Nama : M Rezky Raya Kilwouw

NIM : 222313190

Kelas : 3SI1

PRAKTIKUM 11

PEMROGRAMAN PLATFORM KHUSUS

TUGAS I

I. PENDAHULUAN

1.1 Latar Belakang

Pada era digital saat ini, sebagian besar aplikasi mobile memiliki lebih dari satu layar yang dapat dinavigasi oleh pengguna. Contohnya seperti aplikasi Settings yang memiliki banyak halaman konten yang tersebar di berbagai layar. Dalam pengembangan Android modern, aplikasi multi-layar dibuat menggunakan komponen Jetpack Navigation yang memungkinkan perpindahan antar layar dengan pendekatan deklaratif, sama seperti membangun user interface di Jetpack Compose.

1.2 Tujuan Praktikum

Praktikum ini bertujuan untuk:

1. Memahami cara membuat NavHost composable untuk mendefinisikan routes dan screens dalam aplikasi
2. Melakukan navigasi antar layar menggunakan NavHostController
3. Memanipulasi back stack untuk navigasi ke layar sebelumnya
4. Menggunakan intents untuk berbagi data dengan aplikasi lain
5. Mengkustomisasi AppBar, termasuk judul dan tombol back

1.3 Alat dan Bahan

- Android Studio (versi terbaru)
- Koneksi internet untuk mengunduh starter code
- Emulator Android atau perangkat fisik untuk testing
- Project starter Cupcake App dari GitHub

II. DASAR TEORI

2.1 Komponen Navigation

Navigation component memiliki tiga bagian utama:

NavController: Bertanggung jawab untuk navigasi antar destination (layar) dalam aplikasi. NavController melacak back stack dari composable entries, memindahkan stack ke depan, memungkinkan manipulasi back stack, dan navigasi antar state destination.

NavGraph: Memetakan composable destinations untuk navigasi. NavGraph mendefinisikan semua routes yang tersedia dalam aplikasi dan hubungannya satu sama lain.

NavHost: Composable yang bertindak sebagai container untuk menampilkan destination saat ini dari NavGraph. NavHost menampilkan composable yang berbeda berdasarkan route yang aktif.

2.2 Konsep Route

Route adalah string yang berkorespondensi dengan destination. Konsep ini mirip dengan URL pada website - setiap URL yang berbeda mengarah ke halaman yang berbeda, begitu juga route yang merupakan string yang memetakan ke destination dan berfungsi sebagai identifier unik. Destination biasanya adalah single composable atau grup composables yang berkorespondensi dengan apa yang dilihat pengguna.

2.3 Intent

Intent adalah permintaan untuk sistem melakukan suatu aksi, umumnya untuk menampilkan activity baru. Ada berbagai jenis intent, dan dalam praktikum ini menggunakan ACTION_SEND untuk berbagi data seperti teks dengan aplikasi lain melalui ShareSheet yang disediakan oleh sistem Android.

III. PELAKSANAAN PRAKTIKUM

3.1 Persiapan Project

Tahap pertama adalah mengunduh starter code dari GitHub:

git clone https://github.com/google-developer-training/basic-android-kotlin-compose-training-cupcake.git

cd basic-android-kotlin-compose-training-cupcake

git checkout starter

Project Cupcake App ini berbeda dari aplikasi sebelumnya karena memiliki empat layar terpisah yang dapat dinavigasi pengguna saat memesan cupcake.

3.2 Struktur Aplikasi Cupcake

Aplikasi ini terdiri dari empat layar utama:

1. **Start Order Screen**: Menampilkan tiga tombol untuk memilih jumlah cupcake yang ingin dipesan (1, 6, atau 12 cupcake)
2. **Choose Flavor Screen**: Menampilkan pilihan rasa cupcake menggunakan radio buttons

3. **Choose Pickup Date Screen**: Menampilkan pilihan tanggal pengambilan menggunakan radio buttons
4. **Order Summary Screen**: Menampilkan ringkasan pesanan dengan opsi untuk mengirim ke aplikasi lain atau membatalkan pesanan

3.3 Mendefinisikan Routes dengan Enum Class

Langkah pertama dalam implementasi navigasi adalah mendefinisikan routes menggunakan enum class. Di file CupcakeScreen.kt, saya membuat enum class:

```
enum class CupcakeScreen(@StringRes val title: Int) {
    Start(title = R.string.app_name),
    Flavor(title = R.string.choose_flavor),
    Pickup(title = R.string.choose_pickup_date),
    Summary(title = R.string.order_summary)
}
```

Enum class ini mendefinisikan empat route yang mewakili empat layar dalam aplikasi. Setiap case memiliki parameter title yang merupakan resource ID untuk string judul layar.

