LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 5

Connect to the Internet

Oleh:

Muhammad Rizki Saputra NIM. 2310817310014

PROGRAM STUDI TEKNOLOGI INFORMASI FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN MOBILE I
MODUL 5

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

Nama Praktikan : Muhammad Rizki Saputra
NIM : 2310817310014

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

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL 1

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

- DinoApiService.kt

```
1   package com.example.modul5.data
2
3   import com.example.modul5.data.model.DinoDto
4   import retrofit2.http.GET
5
6   interface DinoApiService {
7       @GET("api/dinosaurs")
8       suspend fun getDinos(): List<DinoDto>
9   }
10
11
```

Table 1 Source Code DinoApiService

- DinoDao.kt

```
1   package com.example.modul5.data
2
3   import androidx.room.Dao
4   import androidx.room.Insert
5   import androidx.room.OnConflictStrategy
6   import androidx.room.Query
7
8   @Dao
9   interface DinoDao {
10      @Query("SELECT * FROM dinos")
11      suspend fun getAll(): List<DinoEntity>
12
13      @Insert(onConflict = OnConflictStrategy.REPLACE)
14      suspend fun insertAll(dinos: List<DinoEntity>)
15
16      @Query("DELETE FROM dinos")
17      suspend fun clear()
    }
```

Table 2 Source Code DinoDao.kt

- DinoDatabase.kt

```
1   package com.example.modul5.data
2
3   import android.content.Context
4   import androidx.room.Database
5   import androidx.room.Room
6   import androidx.room.RoomDatabase
7
8   @Database(entities = [DinoEntity::class], version = 1)
9   abstract class DinoDatabase : RoomDatabase() {
10      abstract fun dinoDao(): DinoDao
11
12      companion object {
13          @Volatile private var INSTANCE: DinoDatabase? = null
14
15          fun getInstance(context: Context): DinoDatabase =
16              INSTANCE ?: synchronized(this) {
17                  INSTANCE ?: Room.databaseBuilder(
18                      context.applicationContext,
19                      DinoDatabase::class.java,
20                      "dino.db"
21                  ).build().also { INSTANCE = it }
22              }
23      }
24  }
```

Table 3 Source Code DInoDatabase.kt

- DinoDto.kt

```
1   package com.example.modul5.data.model
2
3   import kotlinx.serialization.SerialName
4   import kotlinx.serialization.Serializable
5   import com.example.modul5.domain.Dino
6
7   @Serializable
8   data class DinoDto(
9       val name: String,
10      val description: String,
11      val diet: String,
12      val period: String,
13      val image: String
14  ) {
15      fun toDomain() = Dino(name, description, image, diet,
16  period)
17  }
```

Table 4 Source Code DinoDto.kt

- DinoEntity.kt

```
1   package com.example.modul5.data
2
3   import androidx.room.Entity
4   import androidx.room.PrimaryKey
5   import com.example.modul5.domain.Dino
6
7   @Entity(tableName = "dinos")
8   data class DinoEntity(
9       @PrimaryKey val name: String,
10      val description: String,
11      val imageUrl: String,
12      val diet: String,
13      val period: String
14  )
15
16  fun DinoEntity.toDomain() = Dino(
17      name, description, imageUrl, diet, period
18  )
19
20  fun Dino.toEntity() = DinoEntity(
21      name, description, imageUrl, diet, period
22  )
```

Table 5 Source Code DinoEntity.kt

- DinoRepositoryImpl.kt

```
1   package com.example.modul5.data.repository
2
3   import com.example.modul5.data.DinoApiService
4   import com.example.modul5.data.DinoDao
5   import com.example.modul5.data.Resource
6   import com.example.modul5.data.toDomain
7   import com.example.modul5.data.toEntity
8   import com.example.modul5.domain.Dino
9   import com.example.modul5.domain.repository.DinoRepository
10  import kotlinx.coroutines.Dispatchers
11  import kotlinx.coroutines.flow.Flow
12  import kotlinx.coroutines.flow.MutableStateFlow
13  import kotlinx.coroutines.flow.flow
14  import kotlinx.coroutines.flow.flowOn
15  import retrofit2.HttpException
16  import java.io.IOException
17
18  class DinoRepositoryImpl(
19      private val apiService: DinoApiService,
```

```
    private val dao: DinoDao
) : DinoRepository {

    private val selectedDino = MutableStateFlow<Dino?>(null)

    override fun getAllDinos(): Flow<Resource<List<Dino>>> =
flow {
        emit(Resource.Loading())

        val cachedDinos = dao.getAll().map { it.toDomain() }
        emit(Resource.Success(cachedDinos))

        try {
            val remoteDinos = apiService.getDinos()
            dao.clear()
            dao.insertAll(remoteDinos.map {
it.toDomain().toEntity() })
        } catch (e: HttpException) {

            emit(Resource.Error(
                "Terjadi kesalahan koneksi. Menampilkan data
offline.",
                cachedDinos
            ))
        } catch (e: IOException) {

            emit(Resource.Error(
                "Tidak ada internet. Menampilkan data
offline.",
                cachedDinos
            ))
        }

        val newDinos = dao.getAll().map { it.toDomain() }
        emit(Resource.Success(newDinos))

    }.flowOn(Dispatchers.IO)

    override fun selectDino(dino: Dino) {
        selectedDino.value = dino
    }

    override fun getSelectedDino(): Dino? = selectedDino.value
}
```

