LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

CONNECT TO THE INTERNET

Oleh:

Sheila Sabina **NIM. 2310817220028**

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN I
MODUL 5

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

Nama Praktikan : Sheila Sabina
NIM : 2310817220028

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

LEMBAR PENGESAHAN 2
DAFTAR ISI 3
DAFTAR GAMBAR 4
DAFTAR TABEL 5
SOAL 1 6
A. Source Code 6
B. Output Program 33
C. Pembahasan 36
D. Tautan Git 66

DAFTAR GAMBAR

Gambar 1. Screesnshot Output Program Soal 1 35

DAFTAR TABEL

Tabel 1. Source Code AndroidManifest.xml 6
Tabel 2. Source Code GunungDao 7
Tabel 3. Source Code AppDatabase 8
Tabel 4. Source Code GunungEntity.kt 9
Tabel 5. Source Code GunungMapper.kt 9
Tabel 6. Source Code Gunung 10
Tabel 7. Source Code ApiClient 10
Tabel 8. Source Code GunungApiService 11
Tabel 9. Source Code GunungRepository 12
Tabel 10. Source Code RepositoryInstance 13
Tabel 11. Source Code FavoriteGunungFragment.kt 13
Tabel 12. Source Code GunungAdapter 15
Tabel 13. Source Code GunungDetailFragment 17
Tabel 14. Source Code GunungListFragment 18
Tabel 15. Source Code ViewPagerAdapter 20
Tabel 16. Source Code GunungViewModel 21
Tabel 17. Source Code GunungViewModelFactory 23
Tabel 18. Source Code MainActivity 23
Tabel 19. Source Code activity_main.xml 24
Tabel 20. Source Code detail_fragment.xml 25
Tabel 21. Source Code fragment_favorit_gunung.xml 26
Tabel 22. Source Code item_gunung.xml 26
Tabel 23. Source Code list_fragment.xml 29
Tabel 24. Source Code colors.xml 30
Tabel 25. Source Code string.xml 30
Tabel 26. Source Code themes.xml 30
Tabel 27. Source Code build.gradle.kts 31

SOAL 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 untuk status dan error handling pada API dan Flow untuk data stream.
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

MODUL5/app/src/main/AndroidManifest.xml

1. AndroidManifest.xml

Tabel 1. Source Code AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest
3  xmlns:android="http://schemas.android.com/apk/res/android"
4      package="com.example.modul5">
5
6      <uses-permission
7  android:name="android.permission.ACCESS_NETWORK_STATE" />
8      <uses-permission android:name="android.permission.INTERNET"
9  />
10
11     <application
12         android:allowBackup="true"
13
14 android:dataExtractionRules="@xml/data_extraction_rules"
15         android:fullBackupContent="@xml/backup_rules"
16         android:icon="@mipmap/ic_launcher"
17         android:label="@string/app_name"
```

```
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Modul5">

        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

com/example/modul5/data/local/entity/dao

2. GunungDao

Tabel 2. Source Code GunungDao

```
package com.example.modul5.data.local.dao

import androidx.room.*
import com.example.modul5.data.local.entity.GunungEntity
import kotlinx.coroutines.flow.Flow

@Dao
interface GunungDao {

    @Query("SELECT * FROM gunung")
    fun getAllGunung(): Flow<List<GunungEntity>>

    @Query("SELECT * FROM gunung")
    suspend fun getAllGunungOnce(): List<GunungEntity>

    @Query("SELECT * FROM gunung WHERE isFavorite = 1")
    suspend fun getAllFavoriteGunung(): List<GunungEntity>

    @Query("SELECT * FROM gunung WHERE isFavorite = 1")
    fun getAllFavoriteGunungFlow(): Flow<List<GunungEntity>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertGunungList(gunungList:
List<GunungEntity>)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertGunung(gunung: GunungEntity)
```

```
28
29      @Query("DELETE FROM gunung")
30      suspend fun clearGunung()
31
32      @Delete
33      suspend fun delete(gunung: GunungEntity)
34
35      @Update
36      suspend fun update(gunung: GunungEntity)
37
38      @Query("UPDATE gunung SET isFavorite = :isFavorite WHERE
39  name = :name")
40      suspend fun updateFavoriteStatus(name: String, isFavorite:
41  Boolean)
42
43  }
```

com/example/modul5/data/local/entity/database

3. AppDatabase

Tabel 3. Source Code AppDatabase

```
 1  package com.example.modul5.data.local.database
 2
 3  import android.content.Context
 4  import androidx.room.Database
 5  import androidx.room.Room
 6  import androidx.room.RoomDatabase
 7  import com.example.modul5.data.local.dao.GunungDao
 8  import com.example.modul5.data.local.entity.GunungEntity
 9
10  @Database(entities = [GunungEntity::class], version = 1,
11  exportSchema = false)
12  abstract class AppDatabase : RoomDatabase() {
13
14      abstract fun gunungDao(): GunungDao
15
16      companion object {
17          @Volatile
18          private var INSTANCE: AppDatabase? = null
19
20          fun getDatabase(context: Context): AppDatabase {
21              return INSTANCE ?: synchronized(this) {
22                  val instance = Room.databaseBuilder(
23                      context.applicationContext,
24                      AppDatabase::class.java,
25                      "gunung_database"
26                  ).build()
27                  INSTANCE = instance
28                  instance
29              }
```

```
30          }
31      }
32  }
```

com/example/modul5/data/local/entity/entity

4. GunungEntity.kt

Tabel 4. Source Code GunungEntity.kt

```
1  package com.example.modul5.data.local.entity
2
3  import androidx.room.Entity
4  import androidx.room.PrimaryKey
5
6  @Entity(tableName = "gunung")
7  data class GunungEntity(
8      @PrimaryKey val name: String,
9      val lokasi: String,
10     val deskripsi: String,
11     val link: String,
12     val image: String,
13     val isFavorite: Boolean
14 )
```

com/example/modul5/data/mapper

5. GunungMapper.kt

Tabel 5. Source Code GunungMapper.kt

```
1  package com.example.modul5.data.mapper
2
3  import com.example.modul5.data.model.Gunung
4  import com.example.modul5.data.local.entity.GunungEntity
5
6  fun Gunung.toEntity(): GunungEntity {
7      return GunungEntity(
8          name = name,
9          lokasi = lokasi,
10         deskripsi = deskripsi,
11         link = link,
12         image = image,
13         isFavorite = isFavorite
14     )
15 }
16
17 fun GunungEntity.toModel(): Gunung {
18     return Gunung(
19         name = name,
20         lokasi = lokasi,
21         deskripsi = deskripsi,
22         link = link,
```

```
23          image = image,
24          isFavorite = isFavorite
25      )
26  }
```

com/example/modul5/data/model

6. Gunung

Tabel 6. Source Code Gunung

```
 1  package com.example.modul5.data.model
 2
 3  import kotlinx.serialization.SerialName
 4  import kotlinx.serialization.Serializable
 5  import java.io.Serializable as JavaSerializable
 6
 7  @Serializable
 8  data class Gunung(
 9      @SerialName("name")
10      val name: String,
11
12      @SerialName("lokasi")
13      val lokasi: String,
14
15      @SerialName("deskripsi")
16      val deskripsi: String,
17
18      @SerialName("link")
19      val link: String,
20
21      @SerialName("image_url")
22      val image: String,
23
24      val isFavorite: Boolean = false
25  )   : JavaSerializable
```

com/example/modul5/data/remote

7. ApiClient

Tabel 7. Source Code ApiClient

```
 1  package com.example.modul5.data.remote
 2
 3  import kotlinx.serialization.json.Json
 4  import okhttp3.MediaType.Companion.toMediaType
 5  import okhttp3.OkHttpClient
 6  import okhttp3.logging.HttpLoggingInterceptor
 7  import retrofit2.Retrofit
 8  import
 9  com.jakewharton.retrofit2.converter.kotlinx.serialization.asCo
10  nverterFactory
```

```
11
12 import java.util.concurrent.TimeUnit
13
14 object ApiClient {
15     private const val BASE_URL =
16 "https://modul5.free.beeceptor.com/api/path/"
17
18     private val json = Json {
19         ignoreUnknownKeys = true
20     }
21
22     private val loggingInterceptor =
23 HttpLoggingInterceptor().apply {
24         level = HttpLoggingInterceptor.Level.BODY
25     }
26
27     private val okHttpClient = OkHttpClient.Builder()
28         .addInterceptor(loggingInterceptor)
29         .connectTimeout(30, TimeUnit.SECONDS)
30         .readTimeout(30, TimeUnit.SECONDS)
31         .build()
32
33     val retrofit: Retrofit = Retrofit.Builder()
34         .baseUrl(BASE_URL)
35         .client(okHttpClient)
36
37 .addConverterFactory(json.asConverterFactory("application/json
38 ".toMediaType()))
39         .build()
40
41     val gunungApiService: GunungApiService =
42 retrofit.create(GunungApiService::class.java)
43 }
```

8. GunungApiService

Tabel 8. Source Code GunungApiService

```
1 package com.example.modul5.data.remote
2
3 import com.example.modul5.data.model.Gunung
4 import retrofit2.http.GET
5
6 interface GunungApiService {
7     @GET("gunung")
8     suspend fun getGunungList(): List<Gunung>
9 }
```

com/example/modul5/data/repository

9. GunungRepository

Tabel 9. Source Code GunungRepository

```
1  package com.example.modul5.data.repository
2
3  import android.content.Context
4  import com.example.modul5.data.local.database.AppDatabase
5  import com.example.modul5.data.mapper.toEntity
6  import com.example.modul5.data.mapper.toModel
7  import com.example.modul5.data.model.Gunung
8  import com.example.modul5.data.remote.ApiClient
9  import kotlinx.coroutines.Dispatchers
10 import kotlinx.coroutines.flow.Flow
11 import kotlinx.coroutines.flow.map
12 import kotlinx.coroutines.withContext
13
14 class GunungRepository(context: Context) {
15
16     private val apiService = ApiClient.gunungApiService
17     private val gunungDao =
18 AppDatabase.getDatabase(context).gunungDao()
19
20     fun getFavoriteGunungListFlow(): Flow<List<Gunung>> {
21         return gunungDao.getAllFavoriteGunungFlow()
22             .map { it.map { entity -> entity.toModel() } }
23     }
24
25     suspend fun fetchGunungList(): List<Gunung> =
26 withContext(Dispatchers.IO) {
27         try {
28             val remoteList = apiService.getGunungList()
29             val favoriteNames =
30 gunungDao.getAllFavoriteGunung().map { it.name }
31
32             val mergedList = remoteList.map { gunung ->
33                 if (gunung.name in favoriteNames)
34 gunung.copy(isFavorite = true)
35                 else gunung
36             }
37
38             gunungDao.clearGunung()
39             gunungDao.insertGunungList(mergedList.map {
40 it.toEntity() })
41
42             return@withContext mergedList
43         } catch (e: Exception) {
44             return@withContext gunungDao.getAllGunungOnce().map
45 { it.toModel() }
46         }
47     }
48
49     suspend fun updateGunung(gunung: Gunung) =
50 withContext(Dispatchers.IO) {
51         gunungDao.update(gunung.toEntity())
```

```
52      }
53
54      suspend fun insertFavoriteGunung(gunung: Gunung) =
55  withContext(Dispatchers.IO) {
56          gunungDao.updateFavoriteStatus(gunung.name, true)
57      }
58
59      suspend fun deleteFavoriteGunung(gunung: Gunung) =
60  withContext(Dispatchers.IO) {
61          gunungDao.updateFavoriteStatus(gunung.name, false)
62      }
63  }
```

10. RepositoryInstance

Tabel 10. Source Code RepositoryInstance

```
1   package com.example.modul5.data.repository
2
3   import android.content.Context
4
5   object RepositoryInstance {
6       private var repository: GunungRepository? = null
7
8       fun provideRepository(context: Context): GunungRepository {
9           return repository ?: synchronized(this) {
10              val instance =
11  GunungRepository(context.applicationContext)
12              repository = instance
13              instance
14          }
15      }
16  }
```

com/example/modul5/ui

11. FavoriteGunungFragment.kt

Tabel 11. Source Code FavoriteGunungFragment.kt

```
1   package com.example.modul5.ui
2
3   import android.content.Intent
4   import android.net.Uri
5   import androidx.fragment.app.viewModels
6   import android.os.Bundle
7   import android.view.LayoutInflater
8   import android.view.View
9   import android.view.ViewGroup
10  import androidx.fragment.app.Fragment
11  import androidx.lifecycle.lifecycleScope
12  import androidx.recyclerview.widget.LinearLayoutManager
13  import com.example.modul5.R
```

```
14 import
15 com.example.modul5.databinding.FragmentFavoritGunungBinding
16 import com.example.modul5.viewmodel.GunungViewModel
17 import com.example.modul5.viewmodel.GunungViewModelFactory
18 import com.example.modul5.data.repository.RepositoryInstance
19 import kotlinx.coroutines.launch
20
21 class FavoritGunungFragment : Fragment() {
22
23     private var _binding: FragmentFavoritGunungBinding? =
24 null
25     private val binding get() = _binding!!
26
27     private lateinit var adapter: GunungAdapter
28
29     private val viewModel: GunungViewModel by viewModels {
30
31 GunungViewModelFactory(RepositoryInstance.provideRepository(
32 requireContext()))
33     }
34
35     override fun onCreateView(
36         inflater: LayoutInflater, container: ViewGroup?,
37         savedInstanceState: Bundle?
38     ): View {
39         _binding =
40 FragmentFavoritGunungBinding.inflate(inflater, container,
41 false)
42         return binding.root
43     }
44
45     override fun onViewCreated(view: View,
46 savedInstanceState: Bundle?) {
47         super.onViewCreated(view, savedInstanceState)
48
49         adapter = GunungAdapter(
50             onLinkClick = { gunung ->
51                 val intent = Intent(Intent.ACTION_VIEW,
52 Uri.parse(gunung.link))
53                 startActivity(intent)
54             },
55             onDetailClick = { gunung ->
56                 val detail = GunungDetailFragment().apply {
57                     arguments = Bundle().apply {
58                         putString("EXTRA_NAME", gunung.name)
59                         putString("EXTRA_LOKASI",
60 gunung.lokasi)
61                         putString("EXTRA_DESKRIPSI",
62 gunung.deskripsi)
63                         putString("EXTRA_PHOTO",
64 gunung.image)
65                     }
```

```
                }
                parentFragmentManager.beginTransaction()
                    .replace(R.id.fragmentContainerGunung,
detail)
                    .addToBackStack(null)
                    .commit()


requireActivity().findViewById<View>(R.id.fragmentContainerG
unung).visibility = View.VISIBLE
            }
            ,
            onFavoriteClick = { gunung ->
                val updatedGunung = gunung.copy(isFavorite =
!gunung.isFavorite)
                viewModel.updateGunung(updatedGunung)
            }
        )

        binding.recyclerViewFavorit.layoutManager =
LinearLayoutManager(requireContext())
        binding.recyclerViewFavorit.adapter = adapter

        lifecycleScope.launch {
            viewModel.favoriteList.collect { favorites ->
                adapter.submitList(favorites)
            }
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

12. GunungAdapter

Tabel 12. Source Code GunungAdapter

```
package com.example.modul5.ui

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import coil.load
import com.example.modul5.R
import com.example.modul5.data.model.Gunung
import com.example.modul5.databinding.ItemGunungBinding

```

