Offline First App

Praktikum Pemrograman Mobile - 08

Wanda Gusdya

2023

Pendahuluan

Aplikasi offline-first adalah aplikasi yang mampu melakukan semua, atau subset penting dari fungsi intinya tanpa akses ke internet. Artinya, ia dapat melakukan beberapa atau semua logika bisnisnya secara offline.

Pertimbangan untuk membangun aplikasi offline-first dimulai di lapisan data yang menawarkan akses ke data aplikasi dan logika bisnis. Aplikasi mungkin perlu menyegarkan data ini dari waktu ke waktu dari sumber eksternal ke perangkat. Dengan demikian, mungkin perlu memanggil sumber daya jaringan untuk tetap up to date.

Ketersediaan jaringan tidak selalu terjamin. Perangkat umumnya memiliki periode koneksi jaringan jerawatan atau lambat. Pengguna mungkin mengalami hal berikut:

- Bandwidth internet terbatas
- Gangguan koneksi sementara, seperti saat berada di lift atau terowongan.
- Akses data sesekali. Misalnya, tablet khusus WiFi.

Terlepas dari alasannya, seringkali aplikasi dapat berfungsi secara memadai dalam keadaan ini. Untuk memastikan bahwa aplikasi Anda berfungsi dengan benar secara offline, aplikasi harus dapat melakukan hal berikut:

- Tetap dapat digunakan tanpa koneksi jaringan yang andal.
- Sajikan data lokal kepada pengguna dengan segera alih-alih menunggu panggilan jaringan pertama selesai atau gagal.
- Ambil data dengan cara yang sadar akan baterai dan status data. Misalnya, dengan hanya meminta pengambilan data dalam kondisi optimal, seperti saat mengisi daya atau menggunakan WiFi.
- Aplikasi yang dapat memenuhi kriteria di atas sering disebut aplikasi offline-first.

Latihan

Kita telah membuat aplikasi yang dapat menyimpan data ke database lokal menggunakan android room. Untuk menambahka kemampuan mengelola data dari web service, kita perlu menambahkan beberapa library.

```
// network
implementation "com.github.skydoves:sandwich:1.0.9"
implementation "com.squareup.okhttp3:logging-interceptor:4.7.2"
implementation "com.squareup.retrofit2:converter-gson:2.9.0"
testImplementation "com.squareup.okhttp3:mockwebserver:4.7.2"
// whatIf
implementation "com.github.skydoves:whatif:1.0.7"
```

Library okhttp3 akan kita gunakan untuk mengakses web service dengan abstraksi yang disediakan oleh retrofit2. Library retrofit2 dapat menggunakan gson (selain jackson, moshi, dll) sebagai konverter teks ke json dan sebaliknya. Library sandwich dan whatif akan kita gunakan untuk membuat alur akses data yang intuitif.

Selanjutnya, update juga versi dari library lain, sehingga bagian dependencies akan terlihat seperti kode berikut.

```
implementation 'androidx.core:core-ktx:1.10.0'
implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.6.1'
implementation 'androidx.activity:activity-compose:1.7.1'
implementation "androidx.compose.ui:ui:$compose_ui_version"
implementation "androidx.compose.ui:ui-tooling-
preview:$compose_ui_version"
implementation 'androidx.compose.material:material:1.4.2'
testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.5'
androidTestImplementation 'androidx.test.espresso:espresso-
core:3.5.1'
androidTestImplementation "androidx.compose.ui:ui-test-
junit4:$compose_ui_version"
debugImplementation "androidx.compose.ui:ui-
tooling:$compose_ui_version"
debugImplementation "androidx.compose.ui:ui-test-
manifest:$compose_ui_version"

//uuid
implementation "com.benasher44:uuid:0.4.0"

implementation "androidx.compose.runtime:runtime:1.4.2"
implementation "androidx.compose.runtime:runtime-livedata:1.4.2"

//room
implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.6.1"
implementation "androidx.lifecycle:lifecycle-viewmodel-
ktx:2.6.1"
implementation "androidx.room:room-runtime:2.5.1"
implementation "androidx.room:room-ktx:2.5.1"
kapt "androidx.room:room-compiler:2.5.1"
testImplementation "androidx.arch.core:core-testing:2.2.0"

// network
implementation "com.github.skydoves:sandwich:1.0.9"
implementation "com.squareup.okhttp3:logging-interceptor:4.7.2"
implementation "com.squareup.retrofit2:converter-gson:2.9.0"
testImplementation "com.squareup.okhttp3:mockwebserver:4.7.2"
// whatIf
implementation "com.github.skydoves:whatif:1.0.7"

//hilt
implementation "com.google.dagger:hilt-android:2.45"
implementation "androidx.hilt:hilt-navigation-compose:1.0.0"
kapt "com.google.dagger:hilt-compiler:2.45"
```

```
kapt "androidx.hilt:hilt-compiler:1.0.0"
androidTestImplementation "com.google.dagger:hilt-android-
testing:2.45"
kaptAndroidTest "com.google.dagger:hilt-compiler:2.45"
```

