Task ##P/C – Spike: Core 3

Goals:

The aim of this task is to develop an Android app that reads data from a file and displays it in a list using a RecyclerView. The list should be sortable and filterable, with variations between rows based on meeting types. Through this task, the following specific goals will be addressed.

- Understanding how to work with read-only files, lists, and menus in Android app development.
- Implementing RecyclerView to display data efficiently in a list format.
- Filtering data in the RecyclerView.

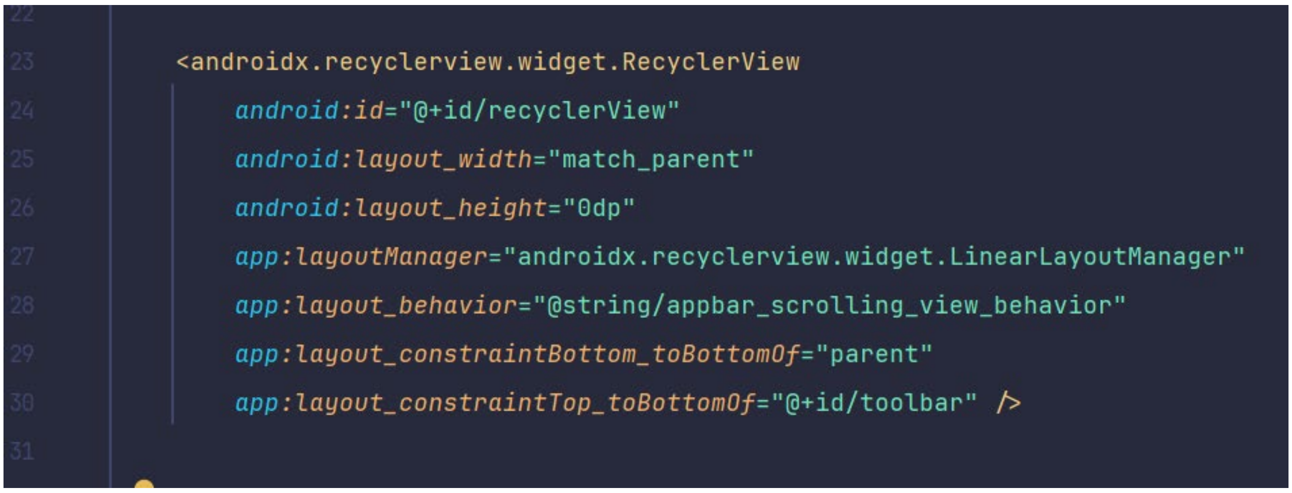
Tools and Resources Used

- Android Studio: Integrated Development Environment (IDE) for Android app development.
- RecyclerView: Android component for efficiently displaying large sets of data in a scrollable list.
- Toolbar: Android component for create menu.

