Task Core 3 – The Clubhouse

Link to GitHub repository: https://github.com/SoftDevMobDevJan2024/core3-104177995

Goals:

- Develop an app that reads data from a file and displays it in a list.
- Implement a RecyclerView with sorting and filtering capabilities.
- Utilize read-only files effectively.
- Design a user-friendly interface with accessible and visually appealing elements.
- Integrate complex UI widgets and differential row styling.
- Ensure robust error handling and user notifications.

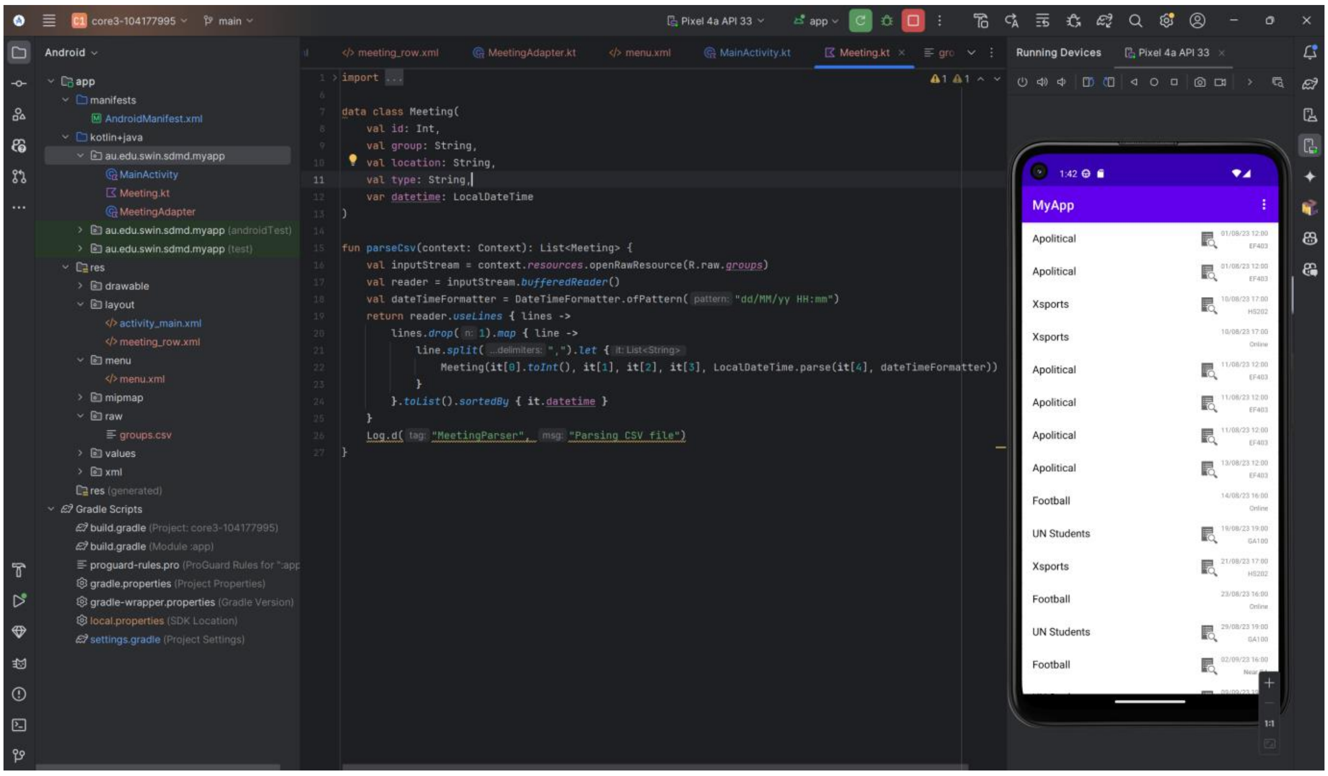
Tools and Resources Used

- The course's modules
- GitHub and Git
- Kotlin programming language and XML files
- Android Studio IDE
- Android UI components, themes, and styles

Knowledge Gaps and Solutions

Gap 1: Understand the filesystem

In this project, I demonstrated my understanding of the Android filesystem by effectively working with read-only files, specifically CSV files stored in the **res/raw** directory. The **parseCsv** function in **Meeting.kt** reads from a CSV file (**groups.csv**) containing meeting data. This function utilizes the Context to access resources, opens the raw resource file using `context.resources.openRawResource(R.raw.groups),` and reads its contents. This approach showcases my proficiency in handling read-only files within the Android resource system, enabling the app to efficiently load and process static data.