Kemudian, bukalah file AndroidManifest.xml, untuk menambahkan permission yang memungkinkan aplikasi mengakses internet.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Jika kita akan menggunakan web service dengan protokol http (bukan https), tambahkan atribut berikut pada tag application di AndroidManifest.xml. Pada modul ini, kami sudah menyediakan web service dengan https di alamat https://setoran-sampah-api.gusdya.net sehingga sebenarnya bagian ini bisa diabaikan, namun penting untuk diingat.

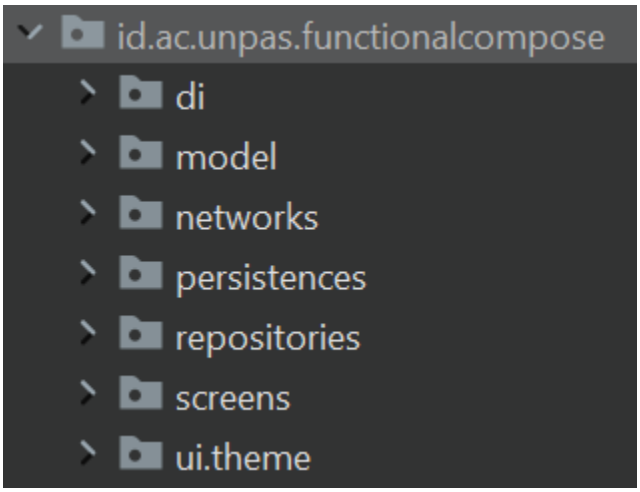
```
android:usesCleartextTraffic="true"
```

Sehingga, file AndroidManifest akan terlihat seperti berikut.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET"
/>

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.FunctionalCompose"
        android:name=".BankSampahApp"
        android:usesCleartextTraffic="true"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.FunctionalCompose">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
```

```
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Selanjutnya, buatlah dua package baru bernama networks dan repositories



Tambahkan kelas data bernama SetoranSampahGetResponse di package networks dengan kode sebagai berikut. Kelas ini akan kita gunakan untuk menampung respon web service yang memberikan banyak data optional SetoranSampah.

```
package id.ac.unpas.functionalcompose.networks

import id.ac.unpas.functionalcompose.model.SetoranSampah

data class SetoranSampahGetResponse(
    val data: List<SetoranSampah>? = null
)
```

Tambahkan kelas data bernama SetoranSampahSingleGetResponse di package networks dengan kode sebagai berikut. Kelas ini akan kita gunakan untuk menampung respon web service yang memberikan satu data optional SetoranSampah.

```
package id.ac.unpas.functionalcompose.networks

import id.ac.unpas.functionalcompose.model.SetoranSampah
```

```
data class SetoranSampahSingleGetResponse(
    val data: SetoranSampah? = null
)
```

