Informatics Ubaya

UBAYA
UNIVERSITAS SURABAYA

ADVANCED NATIVE MOBILE PROGRAMMING

ROOM

WEEK 08
Informatics Engineering
Universitas Surabaya

Before We Begin

- SQLite and Room Library
- Room Annotation
- Data Access Object
- Room Database
- Kotlin Coroutine

SQLite

- SQLite is an open-source relational database
- Embedded by default
- SQlite is Sandbox
- SQlite is low-level access

The Concept

Room Persistence Library

- Abstraction layer over SQLite
- Compile-time verification of SQL queries.
- Convenience annotations that minimize repetitive and error-prone boilerplate code.
- Streamlined database migration paths.
- Room needs at least three file: The Entity class model, the DAO, and the database class

Room Annotation

- Its a way to mark specific part of codes that handleable by Room library
- Room provide a lot of annotation to speed up project programming times
- Example of annotation:
  @Query -> define custom query to db
  @Entity -> mark the class as table on db
  @Insert -> mark the next function as insert action to db, and so on

DAO (Data Access Object)

- DAO provides methods that offer abstract access to your app's database.
- At compile time, Room automatically generates implementations of the DAOs that you define.

Room Database

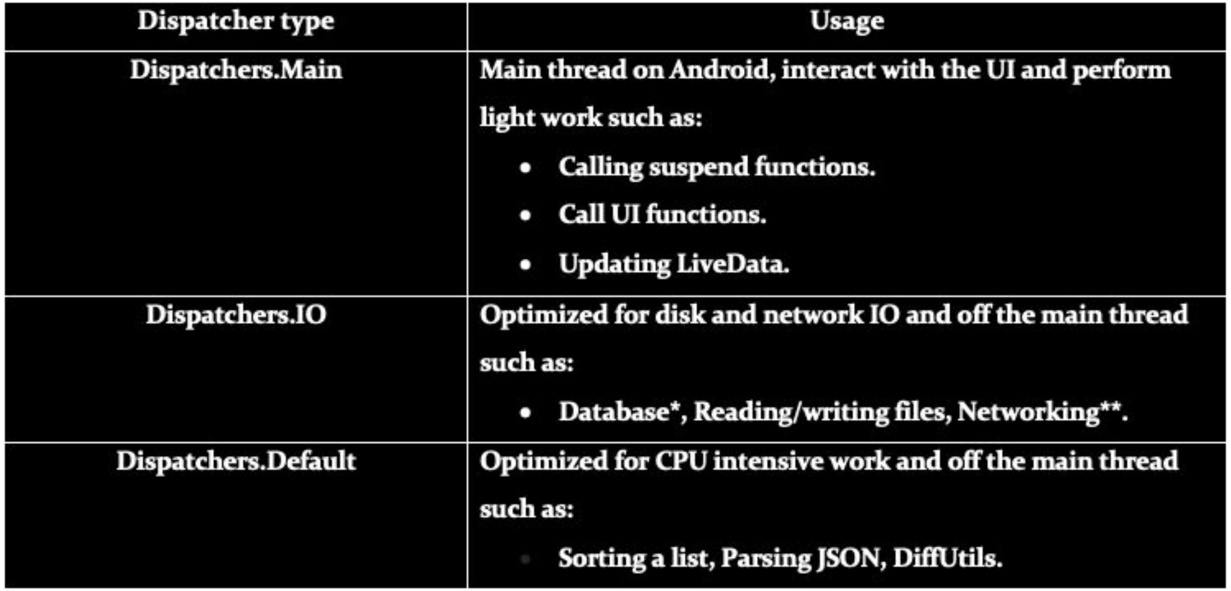
- Base class for all Room databases. All classes that are annotated with Database must extend this class.
- Within this class, you may define database version, migration policy, define tables registered on database, database name, and so on

Kotline Coroutine

- Coroutine manages your long-running tasks in the background thread that might block the main thread and cause your app to freeze
- Coroutines can easily switch threads at any time and pass results back to the original thread

Kotline Coroutine Dispatcher

- There are three different threads in coroutines



Dispatcher type	Usage
Dispatchers.Main	Main thread on Android, interact with the UI and perform light work such as: • Calling suspend functions. • Call UI functions. • Updating LiveData.
Dispatchers.IO	Optimized for disk and network IO and off the main thread such as: • Database*, Reading/writing files, Networking**.
Dispatchers.Default	Optimized for CPU intensive work and off the main thread such as: • Sorting a list, Parsing JSON, DiffUtils.

Setting Up

New Project and Gradle Setting

Create New Project

- Start new Android Studio Project with project name “**TodoApp**”
- Choose “**Empty Views Activity**” template
- Press finish

Setting Up

First we need to setting up dependencies and gradle. Open build.gradle (project)

Add these lines at the topmost:

```
buildscript {
    repositories {
        google()
    }
    dependencies {
        val nav_version = "2.7.1"
        classpath("androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version")
    }
}
```

Don’t forget to click: Sync Now

Setting Up

Next open build.gradle (Module), add following setting (green text)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.jetbrains.kotlin.android)
    id("androidx.navigation.safeargs.kotlin")
    id("kotlin-kapt")
}
```

Kapt (Kotlin Annotation Processing Tool) was a compiler plugin for Kotlin that allowed you to use Java annotation processors in your Kotlin code. Annotation processors are tools that can analyze your code and generate new code or resources based on annotations.

Setting Up

Still in the same file, add following dependencies (modify the version number as suggested by AndStud)

