Anahí Salgado

@anncode

MVVM Clean Architecture and Dagger Hilt

@anncode

Inyección de Dependencias

@anncode

Dependencia

Para que un objeto exista necesita que existа otro u otros

@anncode

Dependencia

Para que un objeto exista necesita que exista otro u otros

Un objeto depende de otro u otros

@anncode

¿Cómo proveemos las dependencias?

¿Cómo proveemos las dependencias?

Podemos pasarlos como parámetros

¿Cómo proveemos las dependencias?

Podemos pasarlos como parámetros

- Constructor
- Método común

@anncode

Inyección de dependencias

Inject ✨

```
data class Agenda(val type: String, val contacts: List<Contact>)
```

Inyección de dependencias

```kotlin
fun createType() {
    type = "Friends"
}
```

```kotlin
fun createContacts(): List<Contact> {
    return listOf(
        Contact( name: "Anahí",  phone: "555-5555"),
        Contact( name: "Mauricio",  phone: "444-4444"),
        Contact( name: "Rodrigo",  phone: "333-3333"),
        Contact( name: "Mirella",  phone: "222-2222")
    )
}
```

```kotlin
data class Agenda(val type: String, val contacts: List<Contact>)
```

@anncode

```kotlin
private lateinit var mainViewModel: MainViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val categoryRepositoryImpl = CategoryRepositoryImpl()
        val getCategoryImpl = GetCategoryImpl(categoryRepositoryImpl)
        val factory = MainViewModel.Factory(getCategoryImpl)
        mainViewModel = ViewModelProvider(this, factory)[MainViewModel::class.java]
        mainViewModel.getCategoryStates().observe(this, Observer {
            when (it) {
                is CategoryState.Loading -> tv_categories.setText(R.string.loading)
                is CategoryState.Categories -> setCategories(it.categories)
                is CategoryState.Error -> tv_categories.text = it.message
            }
        })
    }
```

```kotlin
private lateinit var mainViewModel: MainViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val categoryRepositoryImpl = CategoryRepositoryImpl()
        val getCategoryImpl = GetCategoryImpl(categoryRepositoryImpl)
        val factory = MainViewModel.Factory(getCategoryImpl)
        mainViewModel = ViewModelProvider(this, factory)[MainViewModel::class.java]
        mainViewModel.getCategoryStates().observe(this, Observer {
            when (it) {
                is CategoryState.Loading -> tv_categories.setText(R.string.loading)
                is CategoryState.Categories -> setCategories(it.categories)
                is CategoryState.Error -> tv_categories.text = it.message
            }
        })
    }
```

```kotlin
private lateinit var mainViewModel: MainViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val categoryRepositoryImpl = CategoryRepositoryImpl()
        val getCategoryImpl = GetCategoryImpl(categoryRepositoryImpl)
        val factory = MainViewModel.Factory(getCategoryImpl)
        mainViewModel = ViewModelProvider(this, factory)[MainViewModel::class.java]
        mainViewModel.getCategoryStates().observe(this, Observer {
            when (it) {
                is CategoryState.Loading -> tv_categories.setText(R.string.loading)
                is CategoryState.Categories -> setCategories(it.categories)
                is CategoryState.Error -> tv_categories.text = it.message
            }
        })
    }
```

```kotlin
private lateinit var mainViewModel: MainViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val categoryRepositoryImpl = CategoryRepositoryImpl()
        val getCategoryImpl = GetCategoryImpl(categoryRepositoryImpl)
        val factory = MainViewModel.Factory(getCategoryImpl)
        mainViewModel = ViewModelProvider(this, factory)[MainViewModel::class.java]
        mainViewModel.getCategoryStates().observe(this, Observer {
            when (it) {
                is CategoryState.Loading -> tv_categories.setText(R.string.loading)
                is CategoryState.Categories -> setCategories(it.categories)
                is CategoryState.Error -> tv_categories.text = it.message
            }
        })
    }
```

```kotlin
private lateinit var mainViewModel: MainViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val categoryRepositoryImpl = CategoryRepositoryImpl()
        val getCategoryImpl = GetCategoryImpl(categoryRepositoryImpl)
        val factory = MainViewModel.Factory(getCategoryImpl)
        mainViewModel = ViewModelProvider(this, factory)[MainViewModel::class.java]
        mainViewModel.getCategoryStates().observe(this, Observer {
            when (it) {
                is CategoryState.Loading -> tv_categories.setText(R.string.loading)
                is CategoryState.Categories -> setCategories(it.categories)
                is CategoryState.Error -> tv_categories.text = it.message
            }
        })
    }
```

Dagger

Reutilización
Centralización
Ciclo de vida
de la creación de objetos

Dagger Hilt

Dagger Hilt

@anncode

Dagger Hilt

build.gradle(:app)

```
apply plugin: 'kotlin-kapt'
apply plugin: 'dagger.hilt.android.plugin'

dependencies {

  // Dagger Hilt
  def hilt_version = "2.28-alpha"
  implementation "com.google.dagger:hilt-android:$hilt_version"
  kapt "com.google.dagger:hilt-android-compiler:$hilt_version"

}
```

build.gradle(project)

```
dependencies {
  classpath "com.google.dagger:hilt-android-gradle-plugin:$hilt_version"

}
```

@anncode

Dagger Hilt

build.gradle(:app)

```
apply plugin: 'kotlin-kapt'
apply plugin: 'dagger.hilt.android.plugin'

dependencies {
     // Dagger Hilt
    def hilt_version = "2.28-alpha"
    implementation "com.google.dagger:hilt-android:$hilt_version"
    kapt "com.google.dagger:hilt-android-compiler:$hilt_version"

   // Dagger Hilt for ViewModel
   def hilt_lifecycle = "1.0.0-alpha02"
   implementation "androidx.hilt:hilt-lifecycle-viewmodel:$hilt_lifecycle"
   kapt "androidx.hilt:hilt-compiler:$hilt_lifecycle"

}
```

build.gradle(project)

```
dependencies {
   classpath "com.google.dagger:hilt-android-gradle-plugin:$hilt_version"

}
```

@anncode

Dagger Hilt

```kotlin
@HiltAndroidApp
open class MyApplication: Application()
```

@anncode

Dagger Hilt

```kotlin
@HiltAndroidApp
open class MyApplication: Application()
```

AndroidManifest.xml

```xml
<application
        android:name=".MyApplication"
```

@anncode

Dagger Hilt

```
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
```

Dagger Hilt

```
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
```

Activity
Fragment
View
Service
Broadcast Receiver

Dagger Hilt

```
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
```

Dagger Hilt

```kotlin
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    @Inject lateinit var contacts: List<Contact>
```

@anncode

Dagger Hilt

```
@Inject lateinit var contacts: List<Contact>

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    contacts.forEach {

    }
}
```

Arquitectura MVVM

Dagger Hilt

build.gradle(:app)

```
apply plugin: 'kotlin-kapt'
apply plugin: 'dagger.hilt.android.plugin'

dependencies {
  // Dagger Hilt
  def hilt_version = "2.28-alpha"
  implementation "com.google.dagger:hilt-android:$hilt_version"
  kapt "com.google.dagger:hilt-android-compiler:$hilt_version"




}
```

build.gradle(project)

```
dependencies {
  classpath "com.google.dagger:hilt-android-gradle-plugin:$hilt_version"

}
```

Dagger Hilt

build.gradle(:app)

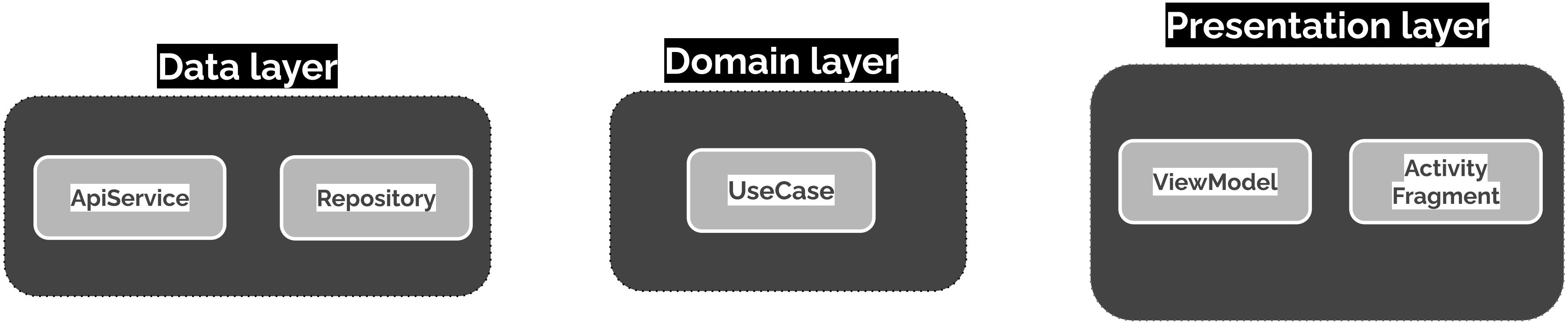
```
apply plugin: 'kotlin-kapt'
apply plugin: 'dagger.hilt.android.plugin'

dependencies {
     // Dagger Hilt
    def hilt_version = "2.28-alpha"
    implementation "com.google.dagger:hilt-android:$hilt_version"
    kapt "com.google.dagger:hilt-android-compiler:$hilt_version"

   // Dagger Hilt for ViewModel
   def hilt_lifecycle = "1.0.0-alpha02"
   implementation "androidx.hilt:hilt-lifecycle-viewmodel:$hilt_lifecycle"
   kapt "androidx.hilt:hilt-compiler:$hilt_lifecycle"

}
```

build.gradle(project)

```
dependencies {
   classpath "com.google.dagger:hilt-android-gradle-plugin:$hilt_version"

}
```