Knowledge Gaps and Solutions

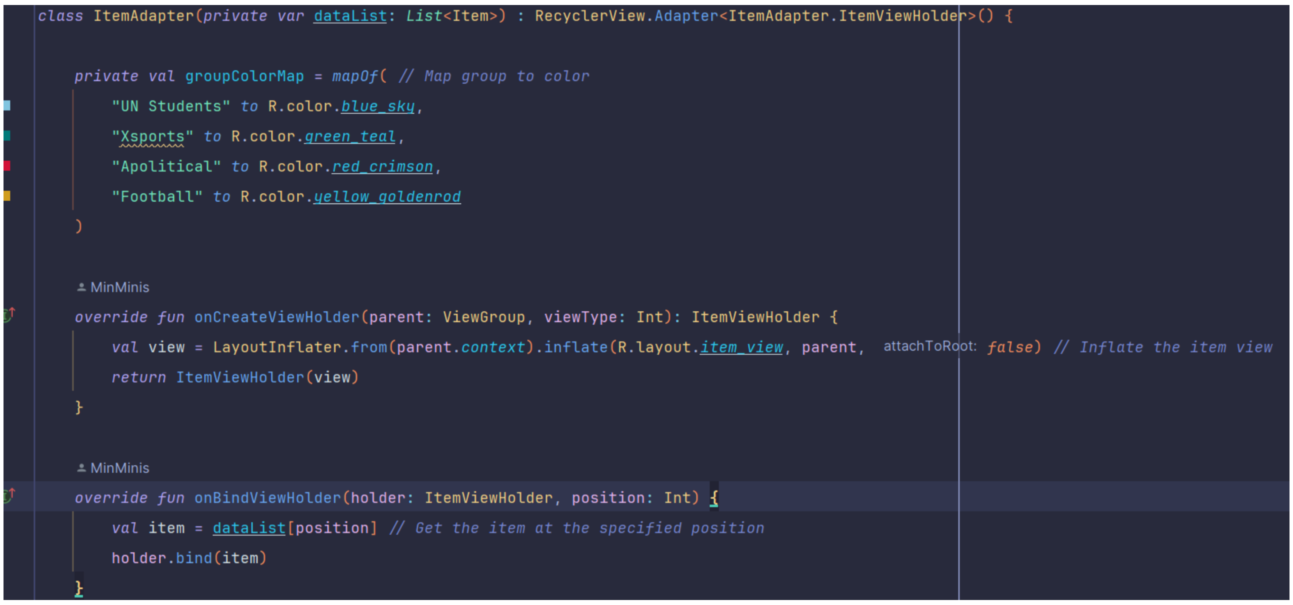
Gap 1: Understanding RecyclerView

To address this gap, I referred to the official Android documentation and completed tutorial exercises provided by Android Developers. This helped me understand the basic concepts of RecyclerView, including its advantages over ListView, such as improved performance and flexibility in managing large datasets.



```xml
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/toolbar" />
```

Image 1: RecycleView code in activity_main.xml



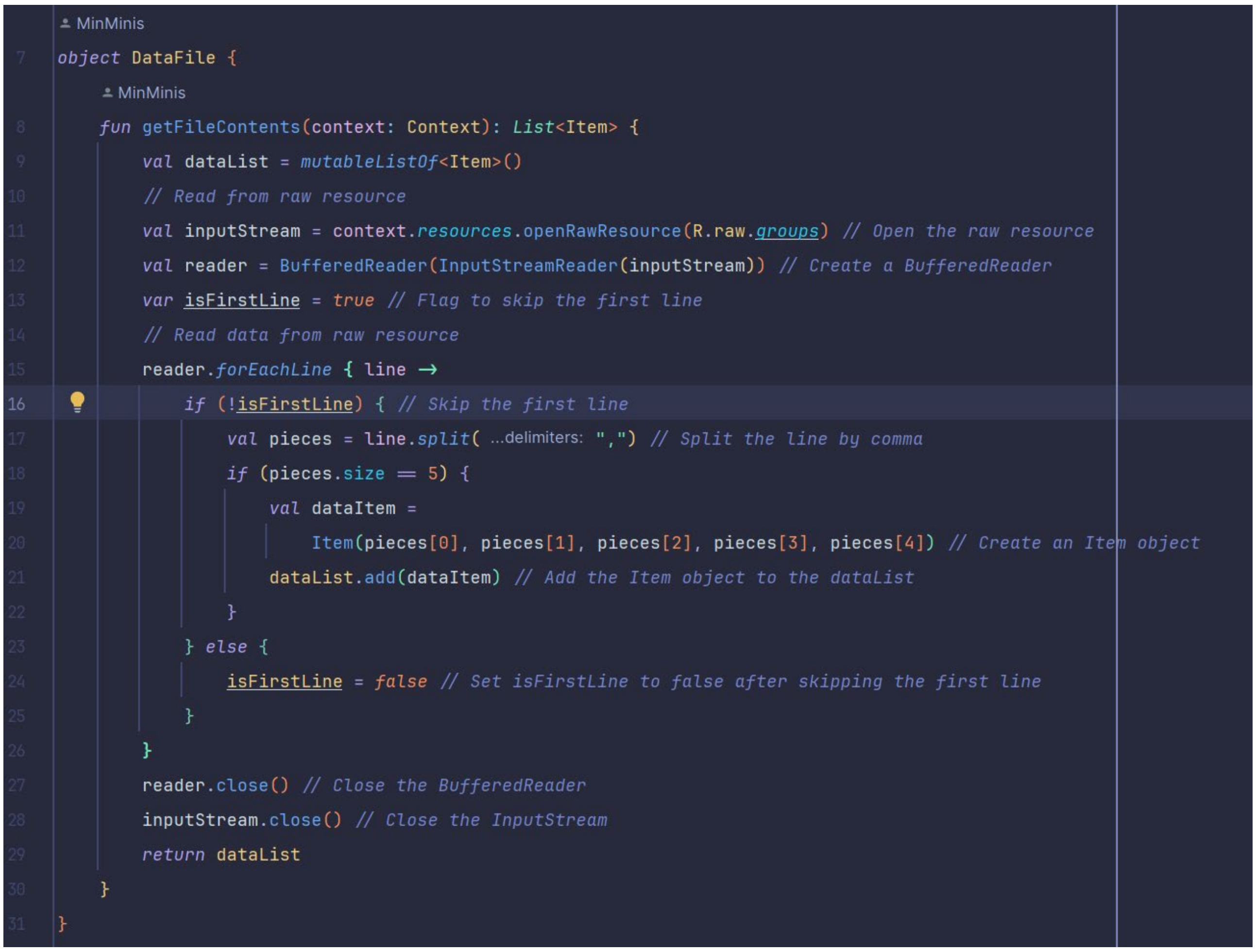
```kotlin
class ItemAdapter(private var dataList: List<Item>) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    private val groupColorMap = mapOf( // Map group to color
        "UN Students" to R.color.blue_sky,
        "Xsports" to R.color.green_teal,
        "Apolitical" to R.color.red_crimson,
        "Football" to R.color.yellow_goldenrod
    )

    MinMinis
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_view, parent,  attachToRoot: false) // Inflate the item view
        return ItemViewHolder(view)
    }

    MinMinis
    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        val item = dataList[position] // Get the item at the specified position
        holder.bind(item)
    }
```

