

MODUL PRAKTIKUM PEMROGRAMAN TEKNOLOGI BERGERAK

NAVIGATION DALAM JETPACK COMPOSE

A. Navigation

Navigation dalam Jetpack Compose memungkinkan kita berpindah antar layar (composable) dalam aplikasi, seperti dari halaman beranda ke halaman profil. Komponen Navigation menyediakan infrastruktur untuk mengelola rute, argumen, dan deep link.

Bayangkan aplikasi kita seperti mal besar dengan banyak toko. Navigation adalah peta yang membantu kita berpindah dari satu toko (composable) ke toko lain dengan jalur yang jelas. kita juga bisa menentukan jalur khusus (deep link) untuk langsung menuju toko tertentu dari luar mal.

Nah, dulunya, setiap layar di aplikasi sering kali dibuat sebagai Activity yang berbeda. Kalau mau pindah dari Activity A ke B, kita pakai Intent. Cara ini valid, tapi punya beberapa tantangan, terutama dalam mengelola data dan state yang kompleks antar Activity.

Sekarang, ada pendekatan yang lebih modern yang disebut **Single-Activity Architecture**. Bayangkan aplikasi kita itu seperti sebuah teater besar (satu Activity). Alih-alih membongkar seluruh panggung untuk setiap adegan (pindah Activity), kita cukup mengganti properti dan aktor di atas panggung yang sama. Nah, Jetpack Compose Navigation adalah 'sutradara' yang mengatur pergantian 'adegan' (layar Composable) itu.

Untuk menggunakan Navigation di Jetpack Compose, kita perlu menambahkan library dependencies pada file build.gradle(Module :app):

```
dependencies {
    // Jetpack Compose Navigation
    implementation("androidx.navigation:navigation-compose:2.7.7")
}
```

Navigasi pada Jetpack Compose ini dibangun oleh tiga komponen utama yang saling bekerja sama:

1. NavController

NavController adalah kelas utama yang berfungsi untuk mengelola state navigasi. Anggap ini sebagai pusat kendali. Tugas Utamanya:

- Mengetahui Posisi: selalu tahu kita sedang berada di layar mana.

- Memberi Perintah: yang kita suruh ketika ingin pindah layar.
- Mengelola Riwayat (Back Stack): mencatat semua layar yang sudah kita kunjungi. Jadi, saat kita menekan tombol "kembali", dia tahu harus kembali ke layar mana.

```
// Contoh perintah ke NavController
navController.navigate("detail") // Pergi ke layar detail
navController.popBackStack() // Kembali ke layar sebelumnya
```

2. NavHost

NavHost adalah sebuah composable function yang bertindak sebagai wadah untuk menampilkan layar yang aktif saat ini. Tugas utamanya:

- Menampilkan Tujuan: NavHost akan menampilkan composable yang sesuai dengan rute (route) yang sedang aktif di NavController.
- Mendefinisikan Grafik Navigasi: Di dalam NavHost, kita mendefinisikan semua kemungkinan tujuan navigasi aplikasi kita.

```
NavHost(navController = navController, startDestination = "home") {
    // Definisi semua layar kita ada di dalam sini
}
```

3. composable

Di dalam NavHost, kita menggunakan fungsi composable() untuk mendefinisikan setiap tujuan navigasi. Tugasnya:

- Memetakan Rute ke Layar: Fungsi ini menghubungkan sebuah alamat unik berbentuk String (disebut route) dengan sebuah Composable function yang akan ditampilkan.
- Membentuk Grafik: Kumpulan dari semua pemetaan composable ini disebut sebagai Grafik Navigasi (Navigation Graph).

```
NavHost(navController = navController, startDestination = "home") {
    // Entri peta pertama
    composable(route = "home") {
        HomeScreen(navController = navController) // Jika rute "home",
        tampilan ini
    }
}
```

```

// Entri peta kedua
composable(route = "detail") {
    DetailScreen(navController = navController) // Jika rute "detail",
tampilkan ini
}
}

```

Contoh sederhana penggunaan Navigation pada Jetpack Compose:

1. Buat dua screen, Home dan Detail

```

@Composable
fun HomeScreen(navController: NavController) {
    Column {
        Text("Ini adalah Layar Home")
        Button(onClick = {
            // Perintah untuk navigasi ke rute "detail"
            navController.navigate("detail")
        }) {
            Text("Pergi ke Detail")
        }
    }
}

