LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

Connect to the Internet

Oleh:

Ghani Mudzakir **NIM. 2310817110011**

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE I
MODUL 5

Laporan Praktikum Pemrograman Mobile Modul 5: Connect to the Internet ini disusun sebagai syarat lulus mata kuliah Praktikum Pemrograman Mobile. Laporan Prakitkum ini dikerjakan oleh:

Nama Praktikan : Ghani Mudzakir
NIM : 2310817110011

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Zulfa Auliya Akbar NIM. 2210817210026	Muti`a Maulida S.Kom M.T.I NIP. 19881027 201903 20 13

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL 1

Soal Praktikum:

1. Lanjutkan aplikasi Android yang sudah dibuat pada Modul 4 dengan menambahkan modifikasi sesuai ketentuan berikut:

a. Gunakan networking library seperti Retrofit atau Ktor agar aplikasi dapat mengambil data dari remote API. Dalam penggunaan networking library, sertakan generic response
b. Gunakan KotlinX Serialization sebagai library JSON.
c. Gunakan library seperti Coil atau Glide untuk image loading.
d. API yang digunakan pada modul ini bebas, contoh API gratis The Movie Database (TMDB) API yang menampilkan data film. Berikut link dokumentasi API: https://developer.themoviedb.org/docs/getting-started
e. Implementasikan konsep data persistence (misalnya offline-first app, pengaturan dark/light mode, fitur favorite, dll)
f. Gunakan caching strategy pada Room..
g. Untuk Modul 5, bebas memilih UI yang ingin digunakan, antara berbasis XML atau Jetpack Compose.

Aplikasi harus mempertahankan fitur-fitur yang dibuat pada modul sebelumnya.

A. Source Code

1. model/Product.kt

Tabel 1 Source Code model/Product.kt

```
1  package com.example.modul_5.data.model
2
3  import kotlinx.serialization.Serializable
4
5  @Serializable
6  data class Product(
7      val id: Int,
8      val title: String,
9      val price: Double,
10     val description: String,
11     val category: String,
       val image: String,
       val rating: Rating
   )
```

```
@Serializable
data class Rating(
    val rate: Double,
    val count: Int
)
```

2. remote/ProductApiService.kt

Tabel 2 Source Code remote/ProductApiService.kt

```
1  package com.example.modul_5.data.remote
2
3  import com.example.modul_5.data.model.Product
4  import retrofit2.http.GET
5
6  interface ProductApiService {
7      @GET("products")
8      suspend fun getProducts(): List<Product>
9  }
```

3. remote/RetrofitInstance.kt

Tabel 3 Source Code remote/RetrofitInstance.kt

```
1  package com.example.modul_5.data.remote
2
3  import
4  com.jakewharton.retrofit2.converter.kotlinx.serializatio
5  n.asConverterFactory
6  import
7  kotlinx.serialization.ExperimentalSerializationApi
8  import kotlinx.serialization.json.Json
9  import okhttp3.MediaType.Companion.toMediaType
10 import retrofit2.Retrofit
11
   object RetrofitInstance {
       private val json = Json {
           ignoreUnknownKeys = true
       }

       @OptIn(ExperimentalSerializationApi::class)
       val api: ProductApiService by lazy {
           Retrofit.Builder()
               .baseUrl("https://fakestoreapi.com/")

   .addConverterFactory(json.asConverterFactory("applicatio
   n/json".toMediaType()))
```

```
            .build()
            .create(ProductApiService::class.java)
    }
}
```

4. repository/ProductRepository.kt

Tabel 4 Source Code repository/ProductRepository.kt

```
package com.example.modul_5.repository

import android.util.Log
import com.example.modul_5.data.model.Product
import com.example.modul_5.data.remote.RetrofitInstance
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow

class ProductRepository {
    fun getProducts(): Flow<Result<List<Product>>> =
flow {
        try {
            val products =
RetrofitInstance.api.getProducts()
            Log.d("ProductRepository", "Success fetching
products: ${products.size} items")
            emit(Result.success(products))
        } catch (e: Exception) {
            Log.e("ProductRepository", "Error fetching
products", e)
            emit(Result.failure(e))
        }
    }
}
```