Tambahkan interface SetoranSampahApi di package networks dengan kode sebagai berikut. Interface ini berisi daftar antarmuka pengaksesan web service yang telah disediakan. Setiap fungsi memiliki anotasi sesuai method http-nya (GET, POST, PUT, dan DELETE) dengan parameter alamat relatif terhadap base url (akan diset di objek module). Jika ada parameter path, kita dapat menggunakan anotasi Path, sedangkan data yang akan disimpan dikirim melalui Body.

```
package id.ac.unpas.functionalcompose.networks

import com.skydoves.sandwich.ApiResponse
import id.ac.unpas.functionalcompose.model.SetoranSampah
import retrofit2.http.Body
import retrofit2.http.DELETE
import retrofit2.http.GET
import retrofit2.http.Headers
import retrofit2.http.POST
import retrofit2.http.PUT
import retrofit2.http.Path

interface SetoranSampahApi {
    @GET("api/setoran-sampah")
    suspend fun all(): ApiResponse<SetoranSampahGetResponse>

    @GET("api/setoran-sampah/{id}")
    suspend fun find(@Path("id") id: String):
ApiResponse<SetoranSampahSingleGetResponse>

    @POST("api/setoran-sampah")
    @Headers("Content-Type: application/json")
    suspend fun insert(@Body item: SetoranSampah):
ApiResponse<SetoranSampahSingleGetResponse>

    @PUT("api/setoran-sampah/{id}")
    @Headers("Content-Type: application/json")
    suspend fun update(@Path("id") pathId: String,
                       @Body item: SetoranSampah):
ApiResponse<SetoranSampahSingleGetResponse>

    @DELETE("api/setoran-sampah/{id}")
    suspend fun delete(@Path("id") id: String):
ApiResponse<SetoranSampahSingleGetResponse>
}
```

Kemudian, tambahkan beberapa fungsi di interface SetoranSampahDao. Fungsi getList akan kita gunakan untuk mendapatkan data, fungsi insertAll akan kita gunakan untuk menyimpan semua data yang kita dapatkan dari web service, sedangkan fungsi delete akan kita gunakan untuk menghapus data lokal berdasarkan id.

```
@Query("SELECT * FROM SetoranSampah ORDER BY tanggal DESC")
suspend fun getList(): List<SetoranSampah>
```

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertAll(items: List<SetoranSampah>)
```

```
@Query("DELETE FROM SetoranSampah WHERE id = :id")
fun delete(id: String)
```

Sehingga interface SetoranSampahDao akan terlihat seperti berikut.

```
package id.ac.unpas.functionalcompose.persistences

import androidx.lifecycle.LiveData
import androidx.room.*
import id.ac.unpas.functionalcompose.model.SetoranSampah

@Dao
interface SetoranSampahDao {
    @Query("SELECT * FROM SetoranSampah ORDER BY tanggal DESC")
    fun loadAll(): LiveData<List<SetoranSampah>>

    @Query("SELECT * FROM SetoranSampah ORDER BY tanggal DESC")
    suspend fun getList(): List<SetoranSampah>

    @Query("SELECT * FROM SetoranSampah WHERE id = :id")
    fun find(id: String): SetoranSampah?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(vararg items: SetoranSampah)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(items: List<SetoranSampah>)

    @Delete
    fun delete(item: SetoranSampah)

    @Query("DELETE FROM SetoranSampah WHERE id = :id")
    fun delete(id: String)
}
```

Tambahkan interface Repository di package repositories sebagai tipe data umum dari kelas Repository. Tipe data ini berguna untuk dependency injection.

```
interface Repository
```

Selanjutnya, tambahkan kelas SetoranSampahRepository di package repositories dengan kode sebagai berikut. Kelas Repository ini yang akan mengumpulkan data dari remote (melalui web service) dan lokal untuk dikembalikan ke ViewModel. Repository juga akan menangani transaksi penyimpanan dan penghapusan data ke database lokal dan web service.