```



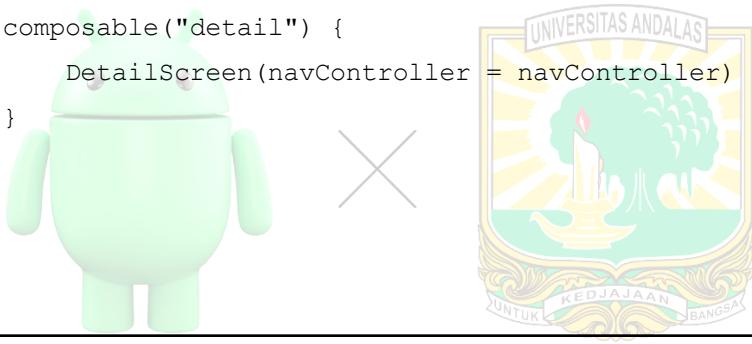
```

@Composable
fun DetailScreen(navController: NavController) {
    Column {
        Text("Ini adalah Layar Detail")
        Button(onClick = {
            // Perintah untuk kembali ke layar sebelumnya
            navController.popBackStack()
        }) {
            Text("Kembali")
        }
    }
}

```

2. Tambahkan Navigation Jetpack Compose pada [MainActivity.kt](#)

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            val navController = rememberNavController()  
  
            NavHost(  
                navController = navController,  
                startDestination = "home"  
            ) {  
                composable("home") {  
                    HomeScreen(navController = navController)  
                }  
                composable("detail") {  
                    DetailScreen(navController = navController)  
                }  
            }  
        }  
    }  
}
```



Kita juga bisa mengirim data ke screen tujuan. Misalnya, mengirim ID produk ke layar detail produk, atau nama pengguna ke layar profil. Ini dilakukan dengan menambahkan **argumen** pada route.

Contoh kode:

```
NavHost(navController = navController, startDestination = "home") {  
    composable(route = "home") {  
        HomeScreen(navController = navController)  
    }  
  
    // 1 & 2: Definisikan rute dengan argumen dan deklarasikan tipenya  
    composable(  
        route = "profile/{userId}", // Placeholder untuk argumen
```

```

        arguments      =      listOf(navArgument("userId")      { type      =
NavType.StringType })
    ) { backStackEntry ->
        // 4. Ambil argumen di sini
        val userId = backStackEntry.arguments?.getString("userId")
        ProfileScreen(navController = navController, userId = userId)
    }
}

```

Deep Link

Selain navigasi antar screen di dalam aplikasi, kita bisa membuat sebuah link yang mengarahkan siapapun yang mengklik link tersebut, maka akan diarahkan ke halaman yang sudah kita atur pada link tersebut di aplikasi kita. Namanya Deep Link. Deep Link adalah sebuah URL khusus yang, saat dibuka, akan langsung membawa pengguna ke layar spesifik di dalam aplikasi, bukan hanya membuka halaman utama. Ini sangat berguna untuk notifikasi, kampanye marketing, atau integrasi dengan website.

Langkah-langkahnya:

1. Tambahkan Definisi Deep Link: Di dalam fungsi composable yang ingin dituju, tambahkan parameter deepLinks. Parameter ini berisi daftar URL yang bisa membuka layar tersebut.
2. Konfigurasi AndroidManifest: Daftarkan URL tersebut di file AndroidManifest.xml menggunakan <intent-filter> agar sistem operasi Android tahu bahwa aplikasi bisa menangani URL tersebut.

Contoh kode penggunaannya:

```

// Di dalam NavHost
composable(
    route = "profile/{userId}",
    arguments = listOf(navArgument("userId") { type = NavType.StringType }),
    // 1. Definisikan pola URI untuk deep link
    deepLinks = listOf(navDeepLink { uriPattern = "https://example.com/profile/{userId}" })
) { backStackEntry ->
    val userId = backStackEntry.arguments?.getString("userId")
    ProfileScreen(navController = navController, userId = userId)
}

```

B. AppBar

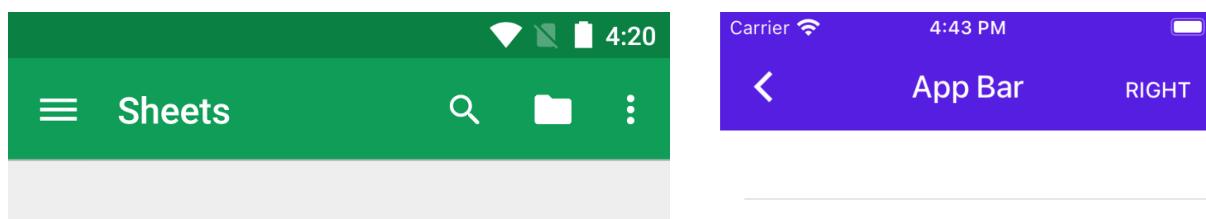
Sekarang kita akan membuat tampilan aplikasi menjadi lebih standar dan profesional dengan menambahkan bar di bagian atas layar, yang disebut TopAppBar. Cara termudah dan paling benar untuk melakukannya di Jetpack Compose adalah dengan menggunakan Scaffold.

Scaffold adalah sebuah composable yang menyediakan kerangka layout standar dari Material Design. Anggap ini adalah "cetakan" untuk sebuah layar aplikasi. Scaffold sudah menyediakan slot-slot khusus untuk elemen umum seperti:

- topBar: Untuk menempatkan TopAppBar.
 - bottomBar: Untuk BottomNavigationView.
 - floatingActionButton: Untuk tombol FAB.
 - drawerContent: Untuk Navigation Drawer.
- TopAppBar

TopAppBar adalah composable yang kita letakkan di dalam slot topBar pada Scaffold. Beberapa parameter penting di dalamnya adalah:

- title: Slot untuk menampilkan judul. Biasanya diisi dengan Text.
- navigationIcon: Slot untuk ikon di sisi kiri (awal), seperti tombol kembali (<-) atau tombol menu (≡). Biasanya diisi dengan IconButton.
- actions: Slot untuk ikon-ikon di sisi kanan (akhir), seperti tombol pencarian atau menu titik tiga.



C. Drawer

Navigation Drawer adalah panel UI yang muncul dari sisi kiri layar dan berisi tujuan navigasi utama dalam aplikasi. Ini adalah pola umum untuk aplikasi dengan beberapa tujuan tingkat atas (misalnya Home, Profile, Settings) dan memungkinkan pengguna beralih antar layar

dengan cepat. Di Jetpack Compose, kita menggunakan composable ModalNavigationDrawer untuk membuat fungsionalitas ini.

Untuk membuat Navigation Drawer yang fungsional, kita memerlukan beberapa komponen dan state:

- ModalNavigationDrawer: Composable utama yang menjadi wadah untuk konten drawer dan konten layar utama.
- rememberDrawerState: Sebuah fungsi untuk membuat dan mengingat state dari drawer, apakah sedang dalam keadaan terbuka (DrawerValue.Open) atau tertutup (DrawerValue.Closed).
- rememberCoroutineScope: Diperlukan untuk mendapatkan scope yang bisa menjalankan fungsi suspend seperti membuka (drawerState.open()) atau menutup (drawerState.close()) drawer secara programmatic.
- ModalDrawerSheet: Composable yang berfungsi sebagai "lembar" atau latar belakang dari menu yang ada di dalam drawer.
- NavigationDrawerItem: Komponen standar untuk setiap item menu yang dapat diklik di dalam ModalDrawerSheet.

Contoh kode penggunaan Drawer:

```
val navController = rememberNavController()
val navBackStackEntry by navController.currentBackStackEntryAsState()
val currentRoute = navBackStackEntry?.destination?.route

val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
val scope = rememberCoroutineScope()

