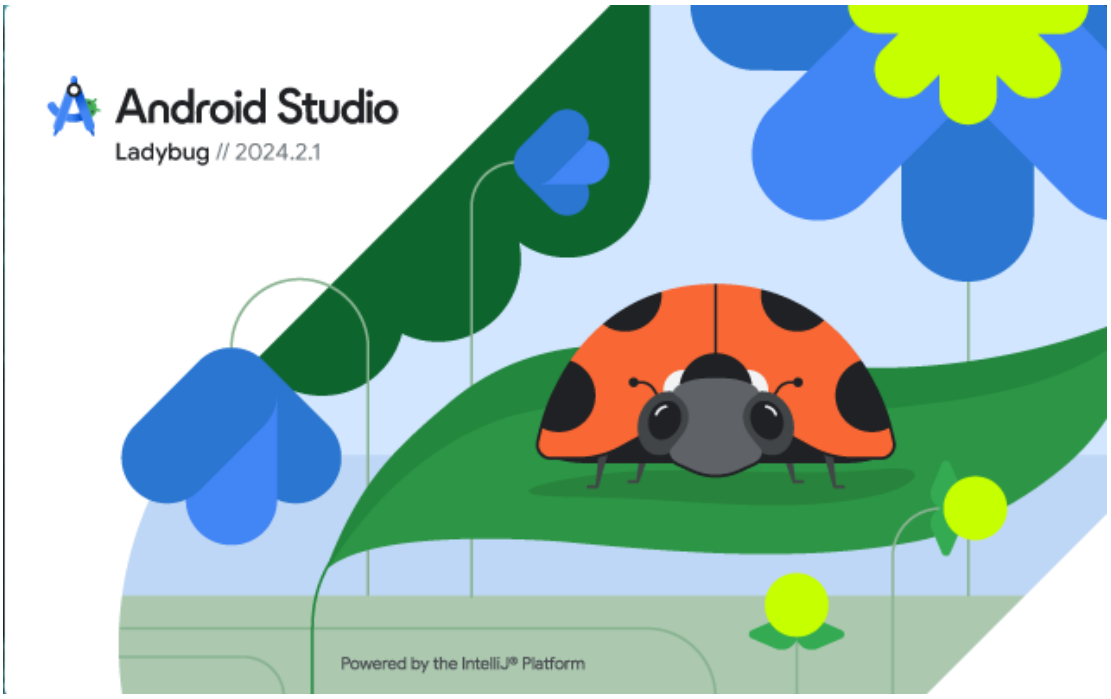
UNIVERSIDAD POLITECNICA DE VALENCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Desarrollo de aplicaciones para dispositivos móviles

PRÁCTICA 2: COMUNICACIONES HTTP



Escuela Técnica Superior de Ingeniería **Informática**

etsinf

Objetivos

- Estructurar la capa de datos en repositorios y fuentes de datos con interfaces definidas.
- Aplicar el principio de inversión de dependencias para facilitar que la aplicación sea fácilmente escalable y testable.
- Inyectar dependencias en código de forma automática mediante bibliotecas de terceros, como *Hilt*.
- Ejecutar tareas de forma paralela y/o en diferentes hilos de ejecución mediante corrutinas y funciones *suspend*.
- Utilizar bibliotecas de terceros, como *Retrofit*, para consumir servicios web.
- Convertir información de formato JSON a objeto *Kotlin* mediante bibliotecas de terceros, como *Moshi*.

Comunicaciones HTTP

En esta práctica introduciremos la capacidad de acceder a las funcionalidades ofrecidas por servicios web. Para ello, al igual que para la ejecución de cualquier tarea que pueda bloquear el hilo principal de la aplicación, será necesario realizar la tarea en segundo plano. La capa de datos se estructurará en fuentes de datos, que proporcionarán operaciones CRUD a los orígenes de datos (servicio web https://forismatic.com/en/api/, al que se accederá mediante *Retrofit* y se transformará la respuesta JSON en un objeto *Kotlin* mediante *Moshi*), y repositorios, que abstraerán el acceso a las fuentes de datos. Las funciones que ambas capas expongan se definirán a través de interfaces que facilitarán la inversión de dependencias. La inyección de las dependencias se hará de forma automática mediante *Hilt*.