```
13  class GunungAdapter(
14      private val onLinkClick: (Gunung) -> Unit,
15      private val onDetailClick: (Gunung) -> Unit,
16      private val onFavoriteClick: (Gunung) -> Unit
17  ) : ListAdapter<Gunung,
18  GunungAdapter.ListViewHolder>(DIFF_CALLBACK) {
19
20      inner class ListViewHolder(private val binding:
21  ItemGunungBinding) :
22          RecyclerView.ViewHolder(binding.root) {
23
24          fun bind(gunung: Gunung) {
25              binding.tvGunungName.text = gunung.name
26              binding.tvGunungLokasi.text = gunung.lokasi
27              binding.tvGunungDeskripsi.text = gunung.deskripsi
28              binding.imgGunung.load(gunung.image)
29
30              val iconRes = if (gunung.isFavorite)
31  R.drawable.ic_star_filled else R.drawable.ic_star_border
32              binding.btnFavorite.setImageResource(iconRes)
33
34              binding.btnLink.setOnClickListener {
35                  onLinkClick(gunung)
36              }
37              binding.btnDetail.setOnClickListener {
38                  onDetailClick(gunung)
39              }
40              binding.btnFavorite.setOnClickListener {
41                  onFavoriteClick(gunung)
42              }
43          }
44      }
45
46      override fun onCreateViewHolder(parent: ViewGroup,
47  viewType: Int): ListViewHolder {
48          val binding =
49  ItemGunungBinding.inflate(LayoutInflater.from(parent.context),
50  parent, false)
51          return ListViewHolder(binding)
52      }
53
54      override fun onBindViewHolder(holder: ListViewHolder,
55  position: Int) {
56          holder.bind(getItem(position))
57      }
58
59      companion object {
60          private val DIFF_CALLBACK = object :
61  DiffUtil.ItemCallback<Gunung>() {
62              override fun areItemsTheSame(oldItem: Gunung,
63  newItem: Gunung) =
64                  oldItem.name == newItem.name
```

```
65
66              override fun areContentsTheSame(oldItem: Gunung,
67  newItem: Gunung) =
68                  oldItem == newItem
69          }
70      }
71  }
```

13. GunungDetailFragment

Tabel 13. Source Code GunungDetailFragment

```
1   package com.example.modul5.ui
2
3   import android.os.Bundle
4   import androidx.fragment.app.Fragment
5   import android.view.LayoutInflater
6   import android.view.View
7   import android.view.ViewGroup
8   import coil.load
9   import com.example.modul5.R
10  import com.example.modul5.databinding.DetailFragmentBinding
11
12  class GunungDetailFragment : Fragment() {
13
14      private var _binding: DetailFragmentBinding? = null
15      private val binding get() = _binding!!
16
17      override fun onCreateView(
18          inflater: LayoutInflater, container: ViewGroup?,
19          savedInstanceState: Bundle?
20      ): View {
21          _binding = DetailFragmentBinding.inflate(inflater, container,
22  false)
23
24          val name = arguments?.getString("EXTRA_NAME") ?: "Tidak ada
25  nama"
26          val lokasi = arguments?.getString("EXTRA_LOKASI") ?: "Tidak
27  ada lokasi"
28          val deskripsi = arguments?.getString("EXTRA_DESKRIPSI") ?:
29  "Deskripsi belum tersedia"
30          val image = arguments?.getString("EXTRA_PHOTO")
31
32          binding.tvName.text = name
33          binding.tvLokasi.text = lokasi
34          binding.tvDeskripsi.text = deskripsi
35
36          image?.let {
37              binding.imgPoster.load(it) {
38                  crossfade(true)
39                  placeholder(android.R.drawable.ic_menu_gallery)
40                  error(android.R.drawable.ic_menu_report_image)
41              }
```

```
42          }
43
44          binding.btnBack.setOnClickListener {
45              parentFragmentManager.popBackStack()
46
47  activity?.findViewById<View>(R.id.fragmentContainerGunung)?.visibility
48  = View.GONE
49          }
50
51          return binding.root
52      }
53
54      override fun onDestroyView() {
55          super.onDestroyView()
56          _binding = null
57      }
58  }
```

14. GunungListFragment

Tabel 14. Source Code GunungListFragment

```
 1  package com.example.modul5.ui
 2
 3  import android.content.Intent
 4  import android.net.Uri
 5  import android.os.Bundle
 6  import android.view.LayoutInflater
 7  import android.view.View
 8  import android.view.ViewGroup
 9  import android.widget.Toast
10  import androidx.fragment.app.Fragment
11  import androidx.lifecycle.ViewModelProvider
12  import androidx.lifecycle.lifecycleScope
13  import androidx.recyclerview.widget.LinearLayoutManager
14  import com.example.modul5.R
15  import com.example.modul5.data.repository.GunungRepository
16  import com.example.modul5.databinding.ListFragmentBinding
17  import com.example.modul5.viewmodel.GunungViewModel
18  import com.example.modul5.viewmodel.GunungViewModelFactory
19  import kotlinx.coroutines.flow.collect
20
21  class GunungListFragment : Fragment() {
22
23      private var _binding: ListFragmentBinding? = null
24      private val binding get() = _binding!!
25
26      private lateinit var viewModel: GunungViewModel
27      private lateinit var adapter: GunungAdapter
28
29      override fun onCreate(savedInstanceState: Bundle?) {
30          super.onCreate(savedInstanceState)
31          val repository = GunungRepository(requireContext())
```

```
        val factory = GunungViewModelFactory(repository)
        viewModel = ViewModelProvider(this,
factory)[GunungViewModel::class.java]
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = ListFragmentBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View,
savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        adapter = GunungAdapter(
            onLinkClick = { gunung ->
                val intent = Intent(Intent.ACTION_VIEW,
Uri.parse(gunung.link))
                startActivity(intent)
            },
            onDetailClick = { gunung ->
                viewModel.selectGunung(gunung)
            },
            onFavoriteClick = { gunung ->
                val updatedGunung = gunung.copy(isFavorite =
!gunung.isFavorite)
                viewModel.updateGunung(updatedGunung)
            }
        )

        binding.rvGunung.layoutManager =
LinearLayoutManager(requireContext())
        binding.rvGunung.adapter = adapter

        lifecycleScope.launchWhenStarted {
            viewModel.gunungList.collect { listGunung ->
                adapter.submitList(listGunung)
            }
        }

        lifecycleScope.launchWhenStarted {
            viewModel.errorMessage.collect { error ->
                error?.let {
                    Toast.makeText(requireContext(), it,
Toast.LENGTH_SHORT).show()
                }
            }
```

```
84          }
85
86          lifecycleScope.launchWhenStarted {
87              viewModel.selectedGunung.collect { selected ->
88                  selected?.let {
89                      val detailFragment =
90  GunungDetailFragment().apply {
91                          arguments = Bundle().apply {
92                              putString("EXTRA_NAME", it.name)
93                              putString("EXTRA_LOKASI",
94  it.lokasi)
95                              putString("EXTRA_DESKRIPSI",
96  it.deskripsi)
97                              putString("EXTRA_PHOTO",
98  it.image)
99                          }
100                     }
101
102
103 requireActivity().supportFragmentManager.beginTransaction()
104
105 .replace(R.id.fragmentContainerGunung, detailFragment)
106                         .addToBackStack(null)
107                         .commit()
108
109
110 requireActivity().findViewById<View>(R.id.fragmentContainerG
111 unung).visibility = View.VISIBLE
112                 }
113             }
114         }
115     }
116
117     override fun onDestroyView() {
118         super.onDestroyView()
119         _binding = null
120     }
121 }
123
 7
```

15. ViewPagerAdapter

Tabel 15. Source Code ViewPagerAdapter

```
1  package com.example.modul5.ui
2
3  import androidx.fragment.app.Fragment
4  import androidx.fragment.app.FragmentActivity
5  import androidx.viewpager2.adapter.FragmentStateAdapter
6
7  class ViewPagerAdapter(activity: FragmentActivity) :
8  FragmentStateAdapter(activity) {
```

```
 9
10      private val fragments = listOf(
11          GunungListFragment(),
12          FavoritGunungFragment()
13      )
14
15      private val fragmentTitles = listOf(
16          "List Gunung",
17          "Favorit"
18      )
19
20      override fun getItemCount(): Int = fragments.size
21
22      override fun createFragment(position: Int): Fragment =
23  fragments[position]
24
25      fun getPageTitle(position: Int): String =
26  fragmentTitles[position]
27  }
```

com/example/modul5/viewmodel

16. GunungViewModel

Tabel 16. Source Code GunungViewModel

```
 1  package com.example.modul5.viewmodel
 2
 3  import android.util.Log
 4  import androidx.lifecycle.ViewModel
 5  import androidx.lifecycle.viewModelScope
 6  import com.example.modul5.data.model.Gunung
 7  import com.example.modul5.data.repository.GunungRepository
 8  import kotlinx.coroutines.flow.*
 9  import kotlinx.coroutines.launch
10
11  class GunungViewModel(private val repository: GunungRepository)
12  : ViewModel() {
13
14      private val _gunungList =
15  MutableStateFlow<List<Gunung>>(emptyList())
16      val gunungList: StateFlow<List<Gunung>> =
17  _gunungList.asStateFlow()
18
19      val favoriteList: StateFlow<List<Gunung>> =
20  repository.getFavoriteGunungListFlow()
21          .stateIn(viewModelScope,
22  SharingStarted.WhileSubscribed(5000), emptyList())
23
24      private val _selectedGunung =
25  MutableStateFlow<Gunung?>(null)
26      val selectedGunung: StateFlow<Gunung?> =
```

```
_selectedGunung.asStateFlow()

    private val _errorMessage = MutableStateFlow<String?>(null)
    val errorMessage: StateFlow<String?> =
_errorMessage.asStateFlow()

    init {
        fetchGunungList()
    }

    private fun fetchGunungList() {
        viewModelScope.launch {
            try {
                val remoteList = repository.fetchGunungList()
                _gunungList.value = remoteList
            } catch (e: Exception) {
                _errorMessage.value = "Gagal memuat data:
${e.localizedMessage}"
            }
        }
    }

    fun selectGunung(gunung: Gunung) {
        _selectedGunung.value = gunung
        Log.d("GunungViewModel", "Gunung dipilih:
${gunung.name}")
    }

    fun updateGunung(updatedGunung: Gunung) {
        viewModelScope.launch {
            try {
                if (updatedGunung.isFavorite) {

repository.insertFavoriteGunung(updatedGunung)
                } else {

repository.deleteFavoriteGunung(updatedGunung)
                }

                _gunungList.value = _gunungList.value.map {
                    if (it.name == updatedGunung.name)
updatedGunung else it
                }

                Log.d("GunungViewModel", "Gunung diperbarui dan
disimpan: ${updatedGunung.name}, Favorite:
${updatedGunung.isFavorite}")
            } catch (e: Exception) {
                _errorMessage.value = "Gagal menyimpan favorit:
${e.localizedMessage}"
            }
        }
```

```
79      }
80
81      fun onLinkClicked(gunung: Gunung) {
82          Log.d("GunungViewModel", "Explicit intent diklik untuk:
83  ${gunung.name}")
84      }
85  }
```

17. GunungViewModelFactory

Tabel 17. Source Code GunungViewModelFactory

```
 1  package com.example.modul5.viewmodel
 2
 3  import androidx.lifecycle.ViewModel
 4  import androidx.lifecycle.ViewModelProvider
 5  import com.example.modul5.data.repository.GunungRepository
 6
 7  class GunungViewModelFactory(private val repository:
 8  GunungRepository) : ViewModelProvider.Factory {
 9      override fun <T : ViewModel> create(modelClass: Class<T>):
10  T {
11          if
12  (modelClass.isAssignableFrom(GunungViewModel::class.java)) {
13              @Suppress("UNCHECKED_CAST")
14              return GunungViewModel(repository) as T
15          }
16          throw IllegalArgumentException("Unknown ViewModel
17  class")
18      }
19  }
```

com/example/modul5/MainActivity.kt

18. MainActivity

Tabel 18. Source Code MainActivity

```
 1  package com.example.modul5
 2
 3  import android.os.Bundle
 4  import androidx.appcompat.app.AppCompatActivity
 5  import com.example.modul5.databinding.ActivityMainBinding
 6  import com.example.modul5.ui.ViewPagerAdapter
 7  import com.google.android.material.tabs.TabLayoutMediator
 8
 9  class MainActivity : AppCompatActivity() {
10
11      private lateinit var binding: ActivityMainBinding
12
13      override fun onCreate(savedInstanceState: Bundle?) {
14          super.onCreate(savedInstanceState)
15
```

```
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val viewPagerAdapter = ViewPagerAdapter(this)
        binding.viewPager.adapter = viewPagerAdapter

        TabLayoutMediator(binding.tabLayout, binding.viewPager)
{ tab, position ->
            tab.text = viewPagerAdapter.getPageTitle(position)
        }.attach()
    }
}
```

MODUL5/app/src/main/res/layout

19. activity_main.xml

Tabel 19. Source Code activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:tabIndicatorColor="@color/blue_500"
        app:tabSelectedTextColor="@color/blue_500"
        app:tabTextColor="@color/black" />

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewPager"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toBottomOf="@id/tabLayout"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <FrameLayout
        android:id="@+id/fragmentContainerGunung"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:visibility="gone"
        app:layout_constraintTop_toBottomOf="@id/tabLayout"
```

```
34          app:layout_constraintBottom_toBottomOf="parent"
35          app:layout_constraintStart_toStartOf="parent"
36          app:layout_constraintEnd_toEndOf="parent"/>
37
38  </androidx.constraintlayout.widget.ConstraintLayout>
```

20. detail_fragment.xml

Tabel 20. Source Code detail_fragment.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <ScrollView
3   xmlns:android="http://schemas.android.com/apk/res/android"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       android:background="#B6D6F1"
8       android:padding="16dp"
9       tools:context="com.example.modul5.ui.GunungDetailFragment">
10
11      <LinearLayout
12          android:layout_width="match_parent"
13          android:layout_height="wrap_content"
14          android:orientation="vertical">
15
16          <ImageView
17              android:id="@+id/imgPoster"
18              android:layout_width="match_parent"
19              android:layout_height="300dp"
20              android:contentDescription="Foto Gunung"
21              android:scaleType="centerCrop" />
22
23          <TextView
24              android:id="@+id/tvName"
25              android:layout_width="match_parent"
26              android:layout_height="wrap_content"
27              android:gravity="center"
28              android:textSize="35sp"
29              android:textStyle="bold"
30              android:textColor="@android:color/black"
31              android:layout_marginTop="12dp"
32              tools:text="Gunung Semeru" />
33
34          <TextView
35              android:id="@+id/tvLokasi"
36              android:layout_width="match_parent"
37              android:layout_height="wrap_content"
38              android:gravity="center"
39              android:textSize="22sp"
40              android:layout_marginTop="8dp"
41              tools:text="Lokasi: Jawa Timur" />
42
43          <TextView
```

```
            android:id="@+id/tvDeskripsi"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="18sp"
            android:layout_marginTop="8dp"
            android:justificationMode="inter_word"
            tools:text="Gunung tertinggi di Pulau Jawa yang
terkenal dengan jalur pendakian yang menantang dan keindahan
pemandangan alam." />