5. ui/detail/DetailScreen.kt

Tabel 5 Source Code ui/detail/DetailScreen.kt

```
package com.example.modul_5.ui.detail

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.example.modul_5.data.model.Product
import com.example.modul_5.ui.home.HomeViewModel
```

11

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DetailScreen(productId: Int, viewModel:
HomeViewModel) {
    val products by viewModel.products.collectAsState()
    val isLoading by
viewModel.isLoading.collectAsState()
    val error by viewModel.error.collectAsState()

    val product = products.find { it.id == productId }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Detail Produk") }
            )
        }
    ) { paddingValues ->
        Box(
            modifier = Modifier
                .padding(paddingValues)
                .fillMaxSize(),
            contentAlignment = Alignment.Center
        ) {
            when {
                isLoading -> {
                    CircularProgressIndicator()
                }
                error != null -> {
                    Text(
                        text = error ?: "Terjadi
kesalahan",
                        color =
MaterialTheme.colorScheme.error
                    )
                }
                product != null -> {
                    ProductDetailContent(product =
product)
                }
                else -> {
                    Text("Produk tidak ditemukan.")
                }
            }
        }
    }
}
```

```
@Composable
fun ProductDetailContent(product: Product) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Text("Nama: ${product.title}", style =
MaterialTheme.typography.titleMedium)
        Text("Harga: $${product.price}", style =
MaterialTheme.typography.bodyMedium)
        Text("Kategori: ${product.category}", style =
MaterialTheme.typography.bodyMedium)
        Text("Deskripsi: ${product.description}", style
= MaterialTheme.typography.bodySmall)
    }
}
```

6. ui/home/HomeScreen.kt

Tabel 6 Source Code ui/home/HomeScreen.kt

```
1  @file:OptIn(ExperimentalMaterial3Api::class)
2
3  package com.example.modul_5.ui.home
4  import androidx.compose.foundation.layout.*
5  import androidx.compose.foundation.lazy.LazyColumn
6  import androidx.compose.foundation.lazy.items
7  import androidx.compose.material3.*
8  import androidx.compose.runtime.*
9  import androidx.compose.ui.Alignment
10 import androidx.compose.ui.Modifier
11 import androidx.compose.ui.text.style.TextOverflow
12 import androidx.compose.ui.unit.dp
13 import androidx.navigation.NavController
14 import androidx.navigation.NavType
15 import androidx.navigation.compose.*
16 import androidx.navigation.navArgument
17 import coil.compose.AsyncImage
18 import com.example.modul_5.data.model.Product
19 import android.content.Intent
20 import android.net.Uri
21 import androidx.compose.foundation.background
22 import androidx.compose.ui.graphics.Color
23 import androidx.compose.ui.platform.LocalContext
```

```
24
25
26  @Composable
27  fun HomeNavHost(
28      viewModel: HomeViewModel,
29      modifier: Modifier = Modifier
30  ) {
31      val navController = rememberNavController()
32
33      NavHost(
34          navController = navController,
35          startDestination = "home",
36          modifier = modifier
        ) {
            composable("home") {
                HomeScreen(viewModel = viewModel,
navController = navController)
            }
            composable(
                route = "detail/{productId}",
                arguments = listOf(navArgument("productId") {
type = NavType.IntType })
            ) { backStackEntry ->
                val productId =
backStackEntry.arguments?.getInt("productId") ?: 0
                DetailScreen(productId = productId, viewModel
= viewModel)
            }
        }
    }



    @Composable
    fun HomeScreen(
        viewModel: HomeViewModel,
        navController: NavController,
    ) {

        val context = LocalContext.current

        val products by viewModel.products.collectAsState()
        val isLoading by viewModel.isLoading.collectAsState()
        val error by viewModel.error.collectAsState()

        Scaffold(
            topBar = {
                TopAppBar(
                    title = { Text("Product List") }
```

