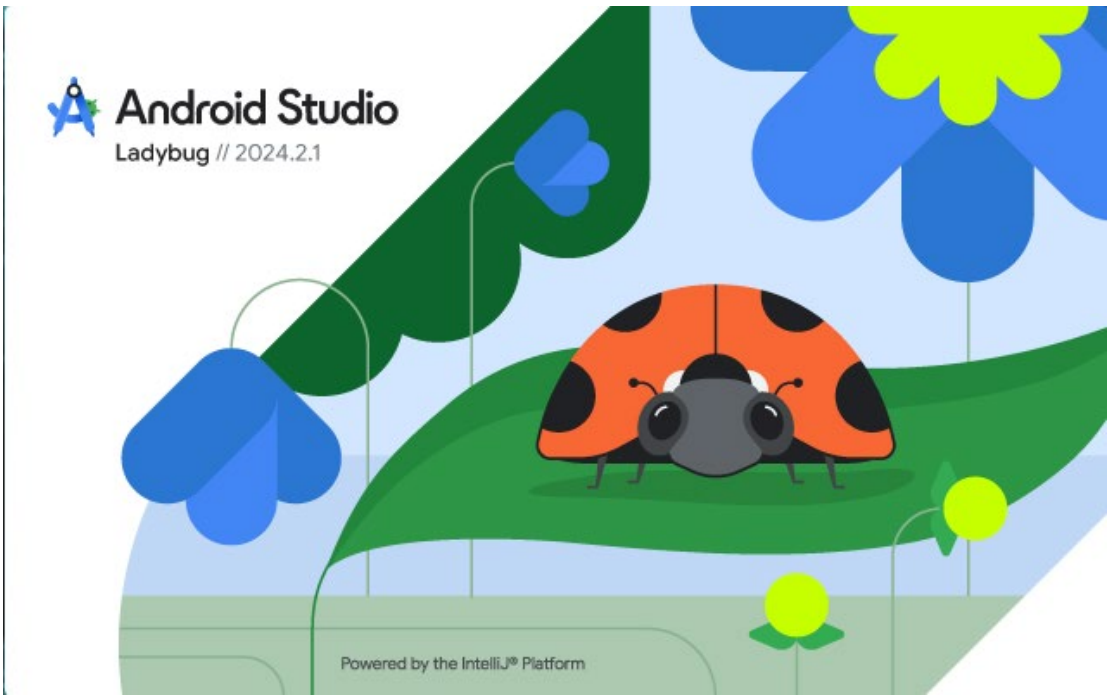
UNIVERSIDAD POLITECNICA DE VALENCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Desarrollo de aplicaciones para dispositivos móviles

PRÁCTICA 1: INTERFAZ GRÁFICA DE USUARIO



Escuela Técnica Superior de Ingeniería **Informática**

etsinf

Objetivos

- Desarrollar una aplicación muy simple que conste de una actividad encargada de la gestión de diversos fragmentos.
- Definir menús de opciones para mostrarlos como elementos de acción en la *ActionBar*.
- Utilizar la biblioteca *Navigation* para facilitar la navegación entre los fragmentos.
- Definir adaptadores personalizados para mostrar información dinámica en *RecyclerView*.
- Utilizar *ViewModel* para mantener el estado de las vistas.
- Utilizar *ConstraintLayout* para alinear y distribuir los componentes gráficos en las pantallas.
- Utilizar *Intents* para solicitar funcionalidades adicionales ya existentes.
- Utilizar estilos y temas para personalizar la apariencia de los componentes gráficos de la aplicación.
- Adaptar los recursos utilizados por la aplicación a las diversas configuraciones del dispositivo, como el lenguaje, la orientación del dispositivo y la densidad de la pantalla.

Interfaz gráfica de usuario

En esta práctica nos centraremos en el desarrollo de la interfaz gráfica de la aplicación que iremos completando en las dos prácticas siguientes con comunicaciones HTTP y almacenamiento local). La aplicación permitirá obtener citas aleatorias de personajes famosos a través de un servicio web disponible en https://forismatic.com/en/. Las citas podrán almacenarse localmente como favoritas que podrán listarse, eliminarse y consultar información acerca de su autor. Una pantalla de configuración permitirá modificar algunos parámetros de uso, y otra mostrará información acerca del desarrollador. El flujo de navegación básico se muestra en la Figura 1.