        <Button
            android:id="@+id/btnBack"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Kembali"
            android:layout_marginTop="16dp"
            android:layout_gravity="center" />
    </LinearLayout>
</ScrollView>


```

21. fragment_favorit_gunung.xml

Tabel 21. Source Code fragment_favorit_gunung.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerViewFavorit"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

22. item_gunung.xml

Tabel 22. Source Code item_gunung.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView

xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    app:cardElevation="4dp"
    app:cardCornerRadius="8dp">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/cream"
        android:padding="16dp">

        <ImageView
            android:id="@+id/imgGunung"
            android:layout_width="120dp"
            android:layout_height="160dp"
            android:scaleType="centerCrop"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
/>

        <TextView
            android:id="@+id/tvGunungName"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="12dp"
            android:textColor="@android:color/black"
            android:textSize="24sp"
            android:textStyle="bold"
            app:layout_constraintTop_toTopOf="@id/imgGunung"

app:layout_constraintStart_toEndOf="@id/imgGunung"
            app:layout_constraintEnd_toEndOf="parent"
            tools:text="Gunung Semeru" />

        <TextView
            android:id="@+id/tvGunungLokasi"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="4dp"
            android:textColor="#666666"
            android:textSize="14sp"
            android:textStyle="bold"

app:layout_constraintTop_toBottomOf="@id/tvGunungName"

app:layout_constraintStart_toStartOf="@id/tvGunungName"

app:layout_constraintEnd_toEndOf="@id/tvGunungName"
            tools:text="Lokasi: Jawa Timur" />

```

```
59          <TextView
60              android:id="@+id/tvGunungDeskripsi"
61              android:layout_width="0dp"
62              android:layout_height="wrap_content"
63              android:layout_marginTop="4dp"
64              android:ellipsize="end"
65              android:maxLines="2"
66              android:textColor="@android:color/black"
67              android:textSize="14sp"
68
69  app:layout_constraintTop_toBottomOf="@id/tvGunungLokasi"
70
71  app:layout_constraintStart_toStartOf="@id/tvGunungLokasi"
72
73  app:layout_constraintEnd_toEndOf="@id/tvGunungLokasi"
74              tools:text="Gunung tertinggi di Jawa Timur
75  dengan pemandangan yang sangat indah dan memiliki trek yang
76  menantang..." />
77
78          <androidx.constraintlayout.widget.Barrier
79              android:id="@+id/barrierStartText"
80              app:barrierDirection="start"
81
82  app:constraint_referenced_ids="tvGunungName,tvGunungLokasi,t
83  vGunungDeskripsi"
84              android:layout_width="wrap_content"
85              android:layout_height="wrap_content" />
86
87          <Button
88              android:id="@+id/btnLink"
89              android:layout_width="wrap_content"
90              android:layout_height="48dp"
91              android:minWidth="70dp"
92              android:layout_marginTop="12dp"
93              android:layout_marginEnd="4dp"
94              android:backgroundTint="@color/teal_700"
95              android:text="@string/btn_link_text"
96              android:textColor="@android:color/white"
97              android:textSize="12sp"
98              android:textStyle="bold"
99              android:includeFontPadding="false"
100
101 app:layout_constraintTop_toBottomOf="@id/tvGunungDeskripsi"
102
103 app:layout_constraintStart_toStartOf="@+id/barrierStartText"
104
105 app:layout_constraintEnd_toStartOf="@id/btnDetail"
106
107 app:layout_constraintBottom_toBottomOf="@id/btnDetail" />
108
109         <Button
110             android:id="@+id/btnDetail"
```

```
            android:layout_width="wrap_content"
            android:layout_height="48dp"
            android:minWidth="100dp"
            android:layout_marginEnd="4dp"
            android:translationY="-12dp"
            android:backgroundTint="@color/blue"
            android:text="@string/btn_detail_text"
            android:textColor="@android:color/white"
            android:textSize="12sp"
            android:textStyle="bold"
            android:includeFontPadding="false"
            app:layout_constraintTop_toTopOf="@id/btnLink"

app:layout_constraintBottom_toBottomOf="@id/btnLink"
            app:layout_constraintStart_toEndOf="@id/btnLink"

app:layout_constraintEnd_toStartOf="@id/btnFavorite" />


        <ImageButton
            android:id="@+id/btnFavorite"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:layout_marginEnd="0dp"
            android:backgroundTint="@color/pink"

android:background="?attr/selectableItemBackgroundBorderless
"

android:contentDescription="@string/btn_favorite_text"
            android:padding="8dp"
            android:scaleType="centerInside"
            android:src="@drawable/ic_star_border"
            app:layout_constraintTop_toTopOf="@id/btnLink"

app:layout_constraintBottom_toBottomOf="@id/btnLink"
            app:layout_constraintEnd_toEndOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.cardview.widget.CardView>

```

23. list_fragment.xml

Tabel 23. Source Code list_fragment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    tools:context="com.example.modul5.ui.GunungListFragment">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvGunung"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:clipToPadding="false"
        android:background="#94C2EB"
        android:padding="16dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        tools:listitem="@layout/item_gunung" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

res/values

24. colors.xml

Tabel 24. Source Code colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="blue_500">#2196F3</color>
    <color name="blue_700">#1976D2</color>
    <color name="black">#1976D2</color>
    <color name="blue">#1976D2</color>
    <color name="pink">#1976D2</color>
    <color name="teal_700">#1976D2</color>
    <color name="cream">#FFEBCD</color>
    <color name="white">#1976D2</color>
</resources>
```

25. string.xml

Tabel 25. Source Code string.xml

```
<resources>
    <string name="app_name">MODUL 5</string>
    <string name="gunung_image_desc">Gambar gunung</string>
    <string name="btn_detail_text">Detail</string>
    <string name="btn_favorite_text">☆</string>
    <string name="btn_link_text">Link</string>
</resources>
```

res/values/themes.xml

26. themes.xml

Tabel 26. Source Code themes.xml

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
```

```
    <style name="Base.Theme.Modul5"
parent="Theme.Material3.DayNight.NoActionBar">
        <item name="colorPrimary">@color/blue_500</item>
        <item name="colorOnPrimary">@android:color/white</item>
        <item
name="colorPrimaryContainer">@color/blue_700</item>
        <item
name="colorOnPrimaryContainer">@android:color/white</item>
    </style>

    <style name="Theme.Modul5" parent="Base.Theme.Modul5" />
</resources>
```

MODUL5/app/build.gradle.kts

27. build.gradle.kts

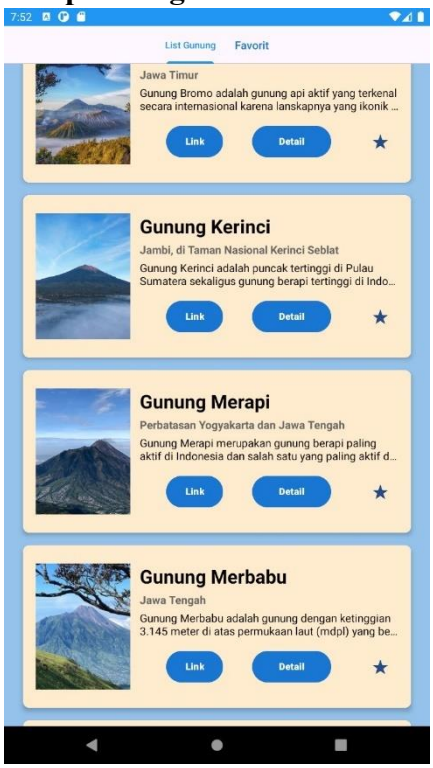
Tabel 27. Source Code build.gradle.kts

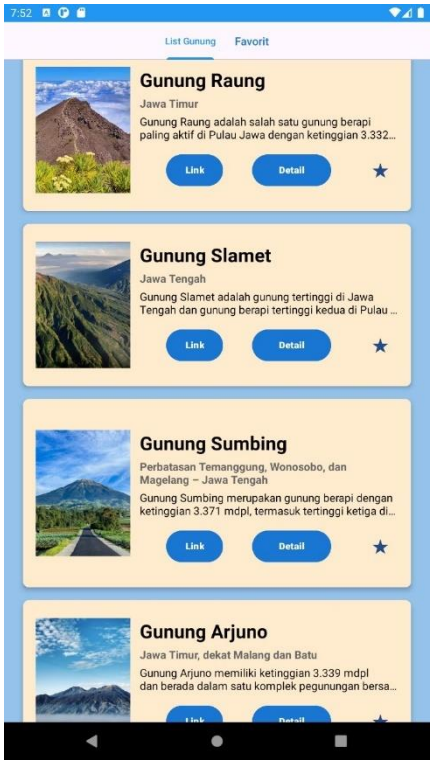
```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("kotlin-parcelize")
    id("org.jetbrains.kotlin.plugin.serialization") version
"1.9.0"
    id("org.jetbrains.kotlin.kapt")
}

android {
    namespace = "com.example.modul5"
    compileSdk = 34

    defaultConfig {
        applicationId = "com.example.modul5"
        minSdk = 30
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner =
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-
optimize.txt"),
                "proguard-rules.pro"
            )
        }
```

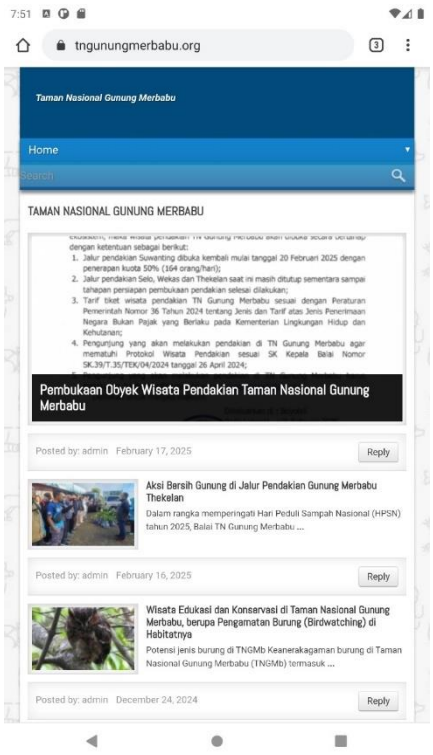
```
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_17
        targetCompatibility = JavaVersion.VERSION_17
    }

    kotlinOptions {
        jvmTarget = "17"
    }

    buildFeatures {
        viewBinding = true
    }
}

dependencies {
    implementation("androidx.core:core-ktx:1.12.0")
    implementation("org.jetbrains.kotlin:kotlin-stdlib:1.9.0")
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation ("androidx.fragment:fragment-ktx:1.6.2")
    implementation("com.google.android.material:material:1.11.0")

implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    implementation("androidx.recyclerview:recyclerview:1.3.2")
    implementation("androidx.cardview:cardview:1.0.0")
    implementation("androidx.lifecycle:lifecycle-runtime-
ktx:2.6.2")
    implementation("androidx.activity:activity-ktx:1.8.2")

    // Retrofit & KotlinX Serialization
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("org.jetbrains.kotlinx:kotlinx-serialization-
json:1.6.0")
    implementation("com.jakewharton.retrofit:retrofit2-kotlinx-
serialization-converter:0.8.0")

    // **Tambahkan OkHttp Logging Interceptor supaya
HttpLoggingInterceptor dikenali**
    implementation("com.squareup.okhttp3:logging-
interceptor:4.11.0")

    // Coil for image loading
    implementation("io.coil-kt:coil:2.5.0")

    // Lifecycle & LiveData & ViewModel
    implementation("androidx.lifecycle:lifecycle-viewmodel-
ktx:2.6.2")
    implementation("androidx.lifecycle:lifecycle-livedata-
ktx:2.6.2")

    // Room (database + coroutine support)
```

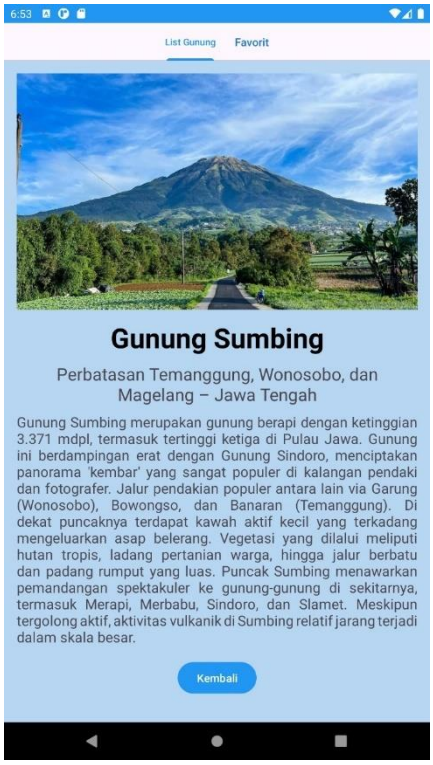
```
86      implementation("androidx.room:room-runtime:2.6.1")
87      kapt("androidx.room:room-compiler:2.6.1")
88      implementation("androidx.room:room-ktx:2.6.1")
89
90      // Kotlin Coroutines (untuk Flow)
91      implementation("org.jetbrains.kotlinx:kotlinx-coroutines-
92  core:1.7.3")
93      implementation("org.jetbrains.kotlinx:kotlinx-coroutines-
94  android:1.7.3")
95
96      // DataStore for theme/favorite preferences
97      implementation("androidx.datastore:datastore-
98  preferences:1.0.0")
99
100     testImplementation("junit:junit:4.13.2")
101     androidTestImplementation("androidx.test.ext:junit:1.1.5")
102     androidTestImplementation("androidx.test.espresso:espresso-
103 core:3.5.1")
104 }
```