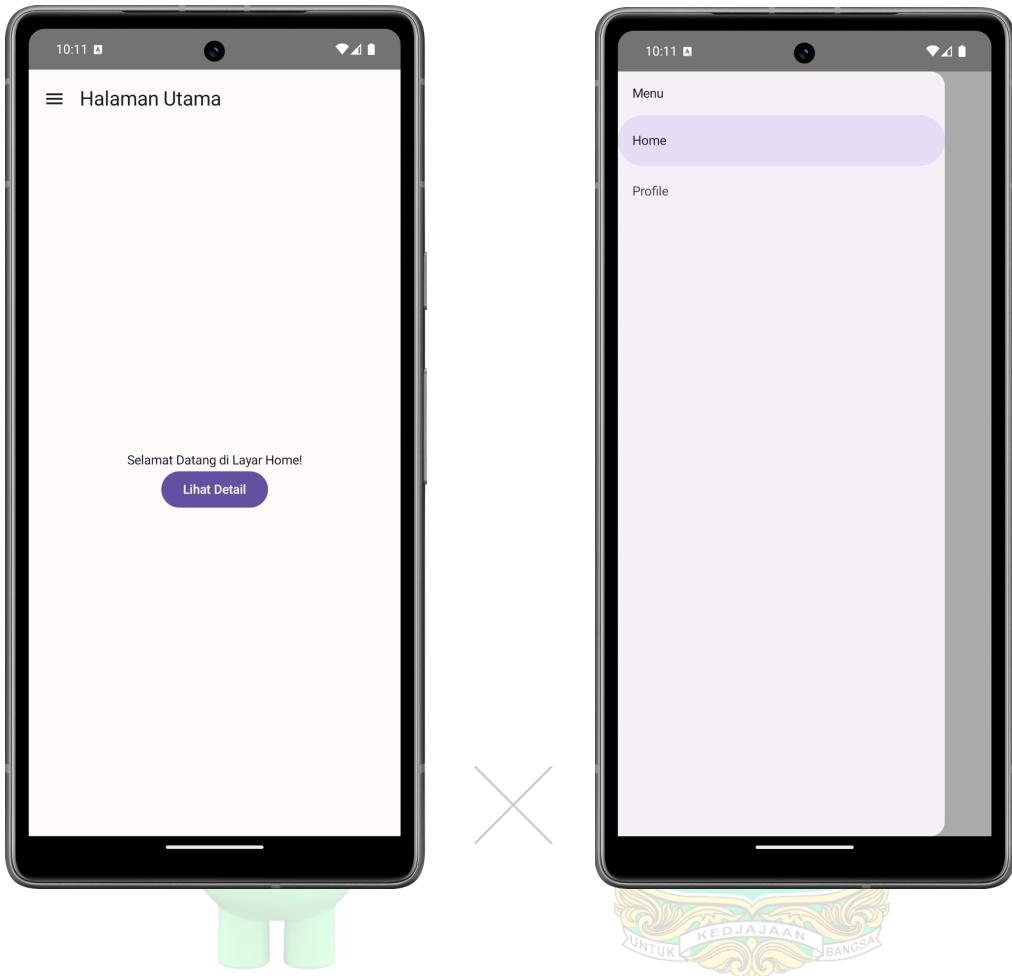
// Drawer → Scaffold → NavHost
ModalNavigationDrawer(
    drawerState = drawerState,
    drawerContent = {
        ModalDrawerSheet {
            Text("Menu", modifier = Modifier.padding(16.dp))

            NavigationDrawerItem(
                label = { Text("Home") },
                onClick = {
                    scope.launch {
                        navController.navigate("home")
                    }
                }
            )
        }
    }
)
```

