Android Fundamental
Networking, Architecture Component, & Local Data Persistent

bangkit

Ground Rules

Observe the following rules to ensure a supportive, inclusive, and engaging classes

Give full attention in class

Mute your microphone when you're not talking

Keep your camera on

Turn on the CC Feature on Meet

Use raise hand or chat to ask questions

Make this room a safe place to learn and share

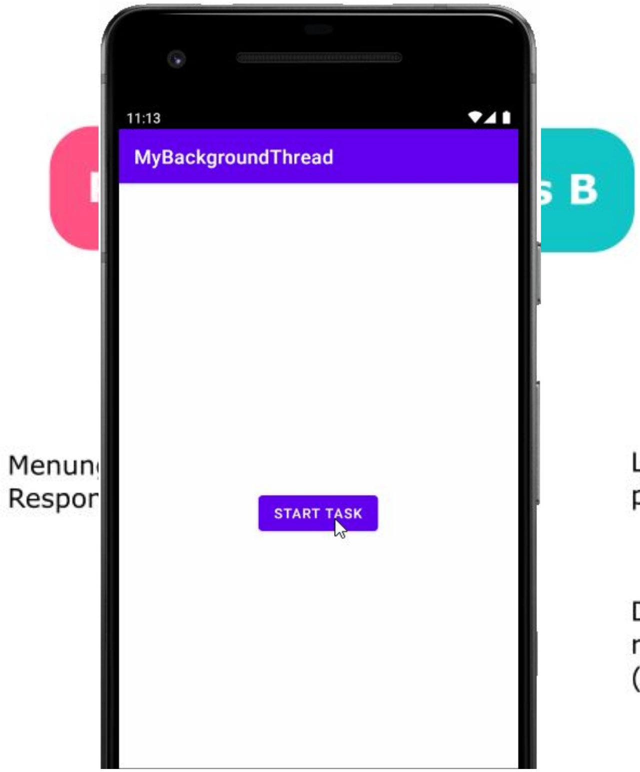
Material/Review

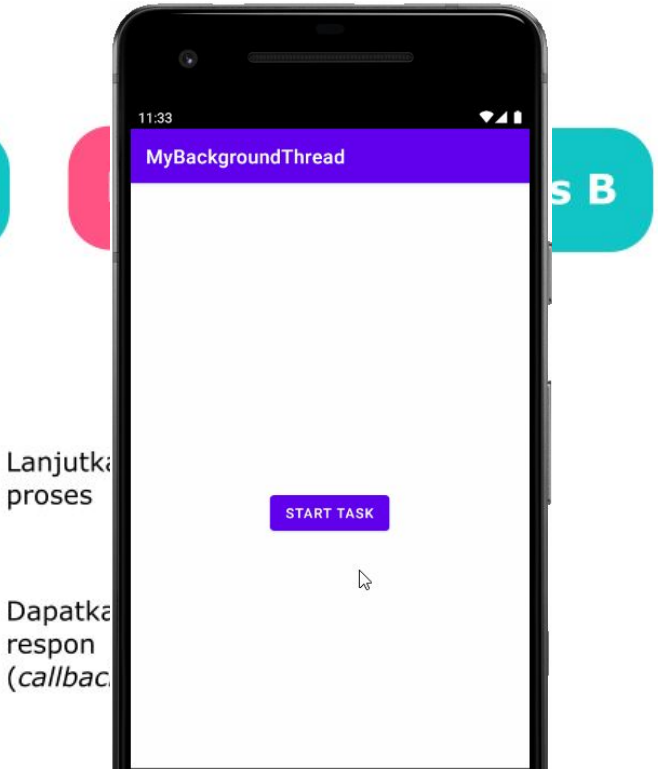
Material that has been studied

- Learn how to **build your first app** with Android Studio.
- Learn about **the basics** of Android Development such as Activity, Intent, Fragment, View, and ViewGroup.
- Learn how to **debug** and **resolve errors**.
- Learn how to build a layout using **ConstraintLayout**.
- Learn how to design attractive applications using **Navigation** elements such as ActionBar, NavigationDrawer, BottomNavigation, and TabLayout.