```
implementation("androidx.navigation:navigation-fragment-ktx:2.5.3")
implementation("androidx.navigation:navigation-ui-ktx:2.5.3")
implementation("androidx.room:room-runtime:2.5.1")
implementation("androidx.room:room-ktx:2.5.1")
kapt("androidx.room:room-compiler:2.5.1")
```

Both navigation-fragment-ktx and navigation-ui-ktx can automatically implemented when you create a new Navigation

Room Library

These dependencies are mandatory to work with room library, especially in the Kotlin environment

Setting Up - (For Version Control)

```
[versions]

navigationFragmentKTX = "2.5.3"

navigationUIKTX = "2.5.3"

roomRuntime = "2.5.1"

roomKTX = "2.5.1"

roomCompiler = "2.5.1"


[libraries]

navigation-fragment-ktx = { group ="androidx.navigation", name="navigation-fragment-ktx",
version.ref="navigationFragmentKTX"}

navigation-ui-ktx = { group ="androidx.navigation", name="navigation-ui-ktx", version.ref="navigationUIKTX" }

room-runtime = { group="androidx.room", name="room-runtime", version.ref="roomRuntime" }
room-ktx = { group="androidx.room", name="room-ktx", version.ref="roomKTX" }

room-compiler = { group = "androidx.room", name="room-compiler", version.ref = "roomCompiler" }
```

Setting Up - (For Version Control)

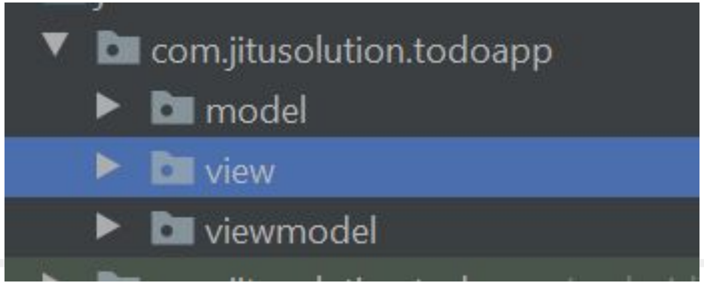
```
dependencies {
    . . .