```
            )
        }
    ) { paddingValues ->
        Box(
            modifier = Modifier
                .padding(paddingValues)
                .fillMaxSize()
        ) {
            when {
                isLoading -> {
                    CircularProgressIndicator(modifier =
Modifier.align(Alignment.Center))
                }
                error != null -> {
                    Text(
                        text = error ?: "Unknown error",
                        modifier =
Modifier.align(Alignment.Center),
                        color =
MaterialTheme.colorScheme.error
                    )
                }
                else -> {
                    LazyColumn(
                        modifier =
Modifier.fillMaxSize(),
                        contentPadding =
PaddingValues(16.dp),
                        verticalArrangement =
Arrangement.spacedBy(12.dp)
                    ) {
                        items(products) { product ->
                            ProductCard(product =
product, onDetailClick = {

navController.navigate("detail/${product.id}")
                            }, onWebsiteClick = {
                                val searchQuery =
Uri.encode(product.title)
                                val url =
"https://shopee.co.id/search?keyword=$searchQuery"
                                val intent =
Intent(Intent.ACTION_VIEW, Uri.parse(url))

context.startActivity(intent)
                            })
                        }
                    }
                }
```

```
            }
        }
    }
}

@Composable
fun ProductCard(
    product: Product,
    onDetailClick: () -> Unit,
    onWebsiteClick: () -> Unit
) {
    val context = LocalContext.current

    Card(
        modifier = Modifier.fillMaxWidth(),
        elevation =
CardDefaults.cardElevation(defaultElevation = 6.dp),
        colors = CardDefaults.cardColors(containerColor =
Color.White)
    ) {
        Column(modifier = Modifier.padding(12.dp)) {
            AsyncImage(
                model = product.image,
                contentDescription = product.title,
                modifier = Modifier
                    .fillMaxWidth()
                    .height(180.dp)
            )
            Spacer(modifier = Modifier.height(8.dp))
            Text(
                text = product.title,
                style =
MaterialTheme.typography.titleMedium,
                maxLines = 2,
                overflow = TextOverflow.Ellipsis,
                color = Color.Black
            )
            Spacer(modifier = Modifier.height(8.dp))
            Text(text = "$${product.price}", style =
MaterialTheme.typography.bodyMedium, color = Color.Black)
            Spacer(modifier = Modifier.height(8.dp))
            Button(onClick = onDetailClick) {
                Text("Detail")
            }
            Button(onClick = {
                val searchQuery =
Uri.encode(product.title)
                val url =
"https://shopee.co.id/search?keyword=$searchQuery"
```

```
                    val intent = Intent(Intent.ACTION_VIEW,
Uri.parse(url))
                    context.startActivity(intent)
                }) {
                    Text("Buy Now on Website")
                }
            }
        }
    }

    @Composable
    fun DetailScreen(productId: Int, viewModel:
    HomeViewModel) {
        val product =
    viewModel.products.collectAsState().value.find { it.id ==
    productId }

        val isLoading by viewModel.isLoading.collectAsState()
        val error by viewModel.error.collectAsState()

        Scaffold(
            topBar = {
                TopAppBar(
                    title = { Text("Product Detail") }
                )
            }
        ) { padding ->
            Box(
                modifier = Modifier
                    .padding(padding)
                    .fillMaxSize()
                    .background(Color.White),
                contentAlignment = Alignment.Center
            ) {
                when {
                    isLoading -> {
                        CircularProgressIndicator()
                    }
                    error != null -> {
                        Text(
                            text = error ?: "Unknown error",
                            color =
    MaterialTheme.colorScheme.error
                        )
                    }
                    product != null -> {
                        Column(
                            modifier =
    Modifier.padding(16.dp)
```