Todas las funcionalidades que se pretende desarrollar se enmarcan en la pantalla encargada de obtener nuevas citas. El aspecto de esta pantalla, al recibir una nueva cita del servicio web, se muestra en la Figura 1.

Figura 1. Pantalla de obtención de nuevas citas mostrando información recibida del servicio web.

Ejercicio 01: Preparación de la arquitectura de la capa de datos e inyección de dependencias mediante *Hilt*

Descripción: Se acondicionará el proyecto para dar soporte a la arquitectura de la capa de datos y la inyección automática de dependencias.

Pasos a seguir:

- Crea un nuevo paquete denominado *data* que, a su vez, contendrá los paquetes relacionados con las funcionalidades ofrecidas que requerirán de acceso a datos: *newquotation*, *favourites* y *settings*. Crea un nuevo paquete denominado *di*, para incluir los módulos de inyección de dependencias.
- Edita el fichero *build.gradle (Project)* e incluye el *plugin* para *Hilt* y el procesador de símbolos de *Kotlin– id("com.google.devtools.ksp") version "1.9.10-1.0.13" apply false* y *id("com.google.dagger.hilt.android") version "2.50" apply false*.
- Edita el fichero *build.gradle (Module: app)* e incluye el plugin de *Hilt* y el procesador de símbolos de *Kotlin – id("com.google.devtools.ksp")* y *id("com.google.dagger.hilt.android") –*, las dependencias de *Hilt* y el procesador de símbolos de *Kotlin – implementation("com.google.dagger:hilt-android:2.50") y ksp("com.google.dagger:hilt-compiler:2.50")*.
- Sincroniza el proyecto para que actualice las dependencias y *plugins*.
- Crea una nueva clase *Kotlin*, en la raíz del proyecto, denominada *<AppName>Application.kt* y que extenderá *Application*. Anótala con *@HiltAndroidApp* para indicar que será la contenedora de las dependencias generadas.
- Modifica el manifiesto para que el atributo *android:name* de *<application/>* se enlace con la clase recién creada.
- Los *ViewModel* definidos deberán utilizar repositorios para acceder a las fuentes de datos, por lo que deberás anotarlos con *@HiltViewModel*. Adicionalmente, deberán inyectarse las dependencias en su constructor, por lo que deberás incluir *@Inject constructor()* después del nombre de la clase del *ViewModel*. Los fragmentos que hacen uso de estos *ViewModel* y la actividad que hace uso de estos fragmentos deberás anotarlos con *@AndroidEntryPoint*.
- Compila la aplicación simplemente para comprobar que compila de forma correcta y sigue funcionando igual que antes.

Ejercicio 02: Esqueleto del repositorio para obtener nuevas citas

Descripción: Para obtener nuevas citas, el *ViewModel* deberá acceder a un repositorio que implementará la interfaz definida, exponiendo los métodos necesarios.

Pasos a seguir:

- Crea una nueva interfaz *Kotlin* en el paquete *data.newquotation* y denomínala *NewQuotationRepository*. Modifica esta interfaz para que disponga de un único

método *suspend* que devuelva *Result<Quotation> – suspend fun getNewQuotation(): Result<Quotation>*. Este tipo permite encapsular una *Quotation* si la operación ha tenido éxito o un *Throwable* (excepción) en caso contrario. Al ser *suspend* se asegura que solo podrá ser invocado desde otras funciones *suspend* o corrutinas.