Image 2: Adapter for RecyclerView

In image 2, the dataList property stores the list of the items to be displayed in the RecyclerView. It also holds the references to the views within the item's layout (R.layout.item_view).

Gap 2: Reading Data from a file.

I learned how to read data from a file stored in the /res/raw directory by using input stream readers in Android. This involved accessing the file using its resource identifier and reading its contents line by line to populate a data structure compatible with RecyclerView.



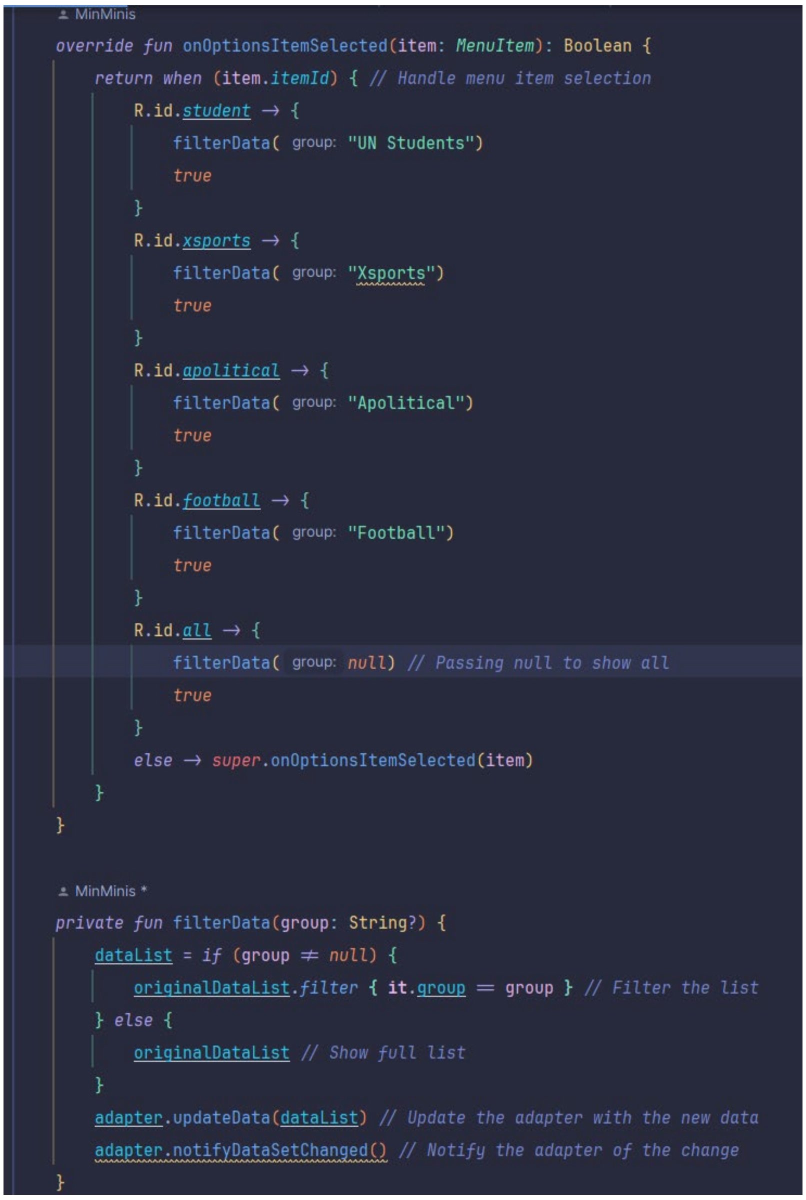
```kotlin
MinMinis
object DataFile {
    MinMinis
    fun getFileContents(context: Context): List<Item> {
        val dataList = mutableListOf<Item>()
        // Read from raw resource
        val inputStream = context.resources.openRawResource(R.raw.groups) // Open the raw resource
        val reader = BufferedReader(InputStreamReader(inputStream)) // Create a BufferedReader
        var isFirstLine = true // Flag to skip the first line
        // Read data from raw resource
        reader.forEachLine { line ->
            if (!isFirstLine) { // Skip the first line
                val pieces = line.split( ...delimiters: ",") // Split the line by comma
                if (pieces.size == 5) {
                    val dataItem =
                        Item(pieces[0], pieces[1], pieces[2], pieces[3], pieces[4]) // Create an Item object
                    dataList.add(dataItem) // Add the Item object to the dataList
                }
            } else {
                isFirstLine = false // Set isFirstLine to false after skipping the first line
            }
        }
        reader.close() // Close the BufferedReader
        inputStream.close() // Close the InputStream
        return dataList
    }
}
```

