Dependency Injection

To add dependency management with Hilt or Dagger for the **SmartHomeController** app using Android Kotlin, here are the necessary steps:

**1. Add Dependencies**

Add the following to your app’s build.gradle file:

otlin

CopyEdit

plugins {

id(“kotlin-kapt”)//annotation processing

id(“com.google.dagger.hilt.android”)//hilt implementation

}

android {

…

}

dependencies {

implementation(“com.google.dagger:hilt-android:2.51.1”)//Hilt core

kapt(“com.google.dagger:hilt-android-compiler:2.51.1”)//Hilt compiler

}

// Allow references to generated code

kapt {

correctErrorTypes = true

}

In the project top-level build.gradle:

kotlin

CopyEdit

plugins {

...

id("com.google.dagger.hilt.android") version "2.51.1" apply false

}

**2. Enable Hilt**

Annotate your Application class with @HiltAndroidApp:

kotlin

CopyEdit

@HiltAndroidApp

class MyApplication : Application()

**3. Inject Dependencies**

* **Define a Module**: A module tells Hilt how to provide an instance of a dependency.

kotlin

CopyEdit

@Module

@InstallIn(SingletonComponent::class)

object AppModule {

@Provides

@Singleton

fun provideRepository(): MyRepository {

return MyRepositoryImpl()

}

}

* **Inject the Dependency**: Use @Inject to request the dependency in your classes.

kotlin

CopyEdit

@AndroidEntryPoint

class MyActivity : AppCompatActivity() {

@Inject

lateinit var repository: MyRepository

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

repository.doSomething()

}

}

**4. Hilt in ViewModel**

Hilt provides a ViewModel integration.

* Add Hilt ViewModel dependency, app’s build.gradle.kts:

kotlin

CopyEdit

implementation "androidx.hilt:hilt-lifecycle-viewmodel:1.0.0-alpha03"

kapt "androidx.hilt:hilt-compiler:1.0.0"

* Example:

kotlin

CopyEdit

@HiltViewModel

class MyViewModel @Inject constructor(

private val repository: MyRepository

) : ViewModel() {

fun fetchData() {

repository.getData()

}

}

**Using Dagger (Plain DI)**

While Hilt simplifies Dagger usage, here’s a simple Dagger example for comparison:

* Define a Module:

kotlin

CopyEdit

@Module

class AppModule {

@Provides

fun provideRepository(): MyRepository = MyRepositoryImpl()

}

* Create a Component:

kotlin

CopyEdit

@Component(modules = [AppModule::class])

interface AppComponent {

fun inject(activity: MyActivity)

}

* Inject in Activity:

kotlin

CopyEdit

class MyActivity : AppCompatActivity() {

@Inject lateinit var repository: MyRepository

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

val component = DaggerAppComponent.create()

component.inject(this)

repository.doSomething()

}

}

**Key Hilt Annotations**

* @HiltAndroidApp: Initializes Hilt in your application.
* @AndroidEntryPoint: Adds DI to Android components like Activity, Fragment.
* @Inject: Requests a dependency.
* @Module: Provides dependencies.
* @InstallIn: Defines the scope of the module (e.g., SingletonComponent).

**Benefits of Using Hilt/Dagger**

1. Automatic DI setup for Android-specific classes.
2. Compile-time safety and faster runtime performance.
3. Encourages modular code architecture.
4. Better testability through constructor injection.
5. **Set up the application class**:

kotlin

CopyEdit

@HiltAndroidApp

class SmartHomeControllerApp : Application()

1. **Inject dependencies into Activities or Fragments**: Annotate the class with @AndroidEntryPoint:

kotlin

CopyEdit

@AndroidEntryPoint

class MainActivity : AppCompatActivity() {

@Inject lateinit var someDependency: SomeDependency

}

**Dagger Dependency Management**

1. **Add Dagger dependencies** to your **app-level build.gradle**:

groovy

CopyEdit

plugins {

id 'com.android.application'

id 'kotlin-android'

id 'kotlin-kapt'

}

dependencies {

implementation "com.google.dagger:dagger:2.48"

kapt "com.google.dagger:dagger-compiler:2.48"

}

1. **Create the Dagger Component**:

kotlin

CopyEdit

@Component

interface AppComponent {

fun inject(activity: MainActivity)

}

1. **Set up the application class**:

kotlin

CopyEdit

class SmartHomeControllerApp : Application() {

lateinit var appComponent: AppComponent

override fun onCreate() {

super.onCreate()

appComponent = DaggerAppComponent.create()

}

}

1. **Inject dependencies into Activities or Fragments**:

kotlin

CopyEdit

class MainActivity : AppCompatActivity() {

@Inject lateinit var someDependency: SomeDependency

override fun onCreate(savedInstanceState: Bundle?) {

(application as SmartHomeControllerApp).appComponent.inject(this)

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

}

}

**Recommendation**: For simpler setup and integration, Hilt is recommended since it is built on top of Dagger and provides additional features like scoped components and easier lifecycle integration.

class NavigationManager @Inject constructor() {

private var navController: NavController? = null

fun setNavController(navController: NavController) {

this.navController = navController

}

fun navigateTo(destination: Int, args: Bundle? = null) {

navController?.navigate(destination, args)

}

fun popBackStack() {

navController?.popBackStack()

}

}

@Module

@InstallIn(SingletonComponent::class)

object AppModule {

@Provides

@Singleton

fun provideNavigationManager(): NavigationManager {

return NavigationManager()

}

}

@AndroidEntryPoint

class MainActivity : AppCompatActivity() {

@Inject

lateinit var navigationManager: NavigationManager

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

val navHostFragment =

supportFragmentManager.findFragmentById(R.id.nav\_host\_fragment) as NavHostFragment

navigationManager.setNavController(navHostFragment.navController)

}

}