extras
    c. parameters
    d. properties

10. StateFlow is a data-holder observable flow that emits the current and new state updates.

    a. True
    b. False

11. Which of the following are true about the Back and Up buttons?

    a. The Back button is a system button
    b. The Up button is provided by the system at the bottom of the screen
    c. The Back button is part of the AppBar
    d. The Up button in the AppBar automatically navigates to the previous screen.
    e. The Back button only appears if you use navigation.
    f. The Up button can be shown or hidden, depending on the current screen.

PATHWAY 3: ADAPT FOR DIFFERENT SCREEN SIZES

Jetpack Compose-д Navigation Graph болон NavHost-гүйгээр дэлгэц солих боломжтой. Энэ нь жижиг аппликейшнд, жишээ нь Reply апп-д хоёр дэлгэцийн хооронд шилжихэд тохиромжтой бөгөөд runtime-д mutable state ашиглан хийгдэнэ.

Window Size Classes

Window Size Classes нь төхөөрөмжийн дэлгэцийн хэмжээг ангилж, апп-ын UI-г тухайн хэмжээнд тохируулан өөрчлөх боломж олгодог. Энэ нь:

- **Responsive Design**: Дэлгэцийн хэмжээнээс хамааран UI-г динамикаар өөрчлөх.
- **Cross-Device Compatibility**: Гар утас, таблет, эвхэгддэг төхөөрөмж зэрэгт ижил код ашиглах.
- **Material Design-д нийцэх**: Google-ийн загварын удирдамжийг дагах.

Window Size Classes нь хоёр хэмжээсийг (өргөн ба өндөр) тус тусад нь ангилдаг бөгөөд тус бүрт гурван төрөл байдаг:

1. **Compact** (жижиг)
2. **Medium** (дунд)
3. **Expanded** (том)

Breakpoints

Breakpoints нь дэлгэцийн хэмжээг пикселээр (dp, density-independent pixels) хэмжиж, тухайн хэмжээг аль ангилалд хамааруулахыг тодорхойлдог. Эдгээр хязгаарын цэгүүд нь Material Design-ийн удирдамжид тодорхойлогдсон бөгөөд дараах байдалтай байна:

1. Өргөний Breakpoints

- **Compact**: 0 dp - 599 dp
    - Жишээ: Ихэнх ухаантай гар утасны босоо байрлал (portrait).
    - Хэрэглээ: Нэг баганатай layout, жижиг дэлгэцэд тохиромжтой.
- **Medium**: 600 dp - 839 dp
    - Жишээ: Таблетын босоо байрлал эсвэл том утасны хэвтээ байрлал (landscape).
    - Хэрэглээ: Хоёр баганатай layout эсвэл навигацийн талбартай UI.
- **Expanded**: 840 dp ба түүнээс дээш
    - Жишээ: Том таблет, эвхэгддэг төхөөрөмж, эсвэл десктоп.
    - Хэрэглээ: Олон баганатай эсвэл бүрэн дэлгэцийн UI.

2. Өндрийн Breakpoints

- **Compact**: 0 dp - 479 dp
    - Жишээ: Гар утасны хэвтээ байрлал.
    - Хэрэглээ: Хязгаарлагдмал босоо зайд тохиромжтой минимал UI.
- **Medium**: 480 dp - 899 dp
    - Жишээ: Гар утасны босоо байрлал эсвэл жижиг таблет.
    - Хэрэглээ: Стандарт босоо layout.
- **Expanded**: 900 dp ба түүнээс дээш
    - Жишээ: Том таблет эсвэл эвхэгддэг төхөөрөмж босоо байрлалд.
    - Хэрэглээ: Том босоо зайд илүү контент харуулах.

Canonical Layouts

Canonical Layouts нь том дэлгэц дээрх UI-ийн стандарт загварууд бөгөөд дизайны эхлэл болон хэрэгжүүлэлтийн удирдамж болдог. Эдгээр загварууд нь Material Design-ийн зарчмууд дээр суурилсан бөгөөд дэлгэцийн хэмжээ, breakpoints (хязгаарын цэгүүд)-ээс хамааран апп-ын элементүүдийг зохион байгуулахад тусална. Гурван үндсэн Canonical Layout байдаг:

1. **List-Detail Layout** (Жагсаалт-Дэлгэрэнгүй Xagd): Жагсаалт ба дэлгэрэнгүйг хамтад нь харуулна.
2. **Supporting Panel Layout**: Хажуугийн туслах хавтанг ашиглан нэмэлт контент харуулна.
3. **Feed Layout**: Мэдээллийн урсгал (feed) хэлбэрээр контентыг зохион байгуулна.

Эдгээр загварууд нь хэрэглэгчийн хүлээлт, апп-ын хэрэглээний тохиолдлуудыг харгалзан дэлгэцийн хэмжээнд дасан зохицох боломжийг олгоно.

List-detail view

List-detail view нь хэрэглэгчдэд жагсаалтаас тодорхой зүйлийг сонгоход түүний дэлгэрэнгүй мэдээллийг хажууд нь эсвэл тусдаа харуулдаг UI загвар юм. Энэ нь:

- **Жижиг дэлгэц дээр (Compact)**: Жагсаалтыг нэг дэлгэц дээр, дэлгэрэнгүйг дараагийн дэлгэц дээр харуулна.
- **Дунд дэлгэц дээр (Medium)**: Жагсаалт ба дэлгэрэнгүйг хуваалцсан байдлаар харуулж болно, гэхдээ зай хязгаарлагдмал.
- **Том дэлгэц дээр (Expanded)**: Жагсаалт ба дэлгэрэнгүйг нэг дэлгэц дээр хоёр баганатай хэлбэрээр харуулж, дэлгэцийн зайг бүрэн ашиглана.