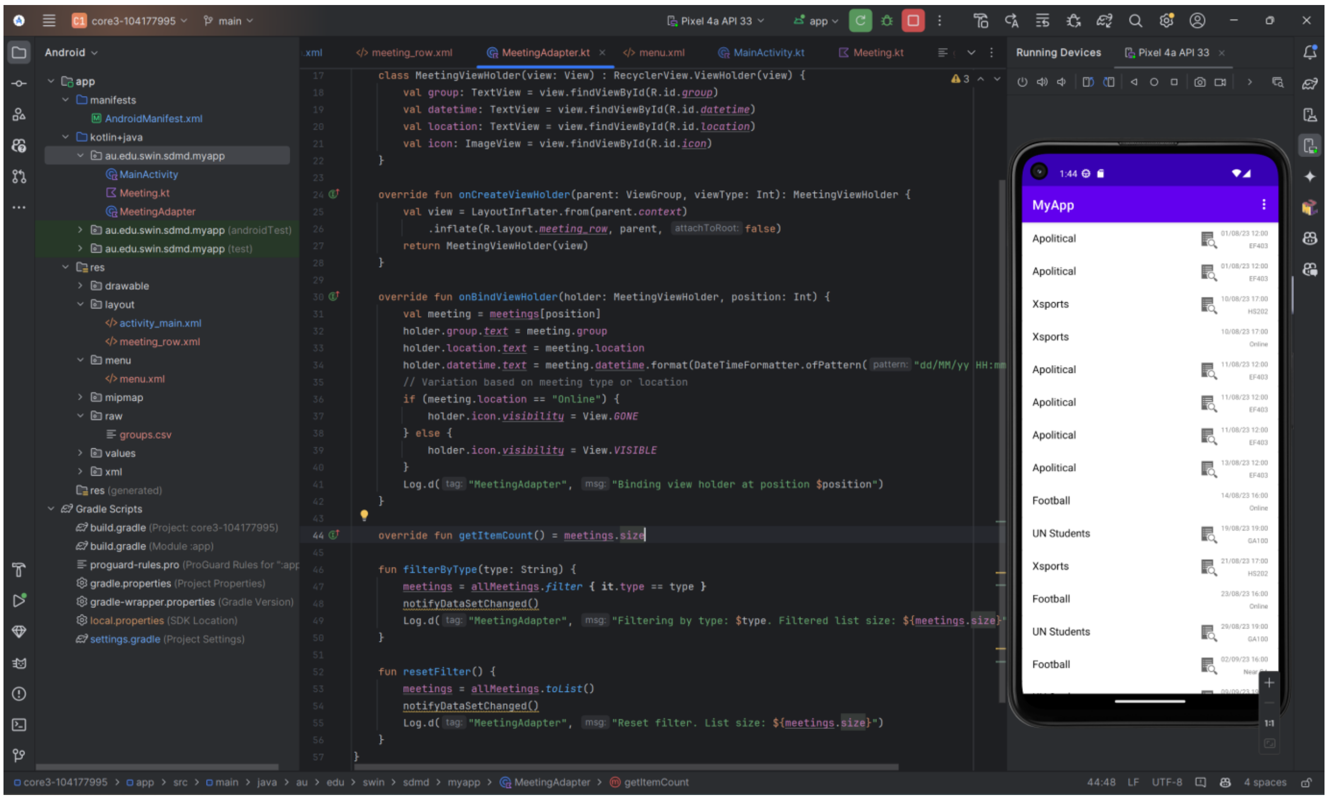


```kotlin
import ...

data class Meeting(
    val id: Int,
    val group: String,
    val location: String,
    val type: String,
    var datetime: LocalDateTime
)

fun parseCsv(context: Context): List<Meeting> {
    val inputStream = context.resources.openRawResource(R.raw.groups)
    val reader = inputStream.bufferedReader()
    val dateTimeFormatter = DateTimeFormatter.ofPattern( pattern: "dd/MM/yy HH:mm")
    return reader.useLines { lines ->
        lines.drop( n: 1).map { line ->
            line.split( ...delimiters: ",").let { it: List<String>
                Meeting(it[0].toInt(), it[1], it[2], it[3], LocalDateTime.parse(it[4], dateTimeFormatter))
            }
        }.toList().sortedBy { it.datetime }
    }
    Log.d( tag: "MeetingParser", msg: "Parsing CSV file")
}
```

```
MyApp
Apolitical     01/08/23 12:00  EF403
Apolitical     01/08/23 12:00  EF403
Xsports        10/08/23 17:00  HS202
Xsports        10/08/23 17:00  Online
Apolitical     11/08/23 12:00  EF403
Apolitical     11/08/23 12:00  EF403
Apolitical     11/08/23 12:00  EF403
Apolitical     13/08/23 12:00  EF403
Football       14/08/23 16:00  Online
UN Students    19/08/23 19:00  GA100
Xsports        21/08/23 17:00  HS202
Football       23/08/23 16:00  Online
UN Students    29/08/23 19:00  GA100
Football       02/09/23 16:00  Near
```