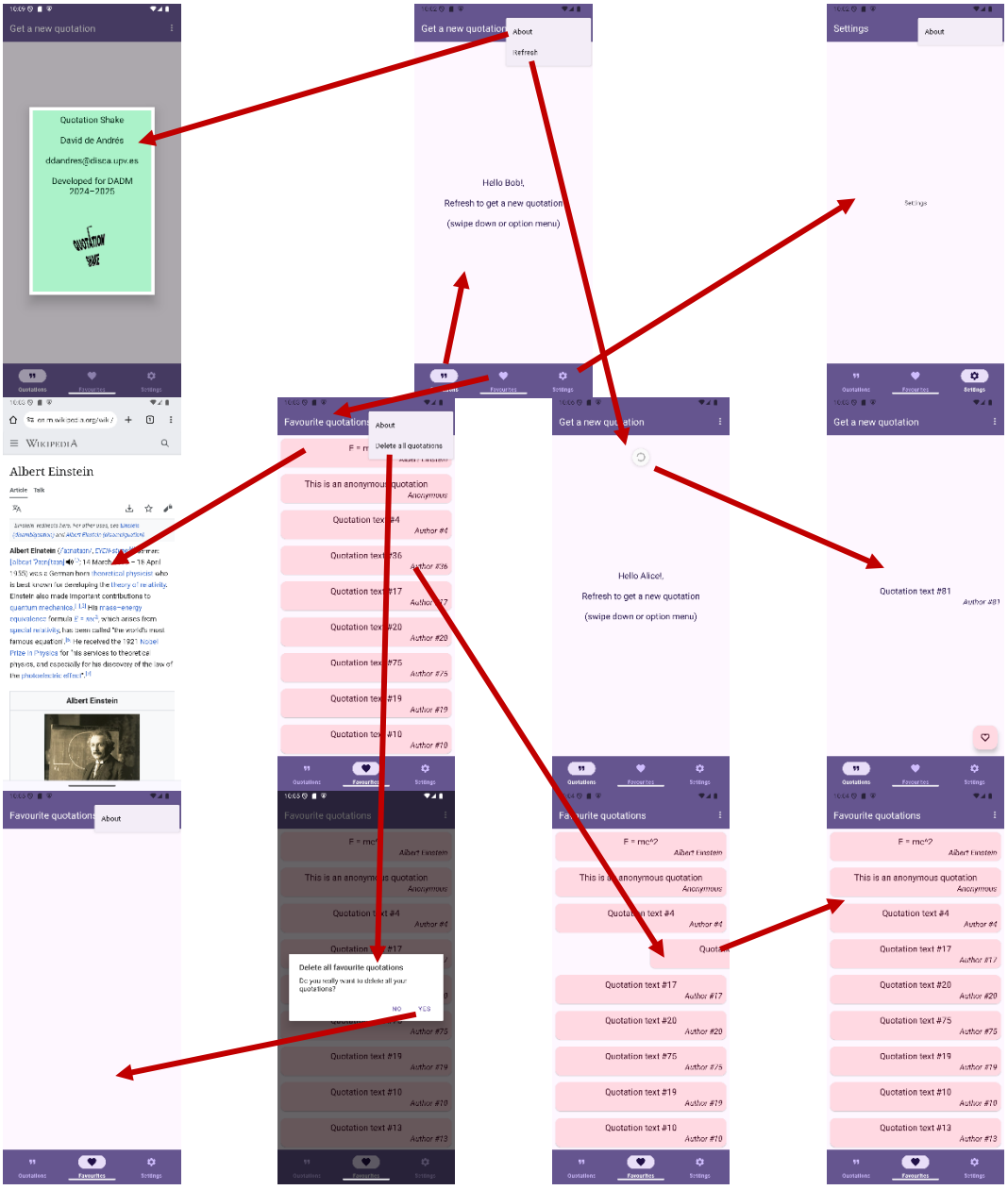


Figura 1. Flujo de navegación de la aplicación a desarrollar.

Ejercicio 01: Creación del proyecto y la actividad principal

Descripción: Este será el punto de acceso a nuestra aplicación, y desde el que se podrá navegar a las pantallas que den soporte a las funcionalidades indicadas (obtener citas, gestionar citas favoritas, configuración y acerca de).

Pasos a seguir:

- Crea un nuevo proyecto que utilice como patrón una nueva *Empty Views Activity* llamada *MainActivity*, que sea punto de entrada a la aplicación. Elige el nombre que quieras para la aplicación (*QuotationShake*, por ejemplo), el paquete debe ser *dadm.<upv_user_name>.<app_name>*), el lenguaje de desarrollo *Kotlin* y la API mínima la *26* (*Android 8.0 Oreo*).

  NOTA: Si seleccionamos una nueva actividad utilizando como patrón *Bottom Navigaton Views Activity* se generará automáticamente toda la infraestructura básica que necesitaremos. Sin embargo, en esta ocasión, la generaremos manualmente para comprender mejor cómo trabajan en conjunto los diferentes componentes y bibliotecas.

- Crea un nuevo paquete para organizar los componentes de la capa de interfaz de usuario denominado *ui*. Mueve (*Refactor*) allí el fichero *MainActivity.kt*.
- Como utilizaremos *ViewBinding* para acceder fácilmente a los recursos definidos en los *layout*, en lugar de utilizar *findViewById()*, modifica el fichero *build.gradle (Module)* para incluir *buildFeatures { viewBinding = true }* dentro de *android { }*. Sincroniza el proyecto para que se actualicen las dependencias (actualízalas todas a la última versión disponible estable si no lo están).
- Modifica el método *onCreate()* de la clase *MainActivity* para obtener una referencia de la clase *ActivityMainBinding* automáticamente generada – *val binding = ActivityMainBinding.inflate(layoutInflater)*. Utiliza esta referencia para establecer la vista de la actividad – *setContenView(binding.root)*.
- Si ejecutas la aplicación, en este momento, debería mostrar una única pantalla con *"Hello World!"*.

Ejercicio 02: Creación de los fragmentos asociados a cada funcionalidad

Descripción: Estas serán las clases que representen la información en pantalla correspondiente a cada una de las funcionalidades soportadas por la aplicación.

Pasos a seguir:

- Crea cuatro nuevos paquetes dentro del paquete *ui*, que corresponderán a las cuatro funcionalidades básicas: *newquotation*, *favourites*, *settings*, y *about*.

NOTA: Otra manera de organizarlo sería crear un paquete para los *fragments*, otro para los *adapters*, otro para los *viewmodels*, etc., pero optamos por una organización basada en las funcionalidades soportadas en lugar de por el tipo de componente.

- Crea un nuevo fichero de recursos (selecciona el directorio */res/layout* y *File -> New -> Layout Resource File*) para definir la vista de la pantalla para obtener nuevas citas (llámalo *fragment_new_quotation.xml*). Modifica este fichero para incluir un *TextView* centrado en la pantalla que muestre el texto *"Get a new quotation"* (crea un recurso de tipo *<string/>* que contenga dicho texto).
- Copia y pega este *layout* tres veces para generar los *layout* correspondientes a la pantalla de citas favoritas (*fragment_favourites.xml*), la pantalla de configuración (*fragment_settings.xml*) y la pantalla acerca de (*fragment_about.xml*). Crea los recursos *<string/>* necesarios para que cada una de ellas muestre el texto: *"Favourite quotations"*, *"Settings"* y *"About"*.
- Crea un fichero en el paquete *ui.newquotation* para definir el fragmento para obtener nuevas citas (*File -> New -> Kotlin Class/File* y denomínalo *NewQuotationFragment.kt*. Modifica el fichero para que la clase extienda *Fragment* y se inicialice con el layout previamente definido *–:Fragment(R.layout.fragment_new_quotation)*.

  NOTA: Existen diversas plantillas que permiten la creación de fragmentos (*File -> New -> Fragment -> …*), pero todas ellas incluyen muchas características que no se utilizarán en esta aplicación básica y, por tanto, requieren más esfuerzo que crearlas de cero.

- Modifica el fichero para incluir una nueva propiedad variable que mantendrá una referencia al *Binding* correspondiente y se inicializará a *null* – *private var _binding : FragmentNewQuotationBinding? = null*. Añade otra propiedad inmutable que dé acceso a esta referencia – *private val binding get() = _binding!!*.
- Modifica el fichero para sobrescribir el método *onViewCreated()* y guardar la referencia al *Binding* – *_binding = FragmentNewQuotationBiding.bind(view)*.
- Modifica el fichero para sobrescribir el método *onDestroyView()* para liberar los recursos asociados al Binding – *_binding = null*.
- Copia y pega este fragmento en los paquetes *ui.favourites* y *ui.settings*, inicializándolos con el *layout* correspondiente y modificando el tipo del *Binding* de forma adecuada.
- En el paquete *ui.about* crea un nueva clase que definirá el fragmento para mostrar la información del desarrollador. Denomina este fichero *AboutDialogFragment.kt* y modifícalo para que extienda *DialogFragment*. En el constructor del *DialogFragment* pásale como parámetro el *layout* definido.

Ejercicio 03: Definición del patrón de navegación

Descripción: El patrón de navegación a través de los fragmentos definidos se basará en un *Bottom bar*. La barra de acción por defecto se sustituirá por una personalizada que ubicará al usuario en todo momento dentro de la aplicación.