Асуудал:

Том дэлгэц дээр энгийн загварыг ашиглавал:

- Контент хэт сунжирч, хоосон зай их үлдэнэ.
- Хэрэглэгчийн туршлага муудна, учир нь дэлгэцийн "real estate" (дэлгэцийн зай) үр ашиггүй ашиглагдана.

Шийдэл:

List-Detail Canonical Layout-ыг ашиглан жагсаалт ба дэлгэрэнгүйг хоёр баганатай хэлбэрээр зохион байгуулж, дэлгэцийн зайг оновчтой болгоно.

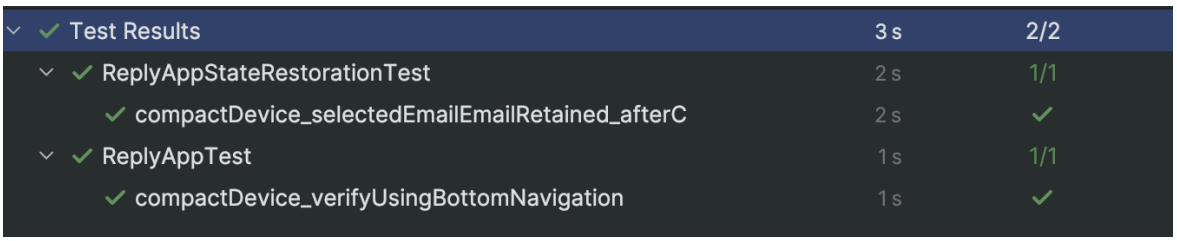
Test

"Build an app with an adaptive layout" codelab-ийн Compact Дэлгэцийн Тест

Compact дэлгэц дээр доод навигацийн элемент (bottom navigation) байгааг шалгах тест:

```
@Test
@TestCompactWidth
fun compactDevice_verifyUsingBottomNavigation() {
    // Set up compact window
    composeTestRule.setContent {
        ReplyApp(
            windowSize = WindowWidthSizeClass.Compact
        )
    }
    // Bottom navigation is displayed
    composeTestRule.onNodeWithTagForStringId(
        R.string.navigation_bottom
    ).assertExists()
}
```

Тестийн үр дүн:



Quiz

1. The ___ composable is used to respond to the Back button, with or without a NavHost.

    a. BackButton
    b. BackHandler
    c. BackNavigator
    d. BackStack

2. Which of the following are true about designing for larger screens?
Choose as many answers as you see fit.

    a. Button positioning is more important on larger screen sizes.
    b. Usually no changes are needed to the UI layout to make the app work well for larger screen sizes.
    c. Adding another layout to the same screen removes the need to navigate between screens.
    d. Large screen layouts should avoid placing commonly used buttons in the center of the screen.

3. A ___ is a specific measurement of width or height where an app's layout should change.

    a. window class
    b. layout point
    c. size bucket
    d. breakpoint

4. The compact width window size class generally refers to smaller devices, such as phones in portrait mode.

    a. True
    b. False

5. The ___ API makes the implementation of adaptive layouts simpler.

    a. SizeClass
    b. WindowSizeState
    c. SizeBucket
    d. WindowSizeClass

6. A navigation rail is often appropriate for ___ width layouts.

    a. compact
    b. standard
    c. medium
    d. expanded

7. When building apps with adaptive layouts, you should use a single preview for each screen.

a. True
b. False

8. The list-detail layout requires Back navigation on compact screens, but not on screens where both the list and detail screens are shown at once.

   a. True
   b. False

9. Assume you have a contacts app that displays a list of contacts and has details to show for each contact. What are appropriate ways to adapt the UI to different screen sizes?

   a. Use the list-detail layout to show one pane or two panes side-by-side depending on the available width of the screen.
   b. The list items should take up the full width of the screen, regardless of how narrow or wide the screen is.
   c. The Up button should always be shown within the app and clicking the button should exit the app.
   d. When rotating the device, the selected item in the list (and the corresponding details of that item shown) should be reset to the first item in the list.
   e. It's required to use the Jetpack Navigation Component to make the UI responsive to different screen sizes.

10. Tests can be configured to run only test functions with custom annotations by configuring the ___.

    a. module
    b. package
    c. instrumentation class
    d. instrumentation arguments

ДҮГНЭЛТ

Энэ лабораторын хүрээнд Activity lifecycle, Log class, ViewModel, StateFlow, rememberSaveableболон Unidirectional Data Flow зэрэг ойлголтуудыг үзлээ. Эдгээр нь аппын төлвийг үр ашигтай хадгалах, UI-г зөв зохистой удирдах, хэрэглэгчийн туршлагыг сайжруулахад чухал ач холбогдолтой. Тухайлбал, ViewModel ашигласнаар аппын мэдээллийг удирдаж, UI болон өгөгдлийн логикийг салгах боломжоор хангадаг. Харин Log class нь аппын ажиллагааг шалгаж, алдааг хурдан илрүүлэхэд тусална. Эдгээрийг ашигласнаар илүү найдвартай, бүтээмж өндөртэй апп хөгжүүлэх боломжтой болно.