Table 6 Source Code DinoRepositoryImpl.kt

- Resource.kt

```
1  package com.example.modul5.data
2
3  sealed class Resource<T>(val data: T? = null, val message:
4  String? = null) {
5      class Success<T>(data: T) : Resource<T>(data)
6      class Error<T>(message: String, data: T? = null) :
7  Resource<T>(data, message)
8      class Loading<T>(data: T? = null) : Resource<T>(data)
9  }
```

Table 7 Source Code Resource.kt

- DinoViewModel,kt

```
1  package com.example.modul5.data
2
3  import
4  com.jakewharton.retrofit2.converter.kotlinx.serialization.asCon
5  verterFactory
6  import kotlinx.serialization.json.Json
7  import okhttp3.MediaType.Companion.toMediaType
8  import retrofit2.Retrofit
9  import com.example.modul5.data.DinoApiService
10 import com.example.modul5.data.model.DinoDto
11
12
13 object RetrofitInstance {
14     private val json = Json { ignoreUnknownKeys = true }
15     private val contentType = "application/json".toMediaType()
16
17     val api: DinoApiService by lazy {
18         Retrofit.Builder()
19             .baseUrl("https://dinoapi.brunosouzadev.com/")
20
21 .addConverterFactory(json.asConverterFactory(contentType))
22             .build()
23             .create(DinoApiService::class.java)
24     }
25 }
26
```

Table 8 Source Code DinoViewModel.kt

- ThemePreferences.kt

```
1  package com.example.modul5.data
2
3  import android.content.Context
4  import
5  androidx.datastore.preferences.core.booleanPreferencesKey
6  import androidx.datastore.preferences.core.edit
7  import androidx.datastore.preferences.preferencesDataStore
8  import kotlinx.coroutines.flow.Flow
9  import kotlinx.coroutines.flow.map
10
11 val Context.dataStore by preferencesDataStore("settings")
12
13 class ThemePreferences(private val context: Context) {
14     companion object {
15         private val DARK_MODE_KEY =
16 booleanPreferencesKey("dark_mode")
17     }
18
19     val darkModeFlow: Flow<Boolean> = context.dataStore.data
20         .map { preferences -> preferences[DARK_MODE_KEY] ?:
21 false }
22
23     suspend fun saveDarkModeSetting(isDarkMode: Boolean) {
24         context.dataStore.edit { preferences ->
25             preferences[DARK_MODE_KEY] = isDarkMode
26         }
27     }
28 }
```

Table 9 Source Code ThemePreferences.kt

- Dino.kt

```
1  package com.example.modul5.domain
2
3  data class Dino(
4      val name: String,
5      val description: String,
6      val imageUrl: String,
7      val diet: String,
8      val period: String
9  )
10
```

Table 10 Source Code MainActivity.kt

- DinoRepository

```
1   package com.example.modul5.domain.repository
2
3   import com.example.modul5.data.Resource
4   import com.example.modul5.domain.Dino
5   import kotlinx.coroutines.flow.Flow
6
7   interface DinoRepository {
8       fun getAllDinos(): Flow<Resource<List<Dino>>>
9
10      fun selectDino(dino: Dino)
11      fun getSelectedDino(): Dino?
12  }
```

Table 11 Source Code DinoRepository.kt

- DinoDetailScreen.kt

```
1   package com.example.modul5.presentation.ui
2
3   import androidx.compose.foundation.layout.*
4   import androidx.compose.foundation.rememberScrollState
5   import androidx.compose.foundation.verticalScroll
6   import androidx.compose.material3.*
7   import androidx.compose.runtime.Composable
8   import androidx.compose.runtime.collectAsState
9   import androidx.compose.ui.Alignment
10  import androidx.compose.ui.Modifier
11  import androidx.compose.ui.draw.clip
12  import androidx.compose.ui.layout.ContentScale
13  import androidx.compose.ui.text.font.FontWeight
14  import androidx.compose.ui.text.style.TextAlign
15  import androidx.compose.ui.unit.dp
16  import androidx.compose.ui.unit.sp
17  import androidx.navigation.NavController
18  import coil.compose.AsyncImage
19  import androidx.compose.foundation.shape.RoundedCornerShape
20  import com.example.modul5.presentation.viewmodel.DinoViewModel
21
22  @Composable
23  fun DinoDetailScreen(viewModel: DinoViewModel, navController:
24  NavController) {
25      val selectedDino =
26  viewModel.selectedDino.collectAsState().value
27
28      if (selectedDino == null) {
29          Text("Data tidak tersedia")
30          return
31      }
```

```
32
33      Column(
34          modifier = Modifier
35              .padding(16.dp)
36              .verticalScroll(rememberScrollState())
37              .fillMaxSize()
38      ) {
39          AsyncImage(
40              model = selectedDino.imageUrl,
41              contentDescription = null,
42              contentScale = ContentScale.Crop,
43              modifier = Modifier
44                  .fillMaxWidth()
45                  .height(250.dp)
46                  .clip(RoundedCornerShape(20.dp))
47          )
48          Spacer(modifier = Modifier.height(16.dp))
49          Text(
505             selectedDino.name,
1               fontSize = 22.sp,
52              fontWeight = FontWeight.Bold,
53              modifier = Modifier.fillMaxWidth(),
54              textAlign = TextAlign.Center
55          )
56          Spacer(modifier = Modifier.height(12.dp))
57          Text(
58              selectedDino.description,
59              fontSize = 14.sp,
60              textAlign = TextAlign.Justify
61          )
62          Spacer(modifier = Modifier.height(16.dp))
63          Button(
64              onClick = { navController.navigate("dino_list") },
65              modifier =
66  Modifier.align(Alignment.CenterHorizontally)
67          ) {
68              Text("Kembali")
69          }
70      }
71  }
```