    implementation(libs.navigation.fragment.ktx)
    implementation(libs.navigation.ui.ktx)
    implementation(libs.room.runtime)
    implementation(libs.room.ktx)
    kapt(libs.room.compiler)
}
```

Fragments

MVVM Package

- Start by create three new package (right click on current package > new >package file). Name those packages as “model”, “view”, “viewmodel”
- Move “MainActivity” into “view” package



Fragments

TodoApp only requires two fragment. One for display list of todo, and other for create new todo.

Also we need two drawable vector asset for FAB button and edit button

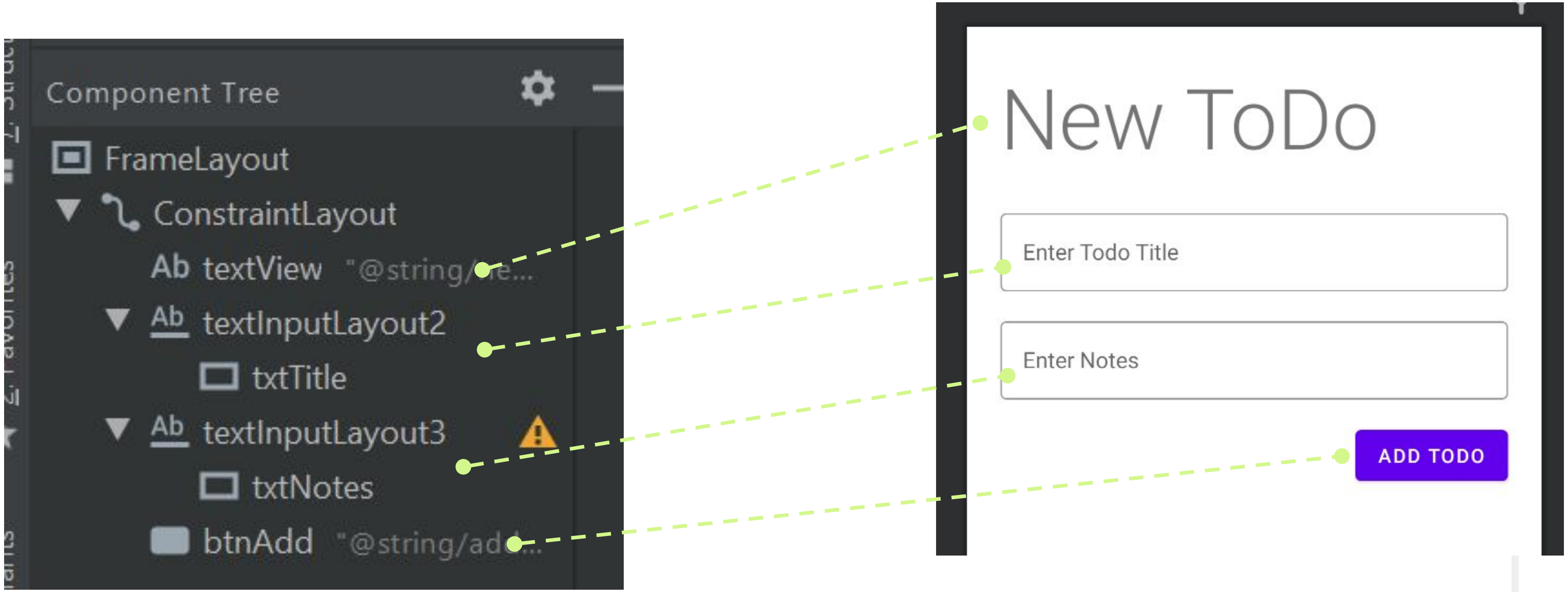
Create those fragments inside view package. Name it as “TodoListFragment” and “CreateTodoFragment”

Clean Up Fragments

Clean up unnecessary codes in both Fragments:

- Delete everything except onCreateView method
- Add override onViewCreated

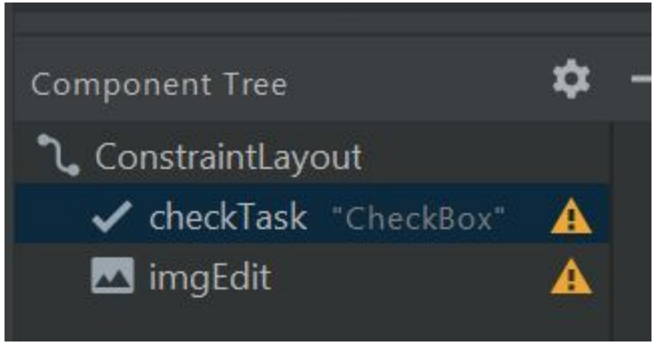
Fragments Layout - CreateTodoFragment



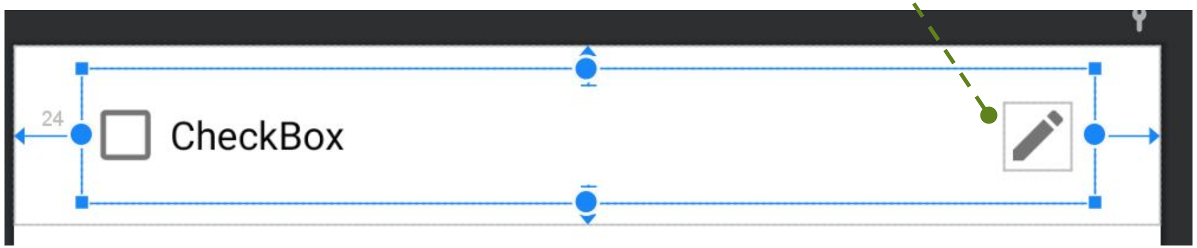
Item Todo Layout

- We need another layout for RecyclerView List to display all todo’s
- Right click on res > layout > new > layout resource file
- Name it as todo_item_layout

Fragments Layout - ItemTodoLayout

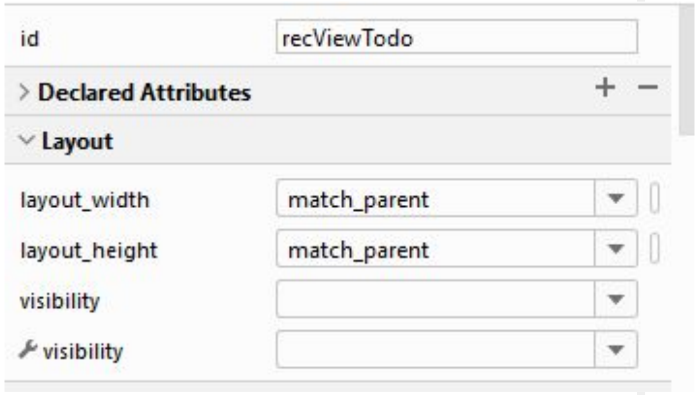


imgEdit requires "pencil"/ edit icon. You may create new on with drawable vector asset



Todo List Layout

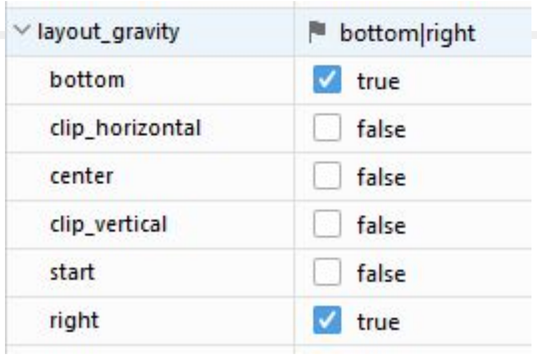
- Now open the todo list layout (fragment_todo_list.xml)
- Drag and drop the **recycler view** into the screen
- Set id to **recViewTodo**
- Set **layout_width** and **layout_height** to **match_parent**



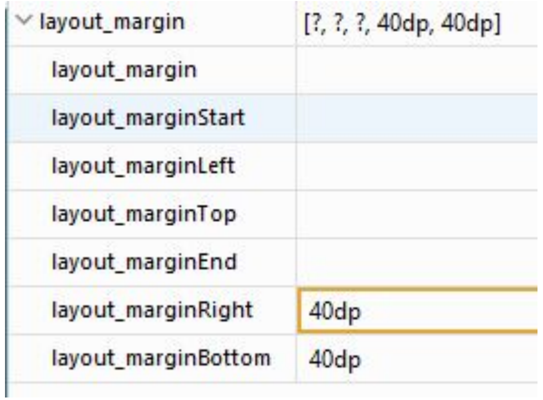
id	recViewTodo
Declared Attributes	
Layout	
layout_width	match_parent
layout_height	match_parent
visibility	
visibility	

Todo List Layout

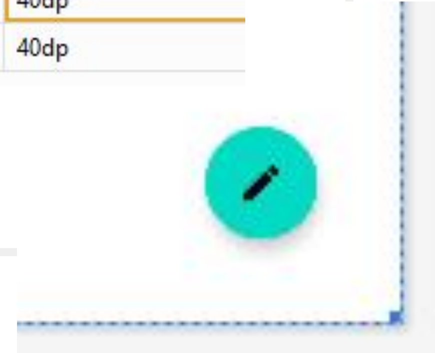
- Add **FAB** (Floating Action Button), set id to **btnFab**