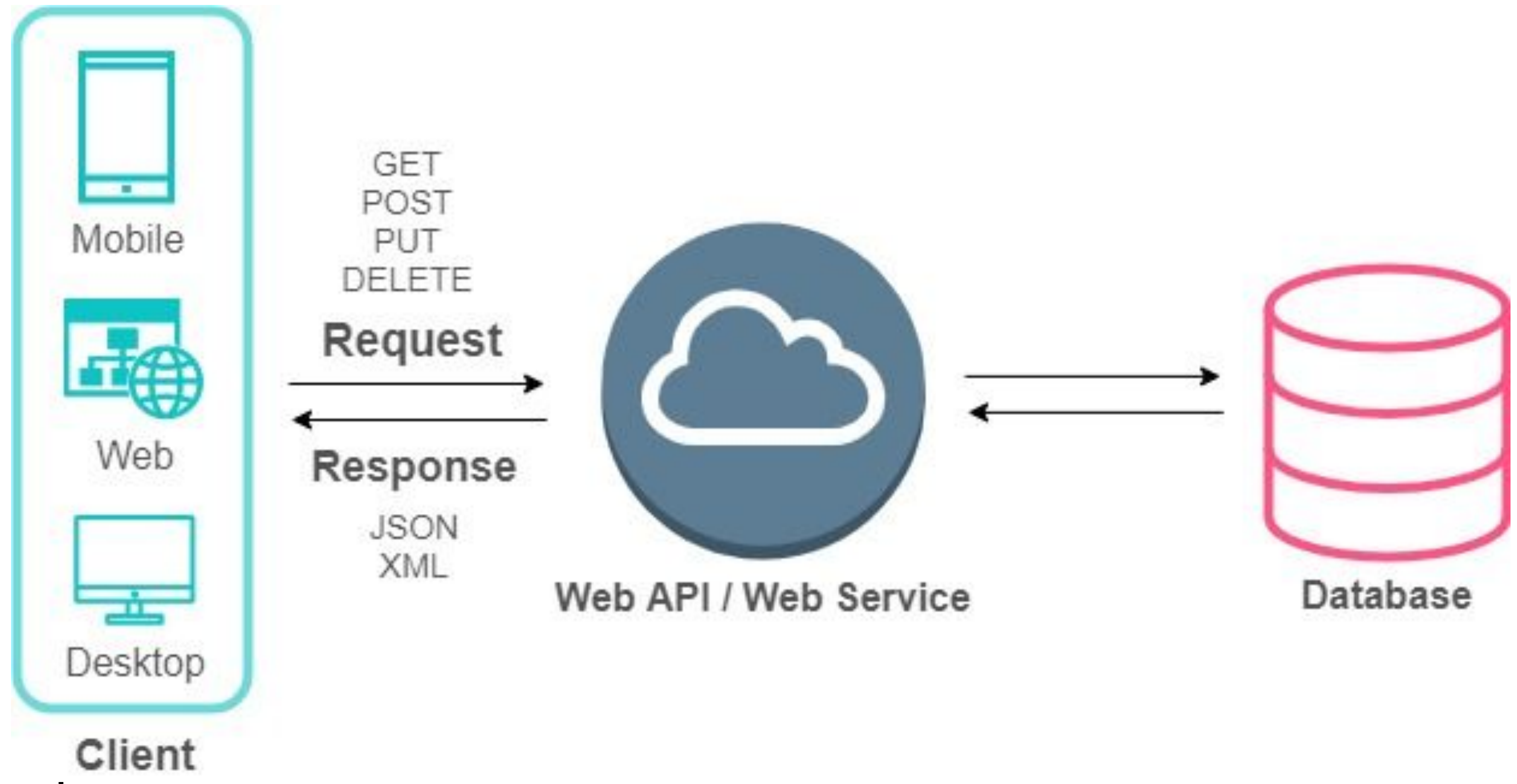
Networking

Synchronous and Asynchronous Comparison





Web API Concept

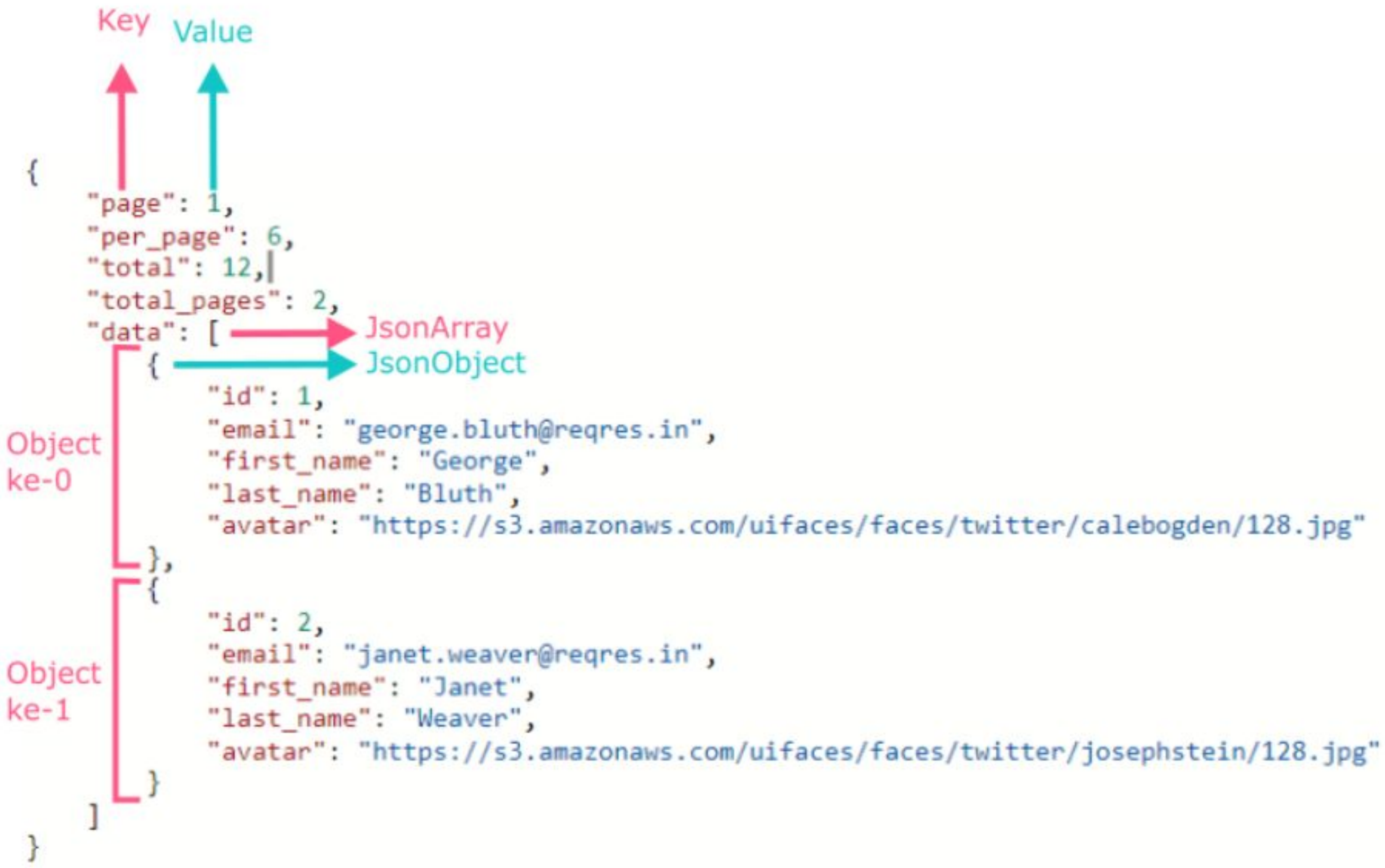


API Parameter

For Example :
https://reqres.in/api/users?page=1&per_page=10

- Path : “users”
- Query 1 : “page” with value “1”
- Query 2 : “per_page” with value “10”

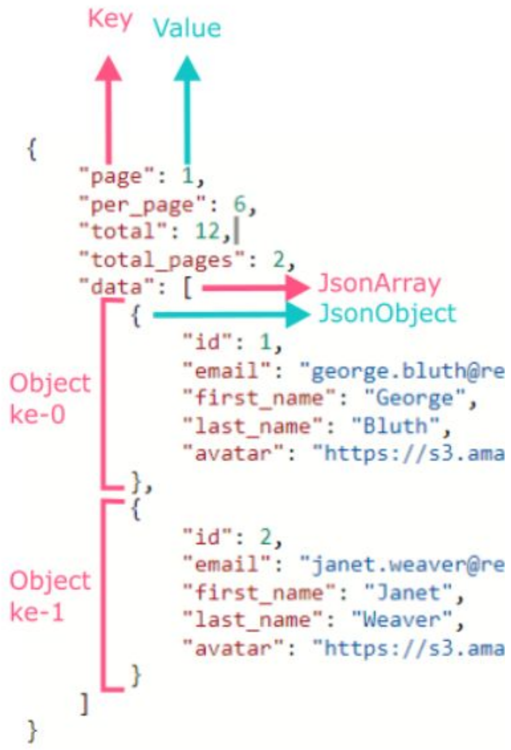
- Use “?” as separator before first parameter
- Use “&” as separator for the next parameter
- Use “=” to fill query with value



```
{
    "page": 1,
    "per_page": 6,
    "total": 12,
    "total_pages": 2,
    "data": [
        {
            "id": 1,
            "email": "george.bluth@reqres.in",
            "first_name": "George",
            "last_name": "Bluth",
            "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/calebogden/128.jpg"
        },
        {
            "id": 2,
            "email": "janet.weaver@reqres.in",
            "first_name": "Janet",
            "last_name": "Weaver",
            "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/josephstein/128.jpg"
        }
    ]
}
```