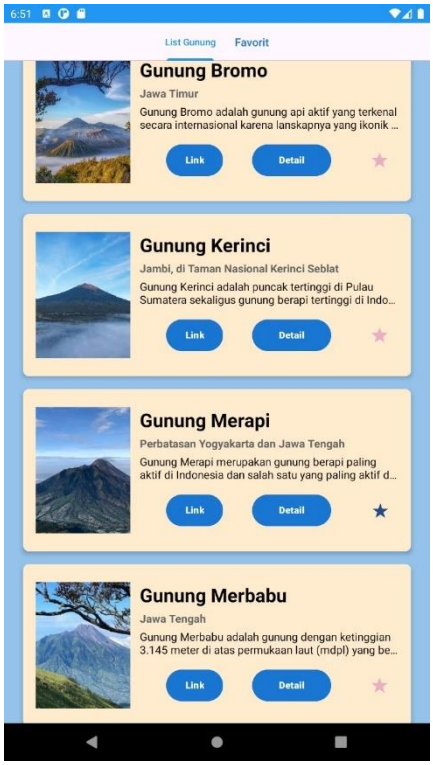
B. Output Program

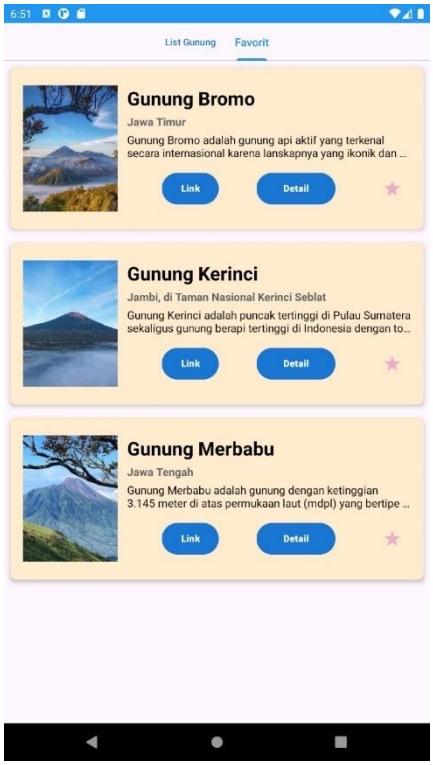


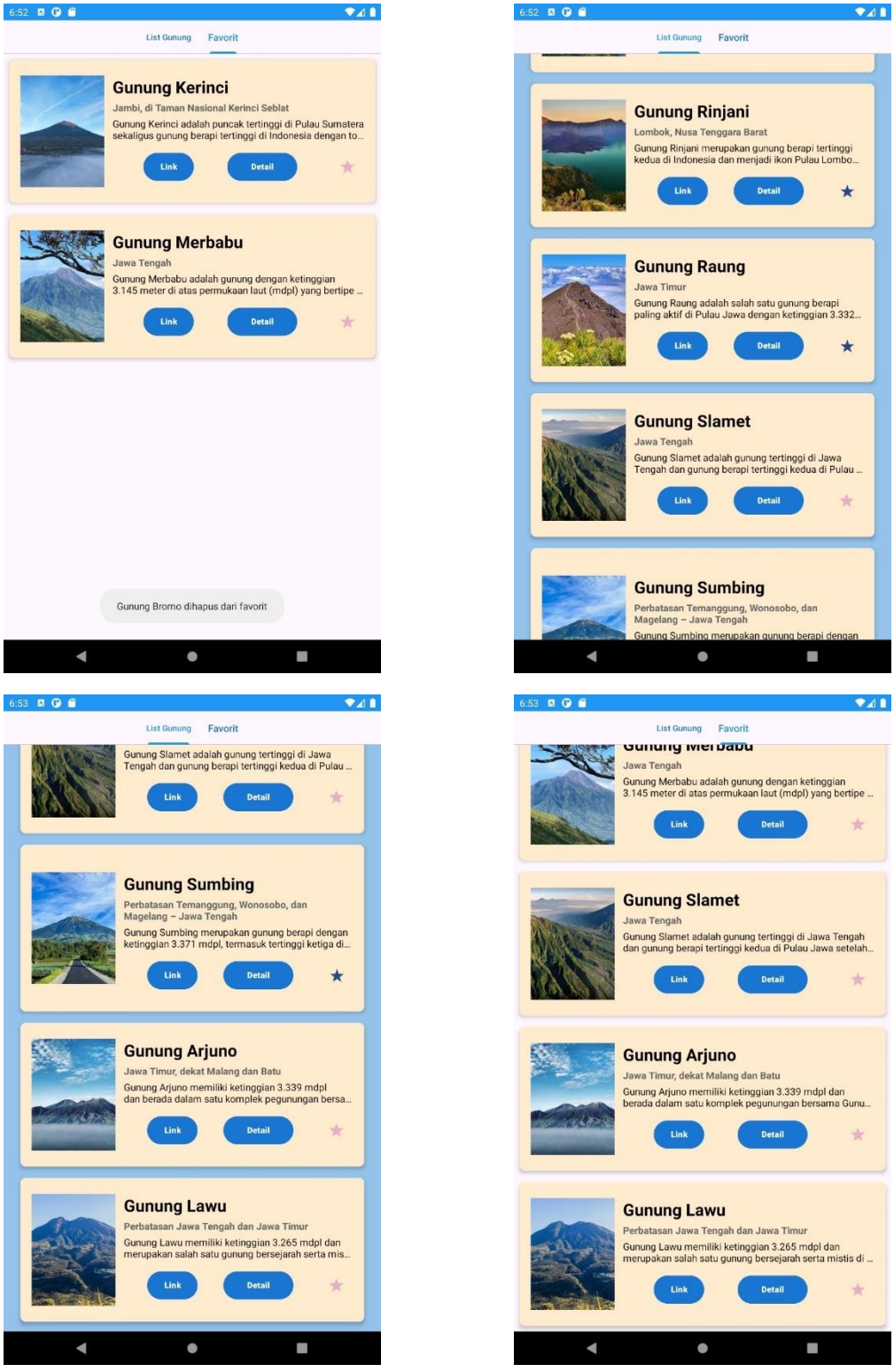



7:51
tngunungmerbabu.org
Taman Nasional Gunung Merbabu
Home
Search
TAMAN NASIONAL GUNUNG MERBABU
dengan ketentuan sebagai berikut:
1. Jalur pendakian Suwanting dibuka kembali mulai tanggal 20 Februari 2025 dengan penerapan kuota 50% (164 orang/hari);
2. Jalur pendakian Selo, Wekas dan Thekelan saat ini masih ditutup sementara sampai tahapan persiapan pembukaan pendakian selesai dilakukan;
3. Tarif tiket wisata pendakian TN Gunung Merbabu sesuai dengan Peraturan Pemerintah Nomor 36 Tahun 2024 tentang Jenis dan Tarif atas Jenis Penerimaan Negara Bukan Pajak yang Berlaku pada Kementerian Lingkungan Hidup dan Kehutanan;
4. Pengunjung yang akan melakukan pendakian di TN Gunung Merbabu agar mematuhi Protokol Wisata Pendakian sesuai SK Kepala Balai Nomor SK.39/T.35/TEK/04/2024 tanggal 26 April 2024;
Pembukaan Obyek Wisata Pendakian Taman Nasional Gunung Merbabu
Posted by: admin February 17, 2025
Reply
Aksi Bersih Gunung di Jalur Pendakian Gunung Merbabu Thekelan
Dalam rangka memperingati Hari Peduli Sampah Nasional (HPSN) tahun 2025, Balai TN Gunung Merbabu ...
Posted by: admin February 16, 2025
Reply
Wisata Edukasi dan Konservasi di Taman Nasional Gunung Merbabu, berupa Pengamatan Burung (Birdwatching) di Habitatnya
Potensi jenis burung di TNGMb Keanekagaman burung di Taman Nasional Gunung Merbabu (TNGMb) termasuk ...
Posted by: admin December 24, 2024
Reply


6:53
List Gunung
Favorit
Gunung Sumbing
Perbatasan Temanggung, Wonosobo, dan Magelang – Jawa Tengah
Gunung Sumbing merupakan gunung berapi dengan ketinggian 3.371 mdpl, termasuk tertinggi ketiga di Pulau Jawa. Gunung ini berdampingan erat dengan Gunung Sindoro, menciptakan panorama 'kembar' yang sangat populer di kalangan pendaki dan fotografer. Jalur pendakian populer antara lain via Garung (Wonosobo), Bowongso, dan Banaran (Temanggung). Di dekat puncaknya terdapat kawah aktif kecil yang terkadang mengeluarkan asap belerang. Vegetasi yang dilalui meliputi hutan tropis, ladang pertanian warga, hingga jalur berbatu dan padang rumput yang luas. Puncak Sumbing menawarkan pemandangan spektakuler ke gunung-gunung di sekitarnya, termasuk Merapi, Merbabu, Sindoro, dan Slamet. Meskipun tergolong aktif, aktivitas vulkanik di Sumbing relatif jarang terjadi dalam skala besar.
Kembali


6:51
List Gunung
Favorit
Gunung Bromo
Jawa Timur
Gunung Bromo adalah gunung api aktif yang terkenal secara internasional karena lanskapnya yang ikonik ...
Link
Detail
Gunung Kerinci
Jambi, di Taman Nasional Kerinci Seblat
Gunung Kerinci adalah puncak tertinggi di Pulau Sumatera sekaligus gunung berapi tertinggi di Indo...
Link
Detail
Gunung Merapi
Perbatasan Yogyakarta dan Jawa Tengah
Gunung Merapi merupakan gunung berapi paling aktif di Indonesia dan salah satu yang paling aktif d...
Link
Detail
Gunung Merbabu
Jawa Tengah
Gunung Merbabu adalah gunung dengan ketinggian 3.145 meter di atas permukaan laut (mdpl) yang be...
Link
Detail


6:51
List Gunung
Favorit
Gunung Bromo
Jawa Timur
Gunung Bromo adalah gunung api aktif yang terkenal secara internasional karena lanskapnya yang ikonik dan ...
Link
Detail
Gunung Kerinci
Jambi, di Taman Nasional Kerinci Seblat
Gunung Kerinci adalah puncak tertinggi di Pulau Sumatera sekaligus gunung berapi tertinggi di Indonesia dengan to...
Link
Detail
Gunung Merbabu
Jawa Tengah
Gunung Merbabu adalah gunung dengan ketinggian 3.145 meter di atas permukaan laut (mdpl) yang bertipe ...
Link
Detail



Gambar 1. Screesnshot Output Program Soal 1

C. Pembahasan

MODUL5/app/src/main/AndroidManifest.xml

1. AndroidManifest.xml

Kode XML dikelas ini adalah file AndroidManifest.xml, yaitu file penting dalam setiap proyek Android yang berfungsi untuk mendeklarasikan informasi dasar aplikasi kepada sistem Android. File dimulai dengan deklarasi XML standar <?xml version="1.0" encoding="utf-8"?>, yang menunjukkan bahwa dokumen ini ditulis dalam format XML dengan encoding UTF-8. Tag <manifest> menjadi elemen utama dan wajib di setiap file manifest. Di dalamnya, terdapat atribut package="com.example.modul5" yang mendefinisikan nama paket unik dari aplikasi ini, yang juga menjadi identitas aplikasi di Play Store maupun sistem Android. Selanjutnya, ada dua deklarasi permission (<uses-permission>) yang memberi tahu sistem bahwa aplikasi membutuhkan izin tertentu. Yang pertama adalah android.permission.ACCESS_NETWORK_STATE, yang memungkinkan aplikasi memeriksa status jaringan (apakah terhubung ke WiFi, mobile data, atau offline). Yang kedua adalah android.permission.INTERNET, yang memberi aplikasi akses untuk terhubung ke internet, penting untuk aplikasi yang mengambil data dari API atau layanan online. Di dalam tag <application>, berbagai atribut disetel untuk mengatur properti aplikasi secara umum. Atribut android:allowBackup="true" mengizinkan pengguna untuk membackup data aplikasi ke akun Google mereka.

Atribut android:dataExtractionRules dan android:fullBackupContent merujuk ke file XML eksternal yang mengatur aturan ekstraksi data dan konten yang diikutsertakan saat backup. Atribut visual seperti android:icon dan android:roundIcon menetapkan ikon aplikasi dalam bentuk standar dan bulat. android:label mengambil nilai dari strings.xml untuk memberi nama aplikasi yang akan ditampilkan ke pengguna. android:supportsRtl="true" menunjukkan bahwa aplikasi ini mendukung layout dari kanan ke kiri, penting untuk bahasa-bahasa seperti Arab atau Ibrani. android:theme menentukan tema global aplikasi yang diambil dari file style (Theme.Modul5). Di dalam application, terdapat deklarasi sebuah activity, yakni MainActivity, yang merupakan pintu masuk utama aplikasi. Atribut android:exported="true" diperlukan untuk menjelaskan apakah activity ini bisa diakses dari

luar aplikasi, dan sejak Android 12, atribut ini wajib dideklarasikan untuk komponen yang memiliki intent-filter. Di dalam tag <intent-filter>, terdapat dua elemen penting: <action android:name="android.intent.action.MAIN" /> dan <category android:name="android.intent.category.LAUNCHER" />. Kombinasi keduanya membuat MainActivity menjadi activity pertama yang dijalankan saat aplikasi dibuka dari launcher. Secara keseluruhan, file ini mendefinisikan hak akses, tampilan, aktivitas utama, serta dukungan backup untuk aplikasi Android yang bernama Modul5. Semua komponen ini saling berkaitan untuk menjamin aplikasi berjalan dengan baik sesuai standar sistem operasi Android.

com/example/modul5/data/local/entity/dao

2. GunungDao

Kode Kotlin di kelas ini mendefinisikan sebuah **interface bernama GunungDao** (Data Access Object) yang berfungsi sebagai penghubung antara aplikasi dengan database lokal yang dibangun menggunakan **Room**, yaitu library ORM (Object Relational Mapping) pada Android. Interface ini menangani segala operasi database terhadap entitas GunungEntity, yaitu representasi dari data gunung yang tersimpan di tabel gunung. Seluruh fungsi yang ada di dalam GunungDao dibuat untuk menangani pengambilan, penyisipan, penghapusan, dan pembaruan data, baik secara sinkron maupun reaktif. Interface ini diawali dengan anotasi @Dao yang menandakan bahwa interface ini adalah komponen DAO. Room akan secara otomatis menghasilkan implementasi konkret dari interface ini saat build time. Fungsi pertama adalah getAllGunung(), yang menggunakan anotasi @Query("SELECT * FROM gunung") untuk mengambil seluruh data dari tabel gunung. Fungsi ini mengembalikan data dalam bentuk Flow<List<GunungEntity>>, yang artinya data tersebut bisa di-*observe* secara real-time. Jika ada perubahan di tabel, maka data yang diterima UI juga akan otomatis terbarui.

Fungsi kedua, getAllGunungOnce(), juga mengambil seluruh data dari tabel gunung, tetapi kali ini menggunakan tipe suspend yang artinya hanya dieksekusi sekali dalam coroutine dan hasilnya dikembalikan sebagai List<GunungEntity>. Tidak seperti Flow, data ini tidak bersifat reaktif. Selanjutnya, getAllFavoriteGunung() adalah fungsi yang

mengambil data gunung yang berstatus favorit, ditandai dengan isFavorite = 1. Sama seperti sebelumnya, fungsi ini bersifat satu kali eksekusi (suspend), dan hasilnya berupa daftar objek GunungEntity favorit. Sebaliknya, getAllFavoriteGunungFlow() mengambil data favorit dengan cara yang reaktif menggunakan Flow. Ini berguna untuk UI yang ingin selalu memperbarui tampilan berdasarkan perubahan status favorit. Fungsi insertGunungList() bertugas untuk menyimpan banyak data gunung sekaligus ke dalam database. Ia menggunakan strategi konflik REPLACE, yang berarti jika ada data dengan primary key yang sama, data lama akan ditimpa dengan yang baru. Demikian pula, insertGunung() melakukan hal yang sama seperti fungsi sebelumnya, tetapi hanya untuk satu entitas GunungEntity. Fungsi clearGunung() adalah perintah untuk menghapus seluruh data dalam tabel gunung. Ini berguna saat aplikasi ingin me-reset database lokal, misalnya setelah logout atau sinkronisasi ulang data. Fungsi delete(gunung: GunungEntity) digunakan untuk menghapus satu entitas gunung tertentu. Room akan mencari berdasarkan primary key dari objek tersebut lalu menghapus datanya dari tabel.

Kemudian ada fungsi update(gunung: GunungEntity) yang akan memperbarui seluruh isi data gunung berdasarkan primary key-nya. Fungsi ini berguna saat ingin mengganti informasi seperti deskripsi atau status gunung. Terakhir, updateFavoriteStatus(name: String, isFavorite: Boolean) memungkinkan perubahan nilai kolom isFavorite secara spesifik berdasarkan nama gunung. Ini sangat efisien saat hanya ingin memperbarui status favorit tanpa harus mengubah keseluruhan entitas. Secara keseluruhan, GunungDao menyediakan fungsi-fungsi yang lengkap untuk menangani seluruh siklus data gunung di aplikasi mulai dari membaca, menyisipkan, memperbarui, hingga menghapus data. Kombinasi antara penggunaan suspend untuk operasi sekali jalan dan Flow untuk data reaktif menunjukkan praktik modern dalam pengelolaan database pada aplikasi Android berbasis Kotlin.

com/example/modul5/data/local/entity/database

3. **AppDatabase**

Kode yang ditampilkan adalah bagian penting dari implementasi database lokal pada aplikasi Android menggunakan library Room. Secara umum, kode ini membentuk kerangka

kerja utama untuk mengatur penyimpanan data dalam database SQLite dengan pendekatan yang lebih terstruktur, aman, dan mudah diakses melalui konsep object-oriented. File ini terletak dalam package com.example.modul5.data.local.database dan berisi deklarasi kelas abstrak AppDatabase yang merupakan turunan dari RoomDatabase. Kelas ini menjadi representasi langsung dari database itu sendiri dan harus ditandai dengan anotasi @Database. Di dalam anotasi ini, ditentukan entitas-entitas yang digunakan dalam database, yakni kelas-kelas data yang mewakili tabel. Dalam hal ini, hanya terdapat satu entitas, yaitu GunungEntity, yang kemungkinan besar menyimpan data tentang gunung-gunung, seperti nama, lokasi, ketinggian, dan status favorit. Selain itu, database ini diberi versi pertama (version = 1) sebagai penanda bahwa ini adalah struktur awal dari database yang dibuat.