- You can use any icon as you like
- Set **layout_gravity** to **right** and **bottom** (because we use frame layout her)
- Set **margin** to **right** and **bottom** about **40 dp**



layout_gravity	bottom|right
bottom	✓ true
clip_horizontal	false
center	false
clip_vertical	false
start	false
right	✓ true

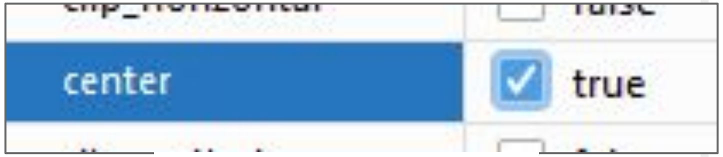


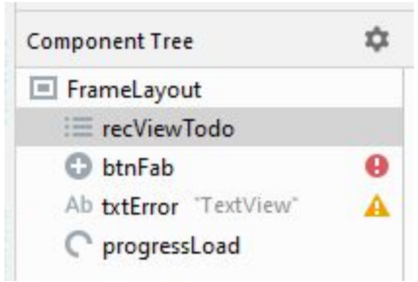
layout_margin	[?, ?, ?, 40dp, 40dp]
layout_margin	
layout_marginStart	
layout_marginLeft	
layout_marginTop	
layout_marginEnd	
layout_marginRight	40dp
layout_marginBottom	40dp



Todo List Layout

- Add TextView set id to txtError
- Set layout_gravity to center
- Add ProgressBar set id to progressLoad
- Set layout_gravity to center

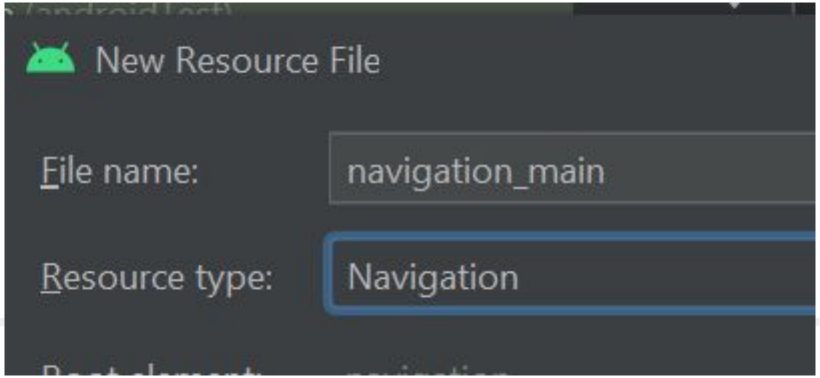

center ✓ true


Component Tree
FrameLayout
recViewTodo
btnFab
txtError "TextView"
progressLoad

Navigation Graph

Navigation Graph

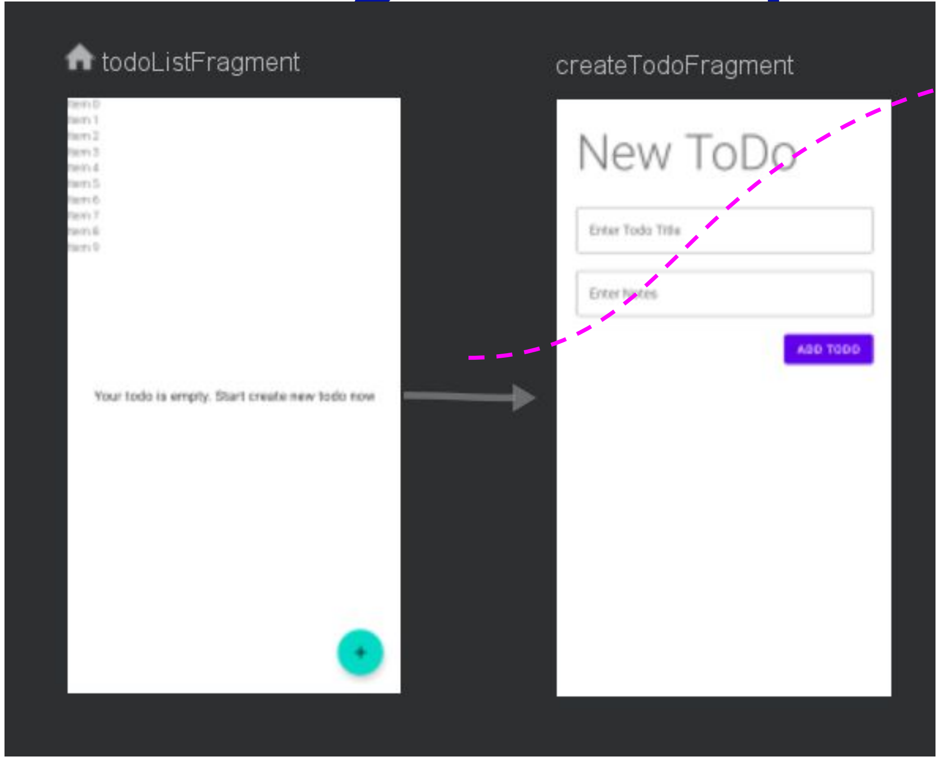
- Right click on res > new > Android Resource File
- Set file name as “navigation_main”
- Set resource type as “Navigation”
- Press OK



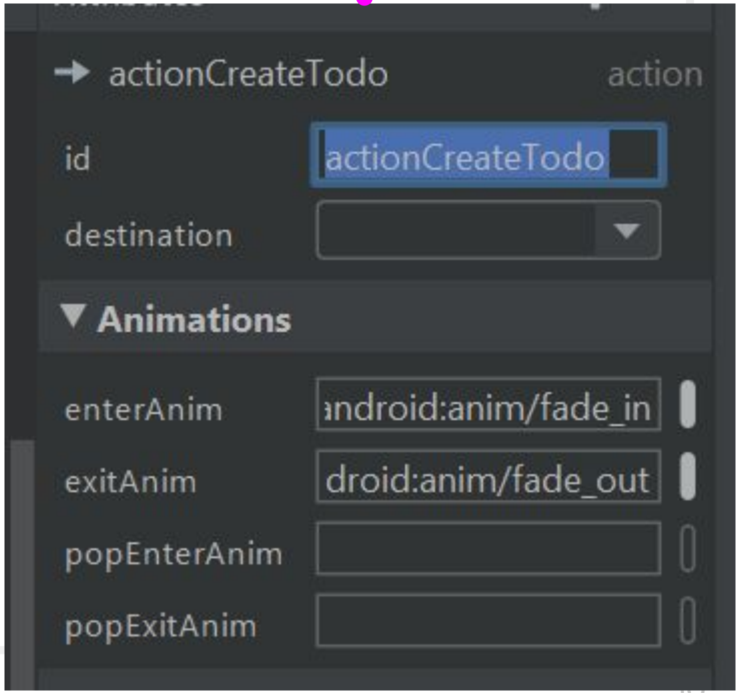
File name: navigation_main

Resource type: Navigation

Draw Navigation Graph



Action (arrow) properties


actionCreateTodo action

id actionCreateTodo

destination

Animations

enterAnim ındroid:anim/fade_in

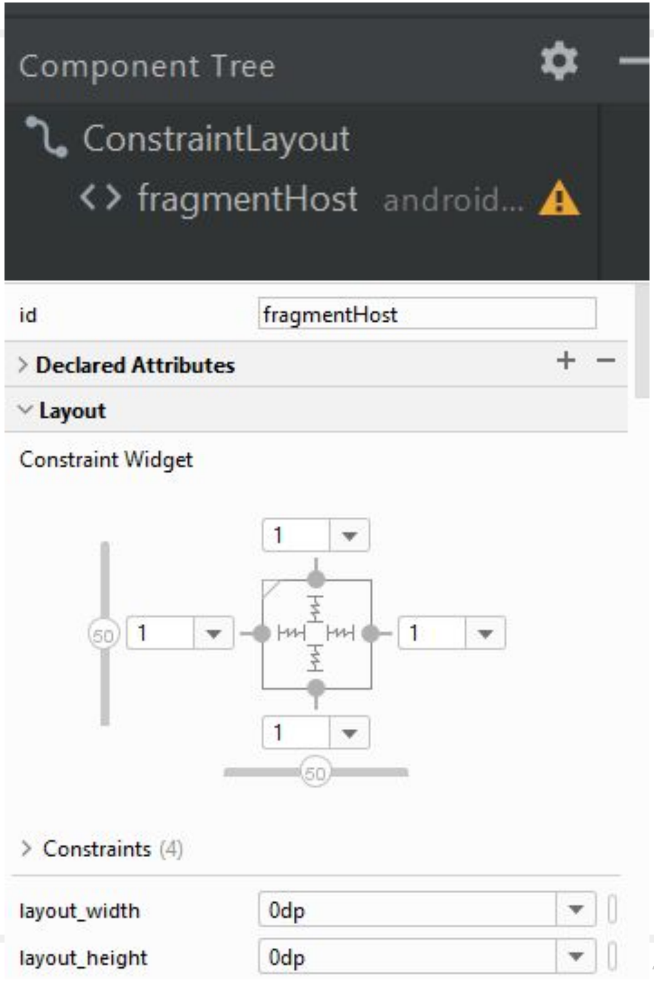
exitAnim droid:anim/fade_out

popEnterAnim

popExitAnim

Main Activity Layout

- Open the MainActivity layout
- Drag and drop **NavHost Fragment** into Main Activity layout
- Choose **navigation_main**
- Set Id to **fragmentHost**, set layout width and height to **0dp**



id: fragmentHost

Declared Attributes

Layout

Constraint Widget

Constraints (4)

layout_width: 0dp

layout_height: 0dp

The Model

Model.kt

- Right click on “Model” package and create new Kotlin file “Model.kt”
- Create Todo class