Gap 2: List performance and adapters

The project highlights the list performance issues and the development of appropriate adapters through the implementation of **MeetingAdapter**. This adapter manages a list of **Meeting** objects and binds them to a **RecyclerView,** known for its efficiency in handling large datasets. The **onBindViewHolder** method ensures smooth item binding, while the **filterByType** and **resetFilter** methods enhance performance by dynamically adjusting the dataset. **filterByType** narrows the displayed items, improving rendering efficiency, while **resetFilter** restores the full dataset without reloading, ensuring the **RecyclerView** remains responsive and efficient during complex data operations like filtering.



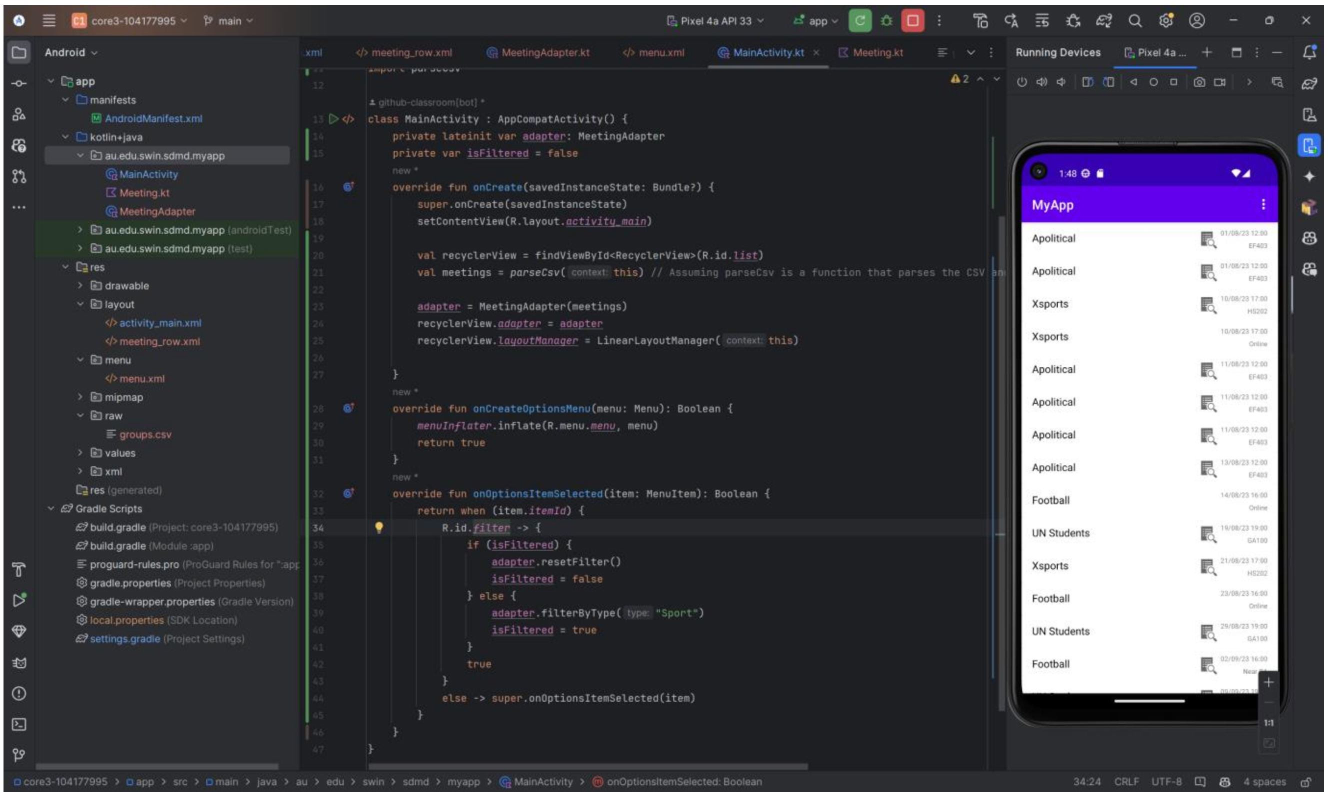
```kotlin
    class MeetingViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val group: TextView = view.findViewById(R.id.group)
        val datetime: TextView = view.findViewById(R.id.datetime)
        val location: TextView = view.findViewById(R.id.location)
        val icon: ImageView = view.findViewById(R.id.icon)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MeetingViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.meeting_row, parent, false)
        return MeetingViewHolder(view)
    }

    override fun onBindViewHolder(holder: MeetingViewHolder, position: Int) {
        val meeting = meetings[position]
        holder.group.text = meeting.group
        holder.location.text = meeting.location
        holder.datetime.text = meeting.datetime.format(DateTimeFormatter.ofPattern("dd/MM/yy HH:mm
        // Variation based on meeting type or location
        if (meeting.location == "Online") {
            holder.icon.visibility = View.GONE
        } else {
            holder.icon.visibility = View.VISIBLE
        }
        Log.d("MeetingAdapter", "Binding view holder at position $position")
    }

    override fun getItemCount() = meetings.size

    fun filterByType(type: String) {
        meetings = allMeetings.filter { it.type == type }
        notifyDataSetChanged()
        Log.d("MeetingAdapter", "Filtering by type: $type. Filtered list size: ${meetings.size}"
    }

    fun resetFilter() {
        meetings = allMeetings.toList()
        notifyDataSetChanged()
        Log.d("MeetingAdapter", "Reset filter. List size: ${meetings.size}")
    }
}
```