- Crea una nueva clase *Kotlin* en ese mismo paquete, denominada *NewQuotationRepositoryImpl*, que implemente la interfaz recién definida. Sobrescribe el método de la interfaz y haz que devuelva una cita aleatoria (el 90% de las veces) y una excepción (el 10% de las veces). Para que *Hilt* pueda inyectar sus dependencias posteriormente anota el constructor de la clase con *@Inject*.
- Crea una nueva clase abstracta en el paquete *di*, denomínala *NewQuotationBinderModule*, y anótala con *@Module*, para indicar que es un módulo *Hilt*, y *@InstallIn(SingletonComponent::class)*, para indicar que las instancias de las dependencias proporcionadas se mantendrán a lo largo del ciclo de vida de la aplicación.
- En esa clase, crea una nueva función abstracta denominada *bindNewQuotationRepository()* que reciba como parámetro un objeto del tipo de la implementación del repositorio y devuelva como resultado un objeto del tipo de la interfaz del repositorio. Al anotarla con *@Binds*, esto indica a *Hilt* que debe crear un objeto del tipo de la implementación cuando se requiera un objeto del tipo de la interfaz.
- Edita la clase *NewQuotationViewModel* para que reciba una instancia de la interfaz del repositorio a través de su constructor (anteriormente ya lo hemos anotado con *@Inject* para permitir que *Hilt* inyecte la dependencia).
- Edita esta clase para que disponga de una propiedad privada, de solo lectura y de tipo *MutableStateFlow<Throwable?>*, que indicará si hay algún mensaje de error a mostrar (si no es *null*). Crea otra propiedad de solo lectura, en este caso inmutable, que dé acceso a la propiedad privada anterior. Crea una nueva función denominada *resetError()* que simplemente asigne el valor de esta propiedad a *null*.
- Edita el método *getQuotation()* de esta clase para eliminar el código relativo a la creación de una cita de forma aleatoria y sustitúyelo por una llamada al método implementado por el repositorio para obtener una nueva cita. Al tratarse de una función *suspend*, debes crear una corrutina que la lance a ejecución. El ámbito de la corrutina será este *ViewModel*, de forma que se cancele la operación si el *ViewModel* se destruye – *viewModelScope.launch { }*. El resultado devuelto por el repositorio es de tipo *Result<Quotation>*, por lo que puedes utilizar su función *fold()* para proporcionar el código que se ejecutará si la operación finalizó con éxito o fracaso – *newQuotationRepository.getNewQuotation().fold(onSuccess = { },onFailure = { })*. En caso de éxito, actualiza el valor de la propiedad que mantiene la cita recibida. En caso de fracaso, actualiza el valor de la propiedad que mantiene el error originado.
- Edita el método *onCreate()* de la clase *NewQuotationFragment* para observar la propiedad que indica la ocurrencia de un error y, en caso de ser distinto de *null*, mostrar un *Snackbar* indicando el tipo de problema al usuario (genera diferentes recursos de tipo *<string/>* para cada tipo de excepción que puedas recibir y otro para aquellas inesperadas). Una vez mostrado, invoca la función *resetError()* del *ViewModel* para indicar que el mensaje ya se ha mostrado.

- Si ejecutas la aplicación e intentas obtener nuevas citas, el comportamiento de la aplicación deberá ser idéntico al anterior, con la diferencia que el 10% de las veces obtendrás un error que se mostrará mediante un *Snackbar*.

Ejercicio 03: Esqueleto de la fuente de datos para obtener nuevas citas mediante *Retrofit*

Descripción: Para obtener nuevas citas, el repositorio deberá acceder a la fuente de datos, que implementará la interfaz definida, exponiendo los métodos necesarios. La información de las citas se recuperará del servicio web disponible en https://forismatic.com/en/api/, aunque ahora seguiremos realizando pruebas con citas generadas aleatoriamente de forma manual.