```
        selected = currentRoute == Screen.Home.route,
        onClick = {
            scope.launch { drawerState.close() }
            navController.navigate(Screen.Home.route) {
                popUpTo(Screen.Home.route) { inclusive = true }
            }
        }
    )
    NavigationDrawerItem(
        label = { Text("Profile") },
        selected = currentRoute == Screen.Profile.route,
        onClick = {
            scope.launch { drawerState.close() }
            navController.navigate(Screen.Profile.route) {
                popUpTo(Screen.Home.route)
            }
        }
    )
}
```



Contoh tampilan aplikasi dengan Drawer + Scaffold + AppBar:



D. BottomBar (Bottom Navigation Bar)

Bottom Navigation Bar adalah komponen UI yang terletak di bagian bawah layar dan menampilkan 3 hingga 5 tujuan navigasi utama (level teratas) dalam sebuah aplikasi. Gunakan Bottom Navigation Bar ketika:

- Memiliki beberapa tujuan utama yang sama pentingnya.
- Pengguna perlu beralih di antara tujuan-tujuan tersebut dengan cepat dan sering.
- Setiap tujuan harus dapat diakses dari mana saja di dalam aplikasi.

Ini berbeda dari Navigation Drawer yang lebih cocok untuk banyak item navigasi atau item yang tidak perlu diakses sesering mungkin.

Komponen Utama:

- Scaffold: Tetap menjadi kerangka utama. NavigationBar akan kita letakkan di dalam slot bottomBar.

- NavigationBar: Composable yang berfungsi sebagai container untuk item-item navigasi di bagian bawah.
- NavigationBarItem: Composable untuk setiap ikon dan label yang dapat diklik di dalam NavigationBar. Parameter utamanya adalah selected, onClick, icon, dan label

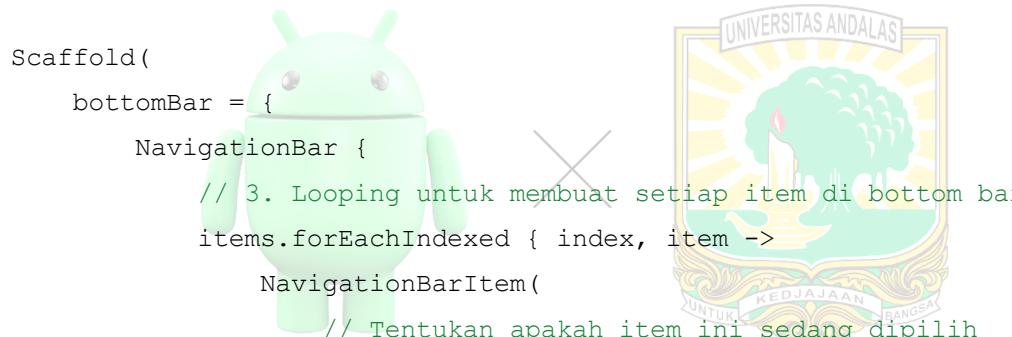
Contoh kode:

```
@Composable
fun SimpleBottomBarScreen() {
    // 1. Definisikan item-item untuk bottom bar
    val items = listOf("Home", "Search", "Profile")
    val icons = listOf(Icons.Default.Home, Icons.Default.Search,
Icons.Default.Person)

    // 2. Buat state untuk mengingat item mana yang sedang dipilih (indeksnya)
    var selectedIndex by remember { mutableStateOf(0) }