Table 12 Source Code DinoDetailScreen.kt

- DinoListScreen.kt

```
1  package com.example.modul5.presentation.ui
2
3  import android.widget.Toast
4  import androidx.compose.foundation.background
5  import androidx.compose.foundation.layout.*
6  import androidx.compose.foundation.lazy.grid.GridCells
7  import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
8  import androidx.compose.foundation.lazy.grid.items
9  import androidx.compose.foundation.shape.RoundedCornerShape
10 import androidx.compose.material.icons.Icons
11 import androidx.compose.material.icons.filled.DarkMode
12 import androidx.compose.material.icons.filled.LightMode
13 import androidx.compose.material3.*
14 import androidx.compose.runtime.*
15 import androidx.compose.ui.Alignment
16 import androidx.compose.ui.Modifier
17 import androidx.compose.ui.draw.clip
18 import androidx.compose.ui.layout.ContentScale
19 import androidx.compose.ui.platform.LocalContext
20 import androidx.compose.ui.text.style.TextAlign
21 import androidx.compose.ui.unit.dp
22 import androidx.compose.ui.unit.sp
23 import androidx.navigation.NavController
24 import coil.compose.AsyncImage
25 import com.example.modul5.data.Resource
26 import com.example.modul5.presentation.viewmodel.DinoViewModel
27 import
28 com.example.modul5.presentation.viewmodel.SettingsViewModel
29 import
30 com.google.accompanist.systemuicontroller.rememberSystemUiContr
31 oller
32
33 @Composable
34 fun DinoListScreen(
35     navController: NavController,
36     viewModel: DinoViewModel,
37     settingsViewModel: SettingsViewModel,
38     isDarkMode: Boolean
39 ) {
40     val context = LocalContext.current
41     var searchQuery by remember { mutableStateOf("") }
42     var selectedDiet by remember { mutableStateOf("All") }
43     var expanded by remember { mutableStateOf(false) }
44
45     val dinoListState by
46 viewModel.dinoListState.collectAsState()
47     val dietOptions = listOf("All", "Herbivora", "Karnivora",
48 "Omnivora")
49     val systemUiController = rememberSystemUiController()
```

```
    val statusBarColor = MaterialTheme.colorScheme.primary
    val colorScheme = MaterialTheme.colorScheme

    SideEffect {
        systemUiController.setStatusBarColor(color =
statusBarColor, darkIcons = !isDarkMode)
    }

    LaunchedEffect(key1 = dinoListState) {
        if (dinoListState is Resource.Error &&
dinoListState.data?.isNotEmpty() == true) {
            Toast.makeText(context, dinoListState.message,
Toast.LENGTH_LONG).show()
        }
    }

    val filteredList = dinoListState.data?.filter { dino ->
        val matchesSearch = dino.name.contains(searchQuery,
ignoreCase = true)
        val apiDiet = when (selectedDiet) {
            "Herbivora" -> "herbivore"
            "Karnivora" -> "carnivore"
            "Omnivora" -> "omnivore"
            else -> "All"
        }
        val matchesDiet = apiDiet == "All" ||
dino.diet.equals(apiDiet, ignoreCase = true)
        matchesSearch && matchesDiet
    } ?: emptyList()

    Column(modifier =
Modifier.background(colorScheme.background).fillMaxSize()) {

        Row(
            modifier = Modifier
                .fillMaxWidth()
                .background(colorScheme.primary)
                .padding(16.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Text(
                text = "Jenis-Jenis Dinosaurus",
                fontSize = 24.sp,
                color = colorScheme.onPrimary,
            )
            IconButton(onClick = {
settingsViewModel.toggleDarkMode() }) {
                Icon(
                    imageVector = if (isDarkMode)
Icons.Default.LightMode else Icons.Default.DarkMode,
                    contentDescription = "Toggle Dark Mode",
```

```
                        tint = colorScheme.onPrimary
                    )
                }
            }


            OutlinedTextField(
                value = searchQuery,
                onValueChange = { searchQuery = it },
                label = { Text("Cari dinosaurus...") },
                shape = RoundedCornerShape(24.dp),
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp),
                colors = OutlinedTextFieldDefaults.colors(
                    focusedBorderColor = colorScheme.primary,
                    unfocusedBorderColor = colorScheme.outline,
                    cursorColor = colorScheme.primary
                )
            )


            Box(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 16.dp)
            ) {
                OutlinedButton(onClick = { expanded = true }) {
                    Text("Filter: $selectedDiet")
                }
                DropdownMenu(expanded = expanded, onDismissRequest
= { expanded = false }) {
                    dietOptions.forEach { option ->
                        DropdownMenuItem(
                            text = { Text(option) },
                            onClick = {
                                selectedDiet = option
                                expanded = false
                            }
                        )
                    }
                }
            }


            Box(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(top = 8.dp),
                contentAlignment = Alignment.Center
            ) {
                LazyVerticalGrid(
```

```
                columns = GridCells.Fixed(2),
                contentPadding = PaddingValues(horizontal =
16.dp, vertical = 8.dp),
                verticalArrangement =
Arrangement.spacedBy(12.dp),
                horizontalArrangement =
Arrangement.spacedBy(12.dp)
            ) {
                items(filteredList) { dino ->
                    Card(
                        modifier = Modifier.fillMaxWidth(),
                        colors =
CardDefaults.cardColors(containerColor = colorScheme.surface),
                        elevation =
CardDefaults.cardElevation(defaultElevation = 4.dp),
                        shape = RoundedCornerShape(16.dp)
                    ) {
                        Column {
                            AsyncImage(
                                model = dino.imageUrl,
                                contentDescription = dino.name,
                                contentScale =
ContentScale.Crop,
                                modifier = Modifier
                                    .fillMaxWidth()
                                    .height(140.dp)

.clip(RoundedCornerShape(topStart = 16.dp, topEnd = 16.dp))
                            )
                            Column(modifier =
Modifier.padding(12.dp)) {
                                Text(
                                    dino.name,
                                    fontSize = 16.sp,
                                    color =
colorScheme.onSurface
                                )
                                Spacer(modifier =
Modifier.height(8.dp))
                                Button(
                                    onClick = {

viewModel.selectDino(dino)

navController.navigate("detail")
                                    },
                                    modifier =
Modifier.fillMaxWidth(),
                                    colors =
ButtonDefaults.buttonColors(containerColor =
colorScheme.primary)
                                ) {
```

```
                                    Text("Info Detail", color =
colorScheme.onPrimary, fontSize = 12.sp)
                                }
                            }
                        }
                    }
                }

                if (dinoListState is Resource.Loading &&
filteredList.isEmpty()) {
                    CircularProgressIndicator()
                }

                if (dinoListState is Resource.Error &&
filteredList.isEmpty()) {
                    Text(
                        text = dinoListState.message ?: "Gagal
memuat data",
                        color = colorScheme.error,
                        textAlign = TextAlign.Center,
                        modifier = Modifier.padding(16.dp)
                    )
                }
            }
        }
}