Pasos a seguir:

- Modifica el fichero *build.gradle.kts (Module)* e incluye las dependencias necesarias para utilizar la biblioteca de navegación – *implementation("androidx.navigation:navigation-ui-ktx:2.8.5")*y *implementation("androidx.navigation:navigation-fragment-ktx:2.8.5")*. Sincroniza el proyecto para descargar estas dependencias.
- Crea un recurso de tipo *<navigation/>* (llámalo *nav_graph.xml*) que definirá el flujo de navegación dentro de la aplicación. Modifica el fichero, desde la interfaz gráfica, para incluir los cuatro fragmentos definidos y selecciona *NewQuotationFragment* como fragmento inicial. Asocia a cada uno de ellos, como etiqueta (*android:label*), el texto que muestra cada fragmento.
- Crea cuatro recursos de tipo *<vector/>* (*File -> New -> Vector Asset*) para poder asociar un icono a cada una de las opciones de navegación disponibles. Selecciona iconos adecuados, de entre los que aparecen disponibles, y asóciales un color claro si el fondo de las barras fuera oscuro (o viceversa). Su tamaño debe ser *24dp*.
- Crea un recurso de tipo *<menu/>* (llámalo *menu_navigation.xml*) que definirá las opciones disponibles en la barra de navegación y las relacionará con cada uno de los fragmentos definidos. Edita el fichero y añade tres *<item/>* correspondientes a los fragmentos de obtención de citas, acceso a las citas favoritas y a la configuración del sistema. Su identificador (*id*) deberá ser el mismo que el creado para identificar los fragmentos en el grafo de navegación, asígnales como título (*title*) la misma etiqueta que muestran los fragmentos (puedes crear otras con texto más corto para facilitar su visualización en la barra de navegación, como *"Quotations"* y *"Favourites"*) y asígnales como icono (*icon*) los recursos gráficos recién creados. Al fragmento que muestra información de la aplicación se accederá a través de la barra de acción.
- Edita el recurso *layout/activity_main.xml* para incluir un *BottomNavigationView* en la parte inferior de la pantalla, encargado de gestionar la navegación entre fragmentos, y un *FragmentContainerView*, que ocupará el resto del espacio y mostrará el fragmento seleccionado.
- Asocia el menú de navegación al *BottomNavigationView* por medio de su atributo *app:menu*.
- Modifica el *FragmentContainerView* para indicar que se trata del fragmento destino de la navegación (*android:name="androidx.navigation.fragment.NavHostFragment"*) para el grafo seleccionado (*app:navGraph="@navigation/nav_graph"*), y que interceptará la pulsación del botón Back (*app:defaultNavHost="true"*).
- Edita el método *onCreate()* de la clase *MainActivity* para configurar la navegación entre fragmentos. Obtén una referencia al controlador de navegación a partir del fragmento que lo alberga – *val navController = binding.navHostFragment.getFragment<NavHostFragment>().navController* – y asócialo a la barra de navegación – *binding.bottomNavigationView.setupWithNavController(navController)*.
- Si ejecutas la aplicación deberías poder navegar entre los tres fragmentos, mediante la barra de navegación.

Ejercicio 04: Personalización de la barra de acción

Descripción: La barra de acción por defecto se sustituirá por una personalizada que ubicará al usuario en todo momento dentro de la aplicación.

Pasos a seguir:

- Edita el recurso *layout/activity_main.xml* para incluir una *MaterialToolbar* – *android:layout_height="?attr/actionBarSize"* y *app:titleTextColor* con color claro si el fondo es oscuro (y viceversa) – dentro de un *AppBarLayout* con *android:fitsSystemWindows="true"* para reservar espacio para la barra de estado del sistema. Adicionalmente, para poder coordinar las acciones entre los diversos componentes, precisaremos de un *CoordinatorLayout* que los englobe a todos y el *FragmentContainerView* deberá disponer del atributo *app:layout_behavior="com.google.android.material.appbar.AppBarLayout$ScrollingViewBehavior"* para coordinar su acción con la barra y dejarle el espacio superior de la pantalla.
- Edita el método *onCreate()* de la clase *MainActivity* para establecer la barra creada como nueva barra de acción – *setSupportActionBar(binding.toolbar)*. Crea una instancia de *AppBarConfiguration* pasándole como parámetro el conjunto de identificadores de los fragmentos que están en el nivel más alto de la jerarquía (no debe mostrarse la opción de navegar hacia arriba en la barra). Asocia esta instancia a la nueva barra de acción para que el controlador de navegación pueda gestionarlo adecuadamente – *setupActionBarWithNavController(navController, appBarConfiguration)*.
- Si ejecutas la aplicación deberías poder navegar entre los tres fragmentos mediante la barra de navegación, igual que antes, pero ahora debería mostrarse el título del fragmento en la barra de acción.

Ejercicio 05: Adaptación de los recursos a la orientación del dispositivo

Descripción: Si el usuario cambia la orientación del dispositivo de vertical a horizontal, entonces la barra de navegación inferior puede ocupar gran parte del espacio disponible. Para evitarlo, se sustituirá automáticamente por un raíl de navegación en la parte izquierda de la pantalla.

Pasos a seguir:

- Edita el fichero *res/layout/activity_main.xml* y desde la vista *Split* o *Design* selecciona *Create Landscape Qualifier* (es lo mismo que si copias el fichero en la carpeta */res/layout-land*).
- Edita este nuevo fichero y sustituye el *BottomNavigationView* por un *NavigationRailView* (debe mantener el mismo identificador, ya que ambos extienden *NavigationBarView* y se utilizarán indistintamente). Reorganiza los componentes para que la barra de acción esté en la parte superior de la pantalla, el raíl de navegación en la parte izquierda y el contenedor de los fragmentos ocupe el resto del espacio disponible.
- Edita el método *onCreate()* de la clase *MainActivity* para que al obtener la referencia al componente de navegación a través del *Binding* se realice una promoción al tipo *NavigationBarView* – *binding.navigationComponent as NavigationBarView*. Con esto se tratará exactamente igual independientemente de que se trate de una instancia de un tipo u otro.
- Si ejecutas la aplicación deberías poder navegar entre los tres fragmentos mediante la barra de navegación tanto en posición vertical (barra de navegación inferior) como horizontal (raíl de navegación). Activa *auto-rotate* en el emulador/dispositivo para poder comprobarlo.

Ejercicio 06: Gestión de *edge-to-edge*

Descripción: A partir de la versión 35 de Android, las aplicaciones se lanzan por defecto en modo *edge-to-edge*. Esto puede hacer que los elementos de la aplicación se solapen con las barras de estado, de navegación y con los recortes de pantalla (cuando la cámara rodeada por parte de la pantalla, por ejemplo). Para evitarlo, deben gestionarse los espacios que se dejan alrededor de los componentes de la aplicación cuando sea necesario.

Pasos a seguir:

- Edita el método *onCreate()* de la clase *MainActivity* y ejecuta el método *enableEdgeToEdge()* antes de asociar la vista a la actividad (*setContentView()*). Esto permitirá que versiones anteriores de Android se muestren en modo *edge-to-edge* y la aplicación ocupe todo el espacio disponible en pantalla.
- Si ejecutas la aplicación en un emulador de *Pixel 9* podrás ver que no hay ningún problema en modo retrato (*portrait*), ya que *MaterialToolBar* y *BottomNavigationView* gestionan automáticamente el espacio alrededor para no solapar con las barras de estado y navegación. Sin embargo, si rotas el dispositivo a la izquierda, la cámara se solapará con el menú de navegación y, si lo rotas a la derecha, se solapará posiblemente con lo que se muestre en el *FragmentContainerView* (cita recibida, citas favoritas, configuración – aunque ahora no se aprecie).

