

# Laboratorio 08 - RecyclerView

## 1. Introducción

En este laboratorio se brindará un acercamiento a la vista de Android llamada **RecyclerView**, dicha vista permite mostrar de manera eficiente y escalable una lista de elementos en pantalla. Además, RecyclerView en lugar de cargar todos los elementos de la lista a la vez, carga únicamente los elementos que están siendo visibles, lo que reduce la sobrecarga de memoria y mejora el rendimiento de la aplicación.

### 1.1 ¿Qué se aprenderá?

- Implementación de RecyclerView
- Envío de información a distintos fragmentos por medio del ViewModel

### 1.2. Requisitos

- Tener previamente instalado Android Studio en tu computadora
- Poder agregar una película usando Model, ViewModel & Repository
- Proyecto dividido en distintos paquetes según su funcionalidad

## 2. Características de RecyclerView

RecyclerView es una herramienta muy versátil que ofrece varias características para mejorar la experiencia de usuario y la eficiencia del desarrollo de aplicaciones. Algunas de las características más importantes son las siguientes:

- **Carga eficiente de elementos:** Únicamente se cargan los elementos que son visibles en pantalla y los que están en dirección de desplazamiento. De esta manera, se reduce la sobrecarga de memoria y se mejora el rendimiento de la aplicación, especialmente en el caso de listas con una gran cantidad de elementos.
- **Personalización de los elementos:** Ofrece la posibilidad de personalizar la apariencia de los elementos de la lista mediante uso de vistas personalizadas (xml item). Lo que permite crear interfaces de usuario atractivas para nuestra aplicación.
- **Animaciones:** RecyclerView permite agregar animaciones a los elementos de la lista, lo que mejora radicalmente la experiencia del usuario al interactuar con la aplicación.
- **Interacciones con elementos:** Es posible agregar interacciones a los elementos de la lista, como clicks y pulsaciones largas. Otorgando al usuario una interfaz interactiva y amigable.

- **Modo de selección:** RecyclerView permite habilitar un modo de selección, en el que el usuario puede seleccionar varios elementos de la lista. Esto es especialmente útil en el caso de aplicaciones que requieren la selección de varios elementos para realizar una acción determinada.

## 3. Componentes de RecyclerView

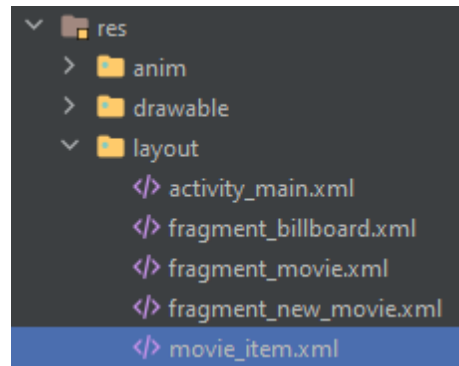
RecyclerView consta de varios componentes que trabajan juntos para mostrar una lista de elementos en pantalla. Sus componentes son los siguientes:

### 3.1 List Item

El archivo **list\_item.xml** se utiliza en conjunto con RecyclerView para definir la vista que se mostrará para cada elemento de la lista. Este archivo de diseño define la estructura visual y los elementos de diseño de la vista de cada elemento, ya sea texto, imágenes, botones, etc.

Procede a crear un **Layout Resource File** llamado **movie\_item** dentro del paquete **layout**. En él puedes utilizar la Card que fue creada previamente para **BillboardFragment**.

La estructura de directorios debería verse de la siguiente manera:



Y dentro de dicho archivo debes construir la siguiente vista, dicho elemento define la apariencia y comportamiento de cada elemento de la lista a la vista.