Image 3: DataFile object read data.

It opens the raw resource (R.raw.groups) which is the file named groups.csv for reading. I also use the InputStreamReader to efficiently read the file line by line. I also skip the first line because it store the headings.

Gap 3: Filtering Data

To implement sorting and filtering functionality, I utilized the ListAdapter or RecyclerView.Adapter classes in conjunction with the notifyDataSetChanged() method. Sorting was achieved by comparing the data based on specific criteria (e.g., date/time), while filtering involved selectively displaying items that match the user's chosen filter criteria.



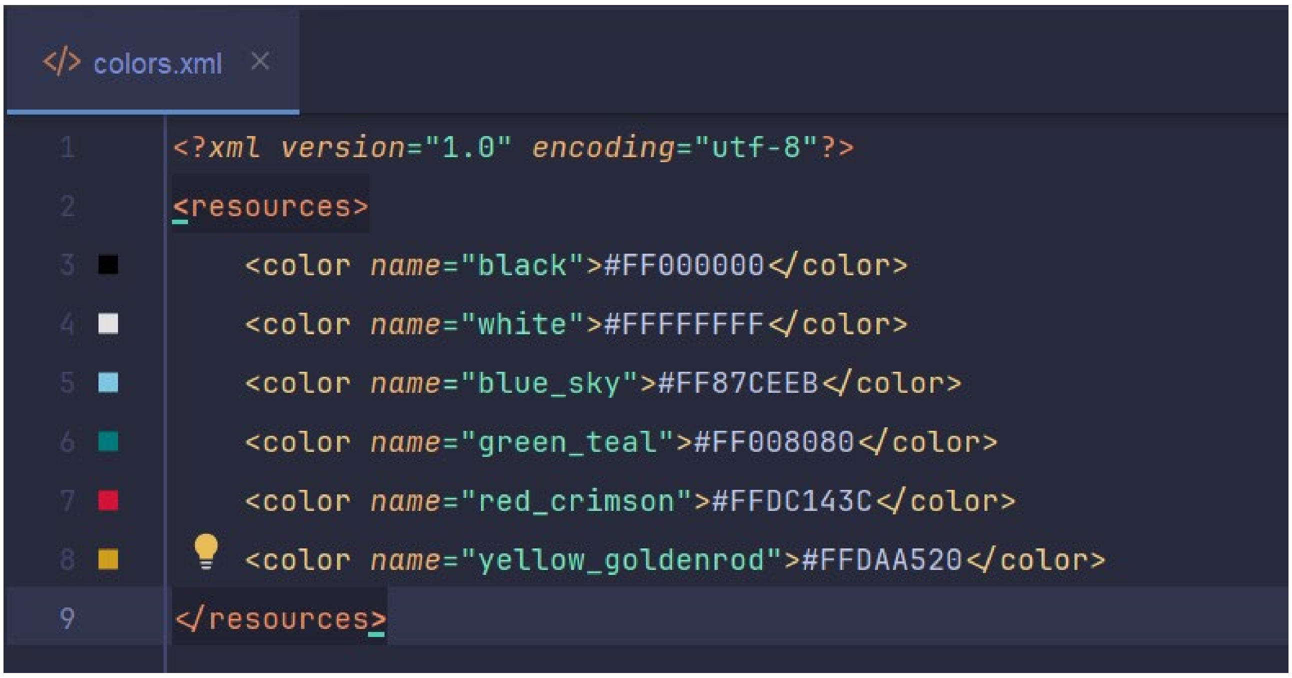
```
MinMinis
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) { // Handle menu item selection
        R.id.student → {
            filterData( group: "UN Students")
            true
        }
        R.id.xsports → {
            filterData( group: "Xsports")
            true
        }
        R.id.apolitical → {
            filterData( group: "Apolitical")
            true
        }
        R.id.football → {
            filterData( group: "Football")
            true
        }
        R.id.all → {
            filterData( group: null) // Passing null to show all
            true
        }
        else → super.onOptionsItemSelected(item)
    }
}

MinMinis *
private fun filterData(group: String?) {
    dataList = if (group ≠ null) {
        originalDataList.filter { it.group == group } // Filter the list
    } else {
        originalDataList // Show full list
    }
    adapter.updateData(dataList) // Update the adapter with the new data
    adapter.notifyDataSetChanged() // Notify the adapter of the change
}
```