Pasos a seguir:

- Edita el fichero *build.gradle (Module: app)* e incluye las dependencias para poder utilizar *Moshi* para generar automáticamente los convertidores de JSON a objetos *Kotlin* – *implementation("com.squareup.moshi:moshi-kotlin:1.14.0")* y *ksp("com.squareup.moshi:moshi-kotlin-codegen:1.14.0")* – y para utilizar *Retrofit* para acceder al servicio web y que utilice *Moshi* para convertir la respuesta – *implementation("com.squareup.retrofit2:retrofit:2.9.0")* e *implementation("com.squareup.retrofit2:converter-moshi:2.9.0")*.
- Sincroniza el proyecto para que actualice las dependencias.
- Edita el manifiesto de la aplicación para indicar que se requieren los permisos para acceder a Internet y poder comprobar el estado de la red –
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
- Crea una nueva interfaz *Kotlin* en el paquete *data.newquotation* y denomínala *NewQuotationDataSource*. Modifica esta interfaz para que disponga de un único método *suspend* que devuelva *Result<Quotation>* – *suspend fun getQuotation(): Result<Quotation>*.
- Crea una nueva clase *Kotlin* en ese mismo paquete, denominada *NewQuotationDataSourceImpl*, que implemente la interfaz recién definida. Para que *Hilt* pueda inyectarla posteriormente anota su constructor con *@Inject*. Sobrescribe el método de la interfaz y copia exactamente el mismo código que has utilizado para hacer las pruebas en el repositorio, ya que esta clase será el origen de datos real.
- Modifica la clase *di.NewQuotationBindModule* para crear una nueva función abstracta denominada *bindNewQuotationDataSource* que reciba como parámetro un objeto del tipo de la implementación de la fuente de datos y devuelva como resultado un objeto del tipo de la interfaz de la fuente de datos. Anótala con *@Binds*.
- Modifica la clase *data.NewQuotationRepositoryImpl* para que reciba en su constructor una instancia de la interfaz de la fuente de datos. Modifica su implementación del método de la interfaz para que devuelva el resultado que le proporcione la interfaz de la fuente de datos al invocar su método *getQuote()*.

- Si ejecutas la aplicación e intentas obtener nuevas citas, el comportamiento de la aplicación deberá ser idéntico al anterior, con la diferencia que ahora es la fuente de datos la que proporciona el resultado y todas las dependencias están inyectándose.

Ejercicio 04: Comprobar si la conexión a Internet está disponible

Descripción: Antes de lanzar cualquier operación que requiera del uso de Internet es recomendable comprobar si se dispone de conexión en ese momento. El repositorio será el encargado de realizar esta comprobación y devolver un error en caso de no disponer de conexión.

Pasos a seguir:

- Crea una nueva clase *Kotlin* en el paquete *data.newquotation*, denominada *ConnectivityChecker*, con el constructor anotado *@Inject* y que reciba como parámetro una instancia de *ConnectivityManager*. Define un nuevo método *isConnectionAvailable()* que devuelva un valor *Boolean* para indicar si hay conexión disponible a través de alguna red. Para ello, utiliza el gestor de conectividad para obtener las capacidades de la red activa – *val capabilities = connectivityManager.getNetworkCapabilities(connectivityManager.activeNetwork)*. Existirá conexión si las capacidades no son *null* y disponen de transporte bien por red Wifi o red móvil – *capabilities.hasTransport(TRANSPORT_WIFI) || capabilities.hasTransport(TRANSPORT_CELLULAR)*.
- Crea una nueva clase *Kotlin* en el paquete *di*, denominada *NewQuotationProviderModule*, anotada como *@Module* e *@InstallIn(SingletonComponent::class)*.
- En esa clase, crea una nueva función denominada *provideConnectivityManager()* que reciba como parámetro un objeto del tipo *Context*, anotado como *@ApplicationContext* para que se inyecte directamente el contexto de la aplicación, y devuelva como resultado una instancia de *ConnectivityManager*. Esta instancia puede obtenerse a partir del contexto recibido – *context.getSystemService(CONNECTIVITY_SERVICE) as ConnectivityManager*. Al anotar la función con *@Provides*, se indica a *Hilt* cómo puede crear objetos del tipo proporcionado por la función, y al anotarla *@Singleton* se indica que siempre se devolverá la misma instancia del resultado.
- Crea un nuevo paquete denominado *utils*. Crea en este paquete una nueva clase *Kotlin* denominada *NoInternetException* que extienda *Exception*. Se utilizará para señalizar que no se dispone de conectividad.
- Edita la clase *data.newquotation.NewQuotationRepositoryImpl* para que reciba en el constructor una instancia de *ConnectivityChecker*. Modifica el método de la implementación de su interfaz para que compruebe si hay conexión a Internet y, en ese caso, realice la llamada a la fuente de datos y, si no, devuelva como resultado de avería una instancia de *NoInternetException*.