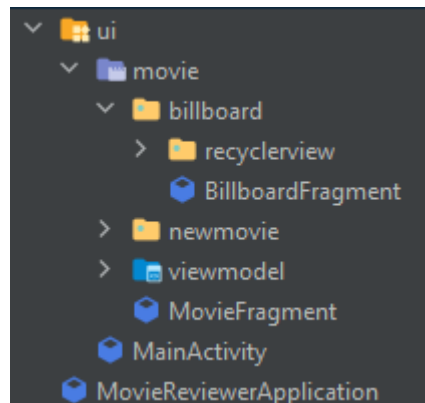


## 3.2 ViewHolder

El ViewHolder es un componente importante para el RecyclerView, ya que se utiliza para almacenar una referencia a la vista de cada elemento de la lista. Su principal función es optimizar el rendimiento del RecyclerView, permitiendo la reutilización de vistas y de reducir la carga en la memoria del dispositivo.

Cada vez que se desplaza la lista de elementos del Recycler, algunos elementos de la lista se mueven fuera de la vista y se vuelven a utilizar para mostrar nuevos elementos en pantalla. **En lugar de crear una nueva vista cada vez que se muestra un elemento, se reutiliza la vista del elemento anterior que ya no es visible,** es aquí donde el ViewHolder entra en juego.

Comienza a distribuir los paquetes de **ui** de la siguiente manera:



Y dentro de la carpeta **recyclerview** crea una nueva **Kotlin Class** llamada **MovieRecyclerViewHolder**, la cual recibe como argumento **MovieItemBinding** que se encarga de vincular al diseño de **movie\_item.xml** y dicha clase extiende de **RecyclerView.ViewHolder** que se inicializa con la vista de dicho elemento.

Además, dentro dicha clase se crea un función llamada **bind** que se encargará de referenciar a los elementos de la vista, dicha función recibe una **MovieModel** y la definición de una **función anónima** que se ejecutará cuando se haga clic en un elemento específico del RecyclerView y toma como parámetro el modelo de datos de la película.

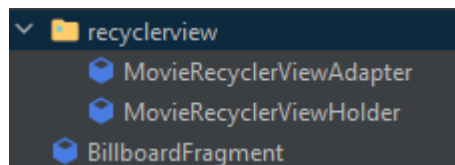
```
1 class MovieRecyclerViewHolder(private val binding: MovieItemBinding): RecyclerView.ViewHolder(binding.root) {
2     fun bind(movie: MovieModel, clickListener: (MovieModel) → Unit) {
3         binding.titleTextView.text = movie.name
4         binding.qualificationTextView.text = movie.qualification
5
6         binding.movieItemCardView.setOnClickListener {
7             clickListener(movie)
8         }
9     }
10 }
```

Seguidamente, se le brinda tanto el **name** como la **qualification** a cada una de las películas que será mostrada en una **Card**. Además, es necesario configurar un **setOnClickListener** para cada elemento del RecyclerView, es aquí donde se hace uso de la función anónima que necesita recibir una **movie** la cual ejecutará una acción cuando dicho elemento sea presionado.


### 3.3 Adapter

El Adapter dentro de un RecyclerView se encarga de vincular los datos de una lista o de un conjunto de datos con las vistas que se muestran en pantalla. El Adapter proporciona una manera eficiente de crear y reutilizar las vistas que se muestran en pantalla al tiempo que se asegura de que los datos se muestran en el orden y cantidad correctos.

Dentro del directorio **recyclerview** crea una **Kotlin Class** llamada **MovieRecyclerViewAdapter**



Dicha clase recibe la función anónima **clickListener** que posteriormente será utilizada, y se crea una nueva **ArrayList** de tipo **MovieModel**, dicho arreglo se encargará de almacenar las películas a mostrar dentro del RecyclerView