Image 4: feature of filtering

By using onOptionsItemSelected for handling menu item selection. If the group is selected, filters the original list, if all is selected,shows the entire list.

Gap 4: Adding Variations between lists items.

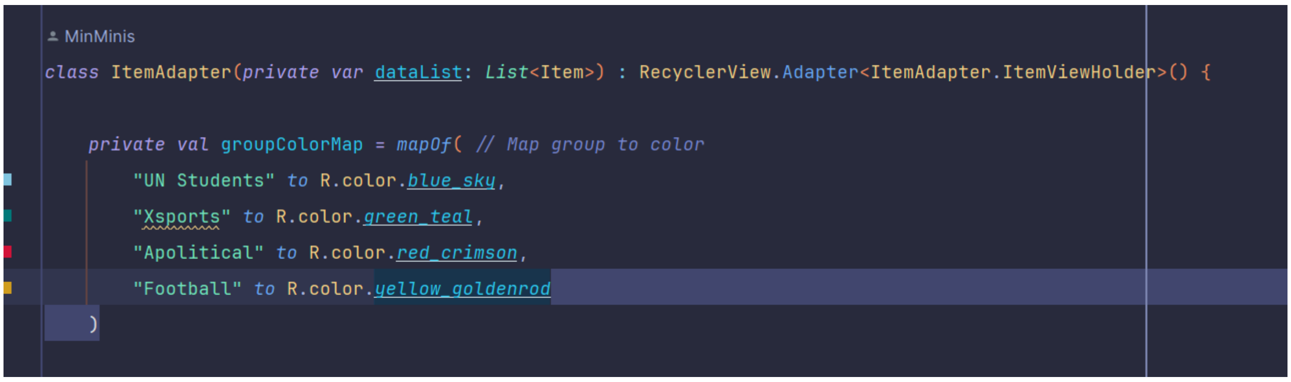
To introduce variations between list items, such as different colors or icons based on meeting types, I customized the RecyclerView's item layout and adapter logic. By dynamically setting properties like background color or icon visibility, I was able to visually differentiate between different types of meetings.



colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="blue_sky">#FF87CEEB</color>
    <color name="green_teal">#FF008080</color>
    <color name="red_crimson">#FFDC143C</color>
    <color name="yellow_goldenrod">#FFDAA520</color>
</resources>
```

Image 5: Define color

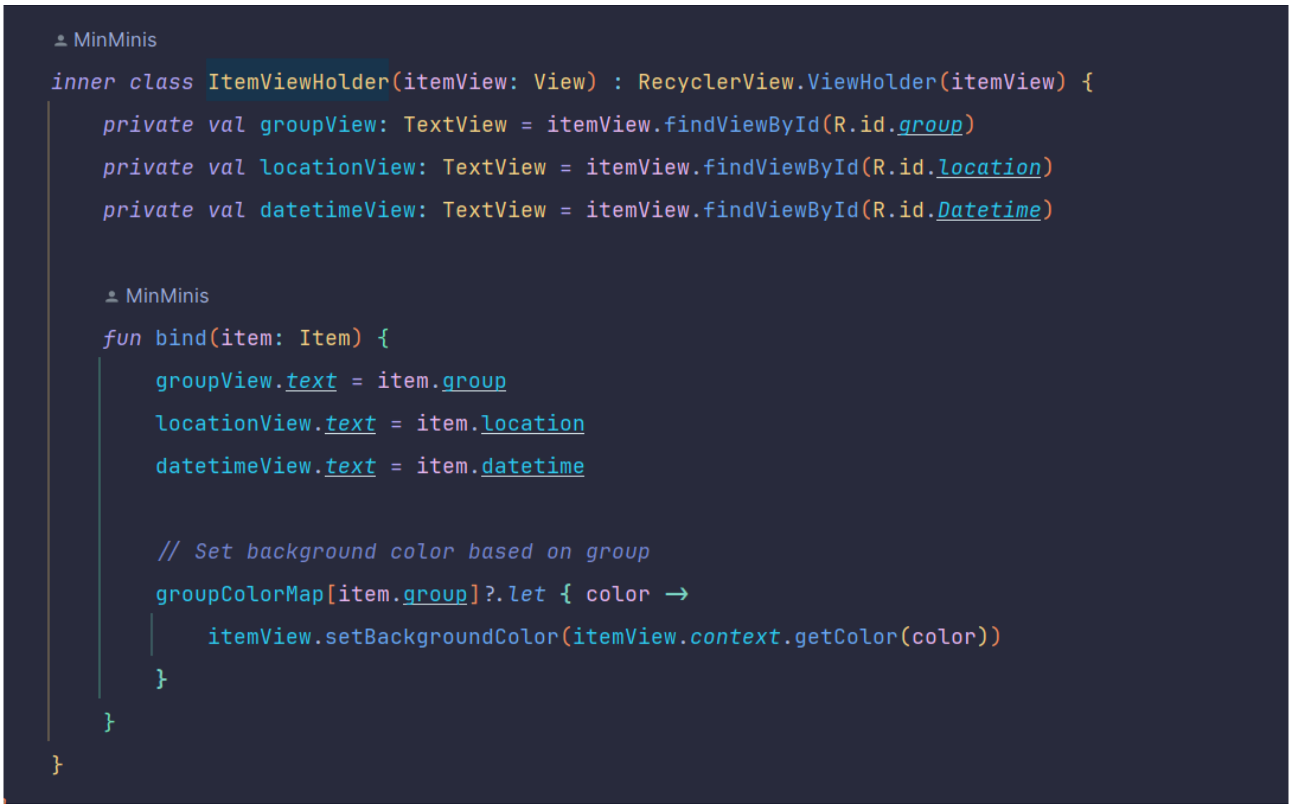


MinMinis

```kotlin
class ItemAdapter(private var dataList: List<Item>) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    private val groupColorMap = mapOf( // Map group to color
        "UN Students" to R.color.blue_sky,
        "Xsports" to R.color.green_teal,
        "Apolitical" to R.color.red_crimson,
        "Football" to R.color.yellow_goldenrod
    )
```

Image 6: Map group to color

In Image 5 and 6, I have defined the colours and map it to the group for displaying it. After that , in the ItemAdapter I have set background colour based on group



```
MinMinis
inner class ItemViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    private val groupView: TextView = itemView.findViewById(R.id.group)
    private val locationView: TextView = itemView.findViewById(R.id.location)
    private val datetimeView: TextView = itemView.findViewById(R.id.Datetime)

    MinMinis
    fun bind(item: Item) {
        groupView.text = item.group
        locationView.text = item.location
        datetimeView.text = item.datetime

        // Set background color based on group
        groupColorMap[item.group]?.let { color ->
            itemView.setBackgroundColor(itemView.context.getColor(color))
        }
    }
}
```

Image 7: Set background color

Open Issues and Recommendations

One open issue is the lack of support for adding new data to the read-only file (groups.csv) used in the app. To address this in the future, the app could incorporate functionality for user input and data persistence, allowing users to add new meeting entries.

Another recommendation is to enhance the filtering mechanism by implementing a more user-friendly interface, such as dropdown menus or search functionality, to make it easier for users to find specific types of meetings. Additionally, considering the accessibility guidelines for colour contrast and icon usage would improve the overall usability of the app for all users.