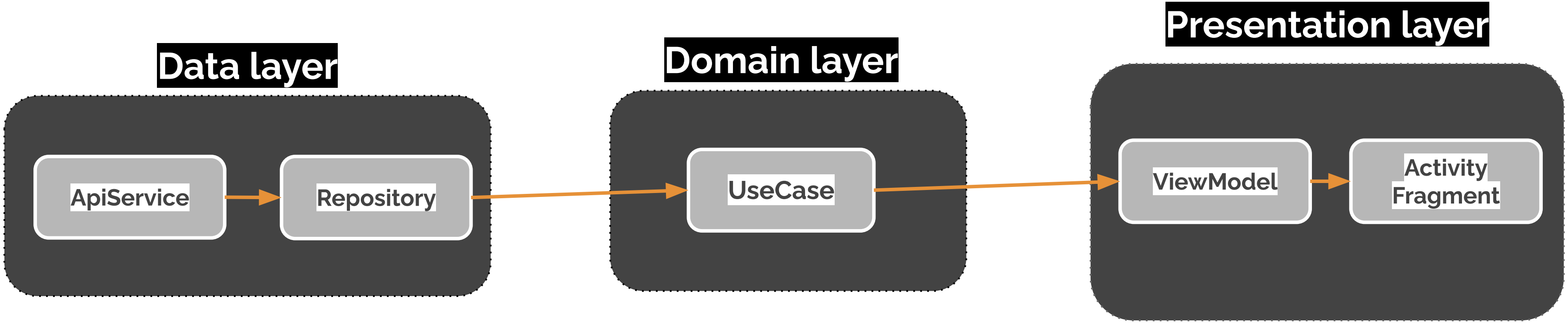
@anncode

Arquitectura



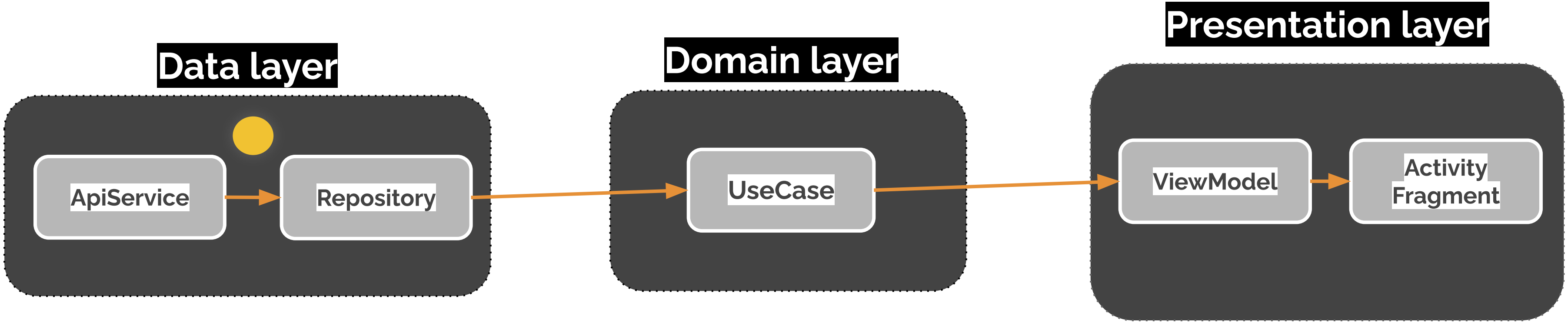
@anncode

Arquitectura



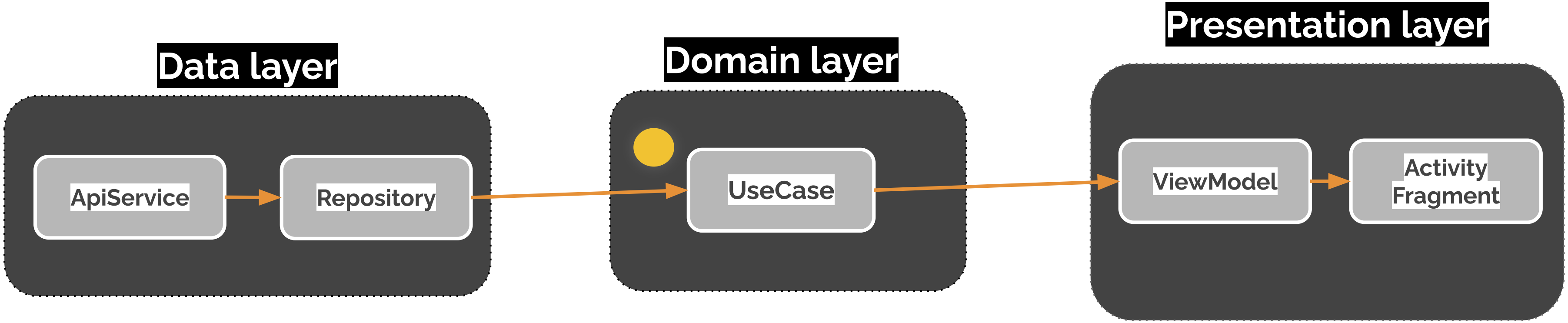
@anncode

Arquitectura



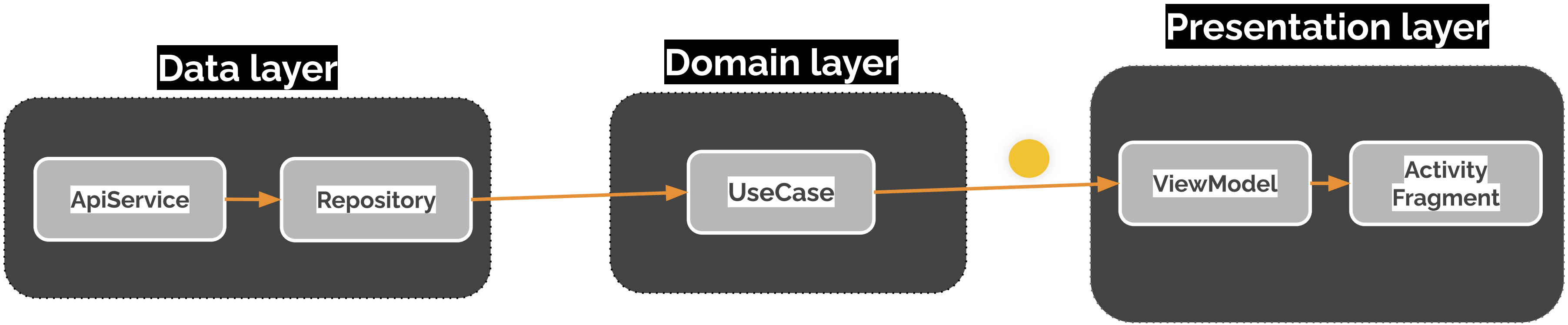
@anncode

Arquitectura



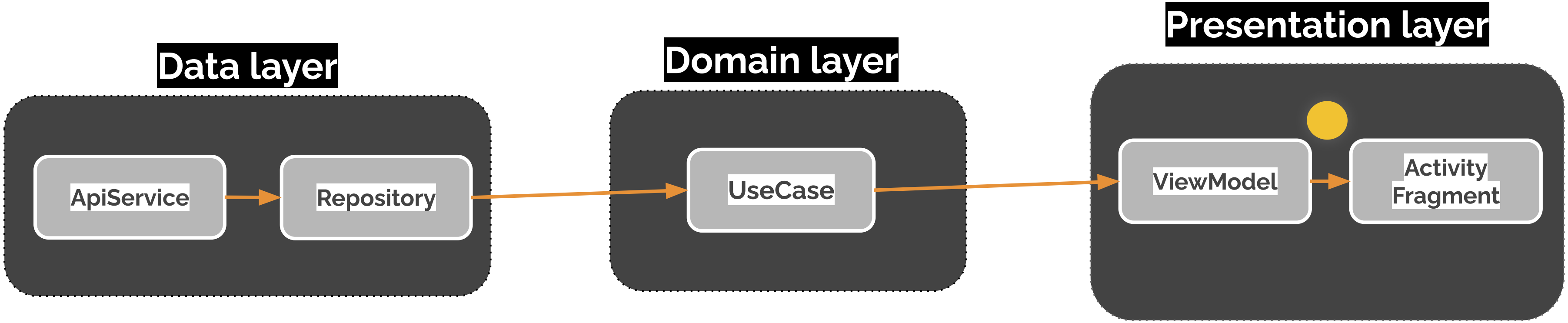
@anncode

Arquitectura



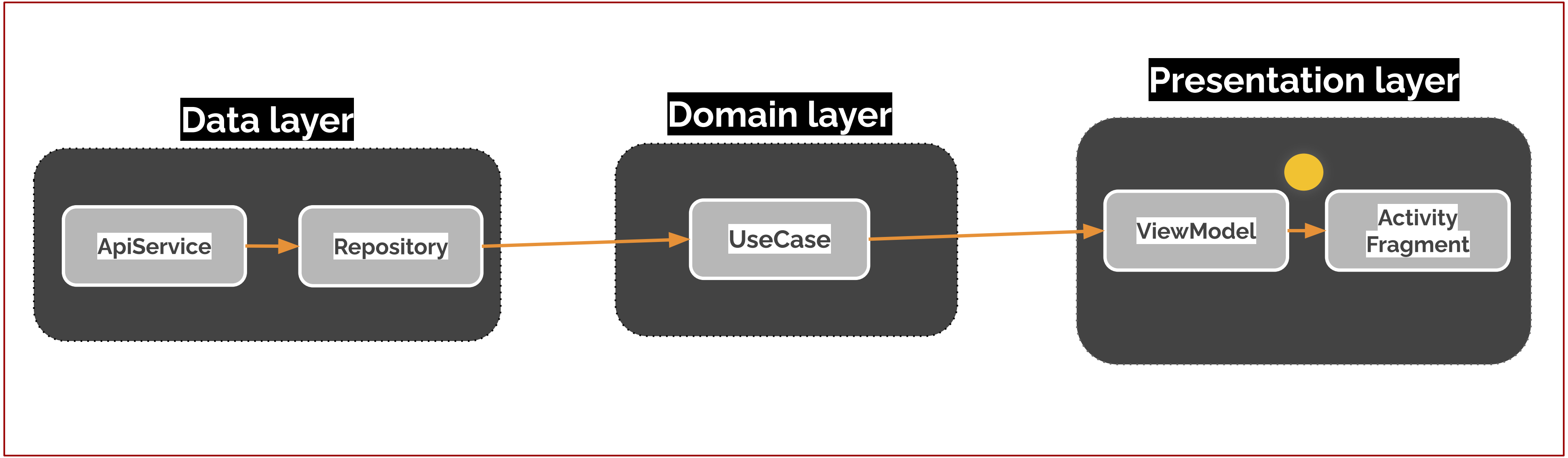
@anncode

Arquitectura



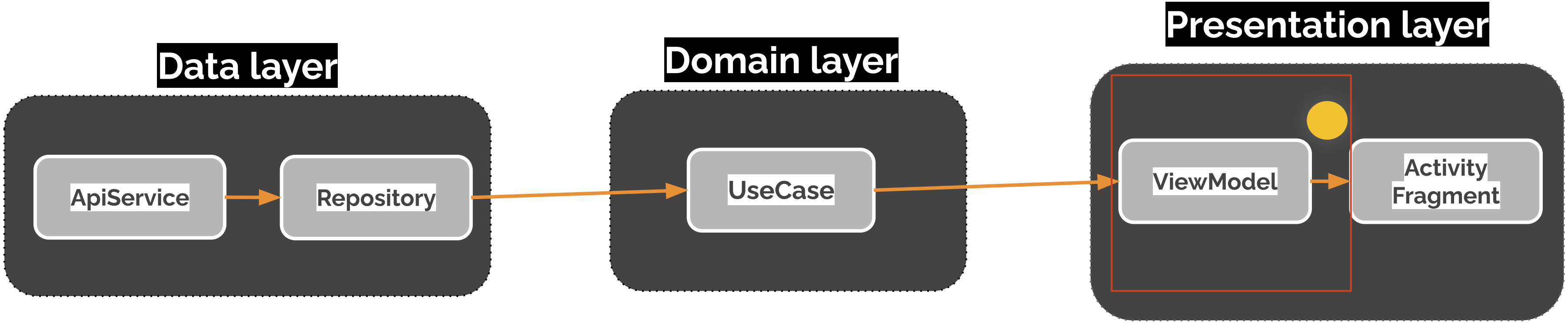
@anncode

Arquitectura



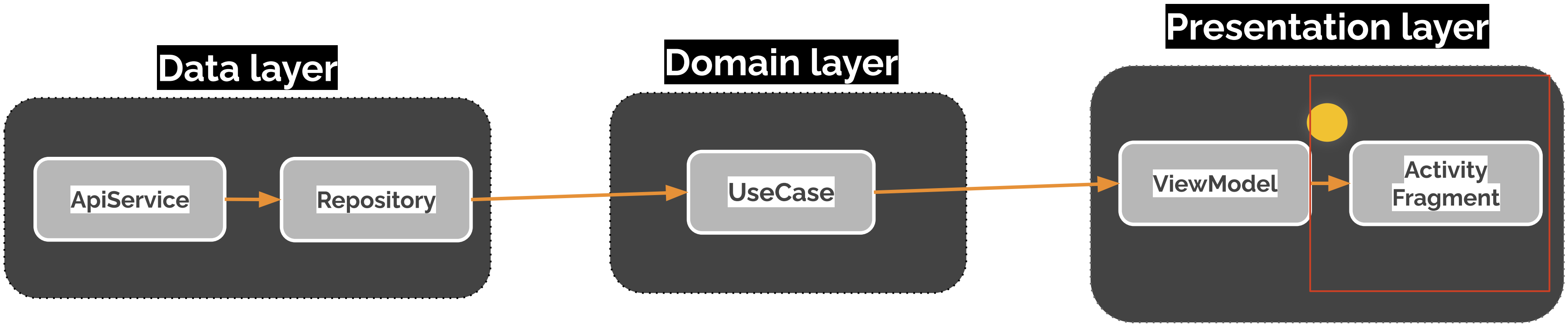