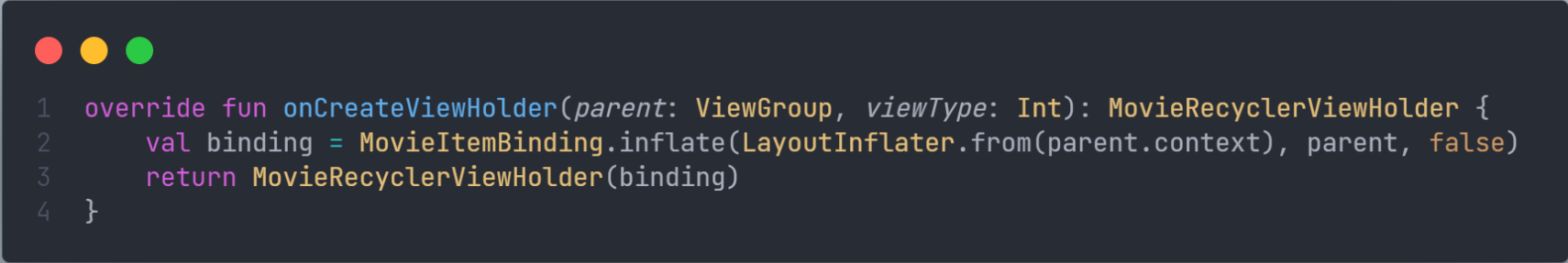
```
1 class MovieRecyclerViewAdapter(  
2     private val clickListener: (MovieModel) → Unit  
3 ) : RecyclerView.Adapter<MovieRecyclerViewHolder>() {  
4     private val movies = ArrayList<MovieModel>()  
5  
6 }
```

Además, dicha clase extiende de **RecyclerView.Adapter** que actúa como una interfaz, por ende es necesario implementar cada uno de sus métodos

Dichos métodos de **Adapter** son los siguientes:



- **onCreateViewHolder:** Dicho método es responsable de crear nuevas instancias de ViewHolder para los elementos de la lista cuando no hay uno disponible para ser reciclado.

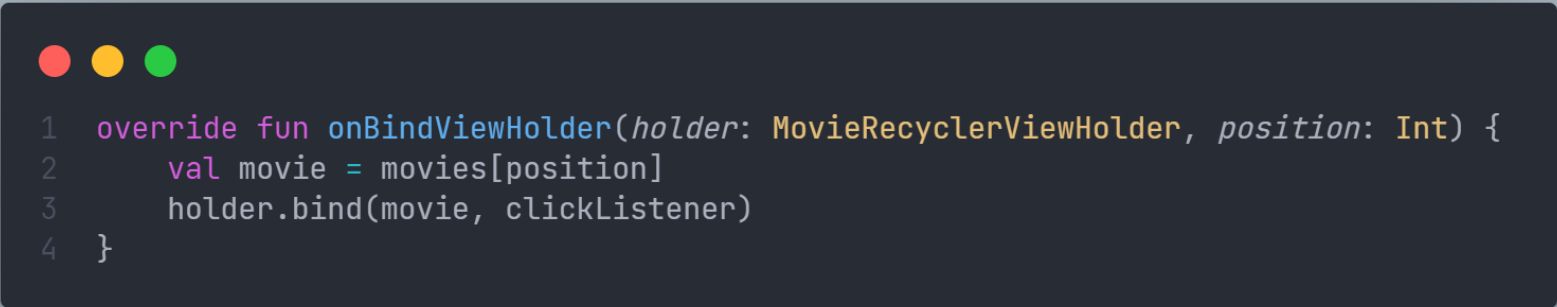


```
1  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieRecyclerViewHolder {  
2      val binding = MovieItemBinding.inflate(LayoutInflater.from(parent.context), parent, false)  
3      return MovieRecyclerViewHolder(binding)  
4  }
```

Dentro de dicho método se crea una variable **binding** a partir de la clase **MovieItemBinding** la cual es utilizada para inflar la vista del elemento de la lista. El método **inflate** recibe como parámetros un objeto **LayoutInflater** que utiliza el método **from** a partir del contexto del padre, el **parent** de la vista que en este caso sería **RecyclerView** y un valor booleano **false** que indica si la vista debe adjuntarse a su padre inmediatamente.

Y finalmente se crea una instancia de **MovieRecyclerViewHolder** enviando la vista inflada como argumento.

- **onBindViewHolder:** Dicho método es responsable de actualizar una vista existente en el RecyclerView con los datos correspondientes a un elemento de la lista.