```
data class Todo(
   var title:String,
   var notes:String
)
```

Implement Room

- Add “entity” annotation on the Todo class to marks this class as an entity.
- This class will have a mapping SQLite table in the database

Implement Room

```
@Entity
data class Todo(
    @ColumnInfo(name="title")
    var title:String,
    @ColumnInfo(name="notes")
    var notes:String
    ) {
    @PrimaryKey(autoGenerate = true)
    var uuid:Int =0
}
```

Entity Annotation

Room will create “todo” table in database

ColumInfo Annotation

specify a column name for the field in table database

PrimaryKey Annotation

Each entity class requires one primary key. Autogenerate config set to true to let SQLite generate the unique id

DAO (Data Access Object)

- Right click on model package > new > Kotlin file/class
- Pick interface
- Set it name as "TodoDao"

Implement Room Dao

```
@Dao
interface TodoDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertAll(vararg todo:Todo)

    @Query("SELECT * FROM todo")
    fun selectAllTodo(): List<Todo>

    @Query("SELECT * FROM todo WHERE uuid= :id")
    fun selectTodo(id:Int): Todo

    @Delete
    fun deleteTodo(todo:Todo)
}
```

Dao Annotation

Marks the class as a Data Access Object. Data Access Objects are the main classes where you define your database interactions. They can include a variety of query methods.

Insert Annotation

The implementation of the method will insert its parameters into the database.

Implement Room Dao