```

Table 13 DinoListScreen.kt

- MainActivity

```
package com.example.modul5.presentation.ui

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Scaffold
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Modifier
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.example.modul5.presentation.theme.Modul5Theme
```

```
import com.example.modul5.presentation.viewmodel.DinoViewModel
import
com.example.modul5.presentation.viewmodel.DinoViewModelFactory
import
com.example.modul5.presentation.viewmodel.SettingsViewModel
import androidx.compose.ui.platform.LocalContext

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            val context = LocalContext.current
            val navController = rememberNavController()

            val viewModel: DinoViewModel = viewModel(factory =
DinoViewModelFactory(context))
            val settingsViewModel: SettingsViewModel =
viewModel()
            val isDarkMode by
settingsViewModel.darkMode.collectAsState(initial = false)

            Modul5Theme(darkTheme = isDarkMode) {
                Scaffold { padding ->
                    NavHost(
                        navController = navController,
                        startDestination = "dino_list",
                        modifier = Modifier.padding(padding)
                    ) {
                        composable("dino_list") {
                            DinoListScreen(navController,
viewModel, settingsViewModel, isDarkMode)
                        }
                        composable("detail") {

                            DinoDetailScreen(viewModel,
navController)
                        }
                    }
                }
            }
        }
    }
}
```

Table 14 Source Code MainActivity.kt

- DinoViewModel

```
1   package com.example.modul5.presentation.viewmodel
2
3   import android.content.Context
4   import androidx.lifecycle.ViewModel
5   import androidx.lifecycle.ViewModelProvider
6   import com.example.modul5.data.DinoDatabase
7   import com.example.modul5.data.RetrofitInstance
8   import com.example.modul5.data.repository.DinoRepositoryImpl
9   import com.example.modul5.domain.repository.DinoRepository
10
11
12  class DinoViewModelFactory(private val context: Context) :
13  ViewModelProvider.Factory {
14      override fun <T : ViewModel> create(modelClass: Class<T>):
15  T {
16          if
17  (modelClass.isAssignableFrom(DinoViewModel::class.java)) {
18
19              val repository: DinoRepository =
20  DinoRepositoryImpl(
21                  apiService = RetrofitInstance.api,
22                  dao =
23  DinoDatabase.getInstance(context).dinoDao()
24              )
25              @Suppress("UNCHECKED_CAST")
26              return DinoViewModel(repository) as T
27          }
28          throw IllegalArgumentException("Unknown ViewModel
29  class")
30      }
31  }
```