3.4 Membuat NavHostController

Untuk navigasi, diperlukan instance NavHostController yang dapat diperoleh dengan memanggil rememberNavController():

```
@Composable
fun CupcakeApp(
    viewModel: OrderViewModel = viewModel(),
    navController: NavHostController = rememberNavController()
){
    // Implementation
}
```

NavHostController ini akan digunakan untuk mengontrol navigasi antar layar dan melacak back stack.

3.5 Implementasi NavHost

NavHost adalah composable yang menampilkan destination berdasarkan route yang diberikan. Di dalam Scaffold, saya menambahkan NavHost:

```
NavHost(
    navController = navController,
    startDestination = CupcakeScreen.Start.name,
```

```
    modifier = Modifier.padding(innerPadding)
) {
    // Destinations will be defined here
}
```

Parameter penting:

- navController: Instance NavHostController untuk kontrol navigasi
- startDestination: Route yang ditampilkan pertama kali (Start screen)
- modifier: Modifier untuk styling

3.6 Mendefinisikan Destinations

Di dalam NavHost, saya mendefinisikan setiap destination menggunakan fungsi composable():

Start Order Screen:

```
composable(route = CupcakeScreen.Start.name) {
    StartOrderScreen(
        quantityOptions = DataSource.quantityOptions,
        onNextButtonClicked = {
            viewModel.setQuantity(it)
            navController.navigate(CupcakeScreen.Flavor.name)
        },
        modifier = Modifier
            .fillMaxSize()
            .padding(dimensionResource(R.dimen.padding_medium))
    )
}
```

Choose Flavor Screen:

```
composable(route = CupcakeScreen.Flavor.name) {
    val context = LocalContext.current
    SelectOptionScreen(
        subtotal = uiState.price,
        onNextButtonClicked = { navController.navigate(CupcakeScreen.Pickup.name) },
        onCancelButtonClicked = {
```

```
            cancelOrderAndNavigateToStart(viewModel, navController)
        },
        options = DataSource.flavors.map { id -> context.resources.getString(id) },
        onSelectionChanged = { viewModel.setFlavor(it) },
        modifier = Modifier.fillMaxHeight()
    )
}
```

Choose Pickup Date Screen:

```
composable(route = CupcakeScreen.Pickup.name) {
    SelectOptionScreen(
        subtotal = uiState.price,
        onNextButtonClicked = { navController.navigate(CupcakeScreen.Summary.name) },
        onCancelButtonClicked = {
            cancelOrderAndNavigateToStart(viewModel, navController)
        },
        options = uiState.pickupOptions,
        onSelectionChanged = { viewModel.setDate(it) },
        modifier = Modifier.fillMaxHeight()
    )
}
```

Order Summary Screen:

```
composable(route = CupcakeScreen.Summary.name) {
    val context = LocalContext.current
    OrderSummaryScreen(
        orderUiState = uiState,
        onCancelButtonClicked = {
            cancelOrderAndNavigateToStart(viewModel, navController)
        },
        onSendButtonClicked = { subject: String, summary: String ->
            shareOrder(context, subject = subject, summary = summary)
```

```
        },
        modifier = Modifier.fillMaxHeight()
    )
}
```

3.7 Implementasi Fungsi Navigasi

Navigate ke Route Lain: Untuk navigasi ke layar lain, cukup memanggil method navigate() pada NavHostController:

```
navController.navigate(CupcakeScreen.Flavor.name)
```

Pop Back Stack: Untuk kembali ke layar awal dan menghapus semua layar dari back stack, digunakan fungsi cancelOrderAndNavigateToStart():

```
private fun cancelOrderAndNavigateToStart(
    viewModel: OrderViewModel,
    navController: NavHostController
) {
    viewModel.resetOrder()
    navController.popBackStack(CupcakeScreen.Start.name, inclusive = false)
}
```

Method popBackStack() menerima dua parameter:

- route: String route destination tujuan
- inclusive: Boolean yang menentukan apakah route yang ditentukan juga dihapus

3.8 Implementasi Intent untuk Sharing

Untuk mengirim data pesanan ke aplikasi lain, digunakan Intent dengan ACTION_SEND:

```
private fun shareOrder(context: Context, subject: String, summary: String) {
    val intent = Intent(Intent.ACTION_SEND).apply {
        type = "text/plain"
        putExtra(Intent.EXTRA_SUBJECT, subject)
        putExtra(Intent.EXTRA_TEXT, summary)
    }

    context.startActivity(
        Intent.createChooser(
```

```
            intent,
            context.getString(R.string.new_cupcake_order)
        )
    )
}
```

Proses pembuatan intent:

1. Membuat intent object dengan ACTION_SEND
2. Set type data menjadi "text/plain"
3. Menambahkan extra data (EXTRA_SUBJECT dan EXTRA_TEXT)
4. Memanggil startActivity() dengan createChooser() untuk menampilkan ShareSheet