MyApp

Apolitical	01/08/23 12:00	EF403
Apolitical	01/08/23 12:00	EF403
Xsports	10/08/23 17:00	HS202
Xsports	10/08/23 17:00	Online
Apolitical	11/08/23 12:00	EF403
Apolitical	11/08/23 12:00	EF403
Apolitical	11/08/23 12:00	EF403
Apolitical	13/08/23 12:00	EF403
Football	14/08/23 16:00	Online
UN Students	19/08/23 19:00	GA100
Xsports	21/08/23 17:00	HS202
Football	23/08/23 16:00	Online
UN Students	29/08/23 19:00	GA100
Football	02/09/23 16:00	Near

Gap 3: Options menu

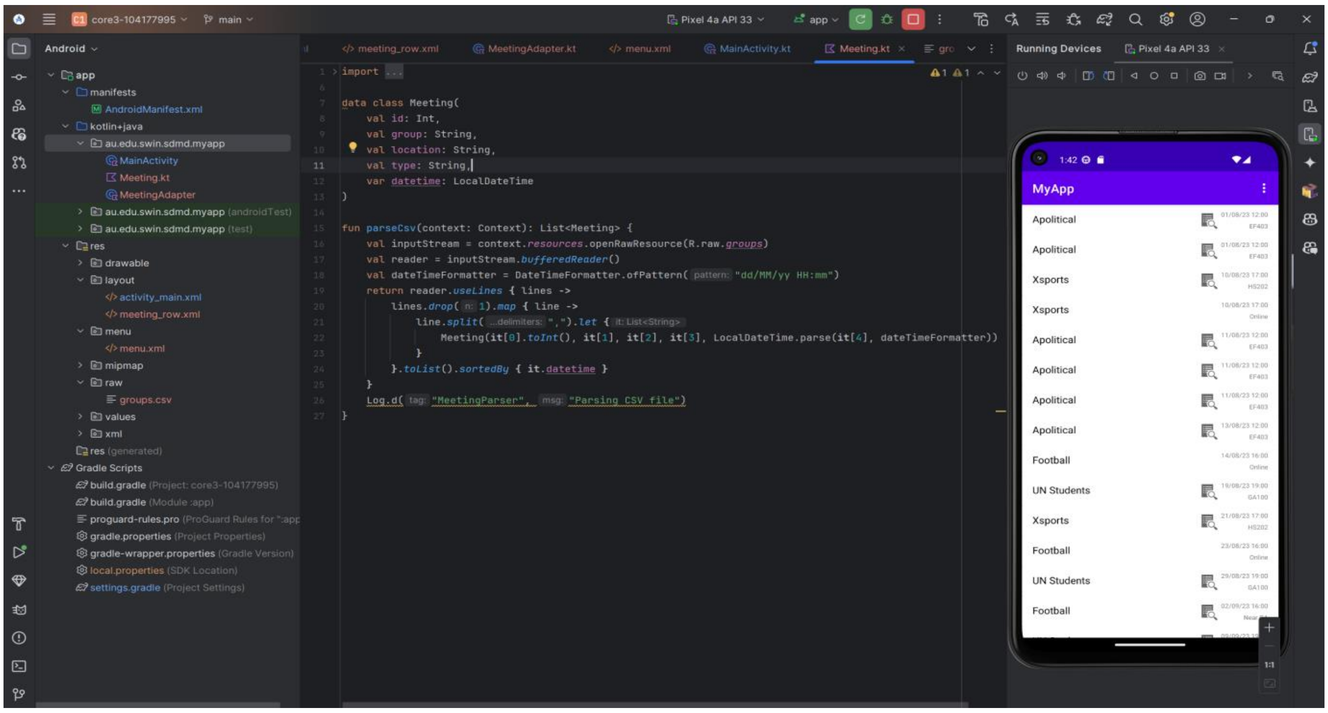
I demonstrated my ability to create an options menu in **MainActivity.kt** through the **onCreateOptionsMenu** and **onOptionsItemSelected** methods. By using `menuInflater.inflate(R.menu.menu, menu)` in **onCreateOptionsMenu** to inflate the menu from an XML resource (**menu.xml**), I added items to the action bar. The **onOptionsItemSelected** method handles item selection events, specifically filtering the list of meetings when the filter option is selected. This implementation showcases my capability to add interactive elements to the app's interface, enhancing user experience.