```
                                .fillMaxSize()
                                .background(Color.White),
                            verticalArrangement =
Arrangement.spacedBy(8.dp)
                        ) {
                            AsyncImage(
                                model = product.image,
                                contentDescription =
product.title,
                                modifier = Modifier
                                    .fillMaxWidth()
                                    .height(180.dp)
                            )
                            Text("Name: ${product.title}",
style = MaterialTheme.typography.titleMedium, color =
Color.Black)
                            Text("Price: $${product.price}",
color = Color.Black)
                            Text("Category:
${product.category}", color = Color.Black)
                            Text("Description:
${product.description}", color = Color.Black)
                            Text("Rating: ${product.rating}",
color = Color.Black   )
                        }
                    }
                    else -> {
                        Text("No product data", color =
Color.Black)
                    }
                }
            }
        }
    }
}
```

7. ui/home/HomeViewModel.kt

Tabel 7 Source Code ui/home/HomeViewModel.kt

```
1  package com.example.modul_5
2
3  import android.os.Bundle
4  import androidx.activity.ComponentActivity
5  import androidx.activity.compose.setContent
6  import androidx.activity.enableEdgeToEdge
7  import androidx.compose.foundation.layout.fillMaxSize
8  import androidx.compose.foundation.layout.padding
9  import androidx.compose.material3.Scaffold
10 import androidx.compose.ui.Modifier
```

```
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.modul_5.ui.home.HomeNavHost
import com.example.modul_5.ui.theme.Modul_5Theme
import com.example.modul_5.ui.home.HomeViewModel

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Modul_5Theme {
                Scaffold(modifier =
Modifier.fillMaxSize()) { innerPadding ->

                    val homeViewModel: HomeViewModel =
viewModel()

                    HomeNavHost(
                        viewModel = homeViewModel,
                        modifier =
Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}
```

8. ui/theme/Theme.kt

Tabel 8 Source Code ui/theme/Theme.kt

```
package com.example.modul_5.ui.theme

import android.app.Activity
import android.os.Build
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.dynamicDarkColorScheme
import androidx.compose.material3.dynamicLightColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.platform.LocalContext

private val DarkColorScheme = darkColorScheme(
    primary = Purple80,
    secondary = PurpleGrey80,
    tertiary = Pink80
```

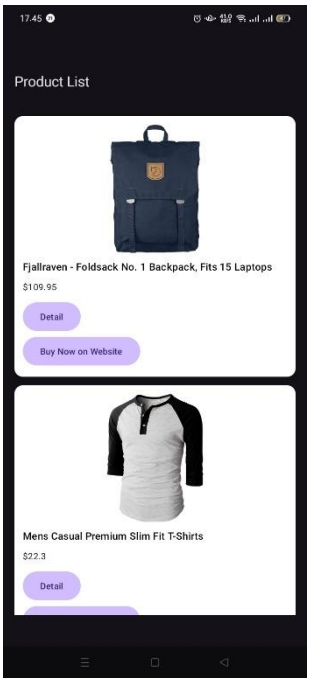
```
)

private val LightColorScheme = lightColorScheme(
    primary = Purple40,
    secondary = PurpleGrey40,
    tertiary = Pink40

    /* Other default colors to override
    background = Color(0xFFFFFBFE),
    surface = Color(0xFFFFFBFE),
    onPrimary = Color.White,
    onSecondary = Color.White,
    onTertiary = Color.White,
    onBackground = Color(0xFF1C1B1F),
    onSurface = Color(0xFF1C1B1F),
    */
)

@Composable
fun Modul_5Theme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >=
Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme)
dynamicDarkColorScheme(context) else
dynamicLightColorScheme(context)
        }

        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}
```

9. MainActivity.kt

Tabel 9 Source Code MainActivity.kt