Table 15 Souce Code DinoViewModel.kt

- DinoViewModelFactory

```
1   package com.example.modul5.presentation.viewmodel
2
3   import androidx.lifecycle.ViewModel
4   import androidx.lifecycle.viewModelScope
5   import com.example.modul5.data.Resource
6   import com.example.modul5.domain.Dino
7   import com.example.modul5.domain.repository.DinoRepository
8   import kotlinx.coroutines.flow.MutableStateFlow
9   import kotlinx.coroutines.flow.StateFlow
10  import kotlinx.coroutines.flow.asStateFlow
11  import kotlinx.coroutines.flow.launchIn
12  import kotlinx.coroutines.flow.onEach
13
14  class DinoViewModel(private val repository: DinoRepository) :
15  ViewModel() {
16
17
18      private val _dinoListState =
19  MutableStateFlow<Resource<List<Dino>>>(Resource.Loading())
20      val dinoListState: StateFlow<Resource<List<Dino>>> =
21  _dinoListState.asStateFlow()
22
23
24      private val _selectedDino = MutableStateFlow<Dino?>(null)
25      val selectedDino: StateFlow<Dino?> =
26  _selectedDino.asStateFlow()
27
28      init {
29
30          getDinos()
31      }
32
33      private fun getDinos() {
34          repository.getAllDinos().onEach { result ->
35              _dinoListState.value = result
36          }.launchIn(viewModelScope)
37      }
38
39      fun selectDino(dino: Dino) {
40
41          _selectedDino.value = dino
42      }
43  }
```

Table 16 Source Code DinoViewModelFactory.kt

- SettingViewModel.kt

```
1   package com.example.modul5.presentation.viewmodel
2
3   import android.app.Application
4   import androidx.lifecycle.AndroidViewModel
5   import androidx.lifecycle.viewModelScope
6   import com.example.modul5.data.ThemePreferences
7   import kotlinx.coroutines.flow.Flow
8   import kotlinx.coroutines.flow.first
9   import kotlinx.coroutines.launch
10
11  class SettingsViewModel(app: Application) :
12  AndroidViewModel(app) {
13      private val prefs =
14  ThemePreferences(app.applicationContext)
15
16      val darkMode: Flow<Boolean> = prefs.darkModeFlow
17
18
19      fun toggleDarkMode() {
20          viewModelScope.launch {
21              val current = darkMode.first()
22              prefs.saveDarkModeSetting(!current)
23          }
24      }
25  }
26
```