```
package id.ac.unpas.functionalcompose.repositories

import com.benasher44.uuid.uuid4
import com.skydoves.sandwich.message
import com.skydoves.sandwich.suspendOnError
import com.skydoves.sandwich.suspendOnException
import com.skydoves.sandwich.suspendOnSuccess
import com.skydoves.whatif.whatIfNotNull
import id.ac.unpas.functionalcompose.model.SetoranSampah
import id.ac.unpas.functionalcompose.networks.SetoranSampahApi
import
id.ac.unpas.functionalcompose.persistences.SetoranSampahDao
import javax.inject.Inject

class SetoranSampahRepository @Inject constructor(
    private val api: SetoranSampahApi,
    private val dao: SetoranSampahDao
) : Repository {

    suspend fun loadItems(
        onSuccess: (List<SetoranSampah>) -> Unit,
        onError: (List<SetoranSampah>, String) -> Unit
    ) {
        val list: List<SetoranSampah> = dao.getList()
        api.all()
            // handle the case when the API request gets a
success response.
            .suspendOnSuccess {
                data.whatIfNotNull {
                    it.data?.let { list ->
                        dao.insertAll(list)
                        val items: List<SetoranSampah> =
dao.getList()
                        onSuccess(items)
                    }
                }
            }
            // handle the case when the API request gets an
```

```
error response.
            // e.g. internal server error.
            .suspendOnError {
                onError(list, message())
            }
            // handle the case when the API request gets an
exception response.
            // e.g. network connection error.
            .suspendOnException {
                onError(list, message())
            }

    }

    suspend fun insert(
        tanggal: String,
        nama: String,
        berat: String,
        onSuccess: (SetoranSampah) -> Unit,
        onError: (SetoranSampah?, String) -> Unit
    ) {
        val id = uuid4().toString()
        val item = SetoranSampah(id, tanggal, nama, berat)
        dao.insertAll(item)
        api.insert(item)
            // handle the case when the API request gets a
success response.
            .suspendOnSuccess {
                onSuccess(item)
            }
            // handle the case when the API request gets an
error response.
            // e.g. internal server error.
            .suspendOnError {
                onError(item, message())
            }
            // handle the case when the API request gets an
exception response.
            // e.g. network connection error.
            .suspendOnException {
                onError(item, message())
            }

    }

    suspend fun update(
        id: String,
```

```
        tanggal: String,
        nama: String,
        berat: String,
        onSuccess: (SetoranSampah) -> Unit,
        onError: (SetoranSampah?, String) -> Unit
    ) {
        val item = SetoranSampah(id, tanggal, nama, berat)
        dao.insertAll(item)
        api.update(id, item)
            // handle the case when the API request gets a
success response.
            .suspendOnSuccess {
                onSuccess(item)
            }
            // handle the case when the API request gets an
error response.
            // e.g. internal server error.
            .suspendOnError {
                onError(item, message())
            }
            // handle the case when the API request gets an
exception response.
            // e.g. network connection error.
            .suspendOnException {
                onError(item, message())
            }

    }

    suspend fun delete(id: String, onSuccess: () -> Unit,
onError: (String) -> Unit) {
        dao.delete(id)
        api.delete(id)
            // handle the case when the API request gets a
success response.
            .suspendOnSuccess {
                data.whatIfNotNull {
                    onSuccess()
                }
            }
            // handle the case when the API request gets an
error response.
            // e.g. internal server error.
            .suspendOnError {
                onError(message())
            }
            // handle the case when the API request gets an
```

```
exception response.
            // e.g. network connection error.
            .suspendOnException {
                onError(message())
            }
    }
}
```

Kemudian, tambahkan object NetworkModule di package di dengan kode sebagai berikut. Module ini berisi objek-objek yang memungkinkan untuk mengakses web service, seperti objek OkHttpClient, Retrofit, dan semua API Client (di modul baru ada satu). Untuk mengubah base URL web service, dapat dilakukan dengan mengubah parameter baseUrl pada fungsi yang membentuk objek Retrofit.