- Edita el método *onCreate()* de la clase *MainActivity* para ajustar el espacio alrededor del *NavigationBar*, del *FragmentContainerView* y del *AppBarLayout* de acuerdo a si solapa con la cámara y las barras del sistema (estado, gestos y navegación):

```
ViewCompat.setOnApplyWindowInsetsListener(binding.Name) { view, insets ->
 val bars = insets.getInsets(WindowInsetsCompat.Type.displayCutout() or
                             WindowInsetsCompat.Type.systemBars())
view.updatePadding(
   left = LEFT,
   top = TOP,
   right = RIGHT,
   bottom = BOTTOM
)
 WindowInsetsCompat.CONSUMED
}
```

  Donde *binding.Name* es la referencia al *NavigationBar* al *FragmentContainerView* y al *AppBarLayout* y (*LEFT*, *TOP*, *RIGHT*, *BOTTOM*) deben tomar los valores (*bars.left*, *0*, *0*, *bars.bottom*), (*0*, *0*, *bars.right*, *if (resources.configuration.orientation == ORIENTATION_LANDSCAPE) bars.bottom else 0*) y (*bars.left*, *bars.top*, *0*, *0*) respectivamente.

- Si ejecutas la aplicación nuevamente en un emulador de Pixel 9, al rotar el dispositivo a la izquierda el raíl de navegación se desplazará a la derecha para no solapar con la cámara. Si rotas el dispositivo a la derecha, el *FragmentContainerView* se desplazará a la izquierda para no solapar con la cámara (aunque no podrás apreciar el efecto hasta que no implementes la lista de citas favoritas).

Ejercicio 07: Menú de opciones para mostrar la información del desarrollador de la aplicación

Descripción: El acceso a la información del desarrollador de la aplicación se realizará a través de un elemento de acción disponible en la barra de acción.

Pasos a seguir:

- Crea un nuevo recurso de tipo *<menu/>*, llámalo *menu_about.xml*, y modifícalo para incluir un único *<item/>*. Asóciale el identificador, título e icono ya definidos, e indica que no debe mostrarse nunca (*app:showAsAction="never"*), por lo que aparecerá en el menú de opciones desplegable asociado a la barra de acción.
- Modifica la clase *MainActivity* para que implemente la interfaz *MenuProvider* y sobrescribe sus dos métodos: *onCreateMenu()* y *onMenuItemSelected()*. Añade esta instancia como proveedor de menús a la barra de acción – *addMenuProvider(this)*.
- El método *onCreateMenu()* debe utilizar los parámetros recibidos para crear el menú correspondiente al recurso definido – *menuInflater.inflate(R.menu.menu_about, menu)*. El método *onMenuItemSelected()* deberá navegar al *DialogFragment* –

navController.navigate(R.id.aboutDialogFragment) – si el elemento seleccionado es el definido en el menú creado, y devolver como resultado *true*, o devolver como resultado *false* en otro caso.

- Si ejecutas la aplicación, el desplegable del menú de opciones debería mostrarse en la barra de acción desde cualquiera de los tres fragmentos. Si seleccionas el elemento de acción disponible, deberías poder visualizar el diálogo (solo muestra *"About* ahora mismo). Pulsando fuera del diálogo o el botón *Back* debería ocultarse el diálogo.

Ejercicio 08: Pantalla de información del desarrollador: adaptación de los recursos a la densidad de la pantalla

Descripción: La pantalla simplemente mostrará información acerca del desarrollador, así como una imagen (no tiene por qué ser la foto del desarrollador, puedes poner un logo o algo similar) que debe adaptarse a la densidad de pantalla del dispositivo.

Pasos a seguir:

- Edita el fichero *res/layout/fragment_about.xml* para incluir el *ConstraintLayout* dentro de un *ScrollView* (tendrás que pasar todas las anotaciones *xmls:* al *ScrollView*). Este componente permite albergar un único componente (o *layout*) y desplazarlo verticalmente en caso de que no quepa en pantalla. El *ScrollView* y el *ConstraintLayout* deberán tener la altura necesaria para mostrar su contenido (*wrap_content*). Incluye un margen alrededor del *ConstraintLayout*.
- Modifica el texto alojado en el *ConstraintLayout* para que esté centrado, situado en la parte superior de su padre, y muestre información relativa al proyecto y su desarrollador, por ejemplo: "*<your app name> by \n\n<your name>\n\n<your contact info>\n\nDeveloped for DADM 2024*".
- Selecciona alguna imagen en **formato PNG** (puede ser una foto del desarrollador, un logo para la aplicación o algo representativo) y tamaño grande (corresponderá a la resolución *xxxhdpi*). Genera las imágenes necesarias para el resto de resoluciones (puedes utilizar el servicio https://nsimage.brosteins.com/ o similares) e inclúyelas en el proyecto en las carpetas correspondientes (*/res/drawable-hdpi*, .etc.)
- Incluye una *ImageView* centrada en el *ConstraintLayout* y sitúala debajo del texto existente. Ocupará el espacio necesario (*wrap_content*) para mostrar la nueva imagen (*android:src="@drawable/picture"*). Deberá disponer de una descripción para facilitar la accesibilidad – *android:contentDescription*.
- Si ejecutas la aplicación y seleccionas el elemento de acción disponible, deberías poder visualizar el diálogo que mostrará la información del desarrollador. SI la imagen es lo suficientemente grande y rotas el dispositivo a una posición horizontal (con *auto-rotate* habilitado), podrás observar cómo aparece la barra de desplazamiento para poder visualizar la información dentro del diálogo. Si no, introduce más retornos de carro en el texto mostrado, simplemente para probar que funciona correctamente el desplazamiento, y luego elimínalos.

Ejercicio 09: Pantalla de configuración

Descripción: La pantalla de configuración deberá permitir gestionar y almacenar las opciones de la aplicación. Hasta el tema 6 no veremos cómo implementarla, así que dejaremos el diseño esta pantalla hasta entonces.

Pasos a seguir:

Nada que hacer por el momento.

Ejercicio 10: Pantalla de obtención de nuevas citas

Descripción: Desde esta pantalla el usuario podrá obtener nuevas citas de personajes famosos, que podrá añadir a su lista de citas favoritas. Al mostrar esta pantalla se mostrará un texto de bienvenida personalizado que indicará cómo operar para la obtención de citas: se podrá realizar la acción de refrescar (*swipe down*) en la pantalla o seleccionar el elemento de acción que aparecerá en la barra de acción (como alternativa de accesibilidad). Un botón flotante permitirá añadir la cita a la lista de favoritas. El estado de la vista se almacenará en un *ViewModel*. Aquí, simplemente se prepararán los elementos necesarios para su implementación.