@anncode

Arquitectura



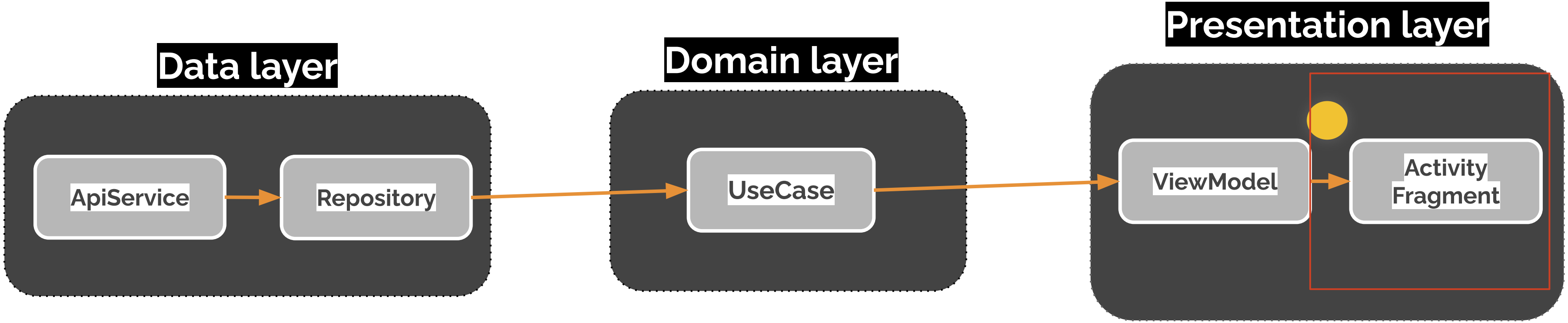
@anncode

Arquitectura



@anncode

Arquitectura



Scope

@anncode

Dagger Hilt

```
@Module
@InstallIn(ApplicationComponent::class)
class MyClassModule {
```

Scope

Dagger Hilt

```
@Module
@InstallIn(ApplicationComponent::class)
class AgendaModule {
```

ApplicationComponent

Application

@anncode

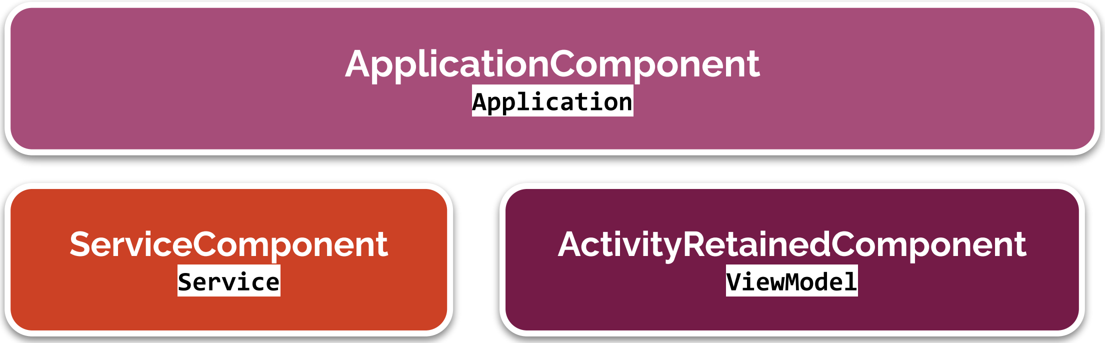
Dagger Hilt

```
@Module
@InstallIn(ServiceComponent::class)
class AgendaModule {
```