```
package id.ac.unpas.functionalcompose.di

import android.content.Context
import
com.skydoves.sandwich.coroutines.CoroutinesResponseCallAdapterFa
ctory
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import id.ac.unpas.functionalcompose.networks.SetoranSampahApi
import okhttp3.OkHttpClient
import okhttp3.logging.HttpLoggingInterceptor
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object NetworkModule {
    @Provides
    @Singleton
    fun provideOkHttpClient(@ApplicationContext context:
Context): OkHttpClient {
        return OkHttpClient.Builder()
            //Hanya untuk development/debug. Tidak disarankan
untuk produksi.
            .addInterceptor(HttpLoggingInterceptor().apply {
                level = HttpLoggingInterceptor.Level.BODY
            })
            .build()
    }
```

```
    @Provides
    @Singleton
    fun provideRetrofit(okHttpClient: OkHttpClient): Retrofit {
        return Retrofit.Builder()
            .client(okHttpClient)
            .baseUrl(
                "https://setoran-sampah-api.gusdya.net/"
            )
            .addConverterFactory(GsonConverterFactory.create())
            .addCallAdapterFactory(CoroutinesResponseCallAdapter
Factory())
            .build()
    }

    @Provides
    @Singleton
    fun provideSetoranSampahApi(retrofit: Retrofit):
SetoranSampahApi {
        return retrofit.create(SetoranSampahApi::class.java)
    }
}
```

Selanjutnya, tambahkan object RepositoryModule di package di dengan kode sebagai berikut. Module ini berisi seluruh Repository yang dimiliki project.

```
package id.ac.unpas.functionalcompose.di

import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.components.ViewModelComponent
import dagger.hilt.android.scopes.ViewModelScoped
import id.ac.unpas.functionalcompose.networks.SetoranSampahApi
import
id.ac.unpas.functionalcompose.persistences.SetoranSampahDao
import
id.ac.unpas.functionalcompose.repositories.SetoranSampahReposito
ry

@Module
@InstallIn(ViewModelComponent::class)
object RepositoryModule {
    @Provides
    @ViewModelScoped
    fun provideSetoranSampahRepository(
        api: SetoranSampahApi,
        dao: SetoranSampahDao
```

```
    ): SetoranSampahRepository {
        return SetoranSampahRepository(api, dao)
    }
}
```

Ubah PengelolaanSampahViewModel menjadi seperti berikut. ViewModel sekarang tidak mengakses Dao, tapi Repository. Kita juga perlu beberapa state tambahan seperti isLoading, success, dan toast untuk memberikan feedback pengaksesan data dari Repository.

```
package id.ac.unpas.functionalcompose.screens

import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import dagger.hilt.android.lifecycle.HiltViewModel
import id.ac.unpas.functionalcompose.model.SetoranSampah
import
id.ac.unpas.functionalcompose.repositories.SetoranSampahReposito
ry
import javax.inject.Inject

@HiltViewModel
class PengelolaanSampahViewModel @Inject constructor(private val
setoranSampahRepository: SetoranSampahRepository) : ViewModel()
{
    private val _isLoading: MutableLiveData<Boolean> =
MutableLiveData(false)
    val isLoading: LiveData<Boolean> get() = _isLoading

    private val _success: MutableLiveData<Boolean> =
MutableLiveData(false)
    val success: LiveData<Boolean> get() = _success

    private val _toast: MutableLiveData<String> =
MutableLiveData()
    val toast: LiveData<String> get() = _toast

    private val _list: MutableLiveData<List<SetoranSampah>> =
MutableLiveData()
    val list: LiveData<List<SetoranSampah>> get() = _list

    suspend fun loadItems()
    {
        _isLoading.postValue(true)
        setoranSampahRepository.loadItems(onSuccess = {
            _isLoading.postValue(false)
            _list.postValue(it)
```

```
        }, onError = { list, message ->
            _toast.postValue(message)
            _isLoading.postValue(false)
            _list.postValue(list)
        })
    }

    suspend fun insert(tanggal: String,
                       nama: String,
                       berat: String){
        _isLoading.postValue(true)
        setoranSampahRepository.insert(tanggal, nama, berat,
onError = { item, message ->
            _toast.postValue(message)
            _isLoading.postValue(false)
        }, onSuccess = {
            _isLoading.postValue(false)
            _success.postValue(true)
        })
    }
}
```

Buka file PengelolaanSampahScreen, tambahkan kode berikut pada bagian atas setelah kurung kurawal buka {

