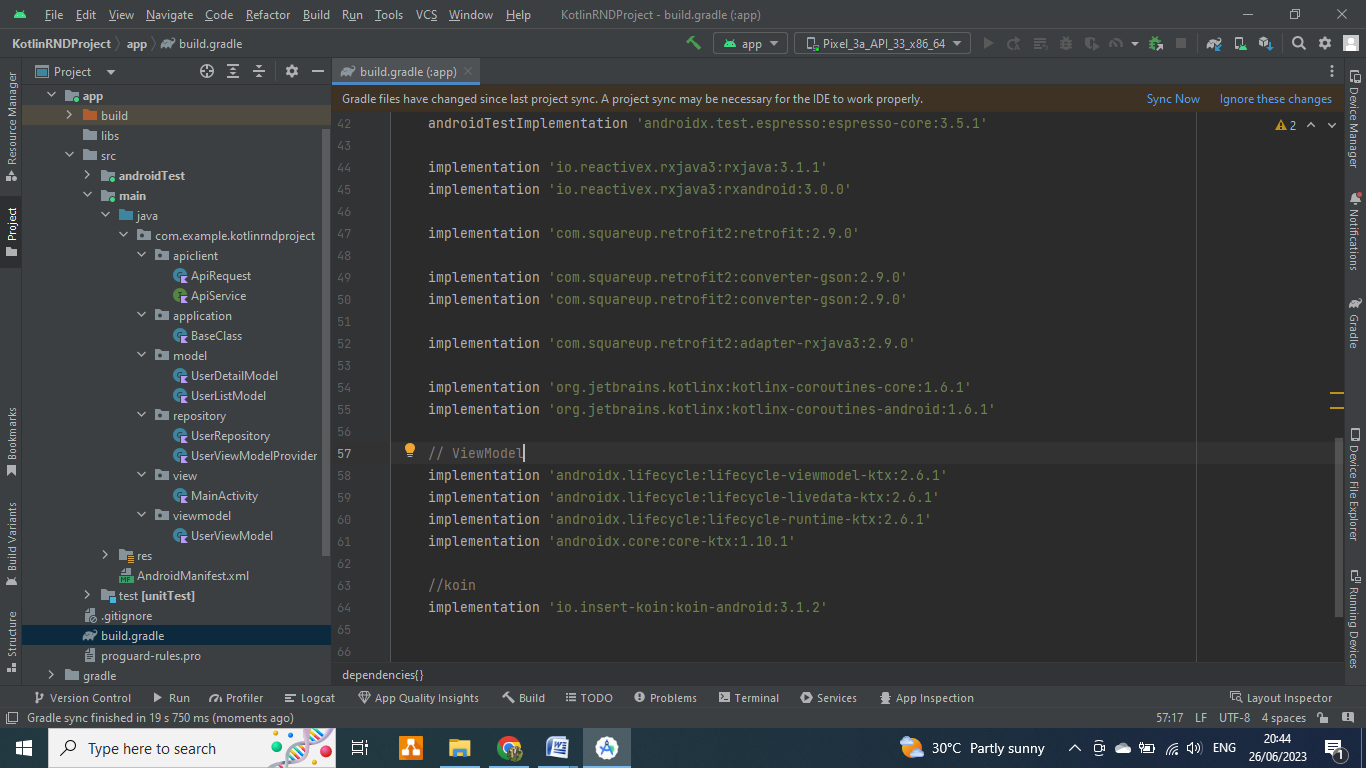
**Depandancy Injection Integration with KOIN**

1st we create package like below

1. **Application** => create application class for inject depandency through KOIN
2. **Model** => The Model layer represents the data and business logic. It includes data classes that define the structure of the data being retrieved from the API, as well as any additional logic or transformations required.
3. **Repository** => The Repository acts as a single source of truth for data. It abstracts the data sources, such as Retrofit API calls or local database queries. The Repository is responsible for coordinating data retrieval and caching.
   1. ViewModelProvider:
   2. Repository:
4. **viewModel** => craete userViewModel class to to co-ordinate with model & view. The ViewModel serves as a bridge between the View and the Model. It retrieves data from the Repository and prepares it for presentation in the View. The ViewModel may expose LiveData or Flow for the View to observe and update the UI accordingly. It also handles user actions and triggers appropriate operations in the Model or Repository
5. **view: Create an activty to show/prsent the data to user**
6. **apiclient:**
   1. **APIInterface:** The API Service is an interface that defines the API endpoints and HTTP methods using Retrofit annotations. It describes how to make network requests and handle responses**.**
   2. **APIRequest:** The Network Client is responsible for creating and configuring the Retrofit instance. It sets up the base URL, converters, interceptors, and other configuration options
7. **Dependency Injection (DI) Modules**: DI modules define how dependencies are provided and managed. They specify the dependencies and their configurations**.**
8. **Dependency Injection (DI) Component:** The DI Component ties everything together by specifying the dependencies and providing them when requested.