Pasos a seguir:

- Modifica el fichero *build.gradle.kts (Module)* e incluye las dependencias necesarias para utilizar el *layout* que permite refrescar mediante *swipe down* y el uso de *ViewModel* y *LiveData* – *implementation("androidx.swiperefreshlayout:swiperefreshlayout:1.1.0")*, *implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.7.0"),*. Sincroniza el proyecto para descargar estas dependencias.
- Crea dos recursos de tipo *<vector/>* (*File -> New -> Vector Asset*) para poder asociar un icono a la acción de refrescar la pantalla para obtener una nueva cita y a la acción de añadir la cita a la lista de favoritos. Selecciona iconos adecuados, de entre los que aparecen disponibles, y asóciales un color claro si el fondo de la barra/botón flotante fuera oscuro (o viceversa).
- Modifica el fichero *res/layout/fragment_new_quotation.xml* para que conste de los siguientes componentes:
    - *SwipeRefreshLayout*: Es el componente de más alto nivel de la jerarquía que englobará a los demás. Permitirá realizar el gesto de desplazar hacia abajo para obtener una nueva cita.
    - *FrameLayout*: Hijo directo del *SwipeToRefreshLayout* y ocupará toda la pantalla.
    - *ConstraintLayout*: Hijo directo del *FrameLayout* y organizará todas las vistas restantes menos el botón flotante.

- *FloatingActionButton*: Hijo directo del *FrameLayout*. Debe ocupar el espacio necesario (*wrap_content*) para mostrar el icono asociado (*android:src*) y debe ubicarse en la parte inferior derecha de la pantalla (*android:layout_gravity="bottom|end"*). Deberá incluir una descripción para accesibilidad (*android:contentDescription*).
- *TextView*: Hijos directos del *ConstraintLayout*. Serán los encargados de mostrar el mensaje de bienvenida, el texto de la cita recibida y su autor. El mensaje de bienvenida se centrará en la pantalla (*android:gravity="center"*). El texto y el autor de la cita formarán una cadena vertical en formato *packed* en el centro de la pantalla. El texto aparecerá justificado (*android:justificationMode="inter_word"*) y el autor aparecerá debajo y a la derecha del texto en cursiva (*android:textStyle="italic"*). El texto de estos tres componentes deberá estar inicialmente vacío.

Ejercicio 11: Mostrar un mensaje de bienvenida

Descripción: Al entrar en la pantalla de obtener nuevas citas, se mostrará al usuario un mensaje de bienvenida personalizado con su nombre.

Pasos a seguir:

- Crea un nuevo recurso de tipo <string/> cuyo texto sea similar al siguiente: "*Hello %1s!,\n\nRefresh to get a new quotation\n\n(swipe down or options menu)*". Este texto incluye un *placeholder* (*%1s*) correspondiente a un *string* que puede introducirse en tiempo de ejecución y que sustituiremos por el nombre que el usuario indique en la configuración de la aplicación.
- Crea una nueva clase *Kotlin* en el paquete *ui.newquotation*, denomínala *NewQuotationViewModel* y modifícala para que extienda *ViewModel* con su constructor vacío.
- Edita esta clase para que disponga de una propiedad privada, de solo lectura, de tipo *MutableStateFlow<String>* correspondiente al nombre de usuario. Inicializa esta propiedad al valor devuelto por el método *getUserName()*. Crea este método privado, que debe devolver el *String* correspondiente al nombre de usuario. Para hacer pruebas, haz que devuelva un nombre aleatorio, por ejemplo: *setOf("Alice", "Bob", "Charlie", "David", "Emma", "").random()*.
- Crea otra propiedad de solo lectura que devuelva el *MutableStateFlow* anterior como un *StateFlow* de solo lectura (.*asStateFlow()*).
- Modifica la clase *NewQuotationFragment* para obtener una referencia al *ViewModel* creado por medio de su factoría – *private val viewModel: NewQuotationViewModel by viewModels()*. Modifica su método *onViewCreated()* para observar cambios en el valor del nombre de usuario mientras el *Fragment* se muestre en pantalla:

```
viewLifecycleOwner.lifecycleScope.launch {
        viewLifecycleOwner.repeatOnLifecycle(Lifecycle.State.STARTED) {
          viewModel.userName.collect { userName ->
```

```
        }
      }
    }
```

Cuando se reciba un nuevo nombre de usuario deberás actualizar el mensaje de bienvenida, recuperando el recurso <string/> creado anteriormente y sustituyendo el *placeholder* por el nombre recibido. Si el nombre de usuario estuviera vacío (*""*), muestra como nombre *"Anonymous"* – *binding.tvGreetings.text = getString(R.string.greetings, userName.ifEmpty{ getString(R.string.anonymous) })*.

- Si ejecutas la aplicación y accedes a la opción de obtener nuevas citas (es la que se muestra por defecto), deberás visualizar el mensaje de bienvenida con el nombre aleatorio que se haya obtenido. Para poder obtener otro nombre aleatorio deberás terminar la ejecución de la aplicación y relanzarla.

Ejercicio 12: Obtener una nueva cita mediante el gesto *swipe down*

Descripción: Desde la pantalla de obtener nuevas citas puede refrescarse la cita mostrada mediante el gesto *swipe down*. Deberá mostrarse un icono que represente que se está recuperando la cita del servidor web (el componente *SwipeRefreshLayout* ya dispone de uno por defecto) y, al recibir la cita, deberá ocultarse el mensaje de bienvenida y mostrar el texto y autor de la cita.

Pasos a seguir:

- Crea un nuevo paquete llamado *domain.model* donde se ubicarán las clases que constituyan el modelo de datos de la aplicación. Crea en ese paquete una nueva clase de datos (*data class*), denominada *Quotation*, que constará de un identificador, texto de la cita y autor de la cita, todos de tipo *String*.
- Modifica la clase *NewQuotationViewModel* para crear las propiedades privadas, de solo lectura y tipo *MutableStateFlow<>*, y sus correspondientes propiedades inmutables de solo lectura – *StateFlow<>* para almacenar: la nueva cita a mostrar en pantalla (*Quotation*, por defecto *null*) y si se debe mostrar el icono que representa que se está obteniendo la nueva cita del servidor (*Boolean*, por defecto *false*).
- Crea un nuevo método *getNewQuotation()* que será el encargado de recuperar la cita del servicio web. Por ahora, generaremos una cita aleatoria que actualizará el valor de la propiedad correspondiente –

```
val num = (0..99).random()
_quotation.update {
   Quotation(
      id = "$num",
      text = "Quotation text #$num",
      author = "Author #$num"
   )
}
```

Al entrar en el método deberás indicar que se debe mostrar el icono de que se está realizando el acceso al servicio web (actualiza la propiedad que controla la visibilidad del icono a *true*) y deberás ocultarlo al obtener la cita (actualiza la propiedad a *false*).