```
val scope = rememberCoroutineScope()
val context = LocalContext.current
```

Kemudian tambahkan kode berikut sebelum kurung kurawal tutup }

```
LaunchedEffect(scope) {
    viewModel.loadItems()
}

viewModel.success.observe(LocalLifecycleOwner.current) {
    if (it) {
        scope.launch {
            viewModel.loadItems()
        }
    }
}

viewModel.toast.observe(LocalLifecycleOwner.current) {
    Toast.makeText(context, it, Toast.LENGTH_LONG).show()
}
```

Sehingga file PengelolaanSampahScreen akan terlihat seperti berikut. Kode di bagian bawah menunjukkan bahwa View berlangganan state success dan toast. Fungsi loadItems dipanggil untuk mentrigger permintaan data ke database lokal dan remote.

```
package id.ac.unpas.functionalcompose.screens

import android.widget.Toast
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.Divider
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalLifecycleOwner
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import id.ac.unpas.functionalcompose.model.SetoranSampah
import kotlinx.coroutines.launch

@Composable
fun PengelolaanSampahScreen() {
    val viewModel = hiltViewModel<PengelolaanSampahViewModel>()
    val scope = rememberCoroutineScope()
    val context = LocalContext.current

    val items: List<SetoranSampah> by
viewModel.list.observeAsState(initial = listOf())

    Column(modifier = Modifier.fillMaxWidth()) {
        FormPencatatanSampah()

        LazyColumn(modifier = Modifier.fillMaxWidth()) {
            items(items = items, itemContent = { item ->

                Row(modifier = Modifier
                    .padding(15.dp)
                    .fillMaxWidth()) {
```

```
                    Column(modifier = Modifier.weight(3f)) {
                        Text(text = "Tanggal", fontSize = 14.sp)
                        Text(text = item.tanggal, fontSize =
16.sp, fontWeight = FontWeight.Bold)
                    }

                    Column(modifier = Modifier.weight(3f)) {
                        Text(text = "Nama", fontSize = 14.sp)
                        Text(text = item.nama, fontSize = 16.sp,
fontWeight = FontWeight.Bold)
                    }

                    Column(modifier = Modifier.weight(3f)) {
                        Text(text = "Berat", fontSize = 14.sp)
                        Text(text = "${item.berat} Kg", fontSize
= 16.sp, fontWeight = FontWeight.Bold)
                    }
                }

                Divider(modifier = Modifier.fillMaxWidth())

            })
        }
    }

    LaunchedEffect(scope) {
        viewModel.loadItems()
    }

    viewModel.success.observe(LocalLifecycleOwner.current) {
        if (it) {
            scope.launch {
                viewModel.loadItems()
            }
        }
    }

    viewModel.toast.observe(LocalLifecycleOwner.current) {
        Toast.makeText(context, it, Toast.LENGTH_LONG).show()
    }
}
```

Kemudian, tambahkan kode berikut di bagian atas FormPencatatanSampah. Kode ini untuk memberikan feedback ketika UI sedang memproses permintaan.

```
val isLoading = remember { mutableStateOf(false) }
val buttonLabel = if (isLoading.value) "Mohon tunggu..." else
"Simpan"
```

Fungsi onClick pada button simpan menjadi seperti berikut. Pengisian field id tidak lagi ditangani oleh View, namun Repository.

```
scope.launch {
    viewModel.insert(tanggal.value.text, nama.value.text,
berat.value.text)
    tanggal.value = TextFieldValue("")
    nama.value = TextFieldValue("")
    berat.value = TextFieldValue("")
}
```

Ubah atribut text button simpan menjadi seperti berikut, sehingga button simpan akan menampilkan teks yang sesuai dengan status permintaan.

```
text = buttonLabel,
```

Kemudian tambahkan kode berikut sebelum kurung kurawal tutup untuk memantau state isLoading dari ViewModel.

```
viewModel.isLoading.observe(LocalLifecycleOwner.current) {
    isLoading.value = it
}
```

Jalankan aplikasi dan perhatikan hasilnya.