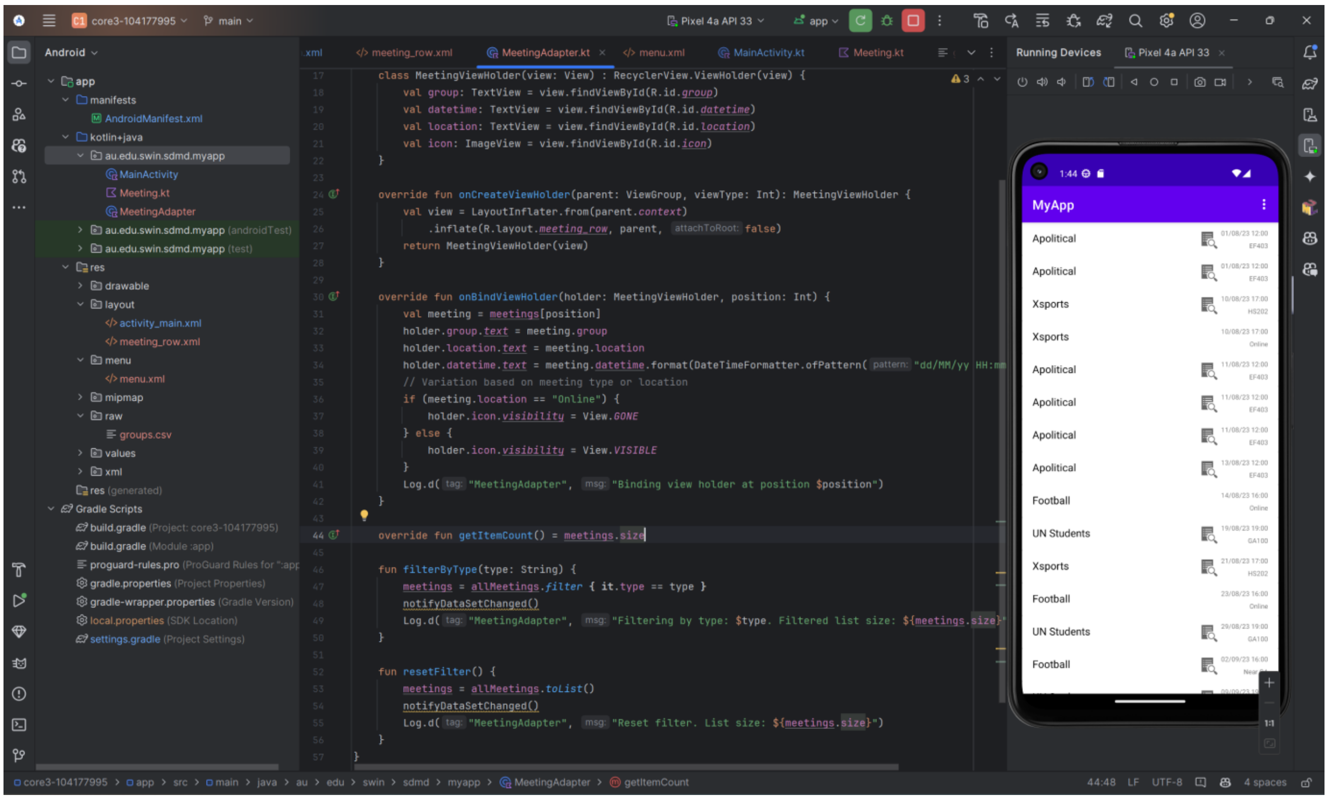
```kotlin
class MainActivity : AppCompatActivity() {
    private lateinit var adapter: MeetingAdapter
    private var isFiltered = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val recyclerView = findViewById<RecyclerView>(R.id.list)
        val meetings = parseCsv(this) // Assuming parseCsv is a function that parses the CSV an

        adapter = MeetingAdapter(meetings)
        recyclerView.adapter = adapter
        recyclerView.layoutManager = LinearLayoutManager(this)

    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.menu, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.filter -> {
                if (isFiltered) {
                    adapter.resetFilter()
                    isFiltered = false
                } else {
                    adapter.filterByType("Sport")
                    isFiltered = true
                }
                true
            }
            else -> super.onOptionsItemSelected(item)
        }
    }
}
```