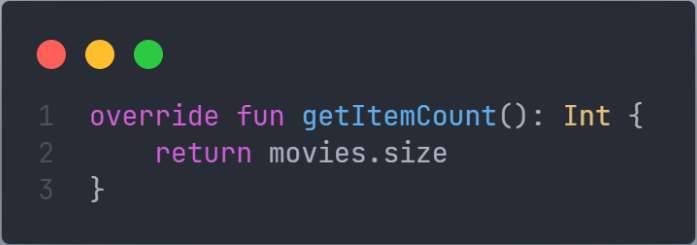


```
1  override fun onBindViewHolder(holder: MovieRecyclerViewHolder, position: Int) {  
2      val movie = movies[position]  
3      holder.bind(movie, clickListener)  
4  }
```

Se obtiene la película correspondiente a la **position** de la lista de películas que se encuentra en el adapter, lo que permite mostrar un orden coherente.

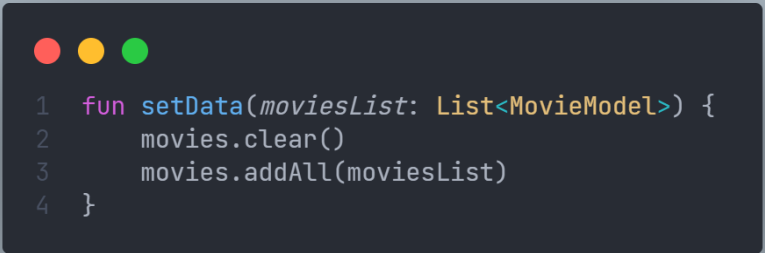
Luego se llama al método **bind** del ViewHolder (**holder**) que se encarga de actualizar los elementos de la vista con los datos de la película correspondiente, recibe los datos de dicha película y un clickListener para lanzar una acción.

- **getItemCount:** Dicho método se encarga de devolver el número de elementos que hay en la lista de datos que se ha proporcionado al Adapter.

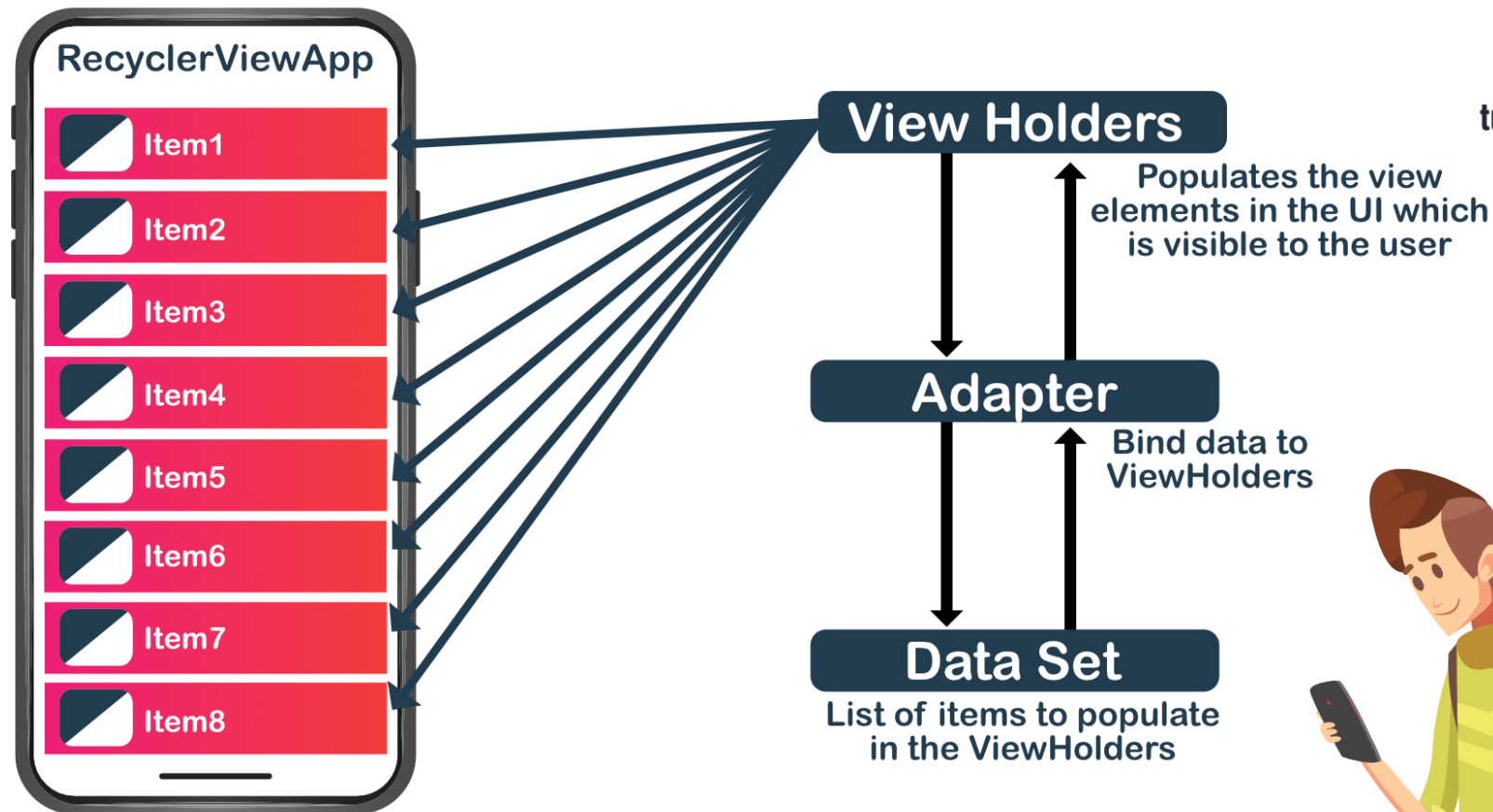


```
1  override fun getItemCount(): Int {  
2      return movies.size  
3  }
```