Table 17 Source Code SettingViewModel.kt

- Theme.kt

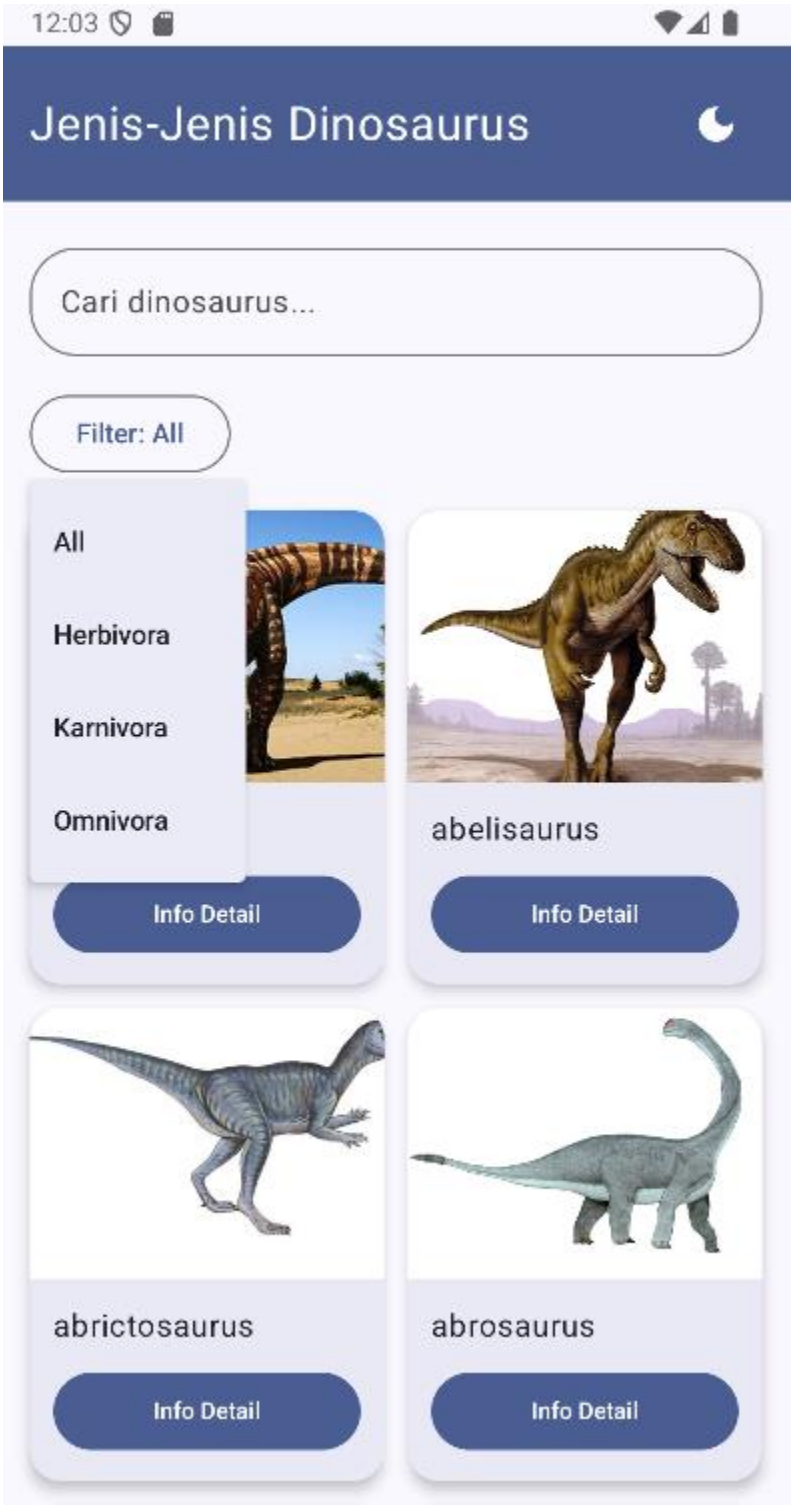
```
1   package com.example.modul5.presentation.theme
2
3   import android.app.Activity
4   import android.os.Build
5   import androidx.compose.foundation.isSystemInDarkTheme
6   import androidx.compose.material3.MaterialTheme
7   import androidx.compose.material3.darkColorScheme
8   import androidx.compose.material3.dynamicDarkColorScheme
9   import androidx.compose.material3.dynamicLightColorScheme
10  import androidx.compose.material3.lightColorScheme
11  import androidx.compose.runtime.Composable
12  import androidx.compose.ui.platform.LocalContext
13
14  private val DarkColorScheme = darkColorScheme(
15      primary = Purple80,
16      secondary = PurpleGrey80,
```

```
    tertiary = Pink80
)

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


)

@Composable
fun Modul5Theme(
    darkTheme: Boolean = isSystemInDarkTheme(),

    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >=
Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else
dynamicLightColorScheme(context)
        }

        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

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

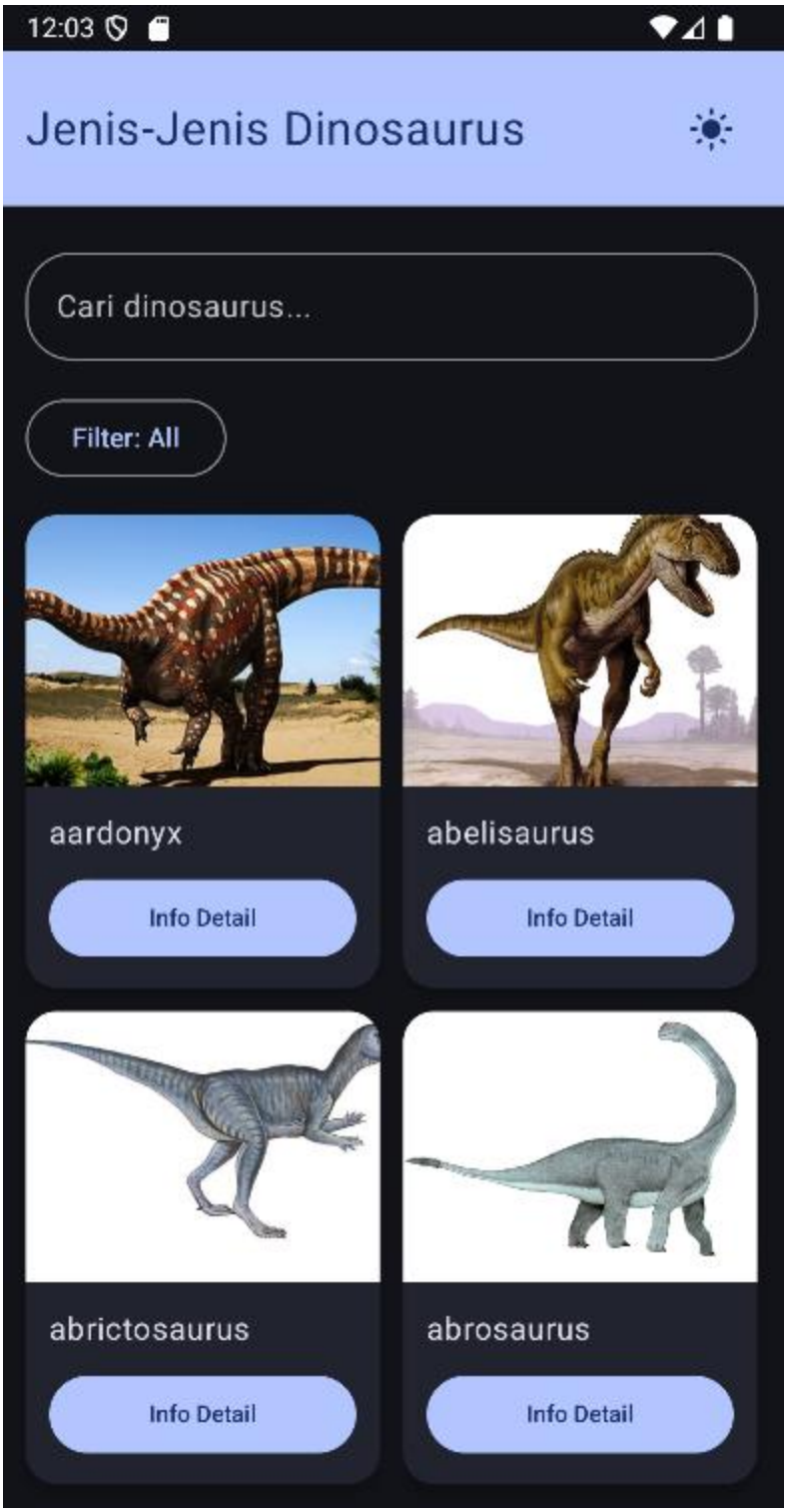
Table 18 Source Code Theme.kt

B. Output Program

Gambar 1 Screenshot Hasil Jawaban Soal 1



Gambar 2 Screenshot Hasil Jawaban Soal 1



Gambar 3 Screenshot Hasil Jawaban Soal 1

Gambar 4 Screenshot Hasil Jawaban Soal 1

C. Pembahasan

- DinoApiService.kt:

  File ini bertujuan untuk berkomunikasi dengan server atau API eksternal menggunakan library Retrofit. interface DinoApiService berisi daftar fungsi yang memetakan ke endpoint API. Fungsi getDinos() dengan anotasi @GET("api/dinosaurs") memberitahu Retrofit untuk membuat permintaan HTTP GET ke alamat tersebut.

- DinoDao.kt

  Data Access Object (DAO) bertujuan sebagai database lokal dan ini menggunakan Room. interface DinoDao berguna untuk berinteraksi dengan tabel dinosaurus di dalam database. Fungsi getAll() mengambil semua data dari tabel, insertAll() menyimpan daftar dinosaurus ke dalam tabel, dan clear() untuk menghapus semua data dari tabel, yang berguna sebelum memasukkan data baru dari API.

- DinoDatabase.kt

  Pada file ini terdapat abstract class DinoDatabase menghubungkan entitas (struktur tabel, yaitu DinoEntity) dengan DAO (DinoDao). Bagian companion object di dalamnya menerapkan pola Singleton, memastikan bahwa hanya ada satu instance database yang dibuat di seluruh aplikasi. Ini penting untuk mencegah kebocoran memori dan konflik data.

- DinoDto.kt

  Data Transfer Object (DTO) yang berfungsi sebagai model data mentah yang persis sama dengan struktur JSON yang diterima dari API. Kelas DinoDto ini memiliki fungsi toDomain()

- DinoEntity.kt

  Sebuah kelas data yang merepresentasikan struktur tabel di dalam database Room (@Entity(tableName = "dinos")). file ini juga berisi fungsi ekstensi (toDomain() dan toEntity()) untuk melakukan konversi antara model database (DinoEntity) dan model domain yang bersih (Dino). Ini memastikan bahwa lapisan database dan domain tetap terpisah.

- DinoRepositoryImpl.kt