```
@Dao
interface TodoDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertAll(vararg todo:Todo)

    @Query("SELECT * FROM todo")
    fun selectAllTodo(): List<Todo>

    @Query("SELECT * FROM todo WHERE uuid= :id")
    fun selectTodo(id:Int): Todo

    @Delete
    fun deleteTodo(todo:Todo)
}
```

Free Query

This query is verified at compile time by Room to ensure that it compiles fine against the database.

Free Query with Parameter

The arguments of the method will be bound to the bind arguments in the SQL statement.

Todo Database with Singleton Concept

- Right click on model package > new > Kotlin file/class
- Pick Class
- Set it name as TodoDatabase

Implement Room Database

```
@Database(entities = arrayOf(Todo::class), version =  1)
abstract class TodoDatabase:RoomDatabase() {
    abstract fun todoDao(): TodoDao

    companion object {
    }
}
```

Database Annotation

Marks a class as a RoomDatabase. Entities config is used to put entity (table) inside database. Version is used for migration purpose.

Implement Singleton

```
companion object {
        @Volatile private var instance: TodoDatabase ?= null
        private val LOCK = Any()

        fun buildDatabase(context:Context) =
              Room.databaseBuilder(
                   context.applicationContext,
                   TodoDatabase::class.java,
                   "newtododb").build()
}
```

Volatile

meaning that writes to this field are immediately made visible to other threads

Database Builder

Database Builder requires context, database class, and database name as String

Implement Singleton

```
companion object {

        . . .

        operator fun invoke(context:Context) {
            if(instance == null) {
                synchronized(LOCK) {
                    instance ?: buildDatabase(context).also {
                        instance = it
                    }
                }
            }
        }
}
```

Invoke

special methode that triggered when the object is initialized

Singleton Implementation

restricts the instantiation of a class to one "single" instance

Synchronized

A thread that enters a synchronized method obtains a lock (an object being locked is the instance of the containing class) and no other thread can enter the method until the lock is released.

ViewModel

Create ViewModels

- Create viewmodel for both fragments
- Right click on viewmodel package new > Kotlin class/file
- Name it as ListTodoViewModel
- Choose class, press OK

TodoListViewModel

```
class ListTodoViewModel(application: Application)
               :AndroidViewModel(application), CoroutineScope {

    val todoLD = MutableLiveData<List<Todo>>()
    val todoLoadErrorLD = MutableLiveData<Boolean>()
    val loadingLD = MutableLiveData<Boolean>()
    private var job = Job()

    override val coroutineContext: CoroutineContext
        get() = job + Dispatchers.IO
```

Coroutine Scope

Its a way to provide control for coroutine

Dispatcher

tell you, on which thread should I run this block of code. i.e(Main, IO, Default)

TodoListViewModel

```
fun refresh() {
        loadingLD.value = true
        todoLoadErrorLD.value = false
        launch {
             val db = TodoDatabase.buildDatabase(
                getApplication()
            )


            todoLD.postValue(db.todoDao().selectAllTodo())
            loadingLD.postValue(false)
        }
}
```

Launch

launch() function to create a job. The launch is known as fire and forgets builder.

postValue

Used postValue to asynchronously handle expected result from DAO.

Call Room DAO

Todo list livedata is populated with data selected from Database.

TodoListViewModel

```
    fun clearTask(todo: Todo) {
            launch {
                val db = TodoDatabase.buildDatabase(
                    getApplication()
                )
                db.todoDao().deleteTodo(todo)


                todoLD.postValue(db.todoDao().selectAllTodo())
            }
        }
```

Delete ToDo

Clear task simply delete single todo based on the selected object

Create DetailTodoViewModels

- Right click on viewmodel package new > Kotlin class/file
- Name it as DetailTodoViewModel
- Choose class, press OK

DetailTodoListViewModel

```
class DetailTodoViewModel(application:  Application)
        :AndroidViewModel(application), CoroutineScope {
    private val job = Job()

    fun addTodo(list:List<Todo>) {
        launch {
            val db = TodoDatabase.buildDatabase(
                getApplication()
            )
            db.todoDao().insertAll(*list.toTypedArray())
        }
    }
```

***list**

Convert individual element of list into its individual object (Todo object) and set it as separated parameter

DetailTodoListViewModel

```
    override val coroutineContext: CoroutineContext
            get() = job + Dispatchers.IO
```

View - Create Todo

Activate ViewBinding

Open gradle script, add following config within android { } scope (don't forget to press sync now):

```
buildFeatures {
        viewBinding = true
}
```

Open CreateTodoFragment

Activate View Binding in the Create Todo Fragment

```
class CreateTodoFragment : Fragment() {
    private lateinit var binding:FragmentCreateTodoBinding

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

Open CreateTodoFragment

Add following viewmodel instance

```
private lateinit var viewModel:DetailTodoViewModel
private lateinit var binding:FragmentCreateTodoBinding


override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel =
         ViewModelProvider(this).get(DetailTodoViewModel::class.java)


}
```

Open CreateTodoFragment

```
binding.btnAdd.setOnClickListener {
      var todo = Todo(
                  binding.txtTitle.text.toString(),
                  binding.txtNotes.text.toString()
                 )
      val list = listOf(todo)
      viewModel.addTodo(list)
      Toast.makeText(view.context, "Data added", Toast.LENGTH_LONG).show()
      Navigation.findNavController(it).popBackStack()
}
```