JSON Parsing

```
implementation 'com.google.code.gson:gson:2.8.6'
...
data class DataItem(
       @field:SerializedName("id")
       val id: Int? = null,
...
private fun parseJson(response: String) {
   val gson = Gson()
   val jsonObject = JSONObject(response);
   val dataArray = jsonObject.getJSONArray("data")

   for (i in 0 until dataArray.length()) {
       val dataObject = dataArray.getJSONObject(i)
       val data = gson.fromJson(dataObject.toString(),
DataItem::class.java)
       adapter.addUser(data)
   }
```



```
{
    "page": 1,
    "per_page": 6,
    "total": 12,
    "total_pages": 2,
    "data": [
       {
          "id": 1,
          "email": "george.bluth@re
          "first_name": "George",
          "last_name": "Bluth",
          "avatar": "https://s3.ama
       },
       {
          "id": 2,
          "email": "janet.weaver@re
          "first_name": "Janet",
          "last_name": "Weaver",
          "avatar": "https://s3.ama
       }
    ]
}
```

Retrofit

Retrofit is a library made by Square, which is popularly used for Networking the Web API.

With Retrofit, setting up API endpoints and parsing JSON is much easier.

```
implementation "com.squareup.retrofit2:retrofit:$retrofitVersion"
implementation "com.squareup.retrofit2:converter-gson:$retrofitVersion"
```

Click for more detail https://square.github.io/retrofit/

Retrofit Service

```kotlin
interface UserService {
   // add information using Header
   @Headers("Authorization: token <Personal Access Token>")
   @GET("users")
   fun getListUsers(@Query("page") page: String): Call<ResponseUser>

   // get list user by id using path
   @GET("users/{id}")
   fun getUser(@Path("id") id: String): Call<ResponseUser>

   // post user using field x-www-form-urlencoded
   @FormUrlEncoded
   @POST("users")
   fun createUser(
      @Field("name") name: String,
      @Field("job") job: String
   ): Call<ResponseUser>
}
```

Retrofit Implementation

```kotlin
val retrofit = Retrofit.Builder()
     .baseUrl("https://reqres.in/api/")
     .addConverterFactory(GsonConverterFactory.create())
     .build()
val userService = retrofit.create(UserService::class.java)
```

```kotlin
userService.getUser(userId).enqueue(object : Callback<UserResponse> {
    override fun onResponse(call: Call<UserResponse>, response:
Response<UserResponse>) {
        if (response.isSuccessful) {
             val data = response.body()
        }
    }
    // Error case is left out for brevity.
    override fun onFailure(call: Call<UserResponse>, t: Throwable) {
        TODO()
    }
})
```

Demo Link

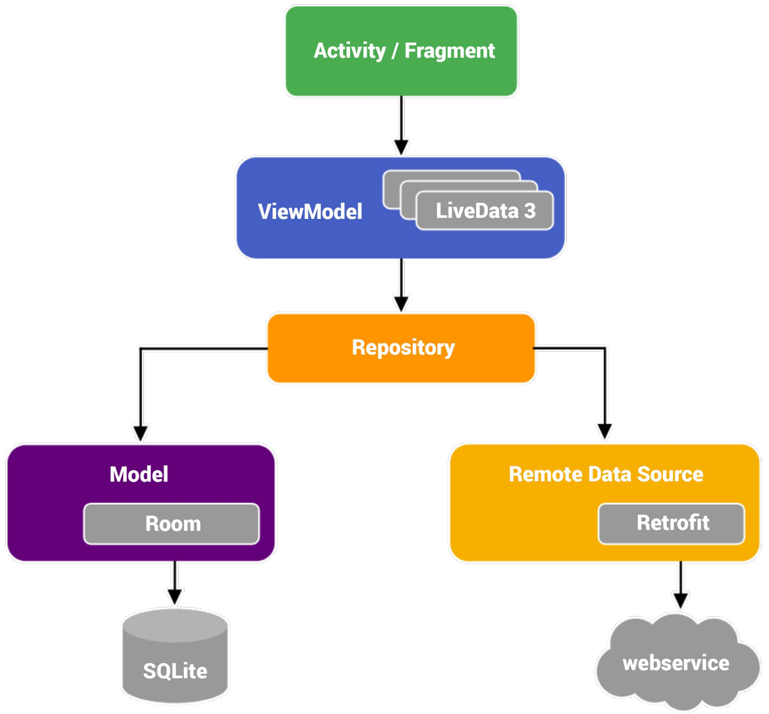
https://github.com/dicodingacademy/demo-ilt-android-bangkit/tree/main/ILT3/LatihanRetrofit

Android Architecture Components

bangkit

Android Architecture Components

- Activity/Fragment
- ViewModel
- LiveData
- Repository
- Room



ViewModel + LiveData

MainViewModel.kt

```kotlin
class MainViewModel : ViewModel() {

   private val _restaurant = MutableLiveData<Restaurant>()
   val restaurant: LiveData<Restaurant> = _restaurant
   ...
   _restaurant.value = // data from repository
}
```