Finalmente, dentro del Adapter se crea una función que se encargará de actualizar la lista de datos que se va a mostrar en el RecyclerView, dicha función elimina los elementos existentes en la lista de películas para evitar que haya elementos duplicados o innecesarios. Y luego agrega todos los elementos que la lista de películas en el Adapter ahora contiene exactamente los mismos elementos que la lista proporcionada.

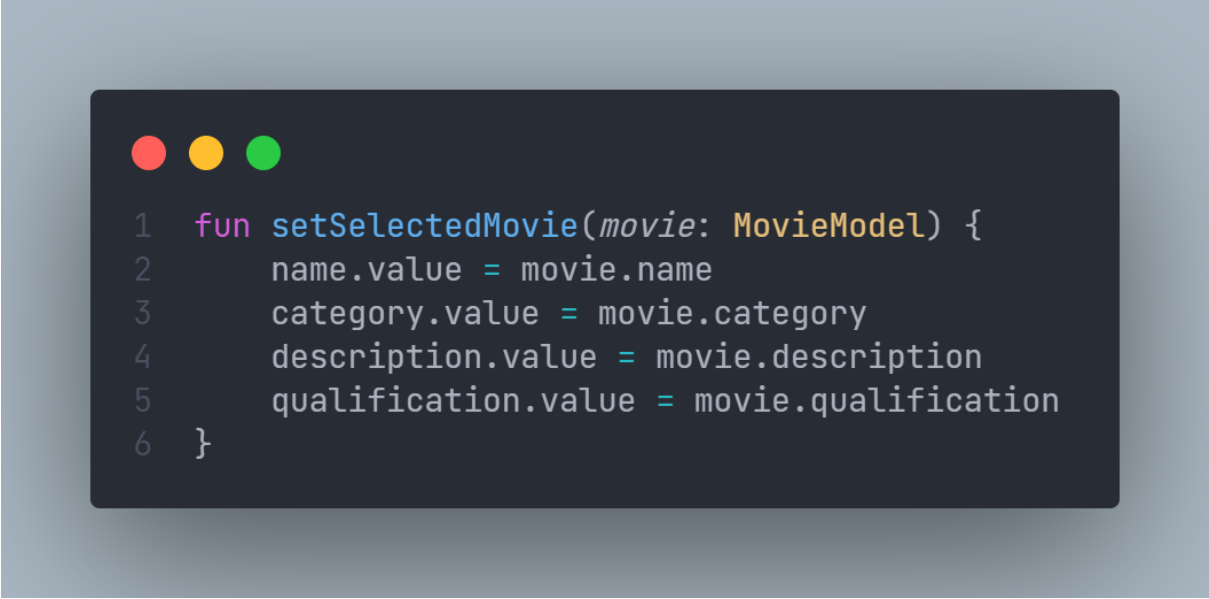


```
1  fun setData(moviesList: List<MovieModel>) {  
2      movies.clear()  
3      movies.addAll(moviesList)  
4  }
```