- Modifica el método *onViewCreated()* de la clase *NewQuotationFragment()* para observar cambios en las propiedades expuestas por el *ViewModel*. Actualiza la visibilidad del mensaje de bienvenida de forma adecuada – *binding.tvGreetings.isVisible*, indica al componente *SwipeRefreshLayout* que debe mostrar/ocultar el icono de refresco – *binding.swipeToRefresh.isRefreshing*, actualiza el texto en los *TextView* restantes para mostrar el texto y el autor de la cita recibida. Si como autor de la cita se recibe una cadena de texto vacía (“”) entonces deberás mostrar el texto *“Anonymous”*.
- En este mismo método, asocia un *OnRefreshListener* al *SwipeRefreshLayout* para que invoque al método *getNewQuotation()* del *ViewModel*.
- Si ejecutas la aplicación y haces el gesto *swipe down* en la pantalla, deberá mostrarse el icono de refresco por un espacio de tiempo muy breve (la cita se crea de forma aleatoria muy rápidamente), desaparecerá el mensaje de bienvenida y se mostrará el texto y autor generados de forma aleatoria. Cada vez que repitas este gesto obtendrás una nueva cita.

Ejercicio 13: Obtener una nueva cita mediante el menú de opciones

Descripción: No todo el mundo puede realizar el gesto *swipe down*, por lo que por problemas de accesibilidad, se podrán obtener citas también desde el menú de opciones.

Pasos a seguir:

- Crea un nuevo recurso de tipo *<menu/>*, llámalo *menu_new_quotation.xml*, y modifícalo para incluir un único *<item/>*. Asóciale un identificador, título *(“Refresh”)* y el icono anteriormente creado, e indica que debe mostrarse en el menú desplegable(*app:showAsAction="never"*).
- Modifica la clase *NewQuotationFragment* para que implemente la interfaz *MenuProvider* y sobrescribe sus dos métodos: *onCreateMenu()* y *onMenuItemSelected()*. Añade esta instancia como proveedor de menús a su actividad solo cuando el fragmento sea interactivo – *requireActivity().addMenuProvider(this, viewLifecycleOwner, Lifecycle.State.RESUMED)*.
- El método *onCreateMenu()* debe utilizar los parámetros recibidos para crear el menú correspondiente al recurso definido – *menuInflater.inflate(R.menu.menu_new_quotation, menu)*. El método *onMenuItemSelected()* deberá invocar al método encargado de obtener una nueva cita – *getNewQuotation()*, si el elemento seleccionado es el definido en el menú creado, y devolver como resultado *true*, o devolver como resultado *false* en otro caso.

- Si ejecutas la aplicación y seleccionas la opción del menú de opciones, deberás obtener el mismo efecto que si haces el gesto *swipe down*. El icono de refresco probablemente no llegue a mostrarse, ya que aparece y desaparece muy rápidamente.

Ejercicio 14: Añadir la cita a la lista de citas favoritas

Descripción: Desde la pantalla de obtener nuevas citas puede añadirse la cita recibida a la lista de citas favoritas por medio del botón flotante. Este se mostrará al recibir una cita y se ocultará si se muestra el mensaje de bienvenida o la cita se ha añadido a la lista de favoritas.

Pasos a seguir:

- Modifica la clase *NewQuotationViewModel* para crear la propiedad privada, de solo lectura y tipo *MutableStateFlow<Boolean>*, y su correspondiente propiedad inmutable – *StateFlow<Boolean>* – de solo lectura que almacenarán la visibilidad del botón flotante que permite añadir la cita a la lista de favoritos. Su valor por defecto será *false*, con lo que se ocultará el botón flotante al mostrar el mensaje de bienvenida.
- Modifica el método *getNewQuotation()* para que, al finalizar el método, se cambie el valor de la propiedad privada a *true*. Crea un nuevo método *addToFavourites()* en el que, por ahora, solo se cambie el valor de la propiedad privada a *false*.
- Modifica el método *onCreateView()* de la clase *NewQuotationFragment()* para observar cambios en la nueva propiedad expuesta por el *ViewModel*. Actualiza la visibilidad del botón flotante de acuerdo con el valor recibido, tal y como hiciste con el mensaje de bienvenida.
- Añade un *OnClickListener* al botón flotante para que, al pulsar en él, se invoque el método *addToFavourites()* del *ViewModel*.
- Si ejecutas la aplicación, el botón flotante debería mostrarse al recibir una nueva cita y ocultarse al pulsarlo.

Ejercicio 15: Pantalla de citas favoritas

Descripción: Esta pantalla mostrará la lista de citas favoritas del usuario a través de un *RecyclerView*. Pulsando sobre cualquiera de ellas se accederá a la Wikipedia para obtener información acerca del autor de la cita. Mediante el gesto *swipe right* se podrán eliminar las citas de la lista. Una opción del menú de opciones permitirá eliminar todas las citas de la lista, aunque un diálogo solicitará confirmación.

Pasos a seguir:

- Crea un nuevo recurso de tipo *<layout/>* y denomínalo *quotation_item.xml*. Este *layout* será el que se utilizará para mostrar cada una de las citas favoritas. Únicamente deberá constar de una *MaterialCardView* que contendrá dos *TextView* encargados de

mostrar el texto y el autor de la cita, respectivamente. Configúralos igual que en el caso de obtener una cita.

- Modifica el fichero *res/values/fragment_favourites.xml* para que conste de un único componente: un *RecyclerView*. Deberá ocupar todo el espacio disponible y dispondrá de un *LinearLayoutManager* vertical (*android:orientation="vertical" app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"*).

Ejercicio 16: Mostrar la lista de citas favoritas

Descripción: Al entrar en la pantalla se mostrará automáticamente la lista de citas favoritas por medio del *RecyclerView*. Para ello, se precisará de un *ListAdapter* que relacione las vistas con el origen de datos, y un *ViewHolder* para mantener las referencias a las vistas.

Pasos a seguir:

- Crea una nueva clase *Kotlin* en *ui.favourites* y denomínala *QuotationListAdapter*. Será la encargada de poner en relación las vistas con el origen de datos. Deberá extender *ListAdapter<Quotation, QuotationListAdapter.ViewHolder>* y el constructor de *ListAdapter* deberá recibir una instancia de *DiffUtil.ItemCallback<Quotation>* para poder optimizar los cambios a realizar en las vistas – escribe, simplemente *(QuotationDiff)*.
- Crea esta clase siguiendo un patrón *Singleton* dentro del propio adaptador – *object QuotationDiff : DiffUtil.ItemCallback<Quotation>() {}* – y sobrescribe los dos métodos necesarios: *areItemsTheSame()* y *areContentsTheSame()*. Supondremos que dos elementos son el mismo si disponen del mismo identificador, mientras que puedes utilizar el operador == de la clase *data Quotation* para determinar si sus contenidos son iguales.
- Crea la clase *ViewHolder* necesaria para optimizar la actualización de la información en las vistas del *RecyclerView*. Recibirá como parámetro una instancia de *QuotationItemBinding* y deberá extender *RecyclerView.ViewHolder*, al que se deberá pasar en el constructor la vista de más alto nivel de la jerarquía del *Binding* – *(binding.root)*. Modifica esta clase para incluir un método denominado *bind()*, que reciba una cita (*Quotation*) como parámetro y que modifique el texto de los *TextView* accesibles a través del *Binding* de acuerdo a la cita recibida.
- Modifica el adaptador para sobrescribir los métodos *onCreateViewHolder()* y *onBindViewHolder()*.
  El primero deberá crear una vista para poder mostrar citas, asociarla al *Binding* correspondiente y devolver el *ViewHolder* que mantendrá su instancia: *ViewHolder(QuotationItemBinding.inflate(LayoutInflater.from(parent.context), parent, false))*.
  El segundo método deberá actualizar la información en la vista de acuerdo a la cita correspondiente a la posición que se reutiliza: *holder.bind(getItem(position))*.
- En el paquete *ui.favourites*, crea una clase *Kotlin* denominada *FavouritesViewModel* que extenderá *ViewModel*.