MainActivity.kt

```kotlin
val mainViewModel = ViewModelProvider(this,
ViewModelProvider.NewInstanceFactory()).get(MainViewModel::class.java)

//OR using activity-ktx
val mainViewModel by viewModels<MainViewModel>()

mainViewModel.restaurant.observe(this, { restaurant ->
   setRestaurantData(restaurant)
})
```

Transform LiveData

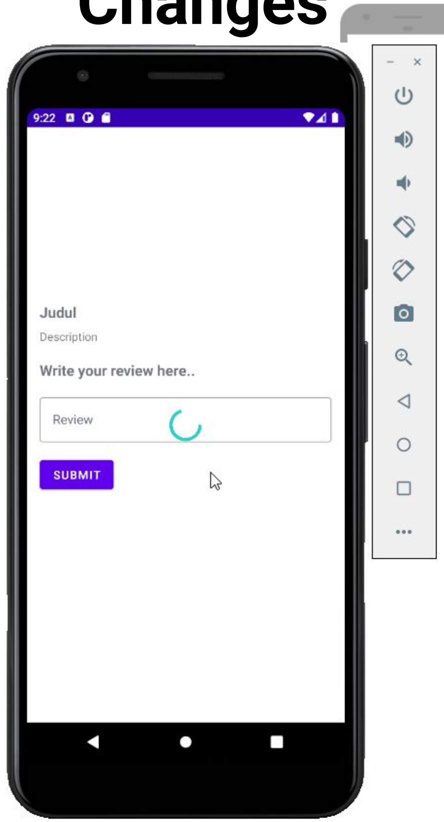
Transformations.map()

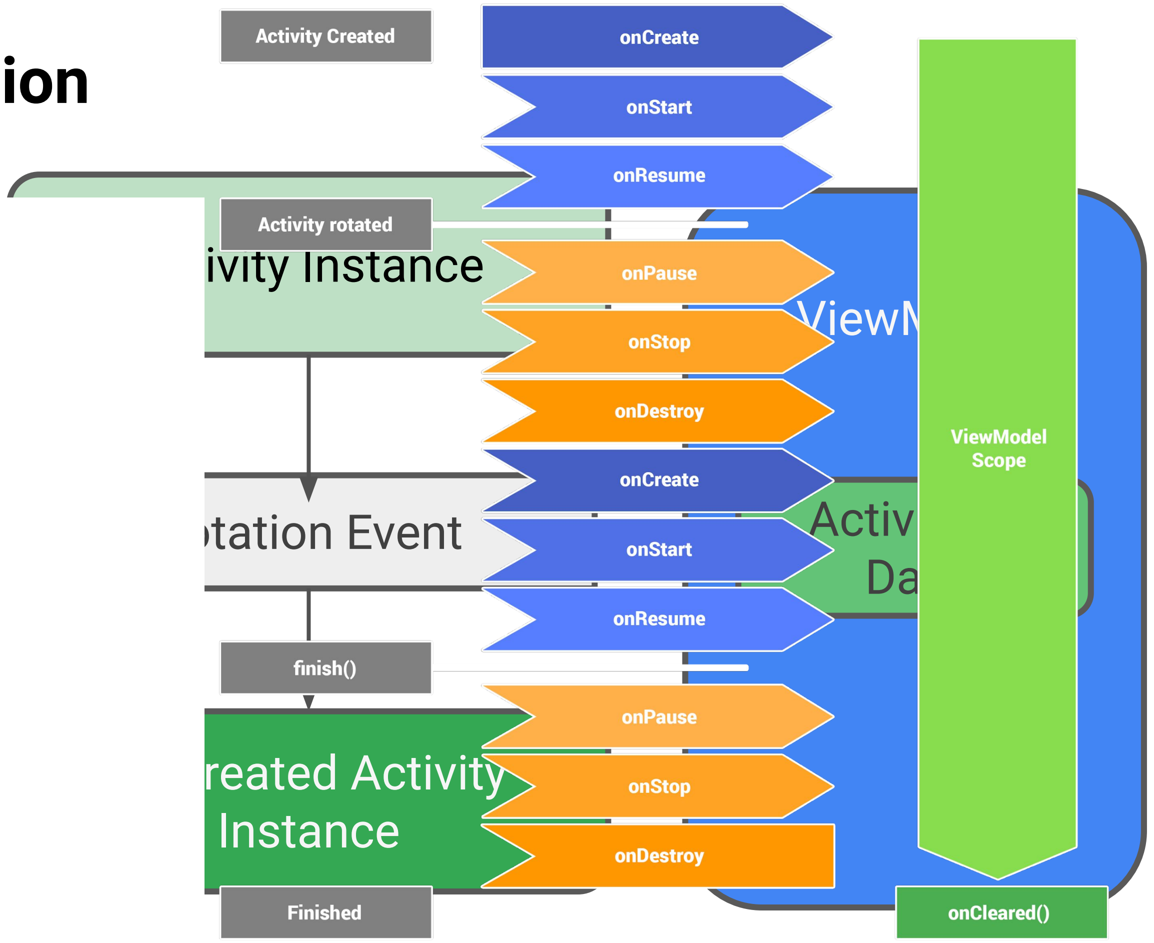
```
val userLiveData: LiveData<UserResponse> = UserLiveData()
val userName: LiveData<String> = Transformations.map(userLiveData) {
    user -> "${user.name} ${user.lastName}"
}
```