## 4. Configuraciones en ViewModel

Dentro del archivo **MovieViewModel.kt** se debe crear una función encargada de actualizar el estado del ViewModel en función de la película que sea seleccionada por el usuario. Dicha función toma como argumento un objeto **MovieModel** y actualiza las variables de tipo **MutableLiveData** con las propiedades correspondientes de la película seleccionada



```
1 fun setSelectedMovie(movie: MovieModel) {  
2     name.value = movie.name  
3     category.value = movie.category  
4     description.value = movie.description  
5     qualification.value = movie.qualification  
6 }
```

## 5. Configuraciones en BillboardFragment

Dentro del archivo **BillboardFragment.kt** se debe crear una variable llamada **adapter** de tipo **MovieRecyclerViewAdapter** dicha clase fue creada previamente

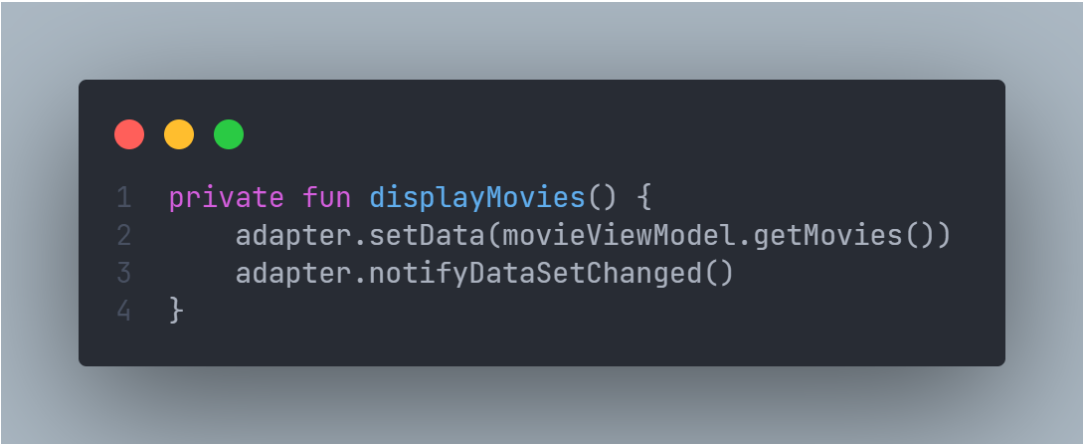
```
1 private lateinit var adapter: MovieRecyclerViewAdapter
```

Además, se crea una función que se encargará de establecer un objeto **movie** y cargar dicha información haciendo uso de la función **setSelectedMovie** creada previamente en el **ViewModel**, una vez se ha cargado la información de la película en el **MovieViewModel**, se procede a navegar al **MovieFragment**

```
1 private fun showSelectedItem(movie: MovieModel) {  
2     movieViewModel.setSelectedMovie(movie)  
3     findNavController().navigate(R.id.action_billboardFragment_to_movieFragment)  
4 }
```

Seguidamente, se crea una función que será la responsable de mostrar las películas en un RecyclerView, donde se llama al método del **Adapter** llamado **setData**, el cual se encarga de actualizar la lista de películas del adaptador con la nueva lista de películas pasada como argumento.

Luego de haber actualizado los datos en el adaptador se llama al método **notifyDataSetChanged** el cual se encarga de notificar a la vista que los datos han sufrido cambios y que deben ser actualizados.



```
1 private fun displayMovies() {  
2     adapter.setData(movieViewModel.getMovies())  
3     adapter.notifyDataSetChanged()  
4 }
```

Cabe recalcar que también es necesario crear una función que se encargará de configurar tanto el RecyclerView como su adaptador.

