Aplicando Hilt a Login

Descargar estos apuntes

Vamos a aplicar la Inyección de dependencia explicada en el tema, al ejercicio de Login de la entrega de ejercicios anterior. **Debes tener en cuenta que si no has usado el proyecto base como base de Login, es posible que ocurran errores al añadir las dependencias y los plugins.**

Como se ve en los apuntes tendremos que añadir los plugins necesarios tanto a gradle del proyecto como al de aplicación. Ademas de descomentar las variables afectadas en el archivo `libs.versions.toml` De forma que:

En `libs.versions.toml` descomentaremos:

```
dagger-hilt-android = { ... }
dagger-hilt-android-compiler = { ... }
androidx-hilt-navigation-compose = { ... }
```

En el `build.gradle.kts` raíz del proyecto:

```
plugins {
    ...
    alias(libs.plugins.devtools.ksp) apply false
    alias(libs.plugins.com.google.dagger) apply false
```

En el `build.gradle.kts` del la app:

```
plugins {
    ...
    alias(libs.plugins.devtools.ksp)
    alias(libs.plugins.com.google.dagger)
}
```

En las dependencias de `build.gradle.kts` del la app:

```
android {
    ...
dependencies {
    ...
    implementation(libs.dagger.hilt.android)
    implementation(libs.androidx.hilt.navigation.compose)
    ksp(libs.dagger.hilt.android.compiler)
    kspAndroidTest(libs.dagger.hilt.android.compiler)
}
```

IMPORTANTE!! **No olvides sincronizar**

A partir de ahí tendremos que seguir los siguientes pasos:

1. Deberemos añadir la clase `Application` como ya hemos hecho en otras ocasiones, pero esta vez le tendremos que agregar la anotación `@HiltAndroidApp`
2. Añadiremos la anotación `@AndroidEntryPoint` a la MainActivity.
3. El siguiente paso sería **exponer la inyección de dependencia** de los objetos que se inyectan en los constructores, donde las necesitemos, en nuestro caso en el repositorio UsuarioRepository del objeto de tipo UsuarioDaoMock.

   ```
   class UsuarioRepository @Inject constructor(
   private val proveedorUsuarios: UsuarioDaoMock)
   ```

4. Ahora pasaremos a definir la clase con los métodos proveedores de las instancias, para ello crearemos el paquete **di** en la raiz de nuestro proyecto y dentro crearemos la clase **AppModule**, como se explica en los apuntes, no olvides etiquetarla como `@Module InstallIn(SingletonComponent::class)`:

   ```
   @Module
   @InstallIn(SingletonComponent::class)
   class AppModule {
       ...
   }
   ```

   Dentro de esta clase crearemos los **métodos proveedores de dependencias**, en nuestro caso solo tenemos un repositorio que debe ser inyectado con un UsuarioDaoMock, por lo que definiremos el método que provee la instancia a inyectar. Y el proveedor de repositorio al que se le inyecta la instancia del UsuarioDaoMock

```
    @Provides
    @Singleton
    fun provideUsuarioDaoMock():UsuarioDaoMock= UsuarioDaoMock()

    @Provides
    @Singleton
    fun provideUsuarioRepository(
            usuarioDaoMock: UsuarioDaoMock
        ): UsuarioRepository =
            UsuarioRepository(usuarioDaoMock)
```

5. Para finalizar nos faltaría etiquetar con `@HiltViewModel` los ViewModel y en el constructor de estos inyectaremos mediante Hilt las instancias de los **objetos colaboradores** (repositorios), *ya **no** crearemos las instancias de los repositorios dentro del ViewModel*.

```
@HiltViewModel
class LoginViewModel  @Inject constructor(private  val usuarioRepository: UsuarioReposi
```