Gap 4: Work with list data

Working with list data, including sorting and filtering, is a key feature of this project. Central to this functionality are the **MeetingAdapter** class and the **parseCsv** function in **Meeting.kt.** The **parseCsv** function sorts meetings by date immediately after parsing, ensuring that data is presented chronologically. Additionally, the **MeetingAdapte**r provides **filterByType** and **resetFilter** methods, allowing users to filter meetings by type and reset the filter, respectively. These features demonstrate the app's ability to dynamically manipulate list data, catering to user preferences and improving data presentation.



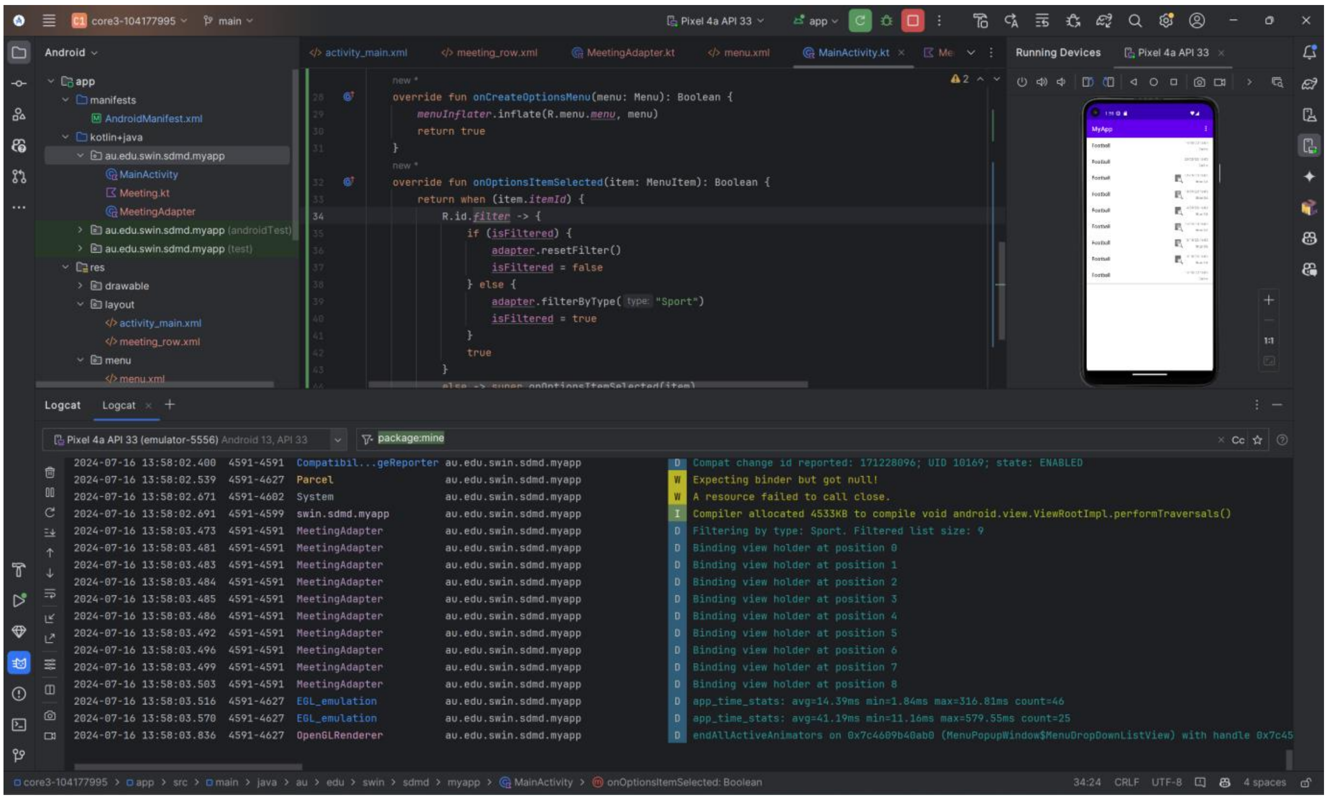
```kotlin
import ...

data class Meeting(
    val id: Int,
    val group: String,
    val location: String,
    val type: String,
    var datetime: LocalDateTime
)

fun parseCsv(context: Context): List<Meeting> {
    val inputStream = context.resources.openRawResource(R.raw.groups)
    val reader = inputStream.bufferedReader()
    val dateTimeFormatter = DateTimeFormatter.ofPattern("dd/MM/yy HH:mm")
    return reader.useLines { lines ->
        lines.drop(1).map { line ->
            line.split(",").let {
                Meeting(it[0].toInt(), it[1], it[2], it[3], LocalDateTime.parse(it[4], dateTimeFormatter))
            }
        }.toList().sortedBy { it.datetime }
    }
    Log.d("MeetingParser", "Parsing CSV file")
}
```



```kotlin