View - Adapter

Create Adapter for RecyclerView

- Right click on view package new > Kotlin class/file
- Name it as TodoListAdapter
- Choose class, press OK

Create Adapter

```
class TodoListAdapter(val todoList:ArrayList<Todo>)
     :RecyclerView.Adapter<TodoListAdapter.TodoViewHolder>() {
    class TodoViewHolder(var binding: TodoItemLayoutBinding):
                            RecyclerView.ViewHolder(binding.root)


     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
TodoViewHolder { }


      override fun onBindViewHolder(holder: TodoViewHolder, position: Int)
      { }


      override fun getItemCount(): Int { }
}
```

Create Adapter

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoViewHolder
{
        var binding = TodoItemLayoutBinding.inflate(
                        LayoutInflater.from(parent.context), parent,false)
        return TodoViewHolder(binding)
}


override fun getItemCount(): Int {
        return todoList.size
}
```

Create Adapter

```
override fun onBindViewHolder(holder: TodoViewHolder, position: Int) {
        holder.binding.checkTask.text = todoList[position].title
}



 fun updateTodoList(newTodoList: List<Todo>) {
        todoList.clear()
        todoList.addAll(newTodoList)
        notifyDataSetChanged()
    }
```

View - Todo List

Open TodoListFragment

Activate View Binding as usual

```
class TodoListFragment : Fragment() {
    private lateinit var binding:FragmentTodoListBinding
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentTodoListBinding.inflate(inflater,container,false)
        return  binding.root
    }
```

Open TodoListFragment

Add following viewmodel instance & adapter

```
private lateinit var viewModel:ListTodoViewModel
private val todoListAdapter  = TodoListAdapter(arrayListOf())
private lateinit var binding:FragmentTodoListBinding

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

        viewModel = ViewModelProvider(this).get(ListTodoViewModel::class.java)
        viewModel.refresh()
        binding.recViewTodo.layoutManager = LinearLayoutManager(context)
        binding.recViewTodo.adapter = todoListAdapter
}
```

Open TodoListFragment

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

        binding.btnFab.setOnClickListener {
            val action = TodoListFragmentDirections.actionCreateTodo()
            Navigation.findNavController(it).navigate(action)
        }

        observeViewModel()

}
```

ObserveViewModel

```
fun observeViewModel() {
        viewModel.todoLD.observe(viewLifecycleOwner, Observer {
            todoListAdapter.updateTodoList(it)
            if(it.isEmpty()) {
                binding.recViewTodo?.visibility = View.GONE
                binding.txtError.setText("Your todo still empty.")
            } else {
                binding.recViewTodo?.visibility = View.VISIBLE
            }
        })
    }
```

ObserveViewModel - Handle Loading and Error

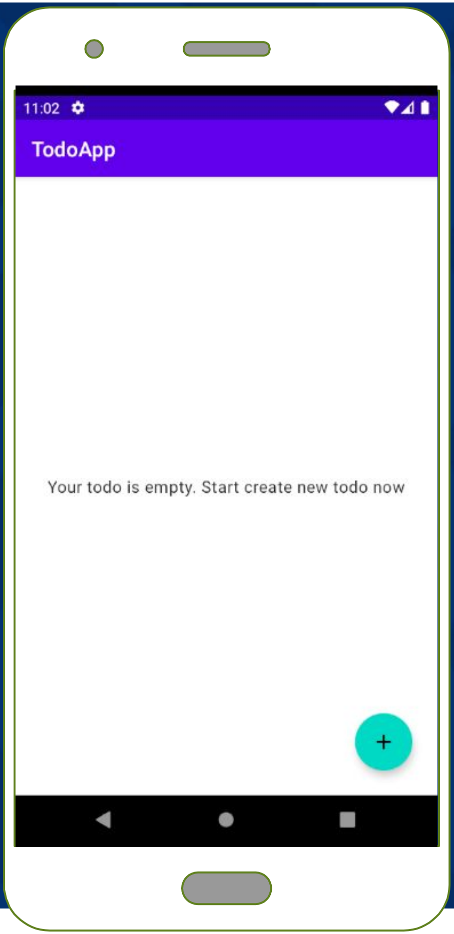
```
viewModel.loadingLD.observe(viewLifecycleOwner, Observer {
    if(it == false) {
        binding.progressLoad?.visibility = View.GONE
    } else {
        binding.progressLoad?.visibility = View.VISIBLE
    }
})
```

ObserveViewModel - Handle Loading and Error

```
viewModel.todoLoadErrorLD.observe(viewLifecycleOwner, Observer {
    if(it == false) {
        binding.txtError?.visibility = View.GONE
    } else {
        binding.txtError?.visibility = View.VISIBLE
    }
})
```