Sementara itu, properti exportSchema diset ke false, artinya Room tidak akan mengekspor skema database ke file JSON—biasanya pengaturan ini digunakan ketika skema tidak diperlukan untuk dokumentasi atau pengujian. Kelas AppDatabase mendefinisikan satu fungsi abstrak, gunungDao(), yang mengembalikan sebuah objek GunungDao. DAO (Data Access Object) adalah antarmuka yang menyediakan berbagai metode untuk berinteraksi dengan tabel gunung dalam database, seperti membaca semua data, menambahkan, memperbarui, atau menghapus data. Dengan pendekatan ini, semua logika query disederhanakan dan dipisahkan dari logika tampilan, sehingga mendukung arsitektur aplikasi yang bersih dan terorganisir. Yang paling krusial dari kode ini adalah bagian companion object, yaitu sebuah objek statis yang memungkinkan akses ke metode dan properti tanpa harus membuat instance dari kelas AppDatabase. Di dalamnya terdapat variabel INSTANCE, yang ditandai dengan anotasi @Volatile. Anotasi ini memastikan bahwa perubahan pada variabel tersebut akan langsung terlihat oleh thread lain, sehingga menjaga konsistensi dan mencegah kondisi tidak stabil saat beberapa thread mencoba mengakses database secara bersamaan. Fungsi getDatabase(context: Context) di dalam companion object ini bertugas mengelola pembuatan dan pengembalian instance tunggal dari database. Fungsi ini menggunakan teknik *lazy initialization*, di mana instance database hanya akan dibuat saat pertama kali dibutuhkan. Mekanisme synchronized(this) digunakan agar hanya satu thread yang dapat mengeksekusi bagian ini pada satu waktu, sehingga menghindari duplikasi pembuatan database yang bisa menimbulkan error atau penggunaan sumber daya yang tidak efisien. Jika INSTANCE masih null, maka Room.databaseBuilder() akan dipanggil untuk

membuat database baru, menggunakan applicationContext dari context yang diberikan agar tidak terjadi memory leak akibat penggunaan context dari activity atau fragment.

Nama database yang digunakan di sini adalah "gunung_database", yang berarti Room akan membuat file SQLite dengan nama tersebut di penyimpanan internal aplikasi. Setelah instance database berhasil dibuat, ia disimpan ke dalam variabel INSTANCE agar dapat digunakan kembali di masa mendatang tanpa harus membuat ulang. Fungsi ini lalu mengembalikan instance tersebut kepada pemanggil. Secara keseluruhan, kode ini adalah struktur penting yang memastikan bahwa aplikasi memiliki satu akses pusat yang aman, efisien, dan terstruktur terhadap database lokal. Ia mengatur cara aplikasi menyimpan, mengambil, dan mengelola data gunung melalui GunungDao, dan menjamin bahwa interaksi dengan database dilakukan melalui satu pintu dengan menggunakan pola singleton. Pendekatan ini sangat sesuai untuk arsitektur modern berbasis MVVM atau Clean Architecture di Android.

com/example/modul5/data/local/entity/entity

4. GunungEntity.kt

Kode ini merupakan bagian dari sistem database lokal menggunakan **Room** di aplikasi Android, dan berfungsi untuk mendefinisikan struktur data atau *entity* yang akan disimpan di dalam tabel database. File ini berada dalam package com.example.modul5.data.local.entity, yang berarti fungsinya terfokus pada mendefinisikan *entitas* yang mewakili satu tabel bernama gunung dalam database lokal aplikasi. Kelas yang dideklarasikan di sini adalah GunungEntity, dan ditandai dengan anotasi @Entity(tableName = "gunung"). Anotasi ini memberi tahu Room bahwa kelas GunungEntity adalah representasi dari sebuah tabel database bernama gunung. Dengan kata lain, setiap objek GunungEntity akan merepresentasikan satu baris data di dalam tabel tersebut.

Kelas ini menggunakan data class, yang merupakan fitur Kotlin untuk membuat kelas yang berfungsi menyimpan data dengan boilerplate code yang minimal. Di dalamnya terdapat beberapa properti atau kolom, yaitu:

- @PrimaryKey val name: String: Kolom name berfungsi sebagai *primary key*, yang berarti nilainya harus unik dan digunakan Room untuk mengidentifikasi tiap entri

secara individual. Dalam konteks ini, name kemungkinan merupakan nama gunung dan digunakan sebagai identitas utama tiap data.

- val lokasi: String: Kolom lokasi menyimpan informasi lokasi gunung, seperti nama provinsi, kabupaten, atau area geografis tempat gunung berada.
- val deskripsi: String: Kolom ini menyimpan deskripsi tentang gunung, yang bisa berupa informasi sejarah, karakteristik, keindahan, atau daya tariknya.
- val link: String: Merupakan kolom yang berisi tautan, kemungkinan ke sumber informasi lebih lanjut seperti artikel, blog, atau situs resmi yang memuat informasi mendetail tentang gunung tersebut.
- val image: String: Kolom ini menyimpan URL atau path gambar gunung, yang akan digunakan oleh aplikasi untuk menampilkan visualisasi gunung di antarmuka pengguna.
- val isFavorite: Boolean: Kolom ini berfungsi sebagai penanda apakah gunung tersebut termasuk dalam daftar favorit pengguna atau tidak. Nilainya berupa true atau false, dan bisa digunakan untuk fitur filter atau penyimpanan personalisasi data.

Dengan struktur ini, Room dapat secara otomatis menghasilkan kode SQL untuk membuat tabel gunung dan mengelola datanya tanpa harus menulis query secara manual dalam setiap operasi. Setiap kali aplikasi menyimpan atau mengambil data dari tabel gunung, ia akan berurusan dengan objek GunungEntity. Pendekatan ini mendukung konsep *type-safe*, yang meminimalkan kesalahan saat berinteraksi dengan database karena semua struktur dan tipe data sudah didefinisikan secara eksplisit di kelas ini. Kode ini juga merupakan bagian penting dalam arsitektur berbasis MVVM (Model-View-ViewModel) karena menjadi lapisan Model yang menyimpan dan menyediakan data secara terstruktur dan konsisten ke seluruh bagian aplikasi.

com/example/modul5/data/mapper

5. GunungMapper.kt

Kode ini berada di dalam package com.example.modul5.data.mapper, yang menunjukkan bahwa fungsinya adalah sebagai *mapper*, atau pengubah data, antara dua

lapisan data berbeda dalam arsitektur aplikasi: yaitu model data dari sumber eksternal (Gunung) dan entitas database lokal (GunungEntity). Mapper ini sangat penting untuk menjaga pemisahan tanggung jawab antar lapisan, terutama dalam arsitektur MVVM atau Clean Architecture, di mana data dari dan ke database perlu dipisahkan dari data yang digunakan di layer domain atau UI. Di dalam file ini terdapat dua fungsi ekstensi Kotlin, yang memungkinkan penulisan metode seolah-olah menjadi bagian dari kelas aslinya, meskipun dideklarasikan secara eksternal. Fungsi pertama adalah fun Gunung.toEntity(): GunungEntity. Fungsi ini adalah ekstensi dari kelas Gunung, yang berarti fungsi ini hanya bisa dipanggil dari objek bertipe Gunung. Tujuannya adalah untuk mengonversi objek Gunung menjadi objek GunungEntity.

Fungsi ini digunakan ketika data dari sumber eksternal (seperti API atau sumber data lainnya) ingin disimpan ke dalam database lokal Room. Semua properti seperti name, lokasi, deskripsi, link, image, dan isFavorite dipetakan secara langsung dari objek Gunung ke GunungEntity. Hal ini memastikan bahwa data dapat disimpan di tabel database gunung dengan struktur yang sesuai. Fungsi kedua adalah fun GunungEntity.toModel(): Gunung. Kebalikan dari fungsi sebelumnya, fungsi ini merupakan ekstensi dari kelas GunungEntity, dan digunakan untuk mengonversi data dari entitas database menjadi model Gunung yang bisa digunakan di domain aplikasi atau ditampilkan di UI. Proses konversi ini penting karena entitas Room biasanya digunakan hanya di lapisan data atau repository, dan tidak langsung digunakan oleh UI. Dengan fungsi ini, data yang sebelumnya tersimpan dalam database dapat diubah ke bentuk yang lebih netral dan sesuai kebutuhan tampilan aplikasi.

Kedua fungsi ini, meskipun tampak sederhana, memiliki peran penting dalam menjaga fleksibilitas dan skalabilitas aplikasi. Dengan memisahkan struktur data yang digunakan untuk menyimpan (entitas) dan struktur data yang digunakan untuk berinteraksi dengan UI atau logika aplikasi (model), maka perubahan pada satu sisi tidak langsung memengaruhi sisi lainnya. Misalnya, jika struktur database perlu diperbarui atau jika model Gunung ingin ditambahkan atribut baru khusus untuk tampilan, perubahan tersebut bisa dilakukan tanpa mengganggu keseluruhan sistem, cukup dengan menyesuaikan fungsi mapper ini. Dengan begitu, kode menjadi lebih modular, mudah diuji, dan lebih tahan terhadap perubahan.

com/example/modul5/data/model

6. Gunung

Kode ini berada di dalam package com.example.modul5.data.model, dan mendefinisikan sebuah *data class* bernama Gunung yang merupakan representasi dari model data utama dalam aplikasi. Kelas ini menampung informasi detail tentang gunung, dan digunakan untuk menyimpan serta memproses data dari sumber eksternal seperti API sebelum ditampilkan ke pengguna atau diolah lebih lanjut dalam aplikasi. Kata kunci @Serializable yang ada di atas kelas menunjukkan bahwa Gunung adalah kelas yang dapat diserialisasi menggunakan pustaka kotlinx.serialization. Artinya, objek dari kelas ini dapat diubah menjadi format lain seperti JSON dan sebaliknya. Ini sangat penting dalam komunikasi client-server, karena data dari API umumnya datang dalam format JSON dan harus dikonversi ke objek Kotlin untuk dapat digunakan dalam kode aplikasi. Setiap properti di dalam kelas Gunung diberi anotasi @SerialName, yang menunjukkan nama sebenarnya dari properti tersebut seperti yang muncul di JSON dari API. Ini memungkinkan Kotlin untuk memetakan field JSON ke nama properti Kotlin yang digunakan di aplikasi. Misalnya:

- @SerialName("name") val name: String menunjukkan bahwa JSON memiliki field "name" yang akan dimasukkan ke properti name.
- @SerialName("lokasi") val lokasi: String memetakan field "lokasi" dari JSON ke properti lokasi.
- @SerialName("deskripsi") val deskripsi: String untuk isi deskripsi gunung.
- @SerialName("link") val link: String yang biasanya merujuk ke tautan sumber atau informasi tambahan.
- @SerialName("image_url") val image: String menunjukkan bahwa data gambar gunung dikirim melalui field JSON bernama "image_url", dan disimpan di properti image.

Satu properti, yaitu isFavorite, tidak memiliki anotasi @SerialName, karena diasumsikan bahwa field ini tidak berasal dari JSON API, melainkan properti tambahan yang digunakan di dalam aplikasi itu sendiri, misalnya untuk menandai gunung favorit. Nilai default dari

properti ini adalah false, sehingga ketika objek Gunung dibuat dari JSON dan field isFavorite tidak ditemukan, nilainya otomatis akan false. Di akhir deklarasi kelas, terdapat pewarisan dari JavaSerializable (java.io.Serializable yang diberi alias agar tidak konflik dengan anotasi KotlinX). Ini berguna ketika objek Gunung perlu dipindahkan antar komponen Android, seperti dari satu fragment ke fragment lain melalui Bundle atau Intent, karena Android memerlukan objek yang bisa diserialisasi dalam proses itu. Secara keseluruhan, kelas Gunung ini merupakan inti dari model data aplikasi, yang merepresentasikan struktur data gunung dari API, mendukung konversi JSON, dan juga siap digunakan dalam konteks navigasi Android. Pendekatan ini menjadikan data model fleksibel untuk kebutuhan backend (API), frontend (UI), maupun penyimpanan sementara (seperti pada Bundle).

com/example/modul5/data/remote

7. **ApiClient**

Kode di atas berada dalam package com.example.modul5.data.remote dan berfungsi sebagai konfigurasi client Retrofit untuk komunikasi dengan API eksternal. Seluruh konfigurasi ini dibungkus dalam sebuah object Kotlin bernama ApiClient, yang berarti kelas ini bersifat singleton—hanya akan dibuat satu instance selama aplikasi berjalan. Tujuan utama dari ApiClient adalah untuk mengatur bagaimana aplikasi akan melakukan request HTTP ke server, dalam hal ini ke alamat https://modul5.free.beeceptor.com/api/path/. Bagian pertama adalah konstanta BASE_URL, yaitu URL dasar yang akan menjadi prefix dari semua endpoint yang digunakan untuk request data gunung dari API. Selanjutnya, disiapkan konfigurasi Json dari kotlinx.serialization.json. Di sini properti ignoreUnknownKeys = true digunakan agar saat melakukan deserialisasi dari JSON ke objek Kotlin, field-field yang tidak dikenali oleh data class tidak menyebabkan error. Hal ini sangat berguna ketika struktur JSON dari API memiliki elemen tambahan yang tidak digunakan oleh aplikasi.

Kemudian ada loggingInterceptor, sebuah instance dari HttpLoggingInterceptor dari pustaka OkHttp, yang diatur pada level BODY. Ini artinya semua detail dari request dan response, termasuk header dan body, akan dicetak ke log. Logging ini sangat membantu saat proses debugging atau pengujian karena developer bisa melihat isi sebenarnya dari

komunikasi HTTP yang sedang berlangsung. Setelah itu, okHttpClient dikonfigurasi menggunakan builder pattern. Di dalamnya, loggingInterceptor ditambahkan agar setiap request dan response bisa tercatat. Timeout untuk koneksi dan pembacaan data juga diatur selama 30 detik agar koneksi tidak langsung gagal saat jaringan lambat. Selanjutnya dibuat instance retrofit menggunakan builder milik Retrofit. Di sini ditetapkan baseUrl dan client yang telah dikonfigurasi sebelumnya. Selain itu, konversi dari JSON ke objek Kotlin dilakukan dengan addConverterFactory, yang memanfaatkan pustaka kotlinx.serialization dan diubah menjadi converter Retrofit menggunakan asConverterFactory. Format konten yang digunakan untuk konversi adalah application/json, yang merupakan standar umum untuk komunikasi data di web API. Di bagian akhir, gunungApiService dideklarasikan sebagai objek service Retrofit yang dibuat dari interface GunungApiService. Interface ini akan berisi definisi endpoint dan metode-metode HTTP seperti GET, POST, atau lainnya yang digunakan untuk mengambil data gunung dari API. Secara keseluruhan, ApiClient ini merupakan komponen utama yang memungkinkan aplikasi untuk berkomunikasi dengan API secara efisien dan aman. Ia memanfaatkan Retrofit dan OkHttp sebagai alat komunikasi, serta kotlinx.serialization sebagai alat konversi data. Kombinasi ini menjadikan aplikasi tidak hanya modular dan mudah di-maintain, tapi juga cukup fleksibel untuk berinteraksi dengan berbagai format data dari server.