@anncode

Dagger Hilt

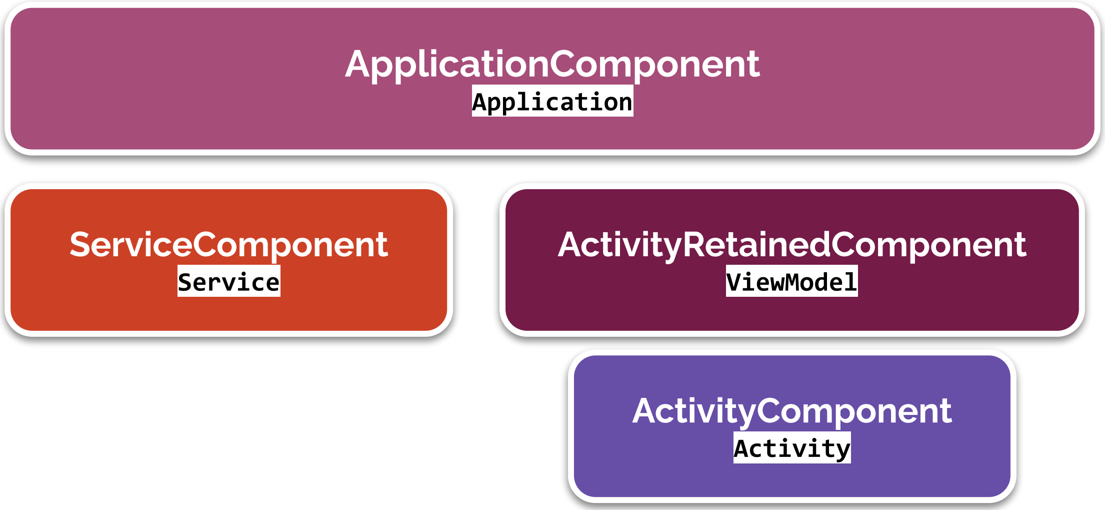
```
@Module
@InstallIn(ActivityRetainedComponent::class)
class AgendaModule {
```



@anncode

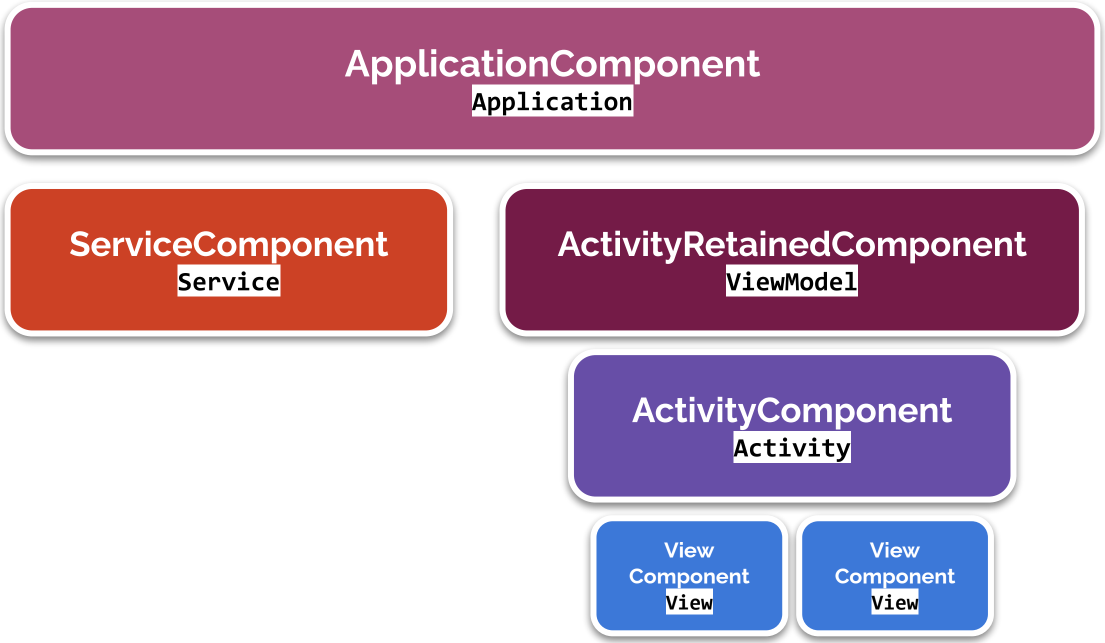
Dagger Hilt

```
@Module
@InstallIn(ActivityComponent::class)
class AgendaModule {
```



Dagger Hilt

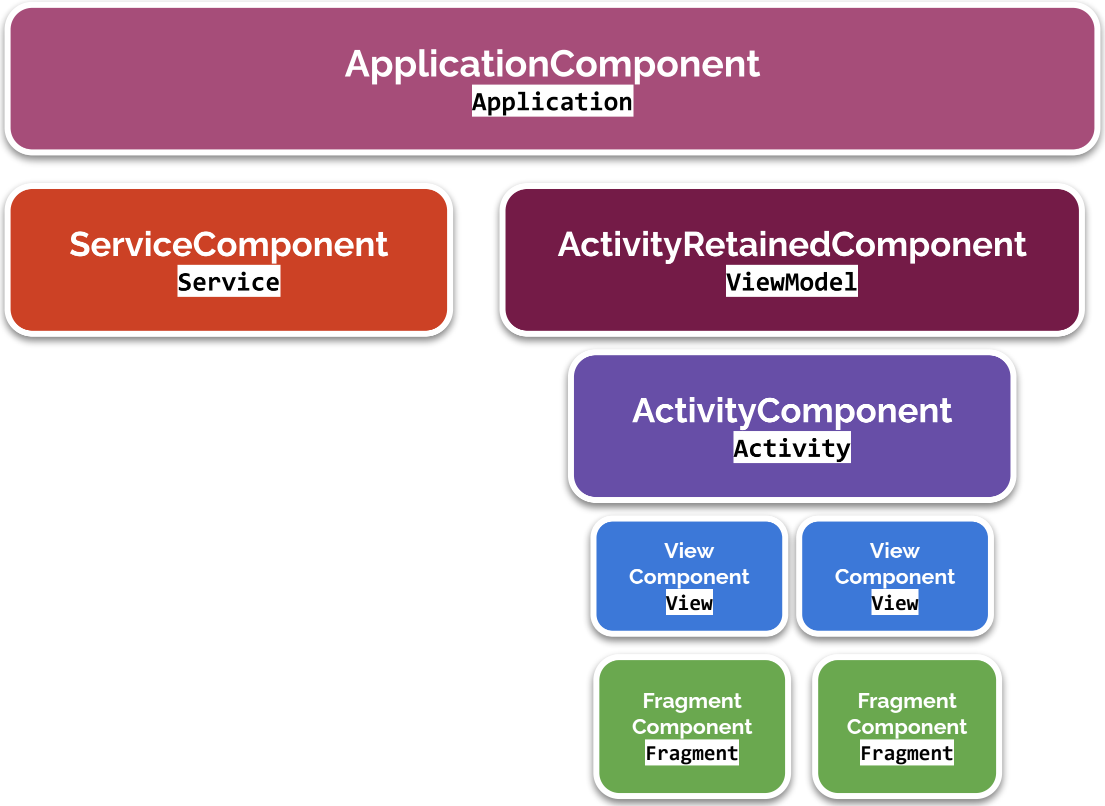
```
@Module
@InstallIn(ViewComponent::class)
class AgendaModule {
```



@anncode

Dagger Hilt

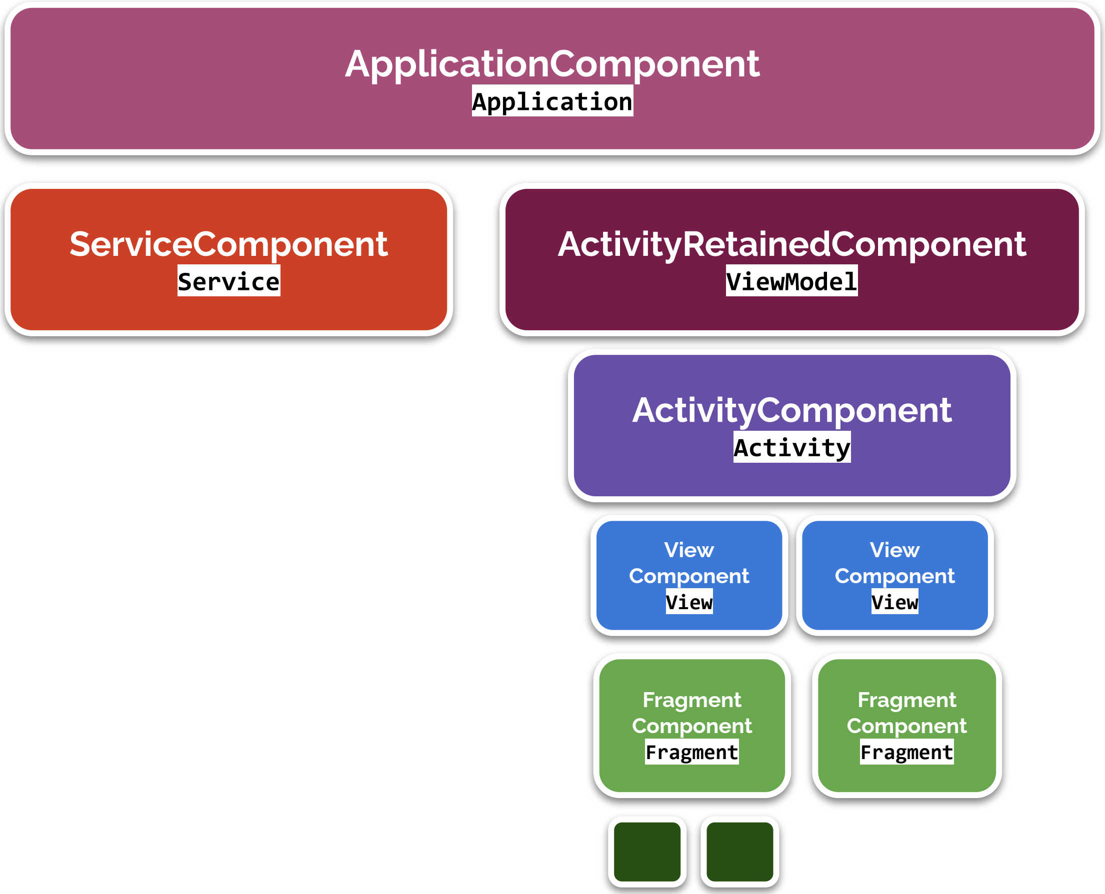
```
@Module
@InstallIn(FragmentComponent::class)
class AgendaModule {
```