    class MeetingViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val group: TextView = view.findViewById(R.id.group)
        val datetime: TextView = view.findViewById(R.id.datetime)
        val location: TextView = view.findViewById(R.id.location)
        val icon: ImageView = view.findViewById(R.id.icon)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MeetingViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.meeting_row, parent, false)
        return MeetingViewHolder(view)
    }

    override fun onBindViewHolder(holder: MeetingViewHolder, position: Int) {
        val meeting = meetings[position]
        holder.group.text = meeting.group
        holder.location.text = meeting.location
        holder.datetime.text = meeting.datetime.format(DateTimeFormatter.ofPattern("dd/MM/yy HH:mm
        // Variation based on meeting type or location
        if (meeting.location == "Online") {
            holder.icon.visibility = View.GONE
        } else {
            holder.icon.visibility = View.VISIBLE
        }
        Log.d("MeetingAdapter", "Binding view holder at position $position")
    }

    override fun getItemCount() = meetings.size

    fun filterByType(type: String) {
        meetings = allMeetings.filter { it.type == type }
        notifyDataSetChanged()
        Log.d("MeetingAdapter", "Filtering by type: $type. Filtered list size: ${meetings.size}"
    }

    fun resetFilter() {
        meetings = allMeetings.toList()
        notifyDataSetChanged()
        Log.d("MeetingAdapter", "Reset filter. List size: ${meetings.size}")
    }
}
```