- Edita esta clase para que disponga de una propiedad privada, de solo lectura, de tipo *MutableStateFlow<List<Quotation>>* correspondiente a la lista de citas favoritas. Inicializa esta propiedad al valor devuelto por el método *getFavoriteQuotations()*. Crea este método privado, que debe devolver la lista de citas favoritas del usuario. Para hacer pruebas, haz que devuelva 20 citas generadas aleatoriamente (como hiciste para obtener una nueva cita).
- Crea otra propiedad de solo lectura, en este caso inmutable – *StateFlow<List<Quotation>>* que dé acceso a la propiedad privada anterior.
- Modifica la clase *FavouritesFragment* para obtener una referencia al *ViewModel* creado por medio de su factoría. Modifica su método *onViewCreated()* para crear una instancia del adaptador necesario y asociarlo al *RecyclerView*. Asimismo, observa cambios en el valor de las citas favoritas para pasarle la nueva lista de citas al adaptador – *adapter.submitList(list)*.
- Si ejecutas la aplicación y accedes a la opción de citas favoritas,), deberás visualizar la lista completa de citas aleatorias generadas.

Ejercicio 17: Eliminar todas las citas favoritas

Descripción: Un elemento de acción, en el menú de opciones, permitirá borrar todas las citas favoritas. Sin embargo, se mostrará un diálogo de confirmación, ya que es una acción peligrosa que no puede deshacerse.

Pasos a seguir:

- Modifica la clase *ui.FavouritesViewModel* para definir un nuevo método – *fun deleteAllQuotations()* – que permita eliminar todas las citas de la lista de favoritos. Por ahora, simplemente cambia el valor de la propiedad encargada de almacenar el estado por una lista vacía.
- Crea una nueva clase *Kotlin* en el paquete *ui.favourites*, que extienda *DialogFragment*, y denomínala *DeleteAllDialogFragment*. Modifica la clase para obtener una referencia de *FavouritesViewModel* creada por medio de la factoría que permite asociar la instancia a la actividad para que pueda ser compartida entre fragmentos – *activityViewModels()*. Sobrescribe el método *onCreateDialog()* que devolverá una instancia del diálogo que se mostrará: utiliza *AlertDialog.Builder()*, asóciale un título (*"Delete all favourite quotations"*), un mensaje (*"Do you really want to delete all your quotations?"*), y dos botones (positivo y negativo) – cuyo *listener* ejecutará el método *deleteAllQuotations()* del *ViewModel* para el botón positivo y *dismiss()* para el botón negativo –, y crea el diálogo mediante *.create()*.
- Modifica el grafo de navegación definido en el fichero */res/navigation/nav_graph.xml* e incluye el *DialogFragment* que acabas de crear. Asóciale una transición desde *FavouritesFragment*.
- Crea un nuevo recurso de tipo *<menu/>*, llámalo *menu_favourites.xml*, y modifícalo para incluir un único *<item/>*. Asóciale un identificador, título *("Delete all quotations"*)

y un icono (que deberás crear), e indica que debe mostrarse en el menú desplegable(*app:showAsAction="never"*).

- Modifica la clase *ui.FavouritesFragment* para que implemente la interfaz *MenuProvider* y sobrescribe sus dos métodos: *onCreateMenu()* y *onMenuItemSelected()*. Añade esta instancia como proveedor de menús a su actividad solo cuando el fragmento sea interactivo – *requireActivity().addMenuProvider(this, viewLifecycleOwner, Lifecycle.State.RESUMED)*. Modifica la propiedad que obtiene la instancia de *FavouritesViewModel* para que utilice la factoría que permite compartir la instancia entre fragmentos – *activityViewModels()*.
- El método *onCreateMenu()* debe utilizar los parámetros recibidos para crear el menú correspondiente al recurso definido – *menuInflater.inflate(R.menu.menu_favourites, menu)*. El método *onMenuItemSelected()* deberá mostrar el diálogo de confirmación – *findNavController().navigate(R.id.<acctionOrFragmentId>),* si el elemento seleccionado es el definido en el menú creado, y devolver como resultado *true*, o devolver como resultado *false* en otro caso.
- Si ejecutas la aplicación, accedes a la pantalla de mostrar la lista de citas favoritas y seleccionas el elemento de acción de borrado del menú de opciones, deberían desaparecer todas las citas de la pantalla.
- Sin embargo, esta opción solo debería aparecer en el menú de opciones si existen citas favoritas. Para gestionarlo, modifica la clase *FavouritesViewModel* y crea una nueva propiedad, de solo lectura (*StateFlow<Boolean>*), que determinará la visibilidad del menú de opciones de acuerdo con el contenido de la lista de citas favoritas (*.map {}* permitirá convertir un *Flow* de un tipo de datos en otro tipo de datos y *.stateIn {}* permitirá convertirlo en un *StateFlow*)–

```
val isDeleteAllMenuVisible = favouriteQuotations.map { list ->
    list.isNotEmpty()
}.stateIn(
    scope = viewModelScope,
    started = SharingStarted.WhileSubscribed(),
    initialValue = true
)
```

- Modifica el método *onViewCreated()* de la clase *ui.FavouritesFragment*, para observar esta propiedad e invalidar el menú de opciones cuando cambie su valor – *requireActivity().invalidateMenu()*. Sobrescribe el método *onPrepareMenu()* para mostrar u ocultar el elemento de acción del menú de opciones en base al valor de la propiedad observada.
- Si ejecutas nuevamente la aplicación, se ocultará el elemento de acción cuando la lista se encuentre vacía.

Ejercicio 18: Eliminar una cita mediante el gesto *swipe right*

Descripción: Una manera rápida y cómoda de eliminar citas, una a una, es mediante el gesto *swipe right*. Un *ItemTouchHelper* permitirá gestionar este gesto.