```
1  package com.example.modul_5
2
3  import android.os.Bundle
4  import androidx.activity.ComponentActivity
5  import androidx.activity.compose.setContent
6  import androidx.activity.enableEdgeToEdge
7  import androidx.compose.foundation.layout.fillMaxSize
8  import androidx.compose.foundation.layout.padding
9  import androidx.compose.material3.Scaffold
10 import androidx.compose.ui.Modifier
11 import androidx.lifecycle.viewmodel.compose.viewModel
12 import com.example.modul_5.ui.home.HomeNavHost
13 import com.example.modul_5.ui.theme.Modul_5Theme
14 import com.example.modul_5.ui.home.HomeViewModel
15
16 class MainActivity : ComponentActivity() {
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         enableEdgeToEdge()
20         setContent {
21             Modul_5Theme {
22                 Scaffold(modifier =
23 Modifier.fillMaxSize()) { innerPadding ->
24
25                     val homeViewModel: HomeViewModel =
26 viewModel()
27
28                     HomeNavHost(
29                         viewModel = homeViewModel,
30                         modifier =
31 Modifier.padding(innerPadding)
32                     )
33                 }
34             }
35         }
36 }
```

B. Output Program



Gambar 1 Screenshot Tampilan Halaman Awal Aplikasi

Gambar 2 Screenshot Tampilan Halaman Detail Product Aplikasi

Gambar 3 Screenshot Tampilan Saat Tombol "Buy On Website" Aplikasi Ditekan

C. Pembahasan

1. model/Produt.kt

Pada file ini kita membuat sebuah data class dengan nama Product yang ditandai dengan anotasi @Serializable, yang artinya class ini bisa digunakan dalam proses serialisasi atau konversi data (misalnya JSON). Class Product ini berisi beberapa properti penting seperti id, title, price, description, category, image, dan rating. Semua properti ini mewakili atribut dari sebuah produk secara lengkap.

Selain itu, di file ini juga terdapat class Rating yang juga diberi anotasi @Serializable. Class ini merupakan bagian dari properti Product, dan berfungsi untuk menyimpan nilai rating dan jumlah rating dari produk tersebut. File ini nantinya akan digunakan saat kita mengambil atau menampilkan data produk, baik dari API ataupun dari repository lokal.

2. remote/ProductApiService.kt

Pada file ini kita membuat sebuah interface dengan nama ProductApiService yang berfungsi sebagai jembatan untuk berkomunikasi dengan API eksternal menggunakan Retrofit. Di dalamnya terdapat satu fungsi bernama getProducts() yang diberi anotasi @GET("products"). Anotasi ini menandakan bahwa fungsi ini akan melakukan request HTTP GET ke endpoint products dan akan mengembalikan daftar data bertipe List<Product>.

Fungsi getProducts() bersifat suspend, artinya hanya bisa dipanggil dari coroutine, karena proses pengambilan data dari internet bersifat asynchronous. File ini akan sangat penting saat menghubungkan aplikasi dengan data dari server atau API eksternal.

3. remote/RetrofitInstance.kt

Pada file ini kita membuat sebuah object dengan nama RetrofitInstance yang berfungsi untuk mengatur dan menyediakan instance Retrofit yang akan digunakan untuk mengakses API. Di dalamnya, kita menggunakan konfigurasi Json dari Kotlin Serialization yang disesuaikan dengan properti ignoreUnknownKeys = true, yang berarti jika ada data dari API yang tidak sesuai dengan struktur data di aplikasi, maka akan diabaikan agar tidak menimbulkan error.

Selanjutnya, kita membuat properti api bertipe ProductApiService yang diinisialisasi secara lazy. Di dalamnya, kita menggunakan Retrofit.Builder() dengan baseUrl yang mengarah ke https://fakestoreapi.com/ dan menambahkan ConverterFactory dari Kotlinx Serialization agar data JSON bisa dikonversi otomatis ke objek Kotlin. File ini menjadi kunci utama untuk menghubungkan aplikasi dengan API, dan akan digunakan saat kita ingin mengambil data produk secara online melalui Retrofit.

4. repository/ProductRepository.kt