3.9 Kustomisasi AppBar

Untuk membuat AppBar responsif terhadap navigasi, diperlukan beberapa modifikasi:

Mendapatkan Current Screen:

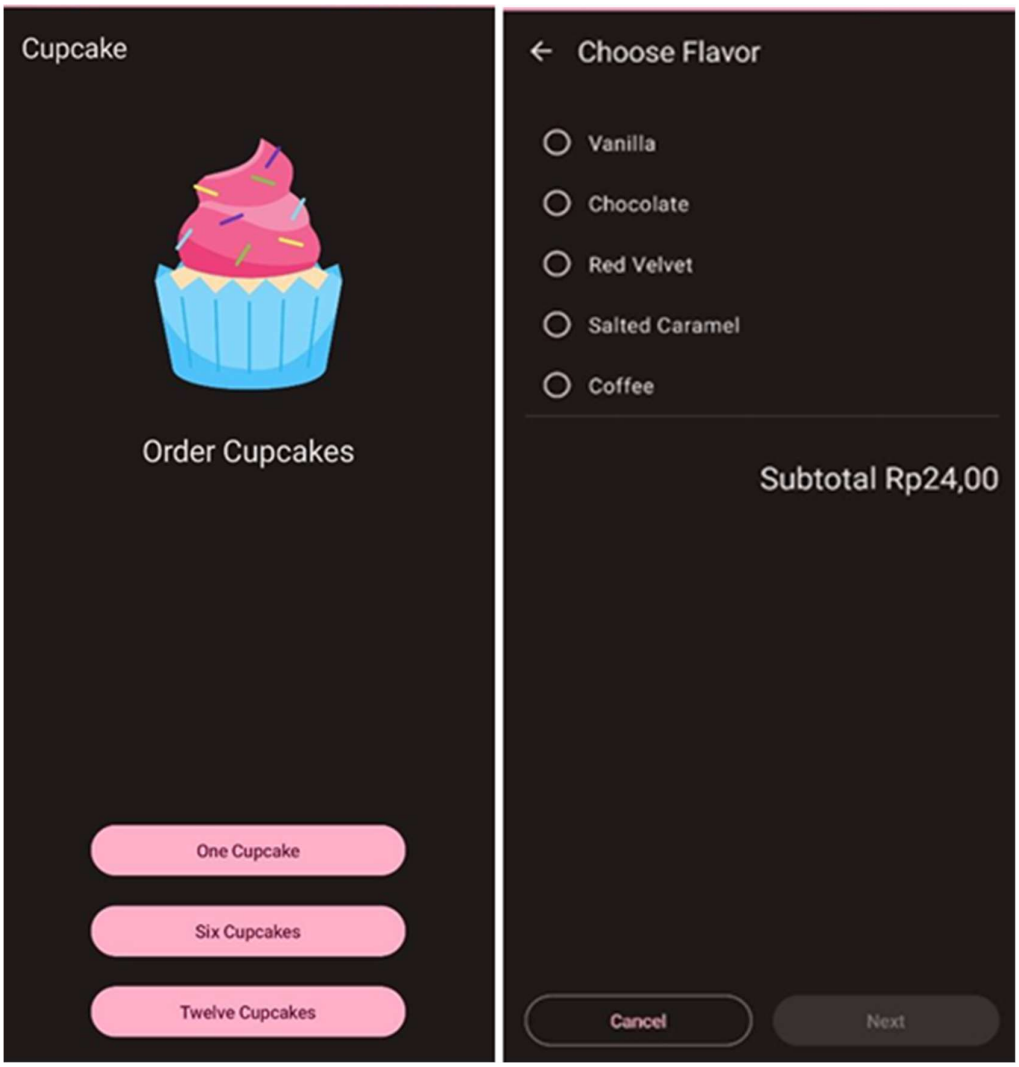
```
val backStackEntry by navController.currentBackStackEntryAsState()
val currentScreen = CupcakeScreen.valueOf(
    backStackEntry?.destination?.route ?: CupcakeScreen.Start.name
)
```

Update CupcakeAppBar:

```
CupcakeAppBar(
    currentScreen = currentScreen,
    canNavigateBack = navController.previousBackStackEntry != null,
    navigateUp = { navController.navigateUp() }
)
```

AppBar akan menampilkan:

- Judul sesuai dengan layar yang aktif
- Up button hanya muncul jika ada layar sebelumnya di back stack
- Up button memanggil navigateUp() untuk kembali ke layar sebelumnya



TUGAS II

Modifikasi Explicit Intent

Pada tugas ini, dilakukan modifikasi terhadap project awal "Cupcake". Struktur navigasi asli yang menggunakan *Single Activity Architecture* (Jetpack Navigation) diubah menjadi arsitektur *Multi-Activity*.