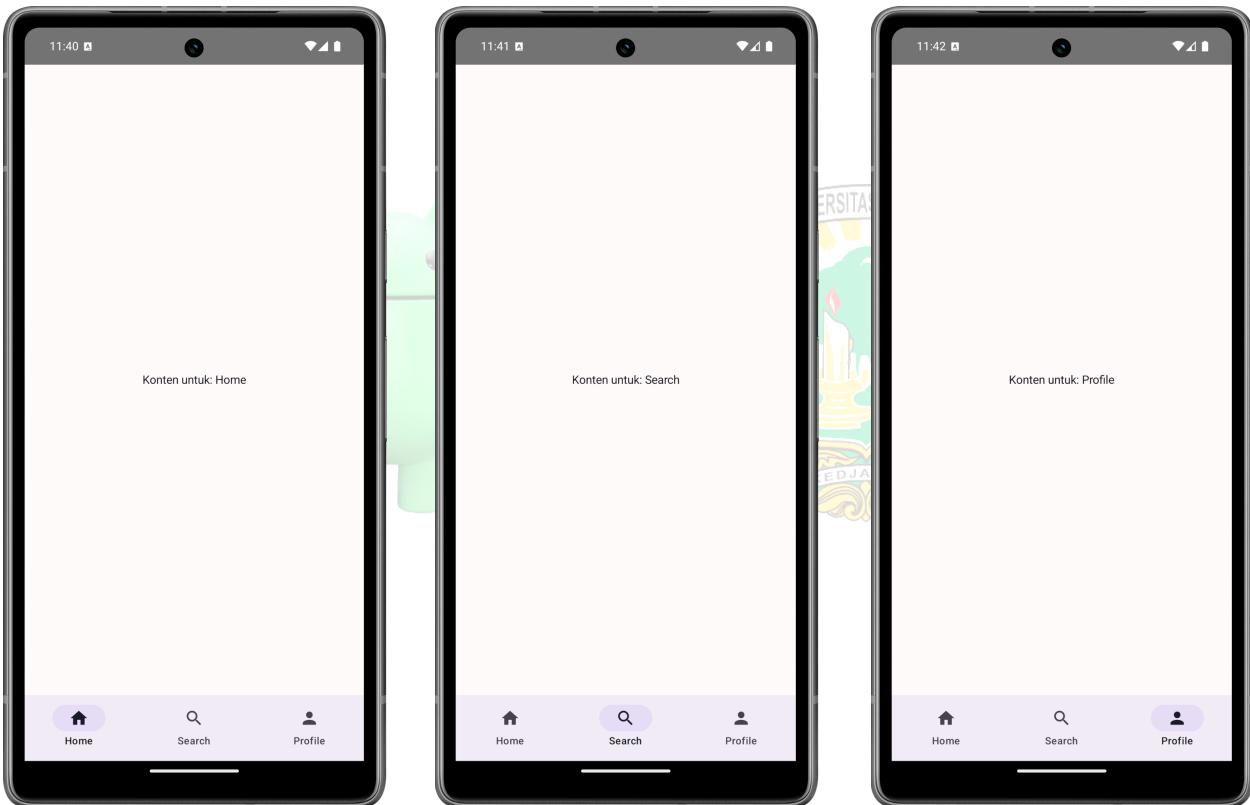
    Scaffold(
        bottomBar = {
            NavigationBar {
                // 3. Looping untuk membuat setiap item di bottom bar
                items.forEachIndexed { index, item ->
                    NavigationBarItem(
                        // Tentukan apakah item ini sedang dipilih
                        selected = selectedIndex == index,
                        // Aksi saat item diklik: ubah state `selectedIndex`
                        onClick = { selectedIndex = index },
                        // Label teks di bawah ikon
                        label = { Text(item) },
                        // Ikon untuk item
                        icon = { Icon(Icons[index], contentDescription = item) }
                    )
                }
            }
        } { innerPadding ->
            // Konten utama layar, ditampilkan berdasarkan item yang dipilih
            Box(

```

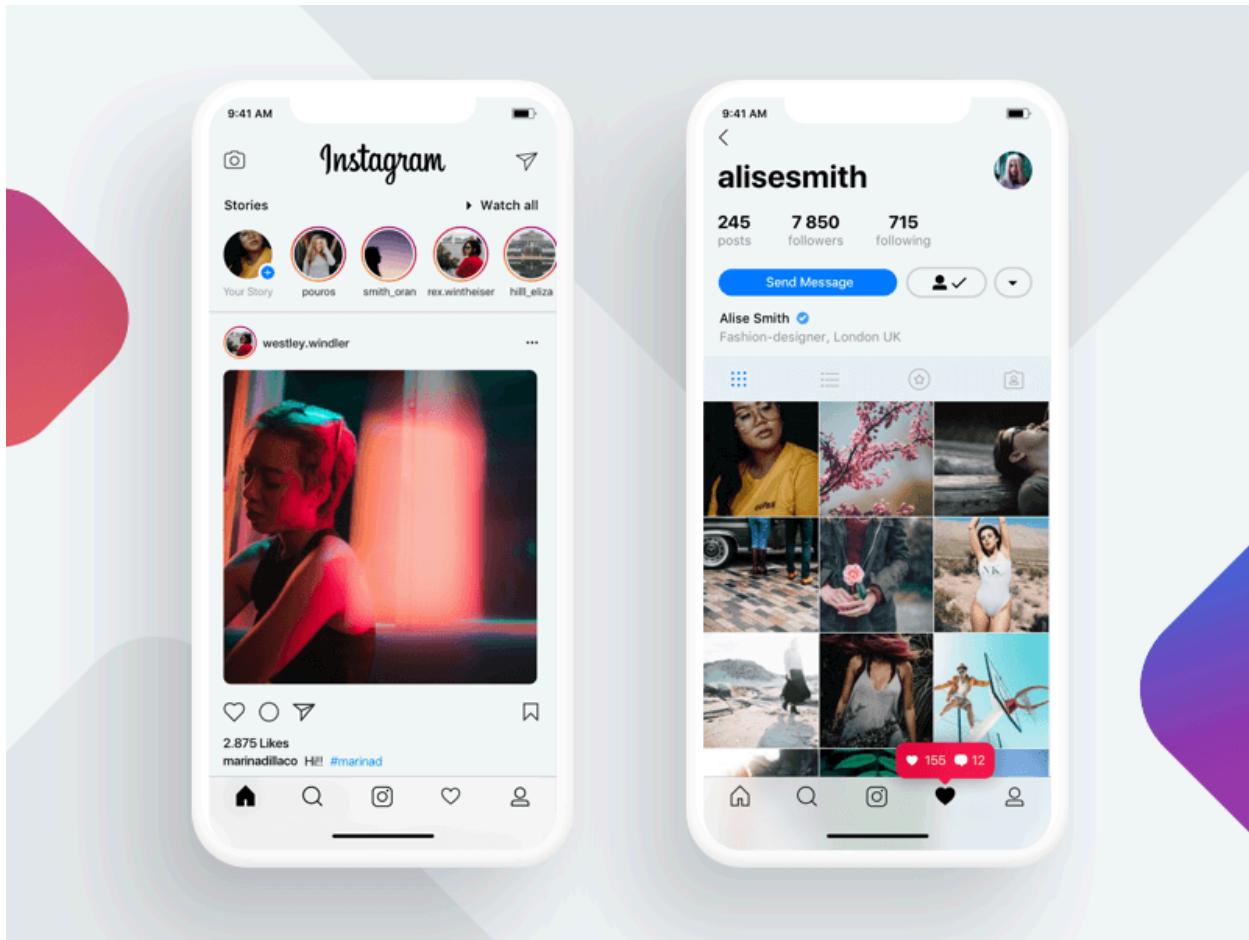