  Fungsi ini menggunakan *builder* flow { ... } dari Kotlin Coroutines untuk menciptakan sebuah aliran data (Flow) yang akan mengeluarkan beberapa status secara berurutan. Saat getAllDinos() mulai dieksekusi (seperti ViewModel), langkah pertama adalah emit(Resource.Loading()). Segera setelah itu, kode mengambil data yang sudah ada di database lokal melalui val cachedDinos = dao.getAll().map { it.toDomain() }. Data dari database (bertipe DinoEntity) ini diubah menjadi model domain (Dino) dan langsung dipancarkan sebagai emit(Resource.Success(cachedDinos)). Tujuannya agar pengguna bisa langsung melihat data lama (jika ada) tanpa harus menunggu koneksi internet selesai. Lalu masuk blok try-catch untuk mencoba mengambil data baru dari internet. Panggilan val remoteDinos = apiService.getDinos() adalah operasi jaringan yang bisa gagal. Jika berhasil, data lama di database akan dihapus melalui dao.clear(), dan data baru dari API akan dimasukkan menggunakan dao.insertAll(...). Sebelum dimasukkan, data dari API (DinoDto) harus diubah dulu menjadi model domain (Dino) lalu ke model database (DinoEntity). Jika terjadi kegagalan jaringan, seperti HttpException (error dari server) atau IOException (tidak ada koneksi), blok catch akan menangkapnya.

- Resource.kt

  Disini terdapat Sealed class Resource membungkus data yang memberitahu kondisi: Loading (sedang dimuat), Success (berhasil didapat), atau Error (terjadi kesalahan). Ini memungkinkan UI untuk menampilkan indikator loading, dan pesan error

- RetrofitInstance.kt

  File ini menggunakan pola Singleton untuk membuat satu objek Retrofit yang akan digunakan di seluruh aplikasi. Objek ini dikonfigurasi dengan URL dasar API (https://dinoapi.brunosouzadev.com/) dan sebuah *converter factory* dari kotlinx.serialization untuk secara otomatis mengubah data JSON dari API menjadi objek DinoDto.

- ThemePreferences.kt

  Pda file ini ada extension property dataStore pada Context melalui preferencesDataStore("settings"), yang memastikan hanya ada satu file preferensi bernama "settings" untuk seluruh aplikasi. Di dalam kelas ThemePreferences, sebuah booleanPreferencesKey dengan nama "dark_mode" didefinisikan untuk menjadi kunci akses yang aman. Properti darkModeFlow kemudian dibuat dengan mengambil aliran data (data.map) dari DataStore, yang secara reaktif akan memancarkan nilai true atau false setiap

kali data dengan kunci tersebut berubah, dengan nilai default false jika belum pernah ada. Untuk menyimpan pengaturan, fungsi suspend saveDarkModeSetting menyediakan cara yang aman untuk menulis ke disk di background thread melalui blok edit, di mana ia menetapkan nilai boolean baru ke DARK_MODE_KEY.

- Dino.kt

File ini berisi data class Dino, yang merupakan model data domain yang ideal dan bersih. Sebagai sebuah data class, disini mendefinisikan struktur objek dinosaurus dengan properti seperti name, description, dan imageUrl.

- DinoRepository.kt

File ini berfungsi sebagai kontrak untuk lapisan data. mendefinisikan fungsi-fungsi yang dapat dipanggil oleh ViewModel, seperti *getAllDinos()*. Signature dari fungsi ini, *Flow<Resource<List<Dino>>>,* berbentuk deskriptif dan sebuah Flow (aliran data asinkron) yang akan memancarkan objek Resource (yang bisa berupa Loading, Success, atau Error) yang membungkus sebuah daftar objek Dino