Pada file ini kita membuat sebuah class dengan nama ProductRepository yang berfungsi sebagai pengelola data dari sumber eksternal, dalam hal ini adalah API. Di

dalam class ini terdapat satu fungsi utama getProducts() yang akan mengembalikan data berupa Flow<Result<List<Product>>>. Ini berarti data produk akan dikirimkan secara asynchronous dan bisa ditangani dengan cara yang reaktif menggunakan Flow.

Fungsi ini mengambil data dari RetrofitInstance.api.getProducts() yang telah didefinisikan sebelumnya, lalu menggunakan emit() untuk mengirimkan hasilnya. Jika data berhasil diambil, maka akan dikirim dalam bentuk Result.success(products). Namun, jika terjadi kesalahan seperti error jaringan atau parsing, maka akan dikirim dalam bentuk Result.failure(e). Logcat juga digunakan untuk mencatat keberhasilan atau kegagalan proses pengambilan data, yang membantu dalam proses debugging.

File ini menjadi penghubung antara lapisan data (API) dan ViewModel, sehingga pemisahan tanggung jawab antar bagian dalam arsitektur aplikasi tetap terjaga.

5. ui/detail/DetailScreen.kt

Pada file ini kita membuat tampilan detail produk yang diberi nama DetailScreen. Fungsi ini merupakan komponen Composable yang digunakan untuk menampilkan informasi lengkap dari salah satu produk berdasarkan ID-nya. Fungsi ini menerima dua parameter, yaitu productId yang merepresentasikan ID produk yang ingin ditampilkan, dan viewModel dari HomeViewModel yang akan digunakan untuk mengambil state produk dari ViewModel. Data yang diamati mencakup tiga hal utama: products yang merupakan daftar seluruh produk, isLoading untuk mengetahui apakah data sedang dimuat, serta error yang akan menampilkan pesan jika terjadi kesalahan saat pengambilan data.

Struktur UI-nya dibangun menggunakan Scaffold yang dilengkapi dengan TopAppBar sebagai header dengan judul "Detail Produk". Kemudian, isi dari tampilan akan menyesuaikan kondisi yang sedang terjadi. Jika data masih dimuat, maka akan ditampilkan CircularProgressIndicator. Jika terjadi kesalahan, maka akan muncul pesan kesalahan yang ditandai dengan warna error. Jika produk berhasil ditemukan berdasarkan productId, maka fungsi ProductDetailContent akan dipanggil untuk menampilkan informasi produk seperti nama, harga, kategori, dan deskripsi secara

terstruktur dalam bentuk kolom. Namun jika produk tidak ditemukan, akan muncul teks informasi bahwa produk tidak tersedia.

6. ui/home/HomeScreen.kt

Pada file ini kita membuat beberapa fungsi composable utama untuk menampilkan tampilan utama aplikasi. Fungsi HomeNavHost berfungsi sebagai pengatur navigasi antar layar dengan menggunakan NavHost dan NavController. Di dalamnya terdapat dua route utama yaitu "home" yang menampilkan daftar produk melalui HomeScreen, serta "detail/{productId}" yang menampilkan detail produk tertentu berdasarkan productId melalui DetailScreen.

Fungsi HomeScreen menampilkan daftar produk dalam bentuk list menggunakan LazyColumn. Data produk diambil dari HomeViewModel yang dipantau melalui collectAsState(). Pada tampilan ini terdapat kondisi loading yang ditandai dengan CircularProgressIndicator, kondisi error yang menampilkan pesan error, dan kondisi sukses yang menampilkan daftar produk menggunakan ProductCard. Setiap item produk dapat diklik untuk menuju detail produk atau membuka halaman website Shopee untuk membeli produk tersebut melalui browser.

ProductCard adalah komponen UI yang merepresentasikan satu produk dalam bentuk kartu yang berisi gambar produk, judul, harga, dan dua tombol aksi yaitu tombol untuk melihat detail produk dan tombol untuk membuka website pembelian. Tombol pembelian mengarahkan pengguna ke browser dengan pencarian produk di Shopee menggunakan Intent.ACTION_VIEW.