@anncode

Dagger Hilt

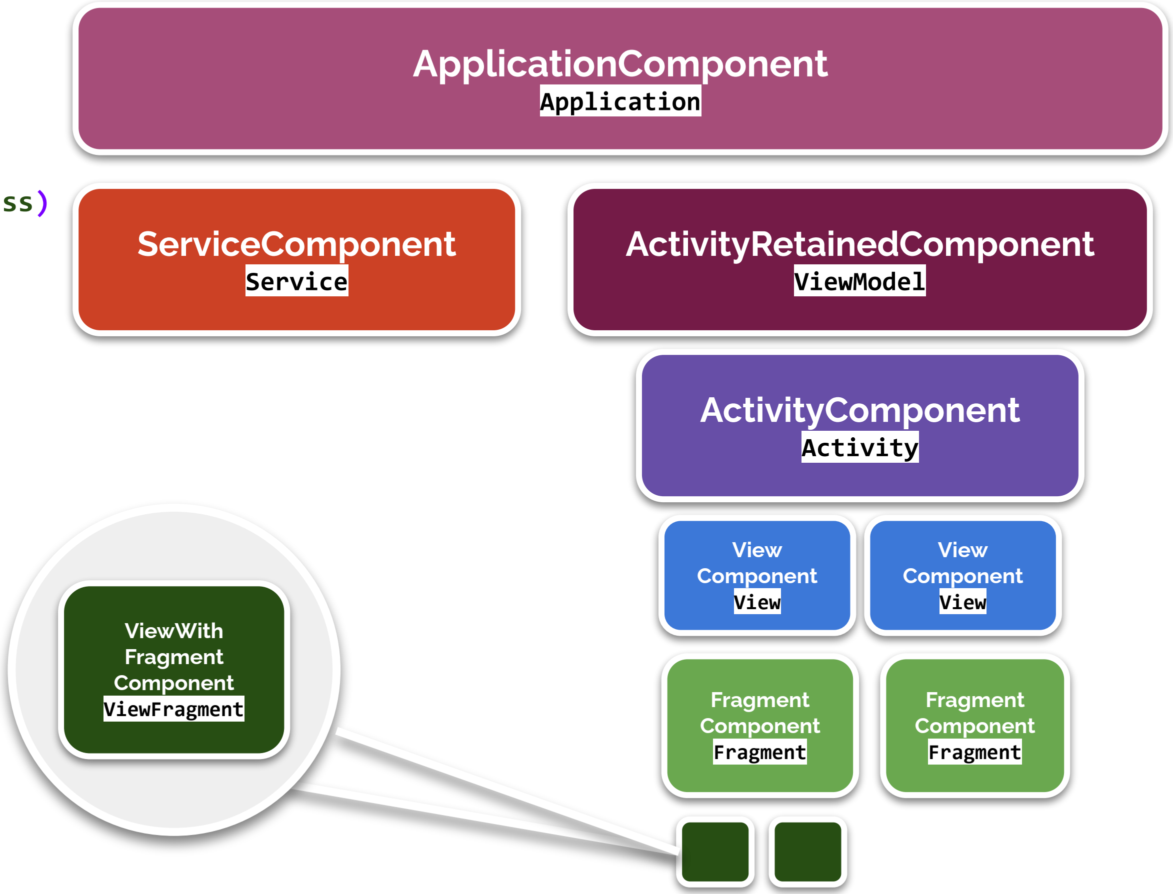
```
@Module
@InstallIn(ViewWithFragmentComponent::class)
class AgendaModule {
```



@anncode

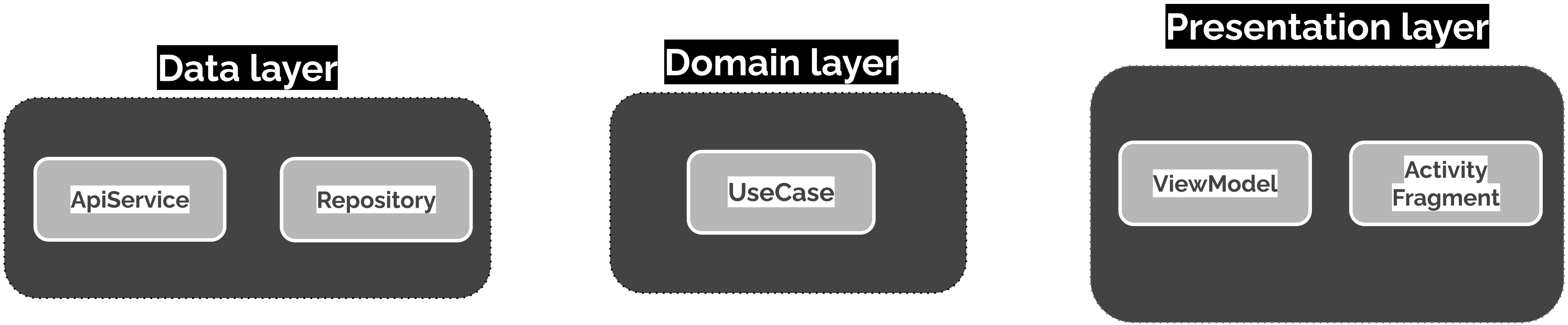
Dagger Hilt

```
@Module
@InstallIn(ViewWithFragmentComponent::class)
class AgendaModule {
```



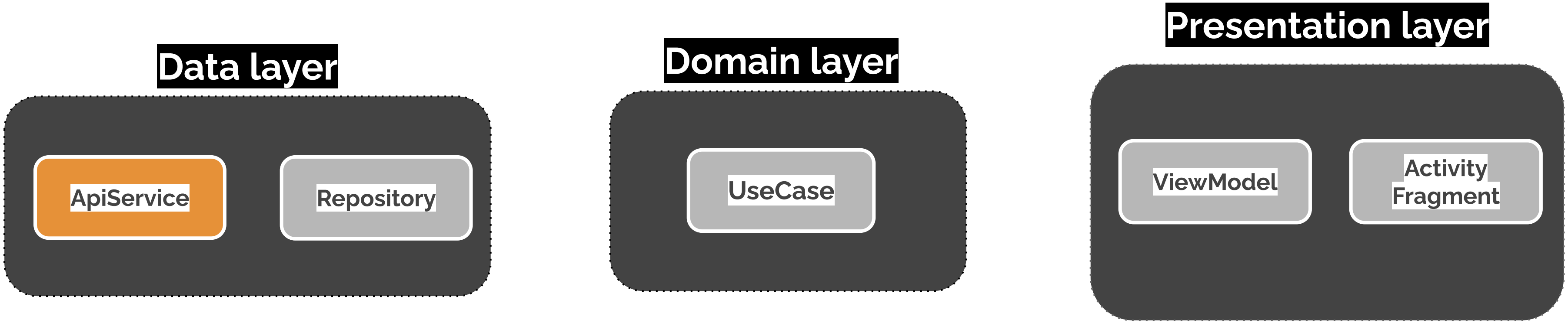
@anncode

Arquitectura



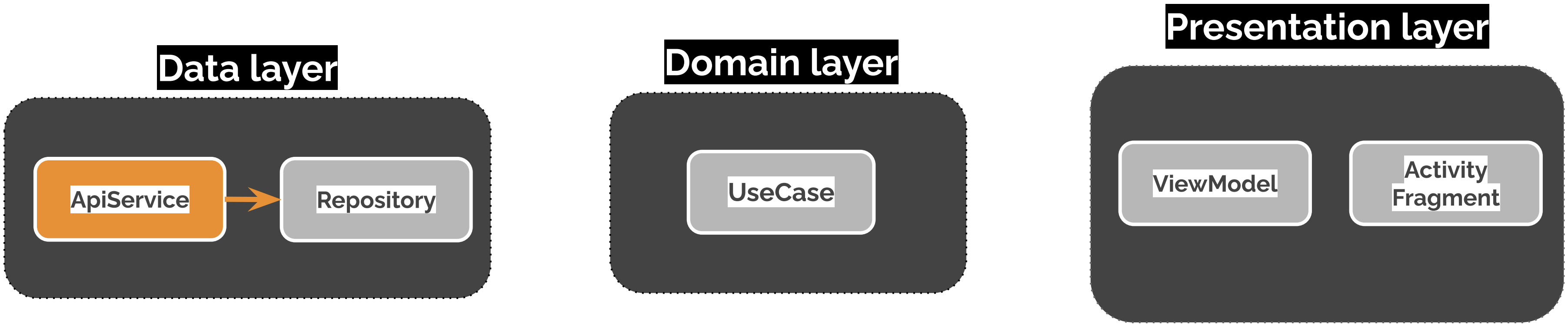
@anncode

Arquitectura



@anncode

Arquitectura



@anncode

Dagger Hilt

ApplicationComponent
`Application`

ActivityRetainedComponent
`ViewModel`

```
@Module
@InstallIn(ApplicationComponent::class)
class NetworkModule {
```