Pasos a seguir:

- Modifica la clase *ui.FavouritesViewModel* para crear un nuevo método que permita eliminar una cita a partir de su posición en el adaptador – *fun deleteQuotationAtPosition(position: Int) { }*. Deberás crear una copia del valor que almacena la propiedad encargada de mantener la lista, borrar la cita de la posición indicada (el método *.minus()* de *Kotlin* devuelve una lista sin el elemento pasado como parámetro) y asigna esa nueva lista como valor a la propiedad.
- Modifica la clase *ui.FavouritesFragment* para incluir una nueva propiedad de tipo *ItemTouchHelper*. Para crear su instancia precisarás de un objeto de tipo *SimpleCallback* – con parámetros *0* (para no utilizar *drag*) y *END* (para reaccionar a *swipe right*)– sobrescribiendo los métodos *onMove()* e *isLongpressDragEnabled()*, que deben devolver *false* para evitar el gesto *drag*, *isItemViewSwipeEnabled()*, que devolverá *true* para habilitar el gesto *swipe*, y *onSwiped()*, que se activará al detectar el gesto y deberá ejecutar el método *deleteQuotationAtPosition()* del *ViewModel* pasando como parámetro la posición del adaptador que proporcionará el *ViewHolder* recibido.
- Modifica el método *onViewCreated()* de este fragmento para asociar el *ItemTouchHelper* al *RecyclerView* y así que responda a los gestos *swipe* – *touchHelper.attachToRecyclerView(binding.recyclerView)*.
- Si ejecutas la aplicación deberías poder eliminar las citas, una a una, de la lista mediante el gesto *swipe right*. Si eliminas todas las citas debería ocultarse el elemento de acción de borrar todas las citas del menú de opciones.

Ejercicio 19: Obtener información del autor de la cita

Descripción: Al pulsar sobre cualquiera de las citas mostradas, se lanzará el navegador por defecto del dispositivo para acceder a la Wikipedia y mostrar la página web asociada al autor de la cita.

Pasos a seguir:

- Modifica el manifiesto (*AndroidManifest.xml*) para incluir una entrada *<queries/>* que indique que deseamos ejecutar *Intent* con acción *VIEW* y con *scheme* "*https*".
- Modifica la clase *ui.QuotationListAdapter* para que reciba como parámetro la función que debe ejecutarse al pulsar sobre algún elemento del *RecyclerView*. Esta función recibirá como parámetro el autor de la cita y no devolverá ningún valor – *val onItemClick: (String) -> Unit*. Modifica el constructor del *ViewHolder* para que también reciba una instancia de esta función. Inicializa el *ViewHolder* para que asocie un *OnClickListener* a la vista de más alto nivel de la jerarquía en el *Binding*. Este listener debe invocar a la función recibida en el constructor, pasándole el texto correspondiente al autor de la cita pulsada.

```
init {
    binding.root.setOnClickListener {
        onItemClick(binding.tvQuotationAuthorItem.text.toString())
    }
}
```

- Edita la clase *ui.FavouritesFragment* para pasar esta función como parámetro al adaptador. En ella se deberá comprobar si el autor de la cita es *"Anonymous"* y, en ese caso, muestra un *Snackbar* que indique que no es posible mostrar información si el autor es anónimo. Si no, entonces genera un *Intent* implícito con acción *ACTION_VIEW*. El dato que se desea visualizar es "*https://en.wikipedia.org/wiki/Special:Search?search=*" *+ authorName*, donde *authorName* será el nombre del autor de la cita. Lanza una actividad utilizando este *Intent* capturando la excepción *ActivityNotFoundException*, por si no hubiera ninguna actividad disponible para acceder a la página web (extraño, pero por si acaso). Al capturar la excepción, muestra un *Snackbar* que indique que no es posible gestionar la acción solicitada.
- Modifica la clase *ui.FavouritesViewModel* para que la lista de citas aleatorias incluya una cita de un personaje famoso, por ejemplo Albert Einstein", y una cita donde el autor sea "Anonymous".
- Ejecuta la aplicación y comprueba que pulsando el botón sobre cualquiera de las citas se lanza una actividad que muestra información acerca del autor (extraída de *Wikipedia*) y se muestra un *Snackbar* si el autor es anónimo.

Ejercicio 20: Internacionalización (*i18n*)

Descripción: No debe codificarse ninguna cadena de texto en el código fuente ni en los recursos, sino definirlas mediante recursos de tipo <*string/*>. El fichero de recursos (*res/values/strings.xml*) contendrá todas aquellas cadenas de texto que se utilicen en la aplicación.

Pasos a seguir:

- Edita el fichero de recursos *res/values/strings.xml*, abre el editor de soporte a la traducción (*Open editor*) y añade un nuevo idioma (*Add Locale*) para español, al menos, y cualquier otro idioma que desees (opcional).
- Traduce todas las cadenas de texto al idioma añadido. Marca *Untranslatable* para aquellas que no tengan traducción (como el nombre de la aplicación, por ejemplo).
- Comprueba que se genera un nuevo directorio *values-<language_code>* con un fichero *strings.xml* correspondiente a las etiquetas traducidas (vista *Project*). En la vista *Android* del entorno de desarrollo se mostrará como un fichero *strings.xml* acompañado del código del idioma añadido.
- Cambia el idioma del dispositivo y lanza a ejecución la aplicación. Comprueba que las diferentes etiquetas se traducen correctamente.

Ejercicio 21: Personalización del tema y estilos de la aplicación

Descripción: Cambios en el estilo de todos los componentes de un tipo dado pueden gestionarse más fácilmente si se incluyen en el tema definido para la aplicación. Cambios particulares para algunos componentes de un tipo dado, pero no para todos ellos, pueden definirse como estilos propios. Cambios en un único componente pueden definirse también como estilos o aplicarse directamente en las vistas implicadas.

Pasos a seguir:

- Edita el fichero de recursos *res/values/colors.xml* y define los colores que quieras utilizar en tu aplicación o utiliza el *Material Theme Builder* (https://material-foundation.github.io/material-theme-builder/) para generar los diferentes colores .
- Edita el fichero de recursos *res/values/themes.xml* y cambia los colores utilizados como primario, secundario, etc., en el tema definido.
- Para personalizar los componentes puedes crear un nuevo estilo basado en alguno ya definido por *Material Design* y modificar los atributos relevantes (consúltalos en *Material Design* para cada componente – *theming*).
    - Por ejemplo, para crear un nuevo estilo para los botones flotantes podrías definir primero un *ThemeOverlay* que indique los colores a modificar:

```
<style name="ThemeOverlay.App.FloatingActionButton" parent="">
  <item name="colorContainer”>
      @color/md_theme_tertiaryContainer
  </item>
  <item name="colorOnContainer”>
    @color/md_theme_onTertiaryContainer
  </item>
</style>
```

    - Este *ThemeOverlay* lo podrías asociar directamente a un único botón flotante mediante el atributo *app:materialThemeOverlay*. Pero podrías definir un nuevo estilo que se aplicara a los componentes seleccionados y que modifique este atributo (tendría más sentido si modificara, adicionalmente, más cosas):

```
<style name="Style.App.FloatingActionButton"
      parent="Widget.Material3.FloatingActionButton.Primary">
  <item name="materialThemeOverlay">
    @style/ThemeOverlay.App.FloatingActionButton
  </item>
</style>
```

    - Pero, si queremos que este estilo se aplique a todos los botones flotantes, en lugar de solo a algunos seleccionados, entonces podemos asociar este estilo directamente como parte del tema de la aplicación:

```
<item name="floatingActionButtonStyle">
  @style/Style.App.FloatingActionButton
</item>
```

- Personaliza tu aplicación conforme al estilo que desees y comprueba que es homogéneo para todos los componentes y pantallas.