```
        modifier = Modifier
            .fillMaxSize()
            .padding(innerPadding), // Penting untuk menerapkan padding
        contentAlignment = Alignment.Center
    ) {
        Text(text = "Konten untuk: ${items[selectedIndex]}")
    }
}
```

Hasil kode:



Contoh tampilan aplikasi dengan BottomNavigation (myUNAND, Gojek, Instagram, dan banyak lagi):



E. Perbandingan Drawer dan Bottom Navigation Bar

Kriteria	Navigation Drawer	Bottom Navigation Bar
Posisi	Muncul dari sisi layar (biasanya kiri).	Tetap di bagian bawah layar.
Visibilitas	Tersembunyi secara default, perlu aksi (geser/klik) untuk membuka.	Selalu terlihat (persisten) di semua layar utama.
Kapasitas Item	Banyak (5+ item), bisa di-scroll.	Sedikit (ideal 3-5 item), tidak bisa di-scroll.
Kasus Penggunaan	Navigasi Sekunder: Untuk tujuan yang tidak sering diakses atau jumlahnya banyak.	Navigasi Primer: Untuk 3-5 tujuan paling penting dan sering digunakan.
Contoh Isi	Settings, About, Bantuan, Arsip, Sampah,	Home, Feed, Search, Pesan,

	Ganti Akun, Logout.	Profile.
Tujuan UX	Membersihkan UI utama dengan menyembunyikan navigasi yang kompleks.	Memberikan akses cepat dan ergonomis untuk beralih antar "mode" utama aplikasi.
Komponen M3	ModalNavigationDrawer	NavigationBar & NavigationBarItem

F. Navigasi dengan Animasi Transisi

Setelah aplikasi memiliki alur navigasi yang fungsional, langkah selanjutnya untuk meningkatkan pengalaman pengguna (UX) adalah dengan menambahkan animasi transisi. Animasi ini membuat perpindahan antar layar terasa lebih mulus, profesional, dan memberikan konteks visual kepada pengguna.

Kenapa Perlu Animasi Transisi?

- Memberi Konteks Spasial: Animasi seperti slide dari kanan ke kiri memberi kesan bahwa pengguna sedang "masuk lebih dalam" ke sebuah alur, dan sebaliknya.
- Feedback Visual: Memberi tahu pengguna bahwa aksi mereka (menekan tombol) telah berhasil memicu sebuah perubahan.
- Estetika Profesional: Aplikasi terasa lebih "hidup" dan dipoles dengan baik, meningkatkan persepsi kualitas secara keseluruhan.

Contoh kode animasi transisi:

```
val navController = rememberNavController()