Aplikasi kini terdiri dari dua *Activity* utama:

1. *Sending Activity* (MainActivity): Berfungsi sebagai formulir input data (Nama, Nama Belakang, Umur).
2. *Receiving Activity* (DetailActivity): Berfungsi untuk menerima dan menampilkan data yang dikirimkan.

Tujuan utama dari modifikasi ini adalah mendemonstrasikan penggunaan **Explicit Intent** untuk memindahkan data antar layar.

Konsep Explicit Intent

Explicit Intent adalah jenis Intent yang digunakan untuk menjalankan komponen aplikasi (seperti Activity) yang sudah diketahui secara pasti nama kelasnya. Dalam tugas ini, Explicit Intent digunakan karena kita ingin menavigasi secara spesifik dari MainActivity menuju DetailActivity yang berada dalam satu paket aplikasi yang sama.

Implementasi

Mengirim Data (Sending Activity)

File: MainActivity.kt

Pada bagian ini, ketika tombol "SAVE AND SHOW REPORT" ditekan, objek Intent dibuat dengan menargetkan DetailActivity. Data disisipkan ke dalam Intent menggunakan metode putExtra dengan format Key-Value.

```
Button(
    onClick = {
        val intent = Intent( packageContext = context,  cls = DetailActivity::class.java).apply {
            putExtra( name = "NAME",  value = name)
            putExtra( name = "SURNAME",  value = surname)
            putExtra( name = "AGE",  value = age)
        }
        context.startActivity( p0 = intent)
    },
    modifier = Modifier.fillMaxWidth()
) {
    Text( text = "SAVE AND SHOW REPORT")
}
```

Menerima Data (Receiving Activity)

File: DetailActivity.kt

Pada Activity penerima, data diambil kembali di dalam method onCreate. Kita mengakses properti intent yang memicu Activity ini, lalu menggunakan getStringExtra sesuai dengan "KUNCI" yang dikirimkan sebelumnya.

```
class DetailActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val name = intent.getStringExtra( name = "NAME") ?: "N/A"
        val surname = intent.getStringExtra( name = "SURNAME") ?: "N/A"
        val age = intent.getStringExtra( name = "AGE") ?: "N/A"

        setContent {
            CupcakeTheme {
                ReceivingScreen(name = name, surname = surname, age = age)
            }
        }
    }
}
```

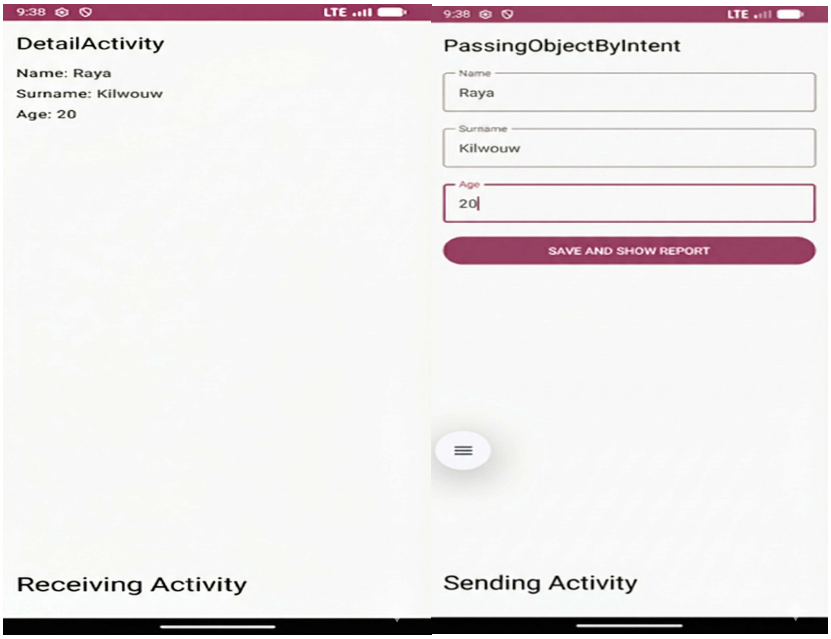
Registrasi Activity (Manifest)

File: AndroidManifest.xml

Setiap Activity baru yang dibuat harus didaftarkan ke dalam manifest agar dikenali oleh sistem operasi Android.

```
<activity
    android:name=".MainActivity"
    android:exported="true"
    android:theme="@style/Theme.Cupcake">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".DetailActivity"
    android:exported="false"
    android:theme="@style/Theme.Cupcake" />
```

Kesimpulan



Modifikasi project berhasil dilakukan dengan mengubah mekanisme navigasi menjadi berbasis Intent. Penggunaan Explicit Intent terbukti efektif untuk mengirimkan data sederhana (String) antar Activity secara langsung, dimana Activity pengirim (MainActivity) mendefinisikan Activity tujuan (DetailActivity) secara jelas pada saat inisialisasi Intent.