1. **Add module label depandency in your gradle file**

implementation 'io.reactivex.rxjava3:rxjava:3.1.1'  
implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'  
  
implementation 'com.squareup.retrofit2:retrofit:2.9.0'  
  
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'  
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'  
  
implementation 'com.squareup.retrofit2:adapter-rxjava3:2.9.0'  
  
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.6.1'  
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.1'  
  
// ViewModel  
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.1'  
implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.6.1'  
implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.6.1'  
implementation 'androidx.core:core-ktx:1.10.1'  
  
//koin  
implementation 'io.insert-koin:koin-android:3.1.2'

1. **Craete two data class**

class UserListModel : ArrayList<UserDetailModel>()

data class UserDetailModel(  
 val completed: Boolean,  
 val id: Int,  
 val title: String,  
 val userId: Int  
)

1. **Create Retrofit for create & manage the retrofit instance**

class ApiRequest {  
  
 private val retrofit: Retrofit = Retrofit.Builder()  
 .baseUrl("https://jsonplaceholder.typicode.com/")  
 .addConverterFactory(GsonConverterFactory.create())  
 .addCallAdapterFactory(RxJava3CallAdapterFactory.create())  
 .build()  
  
 val apiService: ApiService = retrofit.create(ApiService::class.*java*)  
}

1. **Create API Interface Service Class to handle & manage the api request & response**

interface ApiService {  
   
 suspend fun getUsers(): UserListModel  
}

1. **Create User Repository class**

import com.example.kotlinrndproject.apiclient.ApiService  
import com.example.kotlinrndproject.model.UserListModel  
  
class UserRepository (private val apiService: ApiService) {  
  
 suspend fun getUsers(): UserListModel {  
 var isOnlineDataSynck = false  
 // Make a Retrofit API call to fetch users  
 if(isOnlineDataSynck){  
 }else{  
 //userListData = apiService.getUsers();  
 }  
 return apiService.getUsers();  
 }  
}

1. **Craete User ViewModel provider class**

import androidx.lifecycle.ViewModel  
import androidx.lifecycle.ViewModelProvider  
import com.example.kotlinrndproject.viewmodel.UserViewModel  
  
class UserViewModelProvider constructor(private val repository: UserRepository): ViewModelProvider.Factory {  
  
 override fun <T : ViewModel> create(modelClass: Class<T>): T {  
 return if (modelClass.isAssignableFrom(UserViewModel::class.*java*)) {  
 UserViewModel(this.repository) as T  
 } else {  
 throw IllegalArgumentException("ViewModel Not Found")  
 }  
 }  
}

1. **Craete ViewModel class**

import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import com.example.kotlinrndproject.model.UserListModel  
import com.example.kotlinrndproject.repository.UserRepository  
import kotlinx.coroutines.launch  
  
class UserViewModel (private val userRepository: UserRepository) : ViewModel() {  
  
 private val \_usersLiveData = MutableLiveData<UserListModel>()  
 val usersLiveData: LiveData<UserListModel> = \_usersLiveData  
  
 fun fetchUsers() {  
 *viewModelScope*.*launch* **{** val users = userRepository.getUsers()  
 *println*("User Response Data is ${users.get(0).title}")  
 \_usersLiveData.*value* = users  
 **}** }  
}

1. **Application class**

import android.app.Application  
import com.example.kotlinrndproject.apiclient.ApiRequest  
import com.example.kotlinrndproject.apiclient.ApiService  
import com.example.kotlinrndproject.repository.UserRepository  
import com.example.kotlinrndproject.repository.UserViewModelProvider  
import com.example.kotlinrndproject.viewmodel.UserViewModel  
import org.koin.android.ext.koin.androidContext  
import org.koin.androidx.viewmodel.dsl.viewModel  
import org.koin.core.context.GlobalContext.startKoin  
import org.koin.dsl.module  
import retrofit2.Retrofit  
  
class BaseClass : Application() {  
  
  
 val networkModule = *module* **{** single **{** ApiRequest() **}** single **{** get<Retrofit>().create(ApiService::class.*java*) **}  
 }** val repositoryModule = *module* **{** single **{** UserRepository(get()) **}** single **{** UserViewModelProvider(get()) **}  
 }** val viewModelModule = *module* **{** *viewModel* **{** UserViewModel(get()) **}  
 }** override fun onCreate() {  
 super.onCreate()  
  
 startKoin **{** *androidContext*(*applicationContext*)  
 modules(networkModule, repositoryModule, viewModelModule)  
 **}** }  
}

1. **In Your actvity class**

import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
  
import com.example.kotlinrndproject.R  
import com.example.kotlinrndproject.apiclient.ApiRequest  
import com.example.kotlinrndproject.repository.UserViewModelProvider  
import com.example.kotlinrndproject.repository.UserRepository  
import com.example.kotlinrndproject.viewmodel.UserViewModel  
  
import androidx.lifecycle.Observer  
import androidx.lifecycle.ViewModelProvider  
import kotlinx.coroutines.GlobalScope  
import kotlinx.coroutines.delay  
import kotlinx.coroutines.launch  
  
  
class MainActivity : AppCompatActivity() {  
  
 lateinit var userviewModel : UserViewModel  
 private val retrofitService = ApiRequest().apiService  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 userviewModel = ViewModelProvider(this, UserViewModelProvider(UserRepository(retrofitService))).get(UserViewModel::class.*java*)  
  
 // userviewModel = ViewModelProvider(this).get(UserViewModel::class.java)  
 userviewModel.usersLiveData.observe(this, *Observer* **{** user **->** // Update the UI with user data  
 *println*("observe Start emiting data")  
 for(item in user) {  
 val name = item.title  
 //println("Name of user is $name")  
 }  
 *println*("observe end emiting data")  
 **}**)  
  
 userviewModel.fetchUsers()  
  
 GlobalScope.*launch* **{** delay(5000);  
 userviewModel.fetchUsers()  
 delay(5000)  
 userviewModel.fetchUsers()  
 **}** }  
}