Primero se establece el **Layout Manager** del RV como un **LinearLayoutManager** lo que coloca los elementos en una lista vertical.

Luego, se crea una instancia de **MovieRecyclerViewAdapter** que recibe una función que lambda la cual se encarga de llamar a **showSelectedItem** y espera recibir la película que ha sido seleccionada.

A continuación, se establece el adaptador que ha sido creado al RecyclerView.

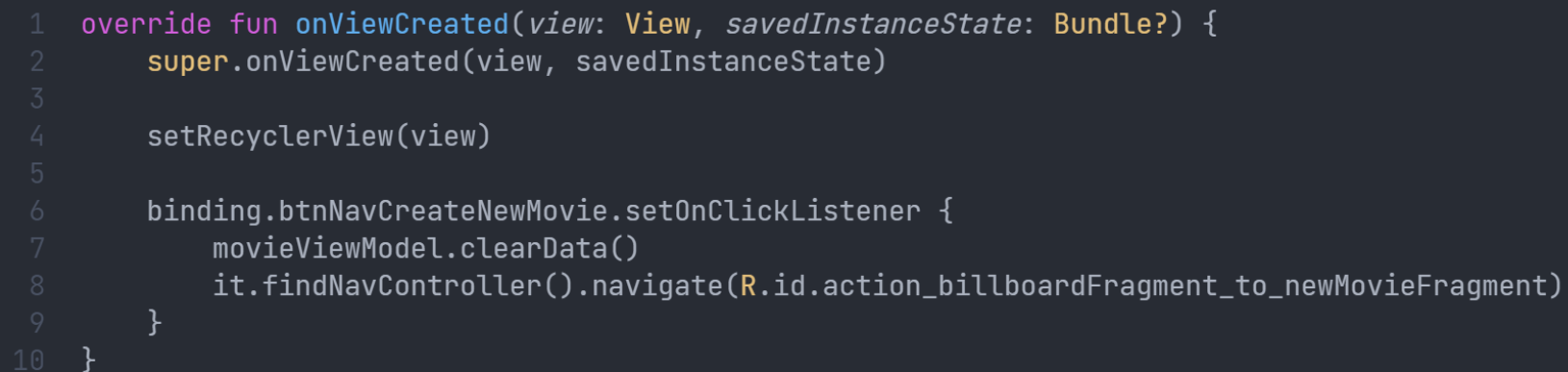
Y finalmente, se llama a la función **displayMovies** que actualiza los datos del adaptador con las películas recuperadas del ViewModel y notifica al RV de los cambios que han surgido con los datos.

No olvides que debes agregar un **widget** de **RecyclerView** en tu archivo `fragment_billboard.xml`

```
1 private fun setRecyclerView(view: View) {
2     binding.recyclerView.layoutManager = LinearLayoutManager(view.context)
3
4     adapter = MovieRecyclerViewAdapter { selectedMovie →
5         showSelectedItem(selectedMovie)
6     }
7
8     binding.recyclerView.adapter = adapter
9     displayMovies()
10 }
```



Además, la función **setRecyclerView** debe ser llamada dentro de **onViewCreated** y recibe como argumento la vista de la interfaz de usuario. Asimismo, cuando el **FloatingButton** sea presionado para navegar el fragmento que permite agregar una nueva película se limpian las variables del ViewModel para evitar cargar información innecesaria