@anncode

Dagger Hilt


```
@Module
@InstallIn(ApplicationComponent::class)
class NetworkModule {

    @Singleton
    @Provides
    fun provideCatFactsApi(): CatFactsService {
        return Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build().create(CatFactsService::class.java)
    }
```

@anncode

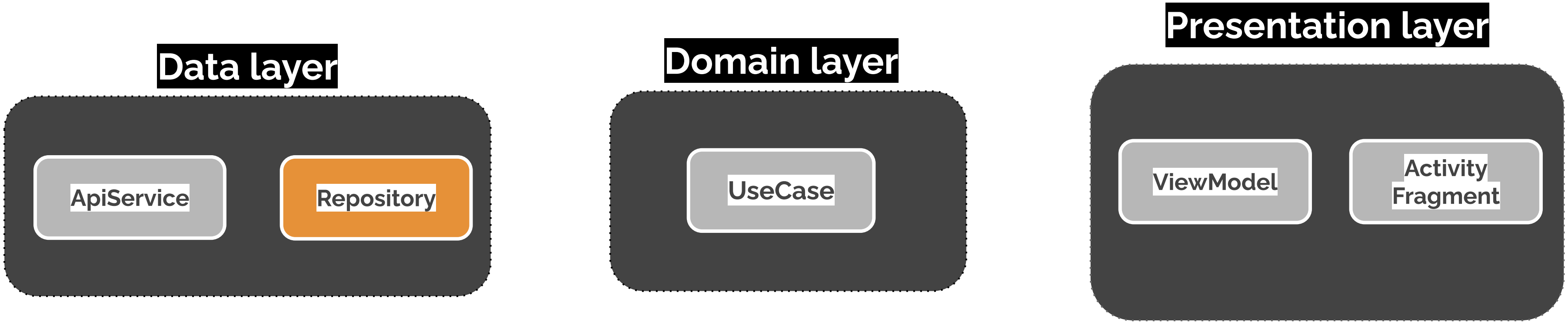
Dagger Hilt

ApplicationComponent
Application

ActivityRetainedComponent
ViewModel

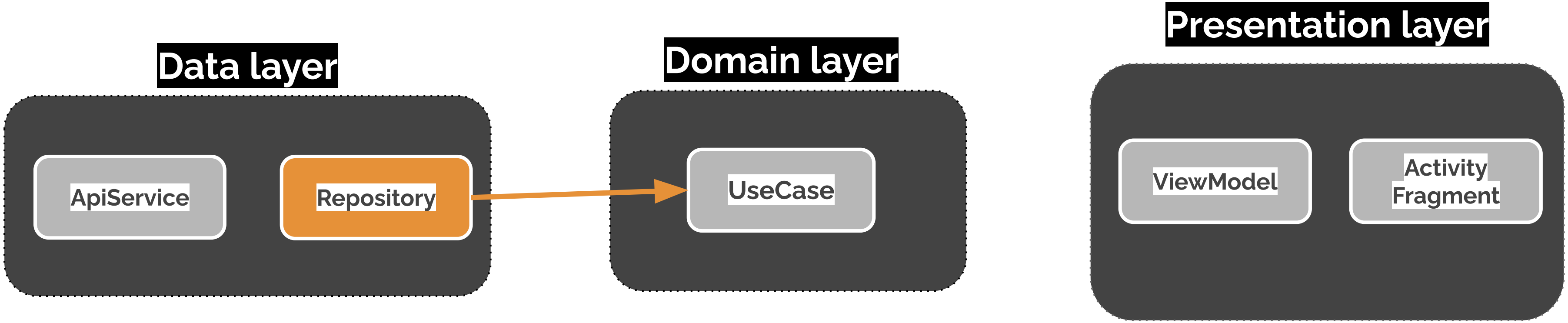
```
@Module
@InstallIn(ApplicationComponent::class)
class NetworkModule {

    @Singleton
    @Provides
    fun provideCatFactsApi(): CatFactsService {
        return Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build().create(CatFactsService::class.java)
    }
```

Arquitectura



@anncode

Arquitectura



@anncode

Dagger Hilt

ApplicationComponent
`Application`

ActivityRetainedComponent
`ViewModel`

```
@Module
@InstallIn(ApplicationComponent::class)
class RepositoryModule {
```

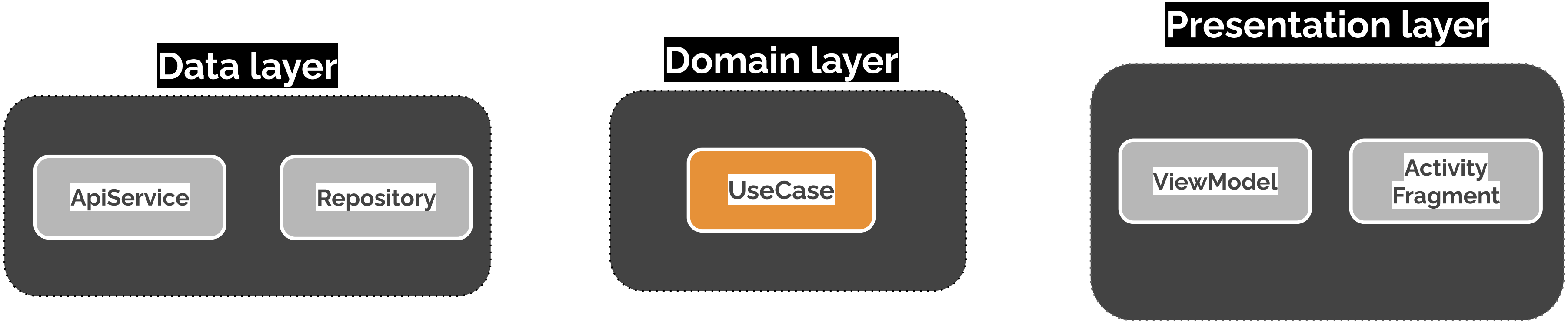
Dagger Hilt

ApplicationComponent
Application

ActivityRetainedComponent
ViewModel

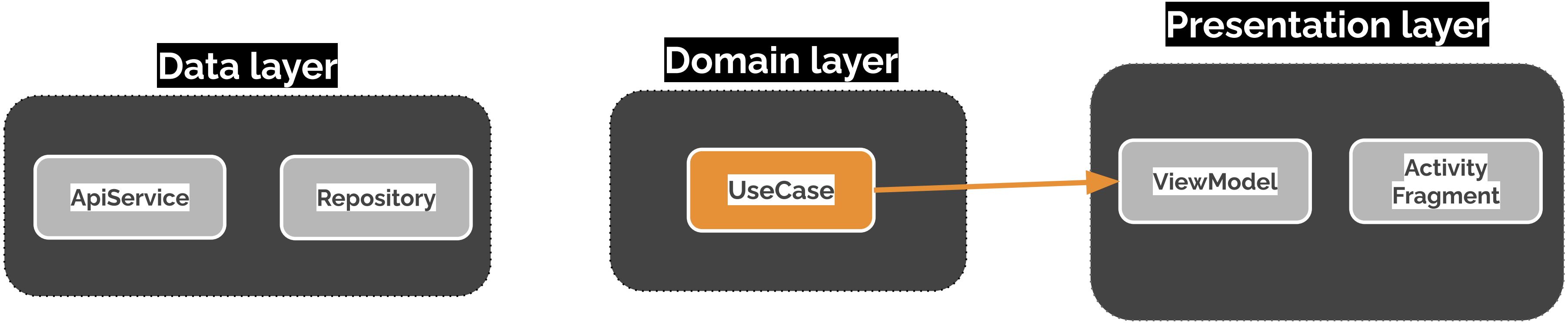
```kotlin
@Module
@InstallIn(ApplicationComponent::class)
class RepositoryModule {

     @Singleton
     @Provides
     fun providesCatFactRepository(
         catFactsService: CatFactsService
     ): CatFactRepository = CatFactRepositoryImpl(catFactsService)
```

Arquitectura



@anncode

Arquitectura



@anncode

Dagger Hilt

```
class UseCaseModule {
```

Dagger Hilt


```kotlin
@Module
@InstallIn(ActivityRetainedComponent::class)
class UseCaseModule {
```