Fungsi DetailScreen menampilkan rincian lengkap dari produk berdasarkan productId yang diterima. Data produk juga diambil dari HomeViewModel dan dipantau status loading serta error-nya. Jika data berhasil didapat, detail produk seperti gambar, nama, harga, kategori, deskripsi, dan rating akan ditampilkan secara rapi. Jika produk tidak ditemukan atau terjadi error, pesan yang sesuai akan ditampilkan. File ini sangat penting sebagai penghubung antara data produk dan tampilan UI di aplikasi dengan

memanfaatkan Jetpack Compose serta Navigation Compose untuk pengalaman navigasi yang lancar dan interaktif.

7. ui/home/HomeViewModel.kt

Pada file ini dibuat sebuah kelas HomeViewModel yang berperan sebagai jembatan antara data repository dan UI. Kelas ini menggunakan ViewModel dari Android Architecture Components dan mengelola state aplikasi dengan memanfaatkan StateFlow dan MutableStateFlow untuk mengawasi perubahan data produk, status loading, serta error. Di dalam HomeViewModel terdapat tiga properti utama yang dibungkus StateFlow: _products yang menyimpan daftar produk dalam bentuk list, _isLoading yang menandakan status pemuatan data sedang berlangsung, dan _error yang menyimpan pesan error jika terjadi kegagalan dalam pengambilan data.

Pada inisialisasi kelas, fungsi fetchProducts() langsung dipanggil untuk memulai pengambilan data produk dari ProductRepository. Fungsi ini menggunakan viewModelScope.launch agar proses fetching berjalan dalam coroutine tanpa mengganggu thread utama. Ketika data berhasil diambil, products diperbarui dengan data terbaru dan error diset null. Jika terjadi kegagalan, daftar produk dikosongkan dan pesan error ditampilkan. File ini sangat penting untuk mengatur alur data dan status aplikasi sehingga UI dapat merespons perubahan data secara reaktif dan efisien dengan menggunakan Jetpack Compose.

8. ui/theme/Theme.kt

File ini bertugas mengatur tema visual aplikasi menggunakan Jetpack Compose Material3, termasuk dukungan untuk mode gelap (dark mode) dan mode terang (light mode). Di dalamnya didefinisikan dua skema warna utama, yaitu DarkColorScheme untuk tema gelap dan LightColorScheme untuk tema terang, dengan warna-warna primer, sekunder, dan tersier yang sudah ditentukan.

Fungsi composable Modul_5Theme adalah pusat pengaturan tema. Fungsi ini secara otomatis mendeteksi apakah perangkat sedang menggunakan mode gelap melalui isSystemInDarkTheme(), sehingga aplikasi bisa menyesuaikan tampilannya

mengikuti preferensi pengguna tanpa pengaturan manual. Selain itu, untuk perangkat Android versi 12 ke atas, aplikasi memanfaatkan fitur dynamic color scheme yang mengambil warna tema dari wallpaper perangkat, membuat tampilan aplikasi lebih menyatu dan personal.

9. MainActivity.kt

File ini merupakan entry point utama aplikasi yang mengatur tampilan dan alur navigasi menggunakan Jetpack Compose. Di dalamnya terdapat class MainActivity yang mewarisi ComponentActivity. Pada fungsi onCreate, pertama-tama dipanggil enableEdgeToEdge() agar aplikasi bisa menampilkan konten hingga ke tepi layar, memberikan tampilan yang lebih modern dan immersive.

Selanjutnya, dengan setContent aplikasi mulai menggunakan tema yang sudah dibuat di Modul_5Theme sehingga tampilan mengikuti pengaturan warna dan tipografi yang konsisten, termasuk dukungan dark mode.

Di dalam tema tersebut, digunakan Scaffold sebagai struktur dasar UI yang menyediakan area standar seperti top bar, bottom bar, dan konten utama. Padding dari scaffold ini dioper ke konten agar tata letak responsif terhadap elemen UI lainnya.

ViewModel HomeViewModel diinisialisasi menggunakan Compose viewModel() yang memastikan data produk dikelola secara lifecycle-aware dan bisa digunakan di seluruh composable.

Terakhir, fungsi HomeNavHost dipanggil untuk mengatur navigasi antar layar (home dan detail produk) dengan ViewModel yang sudah tersedia, sehingga aplikasi dapat menampilkan daftar produk dan detailnya secara dinamis.

D. Tautan Git

https://github.com/KunyitAlami/Praktikum-Pemrograman-Mobile-I-Ghani-Mudzakir.git