Result

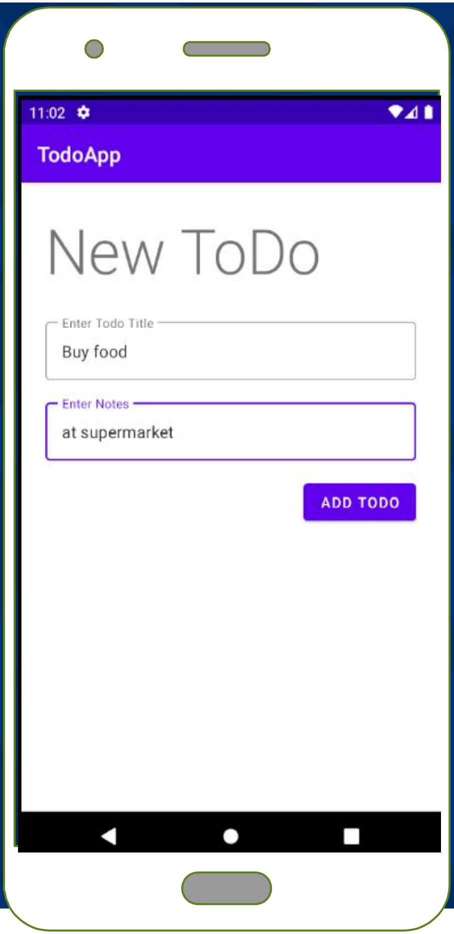
At first its empty



Your todo is empty. Start create new todo now

FAB

Click FAB and add new Todo



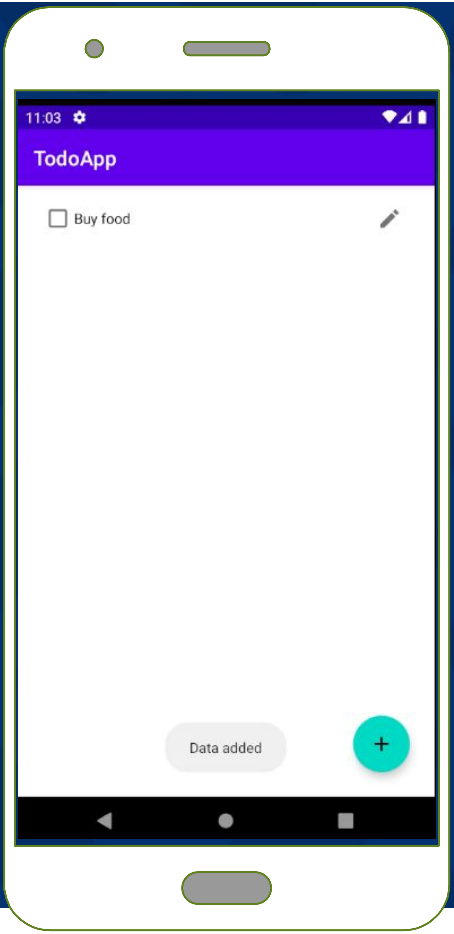
New ToDo

Enter Todo Title: Buy food

Enter Notes: at supermarket

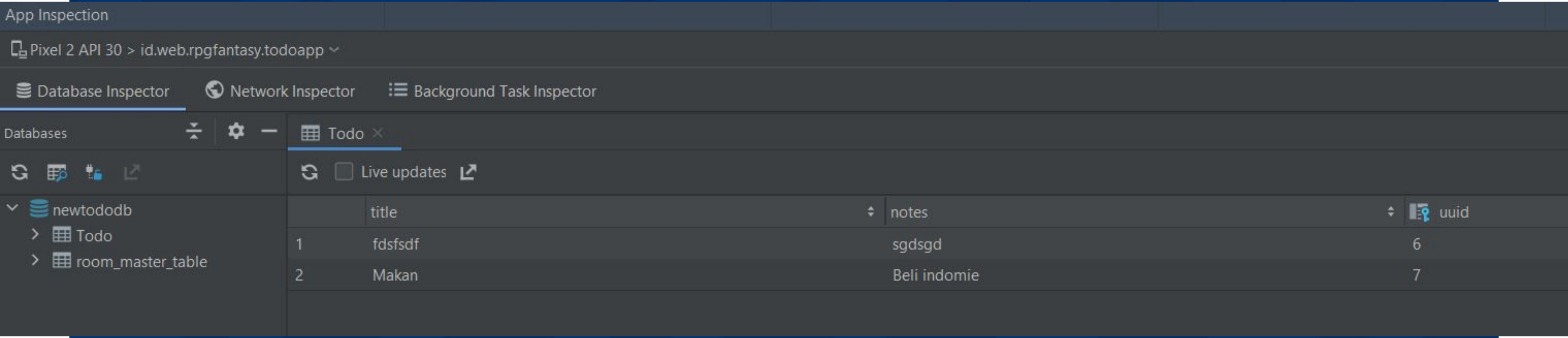
New Todo Added

Displayed as checkable Task



Check With Database Inspector

Click on view > Tools Window > App Inspection, then select Database Inspector. Wait until it is fully loaded



	title	notes	uuid
1	fdsfsdf	sgdsgd	6
2	Makan	Beli indomie	7

Handle Check Task

Handle Check Task

- To clear a todo task, user simply press the checkbox in particular todo
- We already have clearTask function in ViewModel
- However the problem is how to call this function within adapter?
- Solution: interface callback or passing Lambda function

Passing Lambda Function

- Utilize the power of Kotlin
- In Adapter class we can pass a custom Lambda Function as in constructor definition.

```
class TodoListAdapter(
                val todoList:ArrayList<Todo>,
                val adapterOnClick : (Todo) -> Unit)
           :RecyclerView.Adapter<TodoListAdapter.TodoViewHolder>() {
```

Passing Lambda Function

- And call this function if user click on the checkbox

```
override fun onBindViewHolder(holder: TodoViewHolder, position: Int) {
   holder.binding.checkTask.setText(todoList[position].title.toString())

   holder.binding.checkTask.setOnCheckedChangeListener {
          compoundButton, b ->
           if(compoundButton.isPressed) {
               adapterOnClick(todoList[position])
           }
   }
}
```

Passing Lambda Function

- Next define the function in Fragment

```
class TodoListFragment : Fragment() {
    private lateinit var viewModel:ListTodoViewModel
    private val todoListAdapter  = TodoListAdapter(arrayListOf(),
                                        { item -> viewModel.clearTask(item) })
```

Call ClearTask

Inside this lambda function, it safely call clearTask from viewmodel

Thanks.