@anncode

Dagger Hilt

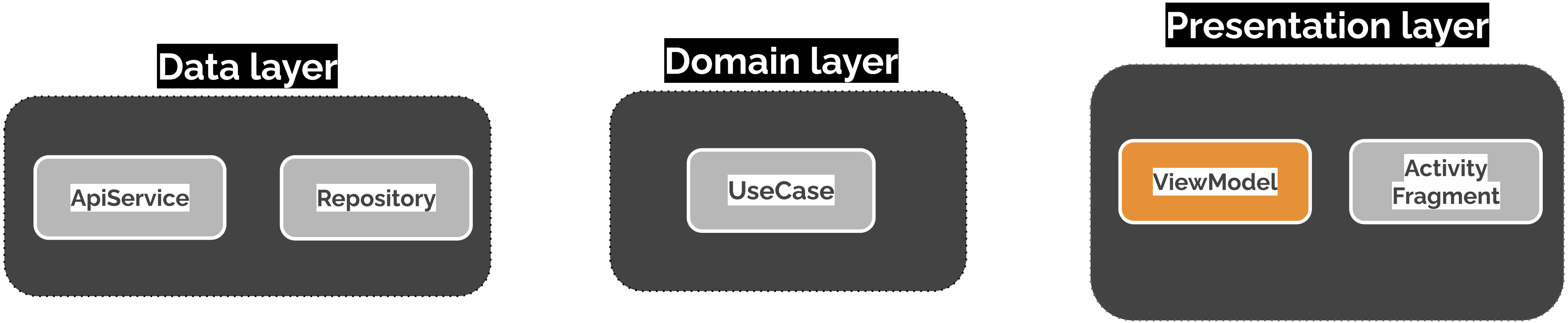
ApplicationComponent
Application

ActivityRetainedComponent
ViewModel

```kotlin
@Module
@InstallIn(ActivityRetainedComponent::class)
class UseCaseModule {

    @Provides
    fun providesGetCatRandomFact(
        catFactRepository: CatFactRepository
    ): GetCatRandomFact = GetCatRandomFactImpl(catFactRepository)
```

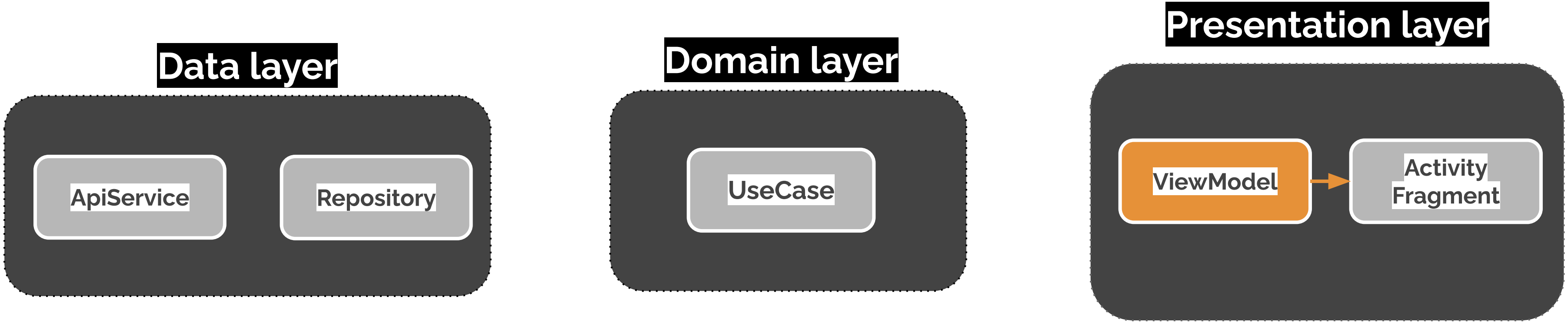
@anncode

Arquitectura



@anncode

Arquitectura



@anncode

Dagger Hilt

```
class ViewModelModule {
```

Dagger Hilt

ApplicationComponent
Application

ActivityRetainedComponent
ViewModel

```
@Module
@InstallIn(ActivityRetainedComponent::class)
class ViewModelModule {
```

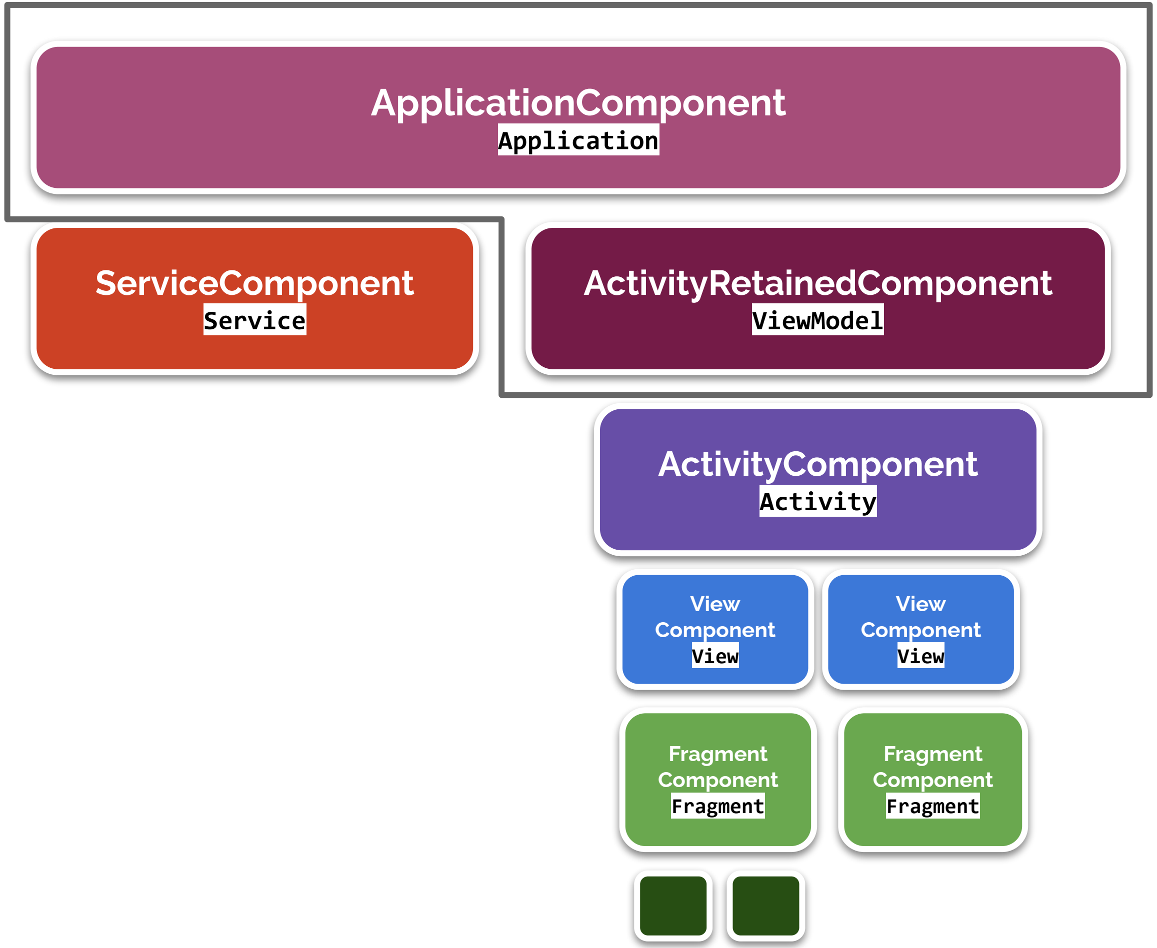
@anncode

Dagger Hilt


```kotlin
@Module
@InstallIn(ActivityRetainedComponent::class)
class ViewModelModule {

    @Provides
    fun provideMainViewModel(
        getCatRandomFact: GetCatRandomFact
    ): MainViewModel = MainViewModel(getCatRandomFact)
```