Transformations.switchMap()

```
private fun getUser(id: String): LiveData<UserResponse> {
    ...
}
val userId: LiveData<String> = ...
val user = Transformations.switchMap(userId) { id -> getUser(id) }
```

Survives Configuration Changes





ViewModelFactory

```
class ViewModelFactory private constructor(private val newsRepository: NewsRepository) :
    ViewModelProvider.NewInstanceFactory() {
    @Suppress("UNCHECKED_CAST")
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(NewsViewModel::class.java)) {
            return NewsViewModel(newsRepository) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class: " + modelClass.name)
    }

    companion object {
        @Volatile
        private var instance: ViewModelFactory? = null
        fun getInstance(context: Context): ViewModelFactory =
            instance ?: synchronized(this) {
                instance ?: ViewModelFactory(Injection.provideRepository(context))
            }.also { instance = it }
    }
}
```

Use Coroutines with ViewModel

- ViewModel includes support for coroutines, namely ViewModelScope.
- A ViewModelScope is defined for each ViewModel in your app. Any coroutine launched in this scope is automatically canceled if the ViewModel is cleared.
- Coroutines are useful here for work that needs to be done only if the ViewModel is active.

```
class MyViewModel: ViewModel() {
    init {
        viewModelScope.launch {
            // do something
        }
    }
}
```

Demo Link

https://github.com/dicodingacademy/demo-ilt-android-bangkit/tree/main/ILT3/LatihanRepository%20(coroutines)

Local Data Persistent

Preferences DataStore

- Initialization

```
val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name =
"settings")
```

- Saving Data

```
private val THEME_KEY = booleanPreferencesKey("theme_setting")

dataStore.edit { preferences ->
    preferences[THEME_KEY] = isDarkModeActive
}
```

- Get Data

```
dataStore.data.map { preferences ->
    preferences[THEME_KEY] ?: false
}
```

DataStore Vs SharedPreference

Feature	SharedPreferences	Preferences DataStore	Proto DataStore
Async API	✅ (only for reading changed values, via listener)	✅ (via `Flow`)	✅ (via `Flow`)
Synchronous API	✅ (but not safe to call on UI thread)	❌	❌
Safe to call on UI thread	❌ *	✅ (work is moved to `Dispatchers.IO` under the hood)	✅ (work is moved to `Dispatchers.IO` under the hood)
Can signal errors	❌	✅	✅
Safe from runtime exceptions	❌ **	✅	✅
Has a transactional API with strong consistency guarantees	❌	✅	✅
Handles data migration	❌	✅ (from SharedPreferences)	✅ (from SharedPreferences)
Type safety	❌	❌	✅ with Protocol Buffers