8. GunungApiService

Kode di atas merupakan sebuah interface bernama GunungApiService yang berada di dalam package com.example.modul5.data.remote. Interface ini digunakan sebagai kontrak API untuk komunikasi antara aplikasi dan server eksternal menggunakan Retrofit, sebuah library populer di Android untuk melakukan HTTP request secara efisien. Pada dasarnya, interface ini mendefinisikan endpoint yang dapat dipanggil oleh aplikasi untuk mendapatkan data gunung dari server. Di dalam interface ini terdapat satu fungsi, yaitu getGunungList(). Fungsi getGunungList() diberi anotasi @GET("gunung"), yang berarti saat fungsi ini dipanggil, Retrofit akan melakukan permintaan HTTP GET ke endpoint yang bernama "gunung", yang merupakan bagian dari URL lengkap yang sudah didefinisikan sebelumnya dalam ApiClient melalui properti BASE_URL. Jadi, URL lengkapnya menjadi https://modul5.free.beeceptor.com/api/path/gunung.

Fungsi ini bersifat suspend, yang artinya hanya bisa dipanggil dari dalam coroutine atau fungsi lain yang juga bersifat suspend. Ini memungkinkan fungsi untuk berjalan secara asynchronous (non-blocking), yang penting dalam pengembangan aplikasi Android agar tidak membekukan UI saat mengambil data dari internet. Hasil dari fungsi ini adalah List<Gunung>, yakni daftar objek Gunung, yang merupakan data class yang telah dibuat sebelumnya dan merepresentasikan model gunung dengan informasi seperti nama, lokasi, deskripsi, link, gambar, dan status favorit. Dengan kata lain, GunungApiService ini adalah penghubung antara aplikasi dan API eksternal. Ia mendeskripsikan bagaimana aplikasi dapat meminta data dari internet (dalam hal ini, daftar gunung), dan Retrofit akan secara otomatis menangani proses permintaan HTTP, parsing JSON ke objek Kotlin (Gunung), serta pengelolaan threading melalui coroutine. Struktur seperti ini membuat kode menjadi bersih, mudah diuji, dan sangat terintegrasi dengan arsitektur modern Android.

com/example/modul5/data/repository

9. GunungRepository

Kode tersebut merupakan implementasi dari sebuah *repository class* bernama GunungRepository, yang memiliki peran penting dalam arsitektur aplikasi modern seperti MVVM (Model-View-ViewModel). Kelas ini mengatur alur data dari dua sumber utama—API (melalui internet) dan database lokal (melalui Room)—dan menyajikannya ke ViewModel secara bersih dan terstruktur. Kelas ini juga bertanggung jawab mengelola transformasi data dan sinkronisasi antara dua sumber data tersebut. Pada bagian atas kelas, terdapat inisialisasi dua properti penting: apiService dan gunungDao. apiService merupakan instance dari GunungApiService yang didapat dari singleton ApiClient, dan digunakan untuk mengambil data dari internet. Sedangkan gunungDao diperoleh dari AppDatabase dengan memanggil fungsi getDatabase(context) dan mengambil gunungDao()-nya, yang berfungsi untuk akses ke database lokal. Salah satu fungsi utama dalam repository ini adalah getFavoriteGunungListFlow(). Fungsi ini mengembalikan data daftar gunung favorit dalam bentuk *Flow*, yang mendukung pengamatan data secara reaktif.

Data yang diambil dari database berupa List<GunungEntity>, lalu diubah menjadi List<Gunung> dengan menggunakan map dan ekstensi toModel() dari package mapper. Ini memungkinkan UI untuk otomatis menerima pembaruan jika data gunung favorit berubah di

database. Fungsi yang paling kompleks di dalam kelas ini adalah fetchGunungList(). Fungsi ini bekerja dalam coroutine dengan konteks Dispatchers.IO untuk operasi I/O, yang artinya dijalankan di thread terpisah agar tidak mengganggu thread utama. Di dalamnya, pertama-tama diambil data gunung dari server melalui apiService.getGunungList(). Setelah itu, daftar nama gunung favorit dari database lokal diambil untuk melakukan pencocokan. Daftar gunung dari API kemudian di-*merge*—jika nama gunung cocok dengan salah satu nama favorit, maka nilai isFavorite akan disetel ke true. Ini memastikan data favorit tetap sinkron setelah mengambil data baru dari server. Setelah penggabungan selesai, database lokal dibersihkan (clearGunung()), lalu data baru yang telah digabung diubah menjadi entitas dan disimpan kembali (insertGunungList()). Jika terjadi error saat pengambilan data dari internet (misalnya karena tidak ada koneksi), fungsi akan mengambil data dari database lokal sebagai *fallback*, dan mengonversinya ke model Gunung. Selain itu, terdapat tiga fungsi lainnya: updateGunung(), insertFavoriteGunung(), dan deleteFavoriteGunung(). Ketiganya digunakan untuk memperbarui data gunung tertentu di database.

updateGunung() digunakan untuk memperbarui seluruh entitas gunung, sedangkan dua fungsi lainnya hanya memodifikasi status favorit sebuah gunung berdasarkan nama—menandainya sebagai favorit (true) atau menghapusnya dari favorit (false). Secara keseluruhan, GunungRepository ini menjadi lapisan penghubung antara ViewModel dengan sumber data, menyatukan logika akses data dari dua arah (API dan database lokal), serta mengatur bagaimana data diolah dan disajikan agar tetap konsisten dan efisien. Pendekatan ini membuat kode aplikasi lebih mudah dirawat, diuji, dan dikembangkan secara modular.

10. RepositoryInstance

Kode di atas mendefinisikan sebuah *singleton object* bernama RepositoryInstance yang digunakan untuk menyediakan instance tunggal dari kelas GunungRepository. Tujuan utama dari pendekatan ini adalah untuk menjamin bahwa hanya satu instance GunungRepository yang digunakan sepanjang siklus hidup aplikasi, sehingga efisiensi memori terjaga dan konsistensi data tetap stabil. Di dalam RepositoryInstance, terdapat sebuah properti repository bertipe nullable GunungRepository, yang diset null secara default. Properti ini menjadi tempat penyimpanan instance repository yang sesungguhnya. Fungsi inti

dari objek ini adalah provideRepository(context: Context). Fungsi ini akan mengembalikan instance GunungRepository yang aktif. Namun sebelum memberikan instance tersebut, fungsi akan memeriksa apakah repository sudah pernah dibuat sebelumnya.

Jika belum (masih null), maka blok synchronized(this) akan dijalankan untuk memastikan bahwa hanya satu thread yang dapat membuat instance pada satu waktu—mencegah kondisi race atau *double instantiation* dalam lingkungan multi-thread. Di dalam blok synchronized, objek GunungRepository dibuat dengan menyuplai applicationContext dari konteks yang diterima sebagai argumen. Penggunaan applicationContext di sini sangat penting untuk mencegah potensi *memory leak*, karena applicationContext memiliki siklus hidup sepanjang aplikasi berjalan, bukan hanya sepanjang siklus hidup activity atau fragment. Setelah instance berhasil dibuat, ia disimpan dalam variabel repository agar bisa digunakan kembali saat fungsi provideRepository() dipanggil lagi di kemudian hari. Fungsi lalu mengembalikan instance tersebut, baik itu instance baru maupun instance yang sudah ada sebelumnya. Dengan pendekatan seperti ini, RepositoryInstance memberikan cara yang aman, efisien, dan terpusat dalam menyediakan akses ke GunungRepository, yang sangat berguna terutama dalam arsitektur MVVM, di mana ViewModel membutuhkan sumber data yang konsisten dari satu repository utama. Pendekatan ini juga sangat cocok diterapkan dalam aplikasi yang mengandalkan dependency injection manual tanpa menggunakan framework seperti Hilt atau Dagger.

com/example/modul5/ui

11. FavoriteGunungFragment.kt

Kode di atas mendefinisikan kelas FavoritGunungFragment, yang merupakan salah satu bagian dari antarmuka pengguna dalam aplikasi Android berbasis arsitektur MVVM. Fragment ini bertugas menampilkan daftar gunung yang telah ditandai sebagai favorit oleh pengguna. Ia memanfaatkan ViewModel, RecyclerView, dan ViewBinding untuk mengelola data dan tampilan secara efisien. Dalam deklarasi kelas, terdapat properti _binding yang merupakan instance dari FragmentFavoritGunungBinding, yaitu binding yang dibuat otomatis berdasarkan file XML fragment_favorit_gunung.xml. Binding ini digunakan untuk

mengakses elemen UI tanpa harus memanggil findViewById, yang membuat kode lebih ringkas dan aman dari kesalahan penamaan. Binding hanya aktif selama fragment memiliki view, dan diatur menjadi null kembali di onDestroyView() untuk mencegah memory leak. Fragment ini menggunakan pola by viewModels untuk menginisialisasi GunungViewModel.

ViewModel disediakan melalui GunungViewModelFactory, yang mengambil instance repository dari RepositoryInstance.provideRepository() dengan menggunakan requireContext() untuk mendapatkan context aplikasi. Ini memastikan bahwa ViewModel memiliki akses ke sumber data yang konsisten. Pada onCreateView, layout fragment di-*inflate* dan view root-nya dikembalikan. Lalu di onViewCreated, adapter dari RecyclerView diatur menggunakan kelas GunungAdapter. Adapter ini menerima tiga parameter fungsi lambda: satu untuk membuka link gunung di browser, satu untuk menampilkan detail gunung di fragment baru, dan satu lagi untuk mengubah status favorit gunung. Setiap aksi direspon dengan membuat intent (untuk membuka link), memanggil FragmentTransaction (untuk navigasi ke detail), atau memperbarui data melalui viewModel.updateGunung(). RecyclerView disiapkan dengan LinearLayoutManager agar item ditampilkan dalam daftar vertikal, dan adapter yang telah dikonfigurasi ditetapkan sebagai adapter-nya. Untuk memuat daftar favorit secara dinamis, fragment menggunakan lifecycleScope.launch yang menjalankan coroutine untuk mengoleksi aliran data favoriteList dari ViewModel.

Setiap kali data favorit berubah, adapter akan diperbarui dengan data terbaru melalui submitList(). Terakhir, onDestroyView bertugas untuk menghapus referensi ke binding ketika fragment dihancurkan, menjaga manajemen memori tetap efisien dan mencegah kebocoran memori karena view yang tidak lagi aktif tetap tertahan di memori. Keseluruhan implementasi ini menunjukkan pendekatan yang bersih dan terstruktur untuk menampilkan daftar data yang reaktif di Android, sambil menjaga keterpisahan logika tampilan dan logika data.

12. GunungAdapter

Kode di atas mendefinisikan GunungAdapter, yaitu kelas adapter untuk RecyclerView yang digunakan dalam aplikasi Android. Adapter ini bertugas mengatur bagaimana data gunung ditampilkan dalam bentuk daftar di antarmuka pengguna. Kelas ini

mewarisi dari ListAdapter<Gunung, GunungAdapter.ListViewHolder>, yang membuatnya secara otomatis mampu mengelola perubahan data secara efisien melalui DiffUtil.

Konstruktor dari GunungAdapter menerima tiga parameter fungsi lambda:

- onLinkClick: dipanggil saat tombol link ditekan (biasanya membuka URL ke browser).
- onDetailClick: dipanggil saat tombol detail ditekan (navigasi ke halaman detail gunung).
- onFavoriteClick: dipanggil saat tombol favorit ditekan (menandai/meniadakan gunung sebagai favorit).

Di dalam kelas GunungAdapter, terdapat inner class ListViewHolder yang bertugas mengatur tampilan satu item gunung menggunakan ItemGunungBinding, yaitu binding otomatis dari layout XML item_gunung.xml. Fungsi bind() menerima sebuah objek Gunung dan mengatur tampilan setiap elemen UI berdasarkan data tersebut:

- Nama, lokasi, dan deskripsi gunung diatur ke TextView.
- Gambar gunung dimuat menggunakan library **Coil** (binding.imgGunung.load(...)), yang mengunduh gambar dari URL secara efisien.
- Ikon favorit ditentukan berdasarkan properti isFavorite. Jika bernilai true, maka ikon bintang penuh ditampilkan, jika false, ikon bintang kosong ditampilkan.
- Tiga tombol memiliki listener masing-masing yang memicu fungsi callback dari konstruktor.

Metode onCreateViewHolder digunakan untuk *inflate* layout item_gunung.xml menjadi objek ViewHolder. Sementara itu, onBindViewHolder digunakan untuk mengikat data gunung pada posisi tertentu ke holder yang sesuai. Di bagian companion object, terdapat DIFF_CALLBACK, yang merupakan implementasi dari DiffUtil.ItemCallback<Gunung>. Ini memungkinkan adapter membandingkan item secara otomatis dan hanya memperbarui tampilan yang berubah, bukan seluruh daftar.

- Fungsi areItemsTheSame() membandingkan identitas objek berdasarkan nama gunung (yang diasumsikan unik).
- Fungsi areContentsTheSame() membandingkan seluruh isi objek (menggunakan ==, berarti semua properti sama persis).

Dengan pendekatan ini, GunungAdapter memungkinkan RecyclerView untuk tampil optimal, efisien, dan responsif terhadap perubahan data, sekaligus tetap menjaga pemisahan logika tampilan dan logika interaksi. Penggunaan ViewBinding juga mengurangi kemungkinan error dan membuat kode lebih bersih.

13. GunungDetailFragment

Kode di atas merupakan implementasi dari GunungDetailFragment, yaitu sebuah fragment dalam aplikasi Android yang digunakan untuk menampilkan informasi detail mengenai sebuah gunung. Fragment ini menggunakan ViewBinding melalui kelas DetailFragmentBinding untuk menghubungkan elemen-elemen UI di layout XML dengan logika program. Pada metode onCreateView, layout di-*inflate* dan disimpan ke dalam variabel _binding, yang kemudian diakses melalui properti binding. Dalam proses ini juga dilakukan pengambilan data dari argument yang dikirim oleh fragment sebelumnya, yaitu nama, lokasi, deskripsi, dan URL gambar dari gunung yang dipilih. Data ini diambil dengan memanggil arguments?.getString() untuk setiap key yang relevan, disertai nilai default jika datanya tidak tersedia agar tampilan tetap informatif. Setelah data berhasil diambil, isi dari variabel tersebut langsung ditampilkan ke elemen UI yang sesuai.

Nama, lokasi, dan deskripsi ditampilkan pada TextView, sementara gambar dimuat ke dalam ImageView menggunakan library Coil. Coil memungkinkan pemuatan gambar dari URL dengan fitur transisi lembut menggunakan crossfade, serta mendukung penanganan kondisi saat gambar sedang dimuat (placeholder) maupun saat terjadi kegagalan memuat (error). Selain itu, fragment ini juga memiliki tombol kembali (btnBack) yang ketika ditekan akan memanggil parentFragmentManager.popBackStack() untuk kembali ke tampilan sebelumnya, serta menyembunyikan kontainer fragment fragmentContainerGunung agar tidak tetap tampil di latar belakang. Fragment ini dirancang untuk memberikan pengalaman pengguna yang lebih informatif dan interaktif, terutama ketika pengguna ingin melihat detail

dari sebuah item gunung yang sebelumnya hanya ditampilkan dalam bentuk daftar. Dengan pendekatan berbasis argument dan binding ini, kode menjadi lebih terstruktur, mudah dibaca, dan lebih aman dari kesalahan umum seperti NullPointerException. Fragment ini juga menunjukkan penggunaan prinsip navigasi fragment yang baik di Android, di mana komunikasi antar-fragment dilakukan dengan cara yang eksplisit dan terkontrol.