// Gunakan AnimatedNavHost
NavHost(
    navController = navController,
    startDestination = Screen.Home.route
) {
    // Layar Home
    composable(
        route = Screen.Home.route,
        exitTransition = {
            // Animasi saat keluar dari Home menuju Detail
        }
    )
}
```

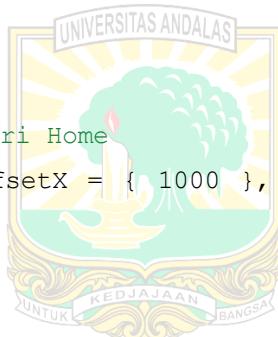
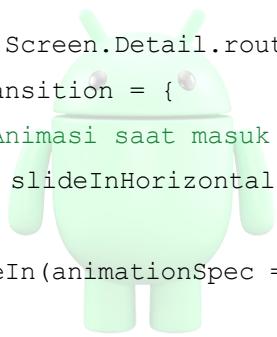
```

        slideOutHorizontally(targetOffsetX = { -1000 }, animationSpec =
tween(300)) +
            fadeOut(animationSpec = tween(300))
        },
        popEnterTransition = {
            // Animasi saat kembali ke Home dari Detail
            slideInHorizontally(initialOffsetX = { -1000 }, animationSpec =
tween(300)) +
                fadeIn(animationSpec = tween(300))
        }
    }

    HomeScreen(navController = navController)
}

// Layar Detail
composable(
    route = Screen.Detail.route,
    enterTransition = {
        // Animasi saat masuk ke Detail dari Home
        slideInHorizontally(initialOffsetX = { 1000 }, animationSpec =
tween(300)) +
            fadeIn(animationSpec = tween(300))
    },
    popExitTransition = {
        // Animasi saat kembali (keluar) dari Detail
        slideOutHorizontally(targetOffsetX = { 1000 }, animationSpec =
tween(300)) +
            fadeOut(animationSpec = tween(300))
    }
)
{
    DetailScreen(navController = navController)
}
}

```



Penjelasan Kode:

- exitTransition → animasi saat Home keluar (menuju Detail):
 - Geser ke kiri (slideOutHorizontally -1000)

- Sambil memudar (fadeOut)
- popEnterTransition → animasi saat kembali dari Detail ke Home:
 - Home masuk dari kiri (slideInHorizontally -1000)
 - Sambil muncul (fadeIn)
- enterTransition → animasi saat Detail masuk (dari Home):
 - Geser dari kanan (slideInHorizontally +1000)
 - Sambil muncul (fadeIn)
- popExitTransition → animasi saat Detail keluar (kembali ke Home):
 - Geser ke kanan (slideOutHorizontally +1000)
 - Sambil memudar (fadeOut)
- slideOutHorizontally(targetOffsetX = { -1000 })
 - Artinya komponen keluar dengan **geser horizontal**.
 - -1000 → arah **ke kiri** sejauh 1000 pixel.
 - Kalau +1000 → arah **ke kanan** sejauh 1000 pixel.
- slideInHorizontally(initialOffsetX = { -1000 })
 - Komponen **masuk** dari luar layar.
 - -1000 → masuk dari kiri.
 - +1000 → masuk dari kanan.
- fadeIn / fadeOut
 - **fadeIn** → perlahan tampil (opacity 0 → 1).
 - **fadeOut** → perlahan hilang (opacity 1 → 0).
- tween(300)
 - tween = interpolasi linear dengan durasi tertentu.
 - Angka 300 = **durasi animasi dalam milidetik (ms)**.
 - Jadi 300ms = 0.3 detik.
 - Kalau diganti tween(1000) → animasinya lebih lambat (1 detik).

Hasil bisa dilihat pada aplikasi yang ditampilkan Asisten.

G. Fungsi-Fungsi Lainnya

Flag	Fungsi Utama	Kenapa Penting?
popUpTo	Membersihkan back stack.	Mencegah tumpukan navigasi yang tak terbatas & error tombol kembali. (Wajib untuk fungsionalitas dasar)
saveState	Menyimpan state dari tab yang ditinggalkan.	Agar posisi scroll, teks input, dll. tidak hilang. (Penting untuk UX)
restoreState	Memulihkan state saat kembali ke tab.	Pasangan dari saveState untuk mengembalikan data yang disimpan. (Penting untuk UX)
launchSingleTop	Mencegah duplikasi layar di puncak stack.	Meningkatkan efisiensi dan mencegah UI berkedip. (Best Practice)