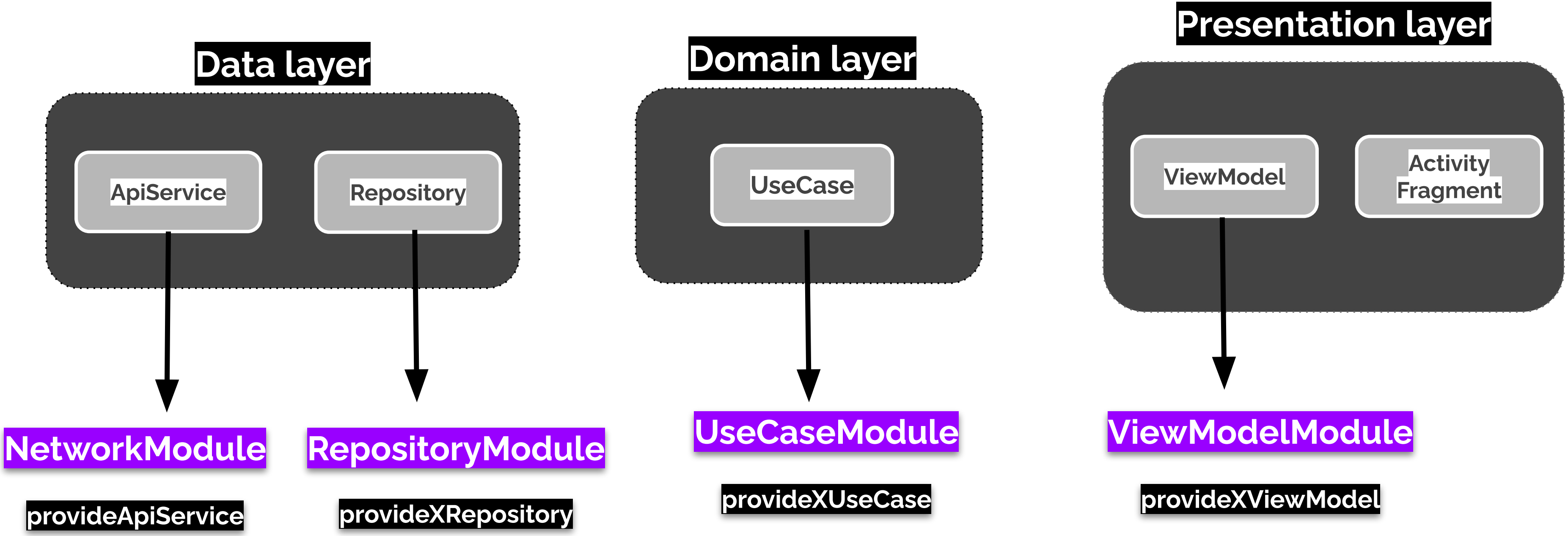
@anncode

Dagger Hilt



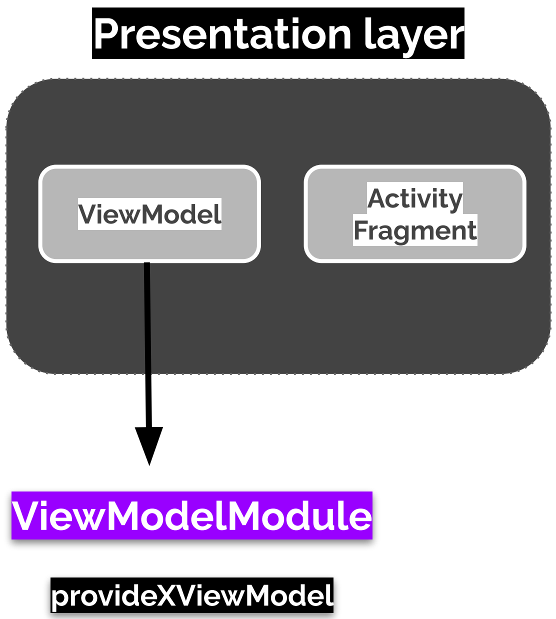
@anncode

Arquitectura



@anncode

Arquitectura



@anncode

—

Dagger Hilt

```
class MainViewModel(
    private val getCatRandomFact: GetCatRandomFact
): ViewModel() {
```

Dagger Hilt

```
class MainViewModel(
    private val getCatRandomFact: GetCatRandomFact
): ViewModel() {
```

```
class Factory(
    private val getCatRandomFact: GetCatRandomFact
): ViewModelProvider.Factory {
```

@anncode

Dagger Hilt

```
class MainViewModel @ViewModelInject constructor(
    private val getCatRandomFact: GetCatRandomFact
): ViewModel() {
```

@anncode

Dagger Hilt

```
class MainViewModel @ViewModelInject constructor(
    private val getCatRandomFact: GetCatRandomFact
): ViewModel() {
```

```
class Factory(
    private val getCatRandomFact: GetCatRandomFact
): ViewModelProvider.Factory {
```

@anncode

Dagger Hilt

```
class MainViewModel @ViewModelInject constructor(
    private val getCatRandomFact: GetCatRandomFact
): ViewModel() {
```

Dagger Hilt

```
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    @Inject lateinit var mainViewModel: MainViewModel
```

Dagger Hilt

```
@Inject lateinit var mainViewModel: MainViewModel

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

     mainViewModel.getCatFact()

}
```

Dagger Hilt

```
@Inject lateinit var mainViewModel: MainViewModel

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    mainViewModel.getCatFact()

}
```

@anncode

Hilt & Dagger Annotations

v1.0 - @AndroidDev

Annotation	Usage	Code Sample
@HiltAndroidApp	Kicks off Hilt code generation. Must annotate the Application class.	`@HiltAndroidApp` `class MyApplication : Application() { ... }`
@AndroidEntryPoint *	Adds a DI container to the Android class annotated with it.	`@AndroidEntryPoint` `class MyActivity : AppCompatActivity() { ... }`

@Inject	Constructor Injection. Tells which constructor to use to provide instances and which dependencies the type has.	`class AnalyticsAdapter @Inject constructor(`
		`  private val service: AnalyticsService`
		`) { ... }`

	Field injection. Populates fields in @AndroidEntryPoint annotated classes.
	
	Fields cannot be private.
		`@AndroidEntryPoint`
		`class MyActivity : AppCompatActivity() {`
		`  @Inject lateinit var adapter: AnalyticsAdapter`
		`  ...`
		`}`

@ViewModelInject **	Tells Hilt how to provide instances of an Architecture Component ViewModel.	`class MyViewModel @ViewModelInject constructor(`
		`  private val adapter: AnalyticsAdapter,`
		`  @Assisted private val state: SavedStateHandle`
		`): ViewModel() { ... }`

@Module	Class in which you can add bindings for types that cannot be constructor injected.	`@InstallIn(ApplicationComponent::class)`
		`@Module`
		`class AnalyticsModule { ... }`

@InstallIn	Indicates in which Hilt-generated DI containers (`ApplicationComponent` in the code) module bindings must be available.	`@InstallIn(ApplicationComponent::class)`
		`@Module`
		`class AnalyticsModule { ... }`

https://medium.com/androiddevelopers/dependency-injection-on-android-with-hilt-67b6031e62d

@anncode

@InstallIn

indicates in which Hilt generated DI containers (`ApplicationComponent` in the code) module bindings must be available.

```
@InstallIn(ApplicationComponent::class)
@Module
class AnalyticsModule { ... }
```

@Provides

Adds a binding for a type that cannot be constructor injected:

- Return type is the binding type.
- Parameters are dependencies.
- Every time an instance is needed, the function body is executed if the type is not scoped.

```
@InstallIn(ApplicationComponent::class)
@Module
class AnalyticsModule {

  @Provides
  fun providesAnalyticsService(
      converterFactory: GsonConverterFactory
  ): AnalyticsService {
    return Retrofit.Builder()
           .baseUrl("https://example.com")
           .addConverterFactory(converterFactory)
           .build()
           .create(AnalyticsService::class.java)
  }
}
```

@Binds

Shorthand for binding an interface type:

- Methods must be in a module.
- @Binds annotated methods must be abstract.
- Return type is the binding type.
- Parameter is the implementation type.

```
@InstallIn(ApplicationComponent::class)
@Module
abstract class AnalyticsModule {

  @Binds
  abstract fun bindsAnalyticsService(
    analyticsServiceImpl: AnalyticsServiceImpl
  ): AnalyticsService
}
```

Scope Annotations:

@Singleton
@ActivityScoped
...

Scoping object to a container.

The same instance of a type will be provided by a container when using that type as a dependency, for field injection, or when needed by containers below in the hierarchy.

```
@Singleton
class AnalyticsAdapter @Inject constructor(
  private val service: AnalyticsService
) { ... }
```

Qualifiers for predefined Bindings:
@ApplicationContext
@ActivityContext

Predefined bindings you can use as dependencies in the corresponding container.

These are qualifier annotations.

```
@Singleton
class AnalyticsAdapter @Inject constructor(
  @ApplicationContext val context: Context
  private val service: AnalyticsService
) { ... }
```

Obtain dependencies in classes that are either not supported directly by Hilt or cannot use Hilt.

The interface annotated with @EntryPoint must also be annotated with @InstallIn passing in the Hilt predefined component

```
class MyContentProvider(): ContentProvider {

  @InstallIn(ApplicationComponent::class)
  @EntryPoint
  interface MyContentProviderEntryPoint {
    fun analyticsService(): AnalyticsService
  }

  override fun query(...): Cursor {
```

https://medium.com/androiddevelopers/dependency-injection-on-android-with-hilt-67b6031e62d

@anncode

```
}
```

Scope Annotations:

@Singleton
@ActivityScoped
...

Scoping object to a container.

The same instance of a type will be provided by a container when using that type as a dependency, for field injection, or when needed by containers below in the hierarchy.

```
@Singleton
class AnalyticsAdapter @Inject constructor(
  private val service: AnalyticsService
) { ... }
```

Qualifiers for predefined Bindings:
@ApplicationContext
@ActivityContext

Predefined bindings you can use as dependencies in the corresponding container.

These are qualifier annotations.

```
@Singleton
class AnalyticsAdapter @Inject constructor(
  @ApplicationContext val context: Context
  private val service: AnalyticsService
) { ... }
```

@Entry Point

Obtain dependencies in classes that are either not supported directly by Hilt or cannot use Hilt.

The interface annotated with @EntryPoint must also be annotated with @InstallIn passing in the Hilt predefined component from which that dependency is taken from.

Access the bindings using the appropriate static method from EntryPointAccessors passing in an instance of the class with the DI container (which matches the value in the @InstallIn annotation) and the entry

```
class MyContentProvider(): ContentProvider {

  @InstallIn(ApplicationComponent::class)
  @EntryPoint
  interface MyContentProviderEntryPoint {
    fun analyticsService(): AnalyticsService
  }

  override fun query(...): Cursor {
    val appContext =
       context?.applicationContext
       ?: throw IllegalStateException()
    val entryPoint =
       EntryPointAccessors.fromApplication(
       appContext,
       MyContentProviderEntryPoint::class.java)
    val analyticsService =
       entryPoint.analyticsService()
    ...
  }
}
```

* The code example for this annotation assumes you're using the Gradle plugin ** This annotation is available using the `androidx.hilt.hilt-lifecycle-viewmodel` library

For more information about DI, Hilt and Dagger: https://d.android.com/dependency-injection

https://medium.com/androiddevelopers/dependency-injection-on-android-with-hilt-67b6031e62d

Anahí Salgado

@anncode