14. GunungListFragment

Kode di atas merupakan implementasi dari GunungListFragment, yaitu fragment utama yang menampilkan daftar gunung dalam aplikasi Android berbasis arsitektur MVVM (Model-View-ViewModel). Fragment ini menggunakan ViewBinding melalui ListFragmentBinding untuk memudahkan manipulasi elemen UI tanpa perlu menggunakan findViewById. Pada siklus hidup onCreate, fragment menginisialisasi GunungViewModel melalui ViewModelProvider, dengan menyuplai instance GunungRepository dan GunungViewModelFactory. Repository ini menjadi perantara antara sumber data (seperti database atau API) dan ViewModel yang mengelola data gunung. Selanjutnya, pada onViewCreated, fragment menginisialisasi GunungAdapter, yaitu adapter untuk RecyclerView yang menampilkan daftar item gunung. Adapter ini dikonfigurasi dengan tiga aksi interaktif: membuka tautan gunung di browser eksternal saat tombol link ditekan, memicu tampilan detail ketika tombol detail ditekan, dan mengubah status favorit saat ikon bintang ditekan.

Ketika status favorit diubah, objek Gunung disalin menggunakan fungsi copy dan nilainya dibalik, kemudian disalurkan kembali ke ViewModel agar data dapat diperbarui secara reaktif. RecyclerView disusun menggunakan LinearLayoutManager agar daftar ditampilkan secara vertikal. Untuk mengamati data secara real-time, digunakan lifecycleScope.launchWhenStarted yang mengamati tiga aliran data (StateFlow) dari ViewModel: pertama, aliran gunungList untuk memperbarui daftar gunung di adapter; kedua, aliran errorMessage untuk menampilkan pesan kesalahan melalui Toast; dan ketiga, aliran selectedGunung yang merespons pemilihan gunung untuk ditampilkan dalam GunungDetailFragment. Ketika sebuah item gunung dipilih, fragment detail dibuat dan dikirimkan data melalui Bundle sebagai argument, yang kemudian ditransaksikan ke dalam fragmentContainerGunung menggunakan FragmentManager.

Kontainer fragment tersebut juga diatur agar terlihat (VISIBLE) agar tampilan detail dapat ditampilkan secara overlay atau berdampingan dengan daftar. Akhirnya, dalam onDestroyView, binding dihapus untuk mencegah memory leak. Secara keseluruhan, fragment ini mencerminkan pola desain yang bersih dan terstruktur dalam pengembangan aplikasi Android modern. Ia memanfaatkan ViewModel untuk pemisahan logika bisnis dan UI, Coroutine untuk observasi asinkron yang aman terhadap lifecycle, serta fragment dinamis untuk navigasi yang fleksibel antar-tampilan.

15. ViewPagerAdapter

Kode di atas merupakan implementasi dari kelas ViewPagerAdapter, yang digunakan untuk mengelola tampilan halaman (page) pada komponen ViewPager2 di aplikasi Android. Adapter ini merupakan turunan dari FragmentStateAdapter, dan menerima FragmentActivity sebagai parameter utama untuk mengelola siklus hidup fragment yang ditampilkan dalam setiap halaman. Di dalam kelas ini, terdapat dua daftar utama: fragments dan fragmentTitles. Daftar fragments menyimpan dua fragment berbeda, yaitu GunungListFragment yang berfungsi untuk menampilkan daftar semua gunung, serta FavoritGunungFragment yang menampilkan daftar gunung yang telah ditandai sebagai favorit oleh pengguna. Sementara itu, fragmentTitles menyimpan judul masing-masing halaman sesuai urutan fragment, yang nantinya dapat digunakan untuk menampilkan tab dengan label yang sesuai. Metode getItemCount() mengembalikan jumlah total halaman atau fragment yang tersedia, yaitu dua dalam kasus ini.

Sedangkan createFragment(position: Int) bertugas mengembalikan instance fragment berdasarkan posisi yang dipilih oleh pengguna di tampilan tab atau swipe. Selain itu, terdapat fungsi tambahan getPageTitle(position: Int) yang mengembalikan string judul halaman berdasarkan indeksnya. Fungsi ini bisa sangat berguna jika adapter dikombinasikan dengan TabLayout yang menampilkan nama tab berdasarkan posisi. Secara keseluruhan, ViewPagerAdapter ini memberikan cara yang bersih dan terstruktur untuk menyusun tampilan dua halaman dengan ViewPager2, memisahkan antara daftar gunung umum dan daftar favorit, sekaligus memudahkan navigasi pengguna melalui tab atau geser halaman. Adapter ini mendukung modularitas dan pengalaman pengguna yang lebih baik dalam menjelajahi konten yang berbeda namun berkaitan di dalam satu layar.

com/example/modul5/viewmodel

16. GunungViewModel

Kode di atas merupakan implementasi dari kelas GunungViewModel, yang berperan sebagai lapisan logika bisnis dan penghubung antara UI dan data dalam arsitektur MVVM (Model-View-ViewModel) pada aplikasi Android. ViewModel ini bertugas mengelola dan menyimpan data yang berkaitan dengan daftar gunung dan status favoritnya, sekaligus menangani aksi-aksi pengguna seperti memilih gunung atau menandai gunung sebagai favorit. ViewModel ini menerima parameter berupa GunungRepository, yang bertanggung jawab terhadap interaksi data baik dari sumber lokal maupun jarak jauh. Di dalamnya terdapat beberapa StateFlow, yaitu struktur data reaktif yang memungkinkan UI mengamati perubahan secara efisien. _gunungList merupakan MutableStateFlow privat yang menyimpan daftar semua gunung, lalu dipublikasikan sebagai gunungList untuk dapat diamati dari luar. favoriteList adalah StateFlow yang langsung mengambil data dari repository, menampilkan hanya gunung-gunung yang ditandai sebagai favorit, dan dikelola menggunakan stateIn() agar tetap aktif selama ada subscriber.

Selain itu, _selectedGunung menyimpan objek gunung yang sedang dipilih oleh pengguna, sementara _errorMessage menyimpan pesan kesalahan jika terjadi masalah saat pengambilan atau pemrosesan data. Ketika ViewModel diinisialisasi, fungsi fetchGunungList() langsung dijalankan untuk mengambil data gunung dari repository. Fungsi ini dijalankan dalam viewModelScope agar aman terhadap siklus hidup dan tidak menyebabkan kebocoran memori. Jika pengambilan data berhasil, maka daftar gunung disimpan dalam _gunungList. Jika gagal, pesan kesalahan akan ditampilkan melalui _errorMessage. Fungsi selectGunung() digunakan saat pengguna memilih salah satu gunung, yang kemudian disimpan ke dalam selectedGunung, dan dicatat ke log untuk keperluan debugging. Fungsi updateGunung() berfungsi untuk menambah atau menghapus gunung dari daftar favorit berdasarkan statusnya. Setelah melakukan operasi penyimpanan ke repository, daftar gunung juga diperbarui agar perubahan langsung tercermin di UI.

Bila terjadi kesalahan saat menyimpan data, maka akan ditangkap dan dilaporkan ke errorMessage. Terakhir, fungsi onLinkClicked() digunakan untuk mencatat ke log saat pengguna menekan tautan informasi gunung, meskipun fungsinya bersifat pelengkap dan

tidak memengaruhi data secara langsung. Secara keseluruhan, GunungViewModel menyediakan alur data yang bersih dan terstruktur, memungkinkan UI merespons perubahan secara reaktif. ViewModel ini juga mengedepankan manajemen data yang aman terhadap siklus hidup dengan memanfaatkan coroutine dan StateFlow, serta mendukung pengalaman pengguna yang mulus saat menjelajahi, memilih, atau menandai gunung favorit.

17. GunungViewModelFactory

Kode di atas merupakan implementasi dari kelas GunungViewModelFactory, yang berperan sebagai *factory class* dalam arsitektur MVVM untuk menghasilkan instance dari GunungViewModel. Kelas ini mengimplementasikan interface ViewModelProvider.Factory, yang merupakan standar dalam Jetpack Architecture Components untuk menciptakan ViewModel dengan parameter konstruktor khusus, dalam hal ini sebuah instance GunungRepository. Secara default, ViewModelProvider hanya mampu membuat ViewModel yang memiliki konstruktor kosong. Namun, GunungViewModel membutuhkan GunungRepository sebagai parameter agar dapat mengakses dan mengelola data. Oleh karena itu, diperlukan kelas factory khusus seperti ini untuk menangani pembuatan ViewModel dengan dependensi tersebut. Pada fungsi create, dilakukan pengecekan apakah modelClass yang diminta merupakan turunan dari GunungViewModel.

Jika ya, maka instance GunungViewModel akan dibuat dengan menyisipkan repository sebagai dependensinya, dan hasilnya dikembalikan sebagai tipe T. Supresi terhadap peringatan UNCHECKED_CAST dilakukan karena proses casting secara eksplisit dari GunungViewModel ke tipe generik T. Jika kelas ViewModel yang diminta bukan GunungViewModel, maka factory ini akan melemparkan IllegalArgumentException dengan pesan "Unknown ViewModel class" sebagai penanganan terhadap kemungkinan kesalahan penggunaan factory oleh komponen lain. Dengan pola ini, kode menjadi lebih fleksibel, terstruktur, dan mudah untuk diuji karena dependensi disuntikkan secara eksplisit dan terkontrol. Factory ini juga memungkinkan penerapan prinsip *dependency injection* dengan lebih rapi, dan menjadi bagian penting dalam menjembatani antara lapisan UI (seperti Fragment) dan ViewModel dalam aplikasi.

com/example/modul5/MainActivity.kt

18. MainActivity

Kode di atas merupakan implementasi dari MainActivity, yaitu aktivitas utama dalam aplikasi Android yang menggunakan arsitektur berbasis fragment dan ViewPager2 untuk menampilkan dua halaman berbeda: daftar gunung dan gunung favorit. Aktivitas ini memanfaatkan ActivityMainBinding, yang merupakan bagian dari View Binding, untuk mengakses elemen-elemen tampilan yang didefinisikan dalam layout activity_main.xml secara aman dan efisien tanpa perlu menggunakan findViewById() secara manual. Di dalam metode onCreate, dilakukan inisialisasi binding dengan memanggil inflate() pada ActivityMainBinding, yang kemudian dijadikan tampilan utama melalui setContentView(binding.root). Selanjutnya, dibuat instance dari ViewPagerAdapter, yaitu adaptor khusus yang menangani logika fragment pada ViewPager2. Adapter ini kemudian dihubungkan dengan binding.viewPager sehingga pengguna dapat menggeser antar halaman secara horizontal. Agar ViewPager2 dapat berfungsi dengan baik bersama TabLayout, digunakan TabLayoutMediator.

Mediator ini menjembatani antara tabLayout dan viewPager dengan menetapkan judul tab berdasarkan posisi, yang diambil dari metode getPageTitle(position) dalam ViewPagerAdapter. Fungsi attach() di akhir baris berfungsi untuk mengikat tab dengan halaman ViewPager secara penuh, sehingga pengguna dapat menavigasi ke fragment yang berbeda baik dengan menggeser halaman maupun mengetuk tab yang sesuai. Secara keseluruhan, MainActivity ini bertindak sebagai container utama yang mengatur alur navigasi antar fragment menggunakan tab, ViewPager2, dan View Binding, sehingga UI terasa modern, intuitif, dan mudah dikembangkan lebih lanjut.

MODUL5/app/src/main/res/layout

19. activity_main.xml

Kode XML di atas adalah definisi layout utama untuk MainActivity pada aplikasi Android berbasis fragment, ViewPager2, dan tab navigasi. Layout ini menggunakan ConstraintLayout sebagai root view, yang memungkinkan penempatan komponen UI secara

fleksibel dengan constraint antar elemen, sehingga tampilan tetap responsif di berbagai ukuran layar. Di bagian atas layout, terdapat elemen TabLayout dari Material Components yang berfungsi sebagai indikator tab navigasi. Elemen ini memiliki lebar yang menyesuaikan parent (0dp dengan constraint start dan end) dan tinggi yang disesuaikan dengan kontennya. Beberapa atribut styling digunakan untuk mengatur warna indikator tab, teks tab saat dipilih (tabSelectedTextColor), dan teks tab saat tidak dipilih (tabTextColor), di mana warna-warna ini diambil dari resource seperti @color/blue_500 dan @color/black.

Di bawah TabLayout, terdapat ViewPager2, yang merupakan komponen untuk menampilkan fragment secara horizontal melalui geseran (swipe). View ini dihubungkan dengan TabLayout melalui TabLayoutMediator di kelas MainActivity, sehingga setiap tab merepresentasikan fragment tertentu. ViewPager2 ini juga dikonstrain ke semua sisi parent, kecuali bagian atas yang dikaitkan ke TabLayout. Komponen ketiga adalah FrameLayout dengan ID fragmentContainerGunung. View ini disiapkan sebagai container tambahan untuk menampilkan fragment detail (misalnya GunungDetailFragment) yang muncul saat pengguna memilih item dari daftar gunung. Awalnya, FrameLayout ini disembunyikan dengan android:visibility="gone" dan hanya akan dimunculkan ketika ada aksi navigasi ke detail, yang diatur melalui logika di kelas fragment seperti GunungListFragment.

Secara keseluruhan, layout ini dirancang agar fleksibel dalam mendukung navigasi antara fragment menggunakan tab dan ViewPager2, serta memberikan ruang tambahan untuk menampilkan fragment detail dengan cara overlay yang dapat dikontrol visibilitasnya. Pendekatan ini membuat UI terasa terstruktur, terorganisir, dan siap untuk pengembangan aplikasi yang modular dan interaktif.

20. detail_fragment.xml

Layout ini merupakan tampilan detail dari sebuah gunung yang ditampilkan menggunakan ScrollView, memungkinkan pengguna menggulir konten secara vertikal, terutama jika deskripsi atau tampilan gambar terlalu panjang untuk satu layar. Root layout ScrollView menggunakan atribut match_parent untuk lebar dan tinggi agar memenuhi seluruh ruang layar, serta diberi latar belakang berwarna biru muda (#B6D6F1) yang

memberikan kesan sejuk dan natural, sesuai dengan tema alam dari konten gunung. Padding sebesar 16dp di sekeliling isi layout membantu menciptakan ruang yang nyaman antara tepi layar dan elemen konten, mencegah tampilan menjadi terlalu rapat. Di dalam ScrollView, terdapat sebuah LinearLayout vertikal yang menyusun seluruh elemen UI dari atas ke bawah. Elemen pertama adalah ImageView dengan ID imgPoster, yang menampilkan gambar gunung berukuran tetap setinggi 300dp dan mengisi seluruh lebar layar.

Atribut scaleType="centerCrop" memastikan gambar di-zoom dan dipotong agar memenuhi ukuran tampilan tanpa mengubah rasio aspek, menghasilkan tampilan visual yang estetis. Selanjutnya, terdapat tiga buah TextView untuk menampilkan nama gunung (tvName), lokasi (tvLokasi), dan deskripsi (tvDeskripsi). Masing-masing teks memiliki penyesuaian tampilan seperti ukuran font besar pada nama (35sp dengan bold) untuk menonjolkan identitas utama gunung, serta sentuhan gravity="center" untuk menyeimbangkan teks secara horizontal pada layar. Untuk teks lokasi, digunakan ukuran sedang (22sp) dan margin atas untuk memberikan jeda antar komponen, sedangkan deskripsi gunung menggunakan justificationMode="inter_word" (berlaku mulai API 26 ke atas) yang meratakan teks secara horizontal seperti pada paragraf di koran atau buku, membuat deskripsi panjang tampak lebih rapi dan mudah dibaca. Terakhir, di bagian bawah terdapat Button dengan ID btnBack yang berfungsi sebagai kontrol navigasi untuk kembali ke tampilan sebelumnya. Button ini diposisikan di tengah menggunakan layout_gravity="center" dan diberi margin atas agar tidak terlalu dekat dengan teks deskripsi. Secara keseluruhan, layout ini dirancang dengan keseimbangan estetika dan fungsionalitas.