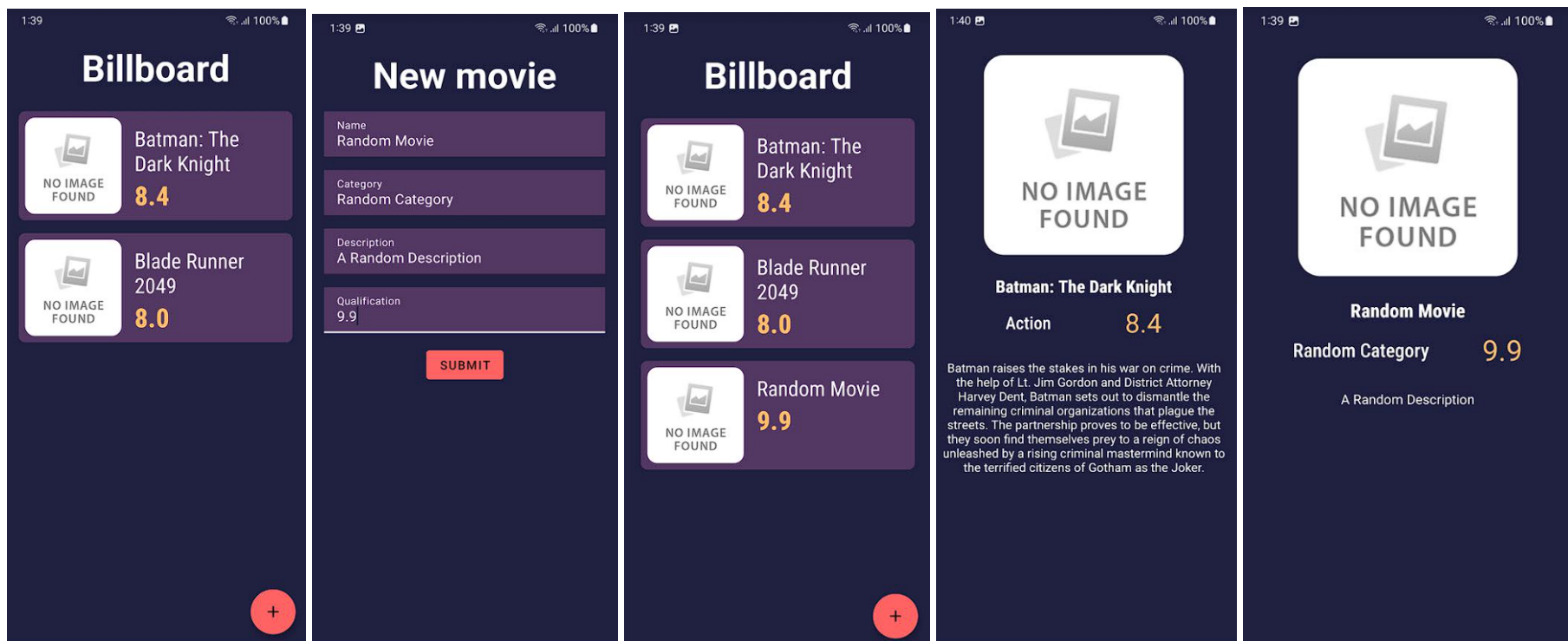


```
1  override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
2      super.onViewCreated(view, savedInstanceState)  
3  
4      setRecyclerView(view)  
5  
6      binding.btnNavCreateNewMovie.setOnClickListener {  
7          movieViewModel.clearData()  
8          it.findNavController().navigate(R.id.action_billboardFragment_to_newMovieFragment)  
9      }  
10 }
```

## 6. Actividad a realizar

Ya es posible renderizar en **BillboardFragment** tanto las películas almacenadas en nuestra fuente de datos como las que agrega el usuario en **NewMovieFragment**. Ahora debes encargarte de que cuando el usuario seleccione un elemento de RV la información de dicha película sea mostrada en **MovieFragment**.

**Tip:** Auxiliarte del **MovieViewModel** y del **Data Binding**



## 7. To do:

1. ¿Cuál es la **principal ventaja** de utilizar **RecyclerView**?

---

---

---

---

2. Menciona los **componentes de RecyclerView** y la **función que tiene cada uno de ellos**

---

---

---

---

---

---

## 8. Rúbrica

Actividad	Porcentaje	Nota
Elementos visuales posicionados correctamente	15%	
Implementación de ViewHolder	10%	
Implementación de Adapter	15%	
Es posible visualizar cuando una película es agregada en el RV de <b>BillboardFragment</b>	20%	
Es posible mostrar la información detallada de una película <b>(MovieFragment)</b>	20%	
Preguntas teóricas	10%	
Puntualidad	10%	
<b>Total</b>	<b>100%</b>	