- Modifica la clase *ui.newquotation.NewQuotationFragment* para que el *Snackbar* que muestra al usuario en caso de error al obtener una nueva cita tenga en cuenta el problema de falta de conectividad.
- Si ejecutas la aplicación, su comportamiento debería se idéntico al anterior. Sin embargo, si desconectas las redes del dispositivo (emulador) se mostrará el mensaje correspondiente de falta de conectividad.

Ejercicio 05: Acceder al servicio web para obtener nuevas citas mediante *Retrofit*

Descripción: Las nuevas citas se recuperarán del servicio web por medio de *Retrofit*. A través de *Moshi*, se obtendrá un objeto *Kotlin* a partir de la respuesta recibida en formato JSON. Finalmente, el repositorio deberá convertir esta respuesta al modelo de datos de la aplicación.

Pasos a seguir:

- Accede a https://api.forismatic.com/api/1.0/?method=getQuote&format=json&lang=en desde un navegador web y podrás observar la respuesta que proporcionará el servidor en formato JSON. Crea un nuevo paquete dentro de *data.newquotation* denominado *model*. Crea una clase de datos *Kotlin* denominada *RemoteQuotationDto* que disponga de propiedades de solo lectura con el mismo nombre que las claves de los objetos JSON que se recibirán (*quoteText*, *quoteAuthor*, *senderName*, *senderLink*, *quoteLink*). Anota la clase con *@JsonClass(generateAdapter = true)* para que *Moshi* genere automáticamente el adaptador necesario para realizar la conversión entre JSON y *Kotlin* (y viceversa).
- Edita la interfaz *data.newquotation.NewQuotationDataSource* y modifica el método definido para que devuelva una instancia de *Response<RemoteQuotationDto>*. Para hacer pruebas se ha estado utilizando *Result<Quotation>*, pero debes adaptarlo al tipo de dato que devuelve *Retrofit*.
- Edita la clase *data.newquotation.NewQuotationDataSourceImpl* y crea una nueva interfaz que represente el servicio web que se quiere consumir. Denomínala *NewQuotationRetrofit* e incluye un método *suspend* denominado *getQuotation()* que devuelva una instancia de *Response<RemoteQuotationDto>*. Anota este método con *@GET("api/1.0/?method=getQuote&format=json&lang=en")* para indicar el servicio web a consumir.
- Modifica el constructor de esta clase para recibir una instancia de *Retrofit*, que es necesario para implementar la interfaz definida. Crea una variable de solo lectura cuyo valor corresponda a la implementación de la interfaz creada mediante la instancia de *Retrofit* – *private val retrofitQuotationService = retrofit.create(NewQuotationRetrofit::class.java)*.
- En esta clase, modifica el método *getQuotation()* de la implementación de la interfaz *NewQuotationDataSource* para que devuelva una instancia de R*esponse<RemoteQuotationDto>*. Además, en lugar de devolver una cita aleatoria,

debe devolver el resultado obtenido al invocar el servicio creado – *retrofitQuotationService.getQuotation()*. La conversión realizada por *Moshi* podría lanzar excepciones en caso de que la respuesta JSON recibida no sea correcta, por lo que deberás encapsular la llamada al servicio *Retrofit* con *try-catch* y devolver *Response.error* en caso de recibir una excepción –

```
return try {
        retrofitQuotationService.getQuotation()
    } catch (e: Exception) {
        Response.error(
            400, // Could be any other code and text, because we are not using it
            ResponseBody.create(MediaType.parse("text/plain"), e.toString())
        )
    }
```