Gap 5: Command of IDE

The project demonstrates adept use of Android Studio's integrated development environment (IDE), particularly leveraging the LogCat for debugging. Throughout the development process, strategic placement of log statements has been employed to monitor the application's behavior and troubleshoot issues effectively.



```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.menu, menu)
    return true
}
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.filter -> {
            if (isFiltered) {
                adapter.resetFilter()
                isFiltered = false
            } else {
                adapter.filterByType( type: "Sport")
                isFiltered = true
            }
            true
        }
```

```
2024-07-16 13:58:02.400  4591-4591  Compatibil...geReporter au.edu.swin.sdmd.myapp  D  Compat change id reported: 171228096; UID 10169; state: ENABLED
2024-07-16 13:58:02.539  4591-4627  Parcel                  au.edu.swin.sdmd.myapp  W  Expecting binder but got null!
2024-07-16 13:58:02.671  4591-4602  System                  au.edu.swin.sdmd.myapp  W  A resource failed to call close.
2024-07-16 13:58:02.691  4591-4599  swin.sdmd.myapp         au.edu.swin.sdmd.myapp  I  Compiler allocated 4533KB to compile void android.view.ViewRootImpl.performTraversals()
2024-07-16 13:58:03.473  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Filtering by type: Sport. Filtered list size: 9
2024-07-16 13:58:03.481  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 0
2024-07-16 13:58:03.483  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 1
2024-07-16 13:58:03.484  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 2
2024-07-16 13:58:03.485  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 3
2024-07-16 13:58:03.486  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 4
2024-07-16 13:58:03.492  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 5
2024-07-16 13:58:03.496  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 6
2024-07-16 13:58:03.499  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 7
2024-07-16 13:58:03.503  4591-4591  MeetingAdapter          au.edu.swin.sdmd.myapp  D  Binding view holder at position 8
2024-07-16 13:58:03.516  4591-4627  EGL_emulation           au.edu.swin.sdmd.myapp  D  app_time_stats: avg=14.39ms min=1.84ms max=316.81ms count=46
2024-07-16 13:58:03.570  4591-4627  EGL_emulation           au.edu.swin.sdmd.myapp  D  app_time_stats: avg=41.19ms min=11.16ms max=579.55ms count=25
2024-07-16 13:58:03.836  4591-4627  OpenGLRenderer          au.edu.swin.sdmd.myapp  D  endAllActiveAnimators on 0x7c4609b40ab0 (MenuPopupWindow$MenuDropDownListView) with handle 0x7c45
```

Question 1: What are the advantages of RecyclerView over ListView ?

RecyclerView offers several advantages over ListView:

- **ViewHolder Pattern**: RecyclerView enforces the ViewHolder pattern, which enhances performance by reducing unnecessary calls to findViewById.
- **Flexibility**: It provides greater flexibility, allowing for various layouts such as grids and staggered grids in addition to the traditional list layout.
- **Animation Support**: Built-in support for animations facilitates easier implementation of dataset changes, such as adding or removing items.
- **Improved Performance**: RecyclerView handles large datasets more efficiently, especially when items are frequently added or removed, due to its optimized recycling mechanism.
- **Item Decoration**: Custom decorations, such as dividers, margins, or offsets, can be easily added to items.
- **Modular Design**: The modular design of RecyclerView allows customization of different parts of its behavior, including layout management and item animations, independently.

Question 2: Why are you unable to write to the file, and what would you need to do to be able to add any new data to the file ?

In this project, writing to files in the **res/raw** directory is not possible because these files are read-only and are bundled with the app's resources. To add new data to a file, the following approaches can be considered:

- **Internal Storage**: Write data to the device's internal storage using context.openFileOutput and context.openFileInput for writing and reading files, respectively.
- **External Storage**: For files that need to be accessible outside the app, use external storage, ensuring the necessary permissions (**READ_EXTERNAL_STORAGE** and **WRITE_EXTERNAL_STORAGE**) are requested and storage availability is checked.
- **Database**: For handling more complex data, a database like SQLite or Room can be used, allowing more advanced operations and efficient management of larger datasets.
- **Shared Preferences**: For small amounts of data, Shared Preferences can be used, though it is not suitable for large or complex datasets.

Question 3: When filtering the data, what changes did you make from a RecyclerView with unchanging data?

When filtering data in a RecyclerView, the following changes were implemented:

- **Dataset Modification**: Methods such as **filterByType** and **resetFilter** were added to the adapter to modify the dataset. **filterByType** narrows the dataset based on filter criteria, while **resetFilter** restores the original dataset.
- **Notify Adapter**: Methods like **notifyDataSetChanged**, **notifyItemInserted**, and **notifyItemRemoved** were used to inform the adapter of dataset changes, ensuring the displayed items are updated accordingly.
- **Efficient Filtering**: The filtering logic was optimized to maintain the RecyclerView's responsiveness, avoiding unnecessary dataset reloads and updating only relevant items.
- **View Binding**: The **onBindViewHolder** method was adjusted to bind data based on the current dataset state, ensuring accurate display of filtered data.