- DinoListScreen.kt

DinoListScreen.kt berguna untuk membangun antarmuka pengguna (UI) utama aplikasi tempat daftar dinosaurus ditampilkan. Pada file ini *val dinoListState by viewModel.dinoListState.collectAsState()* yang menajdi otak pada file ini karena membuat UI menjadi responsif atau reaktif kepada perubahan di ViewModel. Lalu berdasarkan dinoListState yang diterima (apakah Resource.Loading, Success, atau Error), UI secara cerdas akan menampilkan CircularProgressIndicator, LazyVerticalGrid yang berisi daftar dinosaurus, atau sebuah Text dengan pesan error. Interaksi pengguna seperti mengetik di OutlinedTextField atau menekan IconButton untuk mode gelap ditangani dengan memanggil fungsi pada ViewModel atau memperbarui *state* lokal.

- DinoDetailScreen.kt

File ini bertujuan agar membuat halaman yang fokus menampilkan detail dari satu dinosaurus yang dipilih. Dan disini menggunakan collectAsState untuk mendapatkan data selectedDino dari ViewModel. Lalu menggunakan AsyncImage dari pustaka Coil, yang secara otomatis menangani pemuatan gambar dari internet hanya dengan memberikan URL ke properti model. Seluruh informasi seperti nama dan deskripsi panjang disajikan

dalam Column yang dapat di-scroll, dan terdapat tombol "Kembali" yang memanggil NavController untuk mengembalikan pengguna ke layar daftar.

- DinoViewModel

DinoViewModel.kt berfungsi sebagai pengelola *state* dan jembatan antara UI dan lapisan data. Di dalam blok init, memanggil fungsi getDinos() untuk memulai proses pengambilan data. Fungsi ini mengoleksi Flow dari repository menggunakan .launchIn(viewModelScope), dan untuk setiap Resource yang dipancarkan oleh Flow tersebut memperbarui nilai _dinoListState internalnya. *State* ini kemudian diekspos ke UI sebagai StateFlow yang *read-only*, memastikan aliran data yang searah dan terprediksi.

- DinoViewModelFactory.kt

Pada file ini bertujuan untuk membuat instance dari DinoViewModel. Kelas ini diperlukan karena DinoViewModel memiliki dependensi, yaitu DinoRepository, yang perlu disuntikkan saat dibuat. Di dalam metode create, secara manual membangun DinoRepositoryImpl dengan memberikannya semua dependensi yang dibutuhkan, seperti RetrofitInstance.api untuk jaringan dan DinoDao dari database. Repository yang sudah lengkap ini kemudian diteruskan ke dalam konstruktor DinoViewModel.

- SettingViewModel.kt

File ini bertujuan untuk mengelola logika dari pengaturan aplikasi. Lalu file ini mampu membuat instance AndroidViewModel, kemudian akan memunculkan darkModeFlow dari ThemePreferences. Dan juga didalamnya ada toggleDarkMode(), yang di dalamnya menjalankan coroutine untuk mendapatkan nilai dark mode alu memanggil fungsi saveDarkModeSetting untuk menyimpannya secara permanen.

- Theme.kt

Di dalamnya terdapat definisi konstan untuk skema warna, yaitu DarkColorScheme dan LightColorScheme. Komponen utama, Modul5Theme, menerima parameter boolean darkTheme.

D. Tautan Git

https://github.com/Yoruuu00/PemprogramanMobile/tree/main/Modul5