Setiap elemen ditata secara rapi untuk menampilkan informasi penting mengenai gunung secara visual dan tekstual, serta disiapkan untuk berjalan mulus di berbagai ukuran layar dengan bantuan elemen scroll dan layout yang responsif. Kombinasi warna, ukuran font, margin, dan tata letak membuat tampilan ini informatif namun tetap menarik secara visual.

21. fragment_favorit_gunung.xml

File XML ini mendefinisikan tampilan antarmuka pengguna berbasis ConstraintLayout, yang merupakan salah satu layout paling fleksibel dan efisien di Android karena memungkinkan penempatan komponen UI secara relatif terhadap satu sama lain

maupun terhadap parent-nya. Layout ini memiliki lebar dan tinggi yang disetel ke match_parent, yang berarti akan memenuhi seluruh ruang layar perangkat, memastikan semua konten tertata secara penuh tanpa batasan ruang. Di dalam ConstraintLayout hanya terdapat satu elemen utama yaitu RecyclerView dengan ID recyclerViewFavorit, yang berfungsi untuk menampilkan daftar item favorit dalam bentuk list atau grid yang bisa digulir secara vertikal atau horizontal. Penggunaan RecyclerView di sini menandakan bahwa daftar yang ditampilkan bersifat dinamis dan dapat memuat banyak data secara efisien, karena RecyclerView hanya membuat tampilan sebanyak yang diperlukan di layar, sementara sisanya didaur ulang (recycled), menghemat memori dan meningkatkan performa. Lebar dan tinggi RecyclerView diatur dengan 0dp (alias match constraints), artinya ukurannya akan disesuaikan berdasarkan constraint yang diberikan. Empat atribut app:layout_constraintTop_toTopOf, app:layout_constraintBottom_toBottomOf, app:layout_constraintStart_toStartOf, dan app:layout_constraintEnd_toEndOf semuanya mengacu ke parent, yaitu ConstraintLayout itu sendiri.

Dengan pengaturan ini, RecyclerView akan menempati seluruh ruang dari atas ke bawah dan dari kiri ke kanan pada layar, menjadikannya komponen yang dominan dalam tampilan ini. Secara visual, layout ini sangat minimalis dan efisien, hanya berisi elemen tunggal yang bertugas menampilkan daftar favorit pengguna. Struktur seperti ini umum digunakan dalam fragment atau halaman khusus dalam aplikasi Android yang bertugas menyajikan data dalam format list, seperti daftar gunung favorit, item yang disimpan, atau konten yang sering diakses pengguna. Kode ini juga fleksibel untuk dikembangkan lebih lanjut, seperti menambahkan elemen pendukung (misalnya Toolbar, SearchView, atau FloatingActionButton) jika diinginkan dalam pengembangan berikutnya.

22. item_gunung.xml

File XML ini mendefinisikan struktur visual untuk satu kartu data menggunakan CardView sebagai kontainer utama. CardView digunakan karena mampu memberikan efek bayangan (cardElevation) dan sudut membulat (cardCornerRadius), menciptakan tampilan yang modern dan rapi, serta meningkatkan keterbacaan elemen-elemen di dalamnya. Kartu ini dirancang untuk menampilkan informasi singkat mengenai gunung, meliputi gambar,

nama, lokasi, deskripsi pendek, dan tiga tombol aksi: link, detail, serta tombol favorit berbentuk ikon. Di dalam CardView, terdapat sebuah ConstraintLayout yang digunakan sebagai layout utama karena fleksibilitasnya dalam menempatkan elemen-elemen UI secara presisi. ConstraintLayout ini memiliki padding untuk memberikan ruang antara konten dan tepi kartu, serta latar belakang berwarna krem (@color/cream) untuk mempertegas batas visual antar item. Di sisi kiri terdapat ImageView berukuran 120x160dp untuk menampilkan foto gunung, dengan pengaturan centerCrop agar gambar terfokus pada bagian tengah. Sebelah kanan gambar terdapat tiga TextView berisi nama gunung (tvGunungName) dengan ukuran font besar dan tebal untuk penekanan, lokasi (tvGunungLokasi) yang ditampilkan lebih kecil, dan deskripsi (tvGunungDeskripsi) yang dibatasi maksimal dua baris dan akan terpotong dengan ellipsis jika terlalu panjang, menjaga tampilan tetap ringkas.

Ketiganya disejajarkan secara vertikal dan dikaitkan dengan gambar menggunakan constraint start/end, menciptakan keselarasan visual yang konsisten. Selanjutnya, terdapat tiga komponen aksi. Tombol btnLink memungkinkan pengguna membuka tautan eksternal atau navigasi tambahan terkait gunung tersebut, btnDetail untuk melihat detail lengkap, dan btnFavorite berupa ImageButton berbentuk bintang untuk menandai gunung sebagai favorit. Tombol-tombol ini diposisikan sejajar di bawah deskripsi menggunakan constraint, dan bahkan btnFavorite diikat ke sisi kanan parent untuk menempati ujung baris. Tombol-tombol tersebut memiliki warna yang kontras seperti @color/teal_700, @color/blue, dan @color/pink, serta ukuran dan padding yang konsisten untuk menjaga keteraturan dan aksesibilitas. Komponen tambahan berupa Barrier (barrierStartText) digunakan untuk menyelaraskan posisi horizontal tombol berdasarkan ujung kiri teks, memberikan fleksibilitas dalam responsivitas tampilan, terutama jika ada perubahan panjang teks. Elemen ini merupakan teknik lanjutan di ConstraintLayout untuk menjaga konsistensi antar elemen meski konten bersifat dinamis.

Secara keseluruhan, layout ini dirancang tidak hanya untuk tampil menarik, tetapi juga responsif, mudah dinavigasi, dan mendukung pengalaman pengguna yang baik. Kombinasi antara gambar, teks, dan tombol aksi menjadikan setiap kartu informatif sekaligus interaktif, cocok untuk aplikasi katalog wisata alam atau aplikasi data gunung.

23. list_fragment.xml

Layout ini mendefinisikan tampilan utama dari sebuah fragmen (GunungListFragment) dalam aplikasi Android berbasis Jetpack, yang menggunakan ConstraintLayout sebagai root-nya. ConstraintLayout dipilih karena mampu menyusun komponen UI secara fleksibel dan efisien dengan menggunakan constraint antar elemen atau terhadap parent. Layout ini memiliki lebar dan tinggi yang mengisi seluruh layar (match_parent) dan disertai atribut tools:context untuk memberikan referensi konteks saat preview di Android Studio, yaitu GunungListFragment. Di dalamnya terdapat sebuah RecyclerView dengan ID rvGunung, yang berfungsi sebagai komponen utama untuk menampilkan daftar gunung secara vertikal dalam bentuk scrollable list. RecyclerView ini diatur agar lebarnya mengisi penuh dari kiri ke kanan (0dp dengan constraint start/end) dan tingginya dari atas ke bawah (0dp dengan constraint top/bottom), menempel langsung ke batas atas, bawah, kiri, dan kanan ConstraintLayout. RecyclerView ini juga diberi padding sebesar 16dp di semua sisi untuk menciptakan ruang antara isi dan batas layout, serta clipToPadding="false" agar konten seperti efek bayangan (elevation) dari item tetap terlihat di area padding tersebut.

Warna latar belakang RecyclerView diatur ke warna biru muda (#94C2EB), memberikan nuansa visual yang cerah dan mendukung pengalaman pengguna yang menyenangkan. Untuk keperluan preview dan pengembangan di Android Studio, atribut tools:listitem="@layout/item_gunung" ditambahkan. Ini memungkinkan pengembang melihat tampilan representatif dari item yang akan ditampilkan di daftar, berdasarkan layout item_gunung.xml. Secara keseluruhan, layout ini memfasilitasi sebuah daftar gunung dengan desain yang responsif, bersih, dan estetis, siap diisi dengan data dinamis menggunakan adapter di dalam fragment terkait. Struktur ini merupakan fondasi dari user interface berbasis daftar, yang umum digunakan dalam aplikasi pencarian, katalog, atau eksplorasi destinasi wisata alam.

res/values

24. colors.xml

Kelas XML ini berfungsi sebagai definisi sumber daya warna (colors.xml) dalam proyek Android. File ini terletak di direktori res/values/ dan digunakan untuk menyimpan nilai warna yang dapat dipanggil di seluruh aplikasi menggunakan @color/nama_warna. Dalam file ini, terdapat sejumlah entri warna yang diberi nama tertentu agar mudah digunakan dan dikelola secara konsisten. Misalnya, blue_500 memiliki nilai warna #2196F3, yang merupakan salah satu nuansa biru cerah dari palet material design. Warna blue_700, black, blue, pink, teal_700, dan white semuanya diberikan kode warna #1976D2, yaitu warna biru tua yang sama, meskipun nama-namanya berbeda.

Hal ini menunjukkan bahwa proyek ini kemungkinan masih dalam tahap pengembangan atau eksperimen warna, di mana nama-nama placeholder seperti black, white, atau pink masih belum disesuaikan dengan makna warna sebenarnya. Satu-satunya warna yang berbeda adalah cream, yang menggunakan nilai #FFEBCD, yakni warna krem muda yang lembut dan netral, sangat cocok sebagai latar belakang untuk meningkatkan keterbacaan teks dan memberi nuansa hangat pada desain UI. Pendefinisian warna dalam file ini memungkinkan penggunaan warna yang konsisten dan efisien di berbagai elemen layout, style, maupun komponen UI lainnya dalam aplikasi Android. Namun, agar tidak membingungkan, sebaiknya nama warna disesuaikan dengan nilai warnanya untuk menjaga keterbacaan dan maintainability kode.

25. string.xml

File strings.xml merupakan bagian dari sumber daya (resources) dalam proyek Android yang digunakan untuk menyimpan teks statis seperti label, judul, dan deskripsi agar mudah diatur dan diterjemahkan jika aplikasi nantinya mendukung berbagai bahasa (internationalization). Dalam file ini terdapat beberapa string penting. app_name berisi nilai "MODUL 5", yang menandakan nama aplikasi atau proyek yang sedang dikembangkan. Kemudian, gunung_image_desc berisi deskripsi alternatif "Gambar gunung", yang biasanya

digunakan untuk keperluan aksesibilitas atau konten deskriptif pada elemen gambar (ImageView) agar dapat dikenali oleh screen reader.

Selanjutnya, terdapat tiga string untuk teks tombol: btn_detail_text dengan nilai "Detail" yang kemungkinan menampilkan informasi lengkap suatu gunung, btn_favorite_text dengan karakter bintang kosong "☆" yang digunakan sebagai ikon tombol favorit, serta btn_link_text yang bertuliskan "Link", kemungkinan mengarah ke tautan eksternal atau informasi tambahan. Penggunaan string resource seperti ini sangat penting untuk menjaga keteraturan kode, memudahkan proses perubahan konten teks, serta memungkinkan dukungan multi-bahasa secara efisien di masa depan.

res/values/themes.xml

26. themes.xml

File themes.xml ini berfungsi untuk mendefinisikan tema global aplikasi Android kamu, yaitu *MODUL 5*. Di dalamnya terdapat dua style: Base.Theme.Modul5 dan Theme.Modul5. Style Base.Theme.Modul5 menjadi fondasi utama dari tema aplikasi, yang mewarisi dari Theme.Material3.DayNight.NoActionBar. Artinya, aplikasi ini mengadopsi gaya Material Design 3 (Material You) dengan dukungan mode gelap terang otomatis, serta tidak menggunakan *ActionBar* bawaan. Tema ini juga mengatur beberapa elemen warna utama, seperti colorPrimary yang mengambil warna dari @color/blue_500, serta colorPrimaryContainer dari @color/blue_700, yang biasanya digunakan untuk elemen UI seperti tombol dan latar belakang container. Selain itu, colorOnPrimary dan colorOnPrimaryContainer diatur ke warna putih (@android:color/white) agar teks atau ikon yang berada di atas elemen dengan latar belakang biru tetap terlihat jelas dan kontras.

Kemudian, style Theme.Modul5 dideklarasikan sebagai turunan langsung dari Base.Theme.Modul5, yang artinya kamu bisa menggunakannya sebagai tema utama aplikasi di AndroidManifest.xml. Struktur ini memberi fleksibilitas jika suatu saat kamu ingin membuat variasi tema lainnya tanpa harus menulis ulang seluruh konfigurasi warna atau atribut tampilan. Singkatnya, file ini menetapkan identitas visual aplikasi agar tampil konsisten dengan sentuhan warna biru khas yang sudah kamu definisikan di file colors.xml.

MODUL5/app/build.gradle.kts

27. build.gradle.kts

File ini adalah konfigurasi utama proyek Android kamu dalam format Kotlin DSL, yang mendefinisikan bagaimana proyek dikompilasi, dependensi yang digunakan, serta fitur yang diaktifkan. Di bagian plugins, kamu mengaktifkan beberapa plugin penting seperti com.android.application untuk aplikasi Android, org.jetbrains.kotlin.android untuk dukungan Kotlin di Android, kotlin-parcelize untuk kemudahan pengiriman objek melalui Intent, kotlin-kapt untuk annotation processing (khususnya digunakan oleh Room), serta kotlin.plugin.serialization untuk serialisasi/deserialisasi data menggunakan KotlinX Serialization. Di blok android, namespace proyek ditentukan sebagai com.example.modul5, dengan compileSdk 34 dan targetSdk serta minSdk masing-masing 34 dan 30, yang memastikan aplikasi berjalan di Android 11 ke atas. Build type release menonaktifkan ProGuard minification untuk memudahkan debug. Bagian compileOptions dan kotlinOptions menunjukkan bahwa proyek ini menggunakan Java 17 dan Kotlin dengan target JVM 17, yang berarti fitur-fitur modern Java/Kotlin bisa digunakan. viewBinding juga diaktifkan, memungkinkan binding langsung ke view dari layout XML tanpa menggunakan findViewById.

Pada blok dependencies, proyek menggunakan banyak library modern. Untuk core Android, digunakan core-ktx, appcompat, fragment-ktx, dan material untuk Material Design. Untuk UI, digunakan constraintlayout, recyclerview, dan cardview. Manajemen lifecycle difasilitasi oleh lifecycle-runtime-ktx, viewmodel-ktx, dan livedata-ktx. Proyek juga menggunakan Retrofit untuk networking (retrofit, kotlinx-serialization-json, dan retrofit2-kotlinx-serialization-converter) serta okhttp3-logging-interceptor untuk memudahkan debugging HTTP. Untuk memuat gambar, digunakan Coil versi 2.5.0. Untuk penyimpanan data lokal, proyek menggunakan Room database (room-runtime, room-compiler via kapt, dan room-ktx), serta mendukung coroutine lewat kotlinx-coroutines. Proyek juga menggunakan datastore-preferences sebagai cara modern menggantikan SharedPreferences. Terakhir, tersedia dependensi untuk testing, yaitu JUnit untuk unit test serta androidx.test dan espresso untuk pengujian UI. Konfigurasi ini menunjukkan bahwa proyek kamu sudah cukup

lengkap dan modern, mendukung arsitektur berbasis MVVM, konektivitas API, manajemen database lokal, dan penanganan gambar secara efisien.

D. Tautan Git

https://github.com/SheilaSabina/Praktikum-Mobile/tree/master/MODUL5