bangkit

source

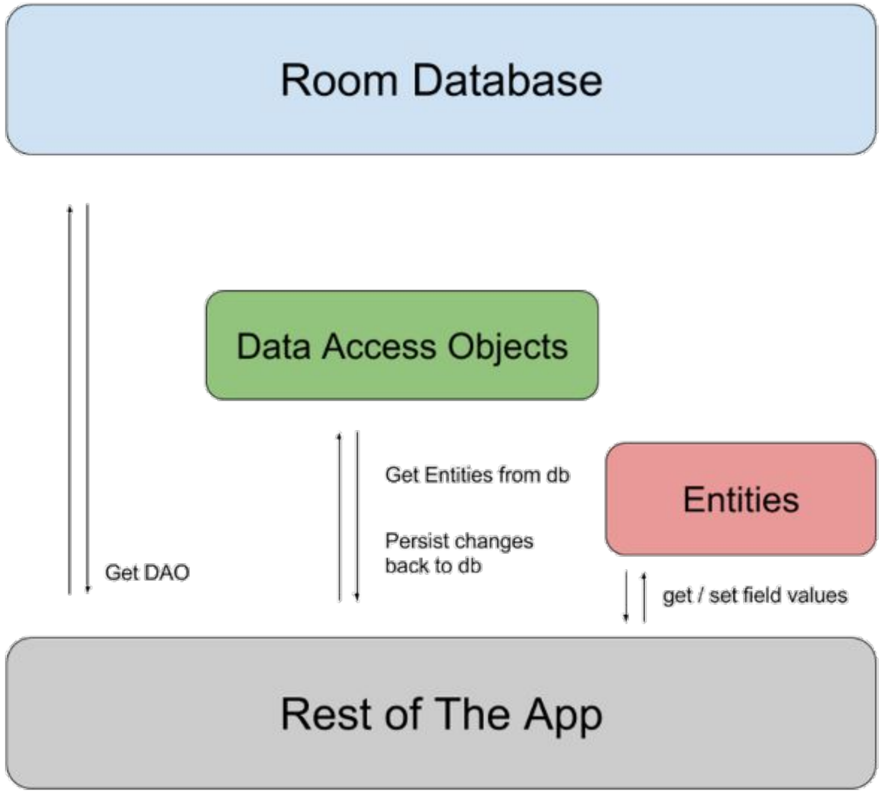
Room

Room is a robust SQL object mapping library:

- Generates SQLite Android code
- Provides a simple API for your database

There are 3 major components in Room:

- **Database**: serves as the main access point for the database.
- **Entity**: Represents a table within the database.
- **DAO**: Contains the methods used for accessing the database.



Room Entity

uid	first_name	last_name
12345	Aleks	Becker
12346	Jhansi	Kumar

```
@Entity(tableName = "user_table")
data class UserEntity(
  @PrimaryKey
  val uid: Int,

  @ColumnInfo(name = "first_name")
  val firstName: String?,

  @ColumnInfo(name = "last_name")
  val lastName: String?
)
```

Room DAO

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user_table")
    fun getAll(): LiveData<UserEntity>

    @Query("SELECT * FROM user_table WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<UserEntity>

    @Query("SELECT * FROM user_table WHERE first_name LIKE :first AND last_name LIKE :last
            LIMIT 1")
    fun findByName(first: String, last: String): UserEntity

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insert(user: UserEntity)

    @Delete
    fun delete(user: UserEntity)
}
```

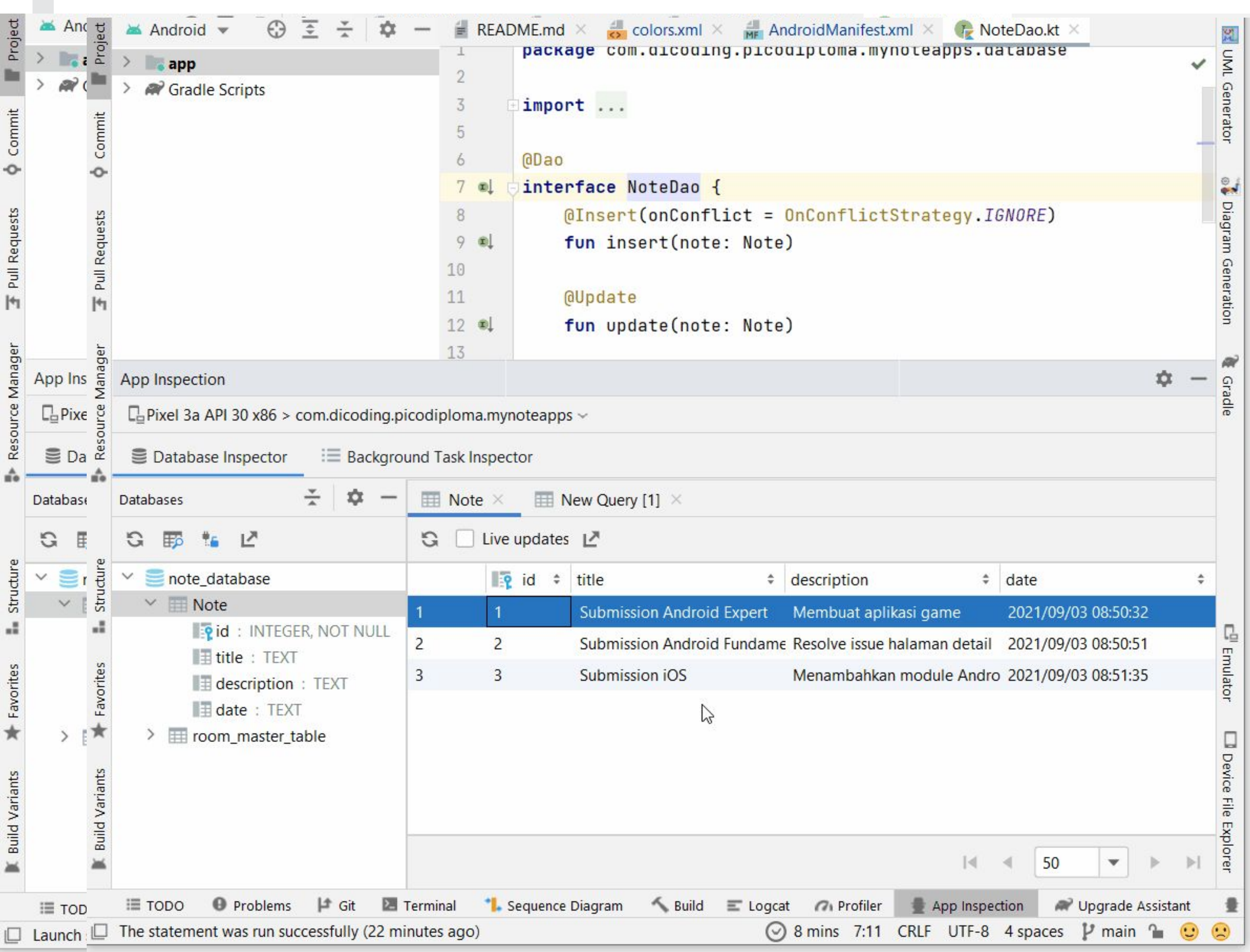
Room Database

Defined as singleton to prevent having multiple instances of the database.