- Modifica la clase *di.NewQuotationProviderModule* para definir un método denominado *provideRetrofit()* que proporcione una instancia de *Retrofit*. Para ello, utiliza *Retrofit.Builder(),* añádele la URL base del servicio web e indica que debe utilizar Moshi para convertir la respuesta de JSON a objeto *Kotlin* –

```
Retrofit.Builder()
        .baseUrl("https://api.forismatic.com/")
        .addConverterFactory(MoshiConverterFactory.create())
        .build()
```

Anota este método con *@Provides* y *@Singleton*.

- Crea un nuevo fichero *Kotlin* (no clase *Kotlin*, solo fichero) en el paquete *data.newquotation.model* denominado *RemoteQuotationDtoMapper.kt*. Crea una función extendida para añadir el método *toDomain()* a la clase *RemoteQuotationDto*. Esta función deberá crear una instancia de *Quotation* a partir de los datos existentes en *RemoteQuotationDto*, de tal forma que se utilizará *quoteLink* como identificador, y el texto y cita del autor para completar los campos homónimos – *fun RemoteQuotationDto.toDomain() = Quotation(id = quoteLink, text = quoteText, author = quoteAuthor)*.
- Crea en este fichero otra función extendida denominada *toDomain()* para poder convertir la respuesta proporcionada por *Retrofit* (*Response<RemoteQuotationDto>*) a *Kotlin* (*Result<Quotation>*). Si la respuesta es correcta entonces se devolverá un resultado correcto que encapsule el cuerpo de la respuesta después de convertirlo a *Quotation* con la función extendida anteriormente creada y, si no es correcta, se devolverá un resultado erróneo como *IOException* –

```
fun Response<RemoteQuotationDto>.toDomain() =
    if (isSuccessful) Result.success((body() as RemoteQuotationDto).toDomain())
    else Result.failure(IOException())
```

- Modifica la *clase data.newquotation.NewQuotationRespositoryImpl* para que la respuesta proporcionada por la fuente de datos se convierta a una instancia del dominio de la aplicación mediante el método *toDomain()*.

- Si ejecutas la aplicación el funcionamiento debería ser idéntico al anterior, pero ahora las citas aleatorias se estarán obteniendo directamente del servicio web.

Ejercicio 06: Obtener citas en diferentes idiomas

Descripción: El servicio web utilizado permite obtener citas en inglés (*"en"*) y en ruso (*"ru"*) a través de la clave *"lang"*. El usuario debería poder seleccionar el idioma en que desea recibir estas citas, lo que veremos en el apartado de configuración del tema de almacenamiento local de la información. Por ahora, introduciremos el parámetro *hardcoded* para hacer pruebas.

Pasos a seguir:

- Modifica la interfaz *data.newquotation.NewQuotationDataSource* para que el método *getQuotation()* reciba como parámetro un *String* que indique el idioma en el que se desea obtener la cita.
- Modifica la clase *data.newquotation.NewQuotationDataSourceImpl* para incluir este parámetro de la implementación de la interfaz y pasar este parámetro a la función *getQuotation()* proporcionada por la implementación del servicio *Retrofit*. Modifica la definición del servicio *NewQuotationRetrofit* para que reciba este parámetro anotado como *@Query("lang")*, que corresponde a la clave que espera el servicio web. Modifica la anotación @GET para eliminar *"lang=en"* de la petición.
- Modifica la clase *data.newquotation.NewQuotationRepositoryImpl* para que la implementación del método de la interfaz pase un idioma aleatorio a la fuente de datos para obtener una nueva cita – *arrayOf("en", "ru", "xx").random()*.
- Si ejecutas la aplicación el funcionamiento debería ser idéntico, pero ahora las citas se recibirán, de forma aleatoria, en inglés o ruso, o aparecerá un *Snackbar* notificando que algo inesperado ha ocurrido si se utiliza un código de lenguaje desconocido.