```
@Database(entities = arrayOf(UserEntity::class), version = 1, exportSchema = false)
public abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null
        fun getDatabase(context: Context): AppDatabase =
           INSTANCE ?: synchronized(this) {
               INSTANCE ?: Room.databaseBuilder(
                    context.applicationContext,
                    Appatabase::class.java,
                    "database-name"
               ).build()
           }
    }
}
```

Demo Link

https://github.com/dicodingacademy/demo-ilt-android-bangkit/tree/main/ILT3/LatihanRoom

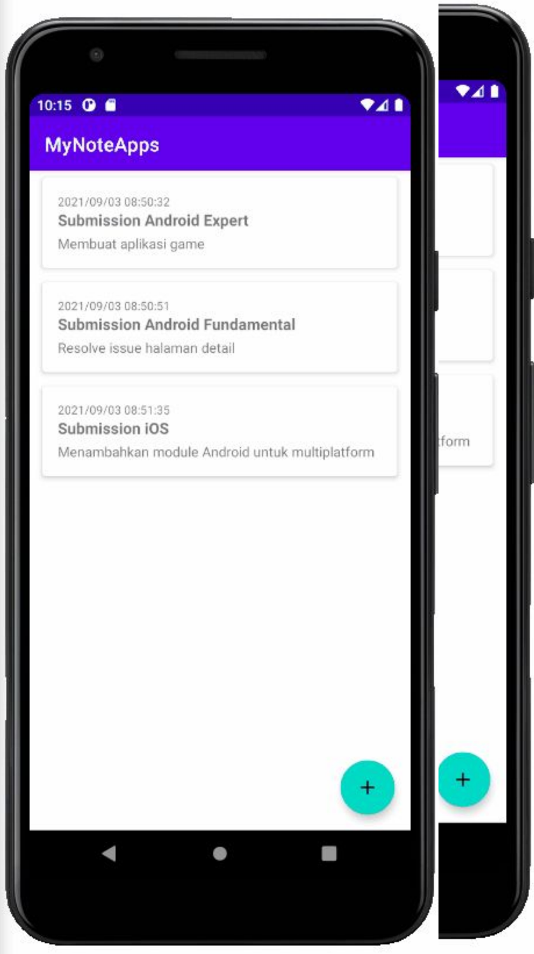


```kotlin
package com.dicoding.picodiploma.mynoteapps.database

import ...

@Dao
interface NoteDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insert(note: Note)

    @Update
    fun update(note: Note)
```

```
note_database
  Note
    id : INTEGER, NOT NULL
    title : TEXT
    description : TEXT
    date : TEXT
  room_master_table
```

	id	title	description	date
1	1	Submission Android Expert	Membuat aplikasi game	2021/09/03 08:50:32
2	2	Submission Android Fundame	Resolve issue halaman detail	2021/09/03 08:50:51
3	3	Submission iOS	Menambahkan module Andro	2021/09/03 08:51:35

The statement was run successfully (22 minutes ago)



2021/09/03 08:50:32
Submission Android Expert
Membuat aplikasi game

2021/09/03 08:50:51
Submission Android Fundamental
Resolve issue halaman detail

2021/09/03 08:51:35
Submission iOS
Menambahkan module Android untuk multiplatform

Sharing

Quiz

Discussion

Thank You

bangkit