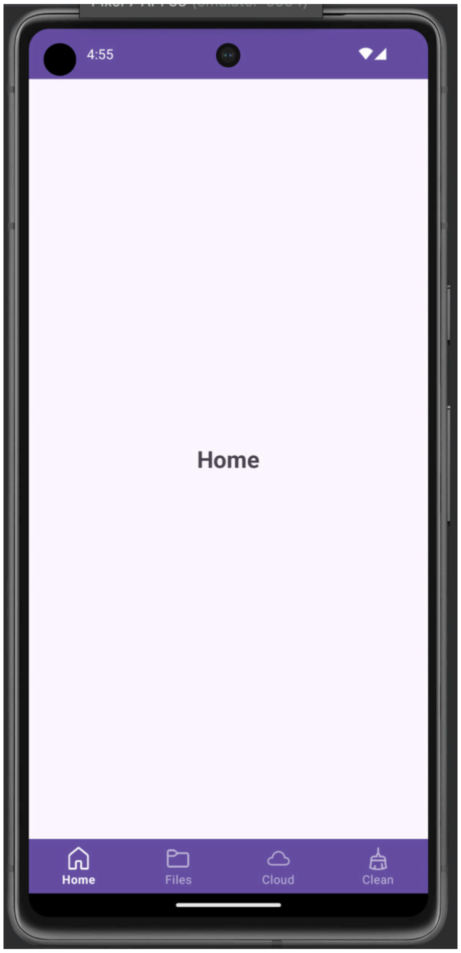
List of features needed for the MVP + Breakdown of tasks:

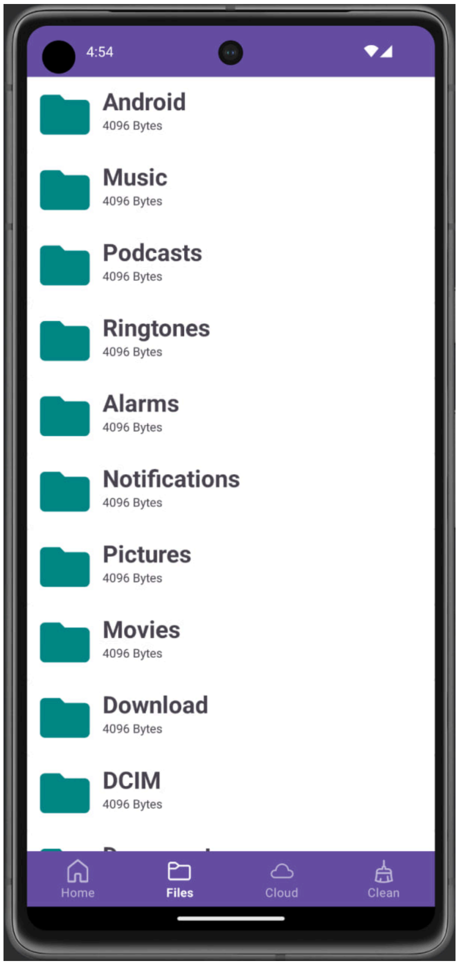
- Display all possible* files and directories on chosen storage
- Modify these files and directories: rename, delete, move etc.
- Create files and directories
- Allow to switch storages, e.g internal to external
- Display memory consumption chart
- Allow to connect at least one cloud storage, such as Google Drive, in ideal to all of them

Mapping between features and value(s) to be delivered by your app (justification):

- Primary value was open-source nature of the app, and all these features' implementations can be examined by anyone
- Another value is ability to connect to a remote cloud storage of your own, which is supported by this app

For the Checkpoint 1, I implemented data retrieving, Basic UI/UX and some aspects of navigation and event-based action
Screenshots from app:





Android
4096 Bytes

Music
4096 Bytes

Podcasts
4096 Bytes

Ringtones
4096 Bytes

Alarms
4096 Bytes

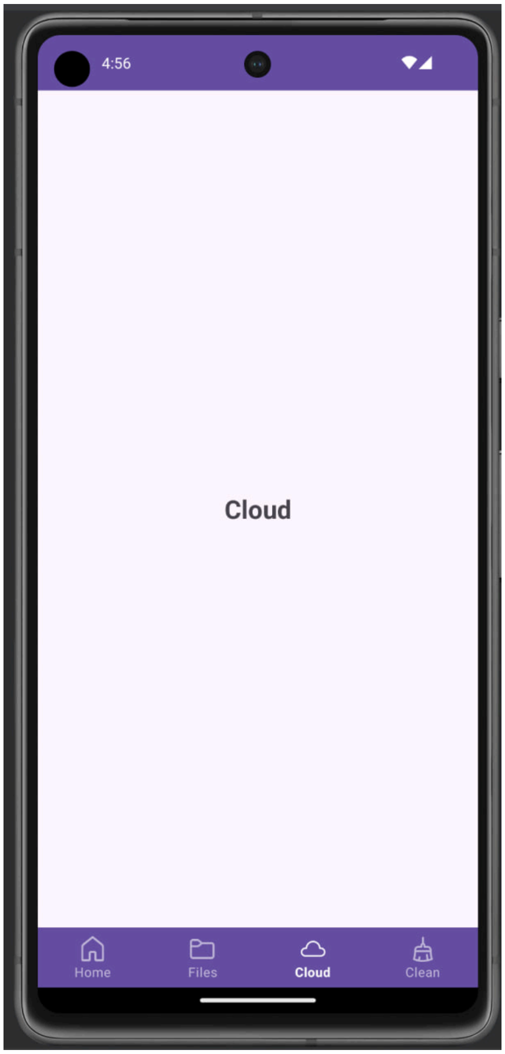
Notifications
4096 Bytes

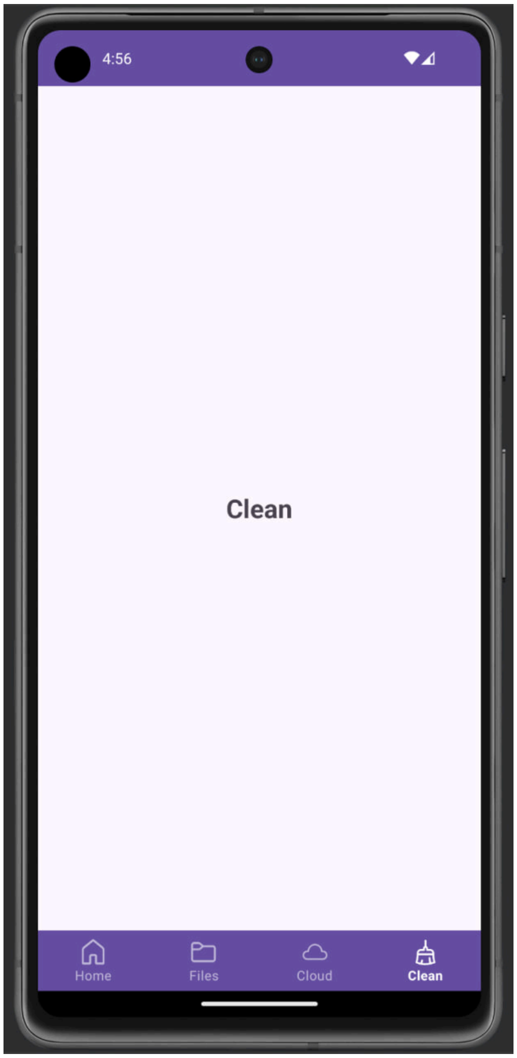
Pictures
4096 Bytes

Movies
4096 Bytes

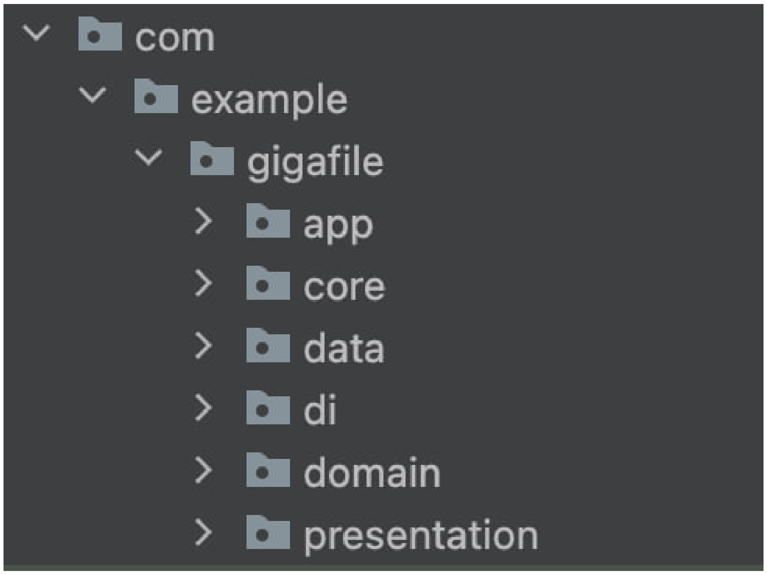
Download
4096 Bytes

DCIM
4096 Bytes

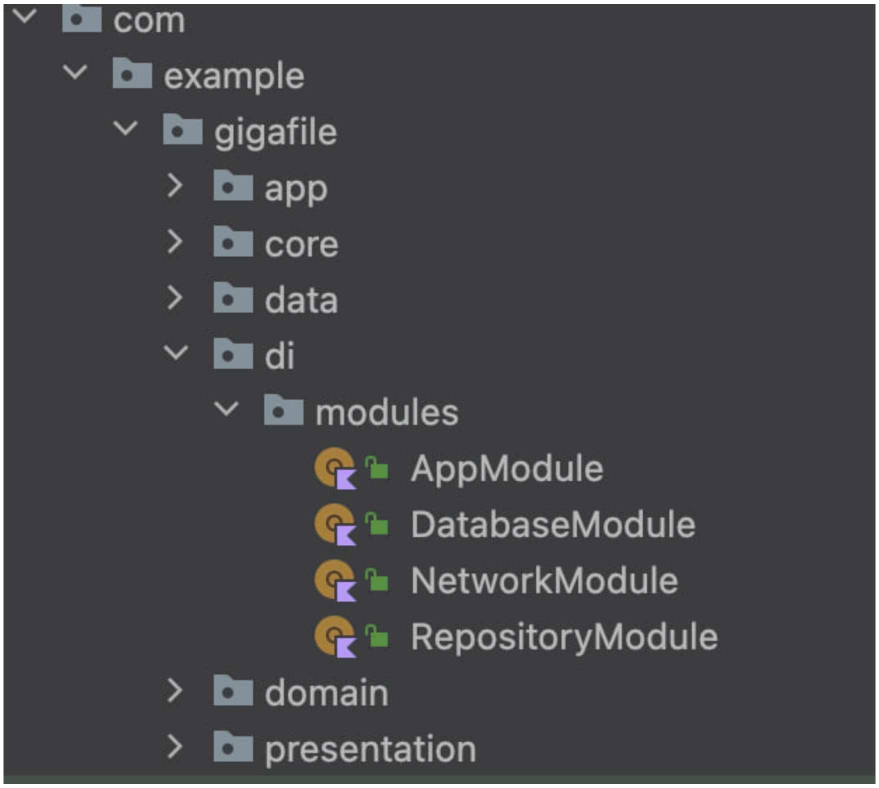




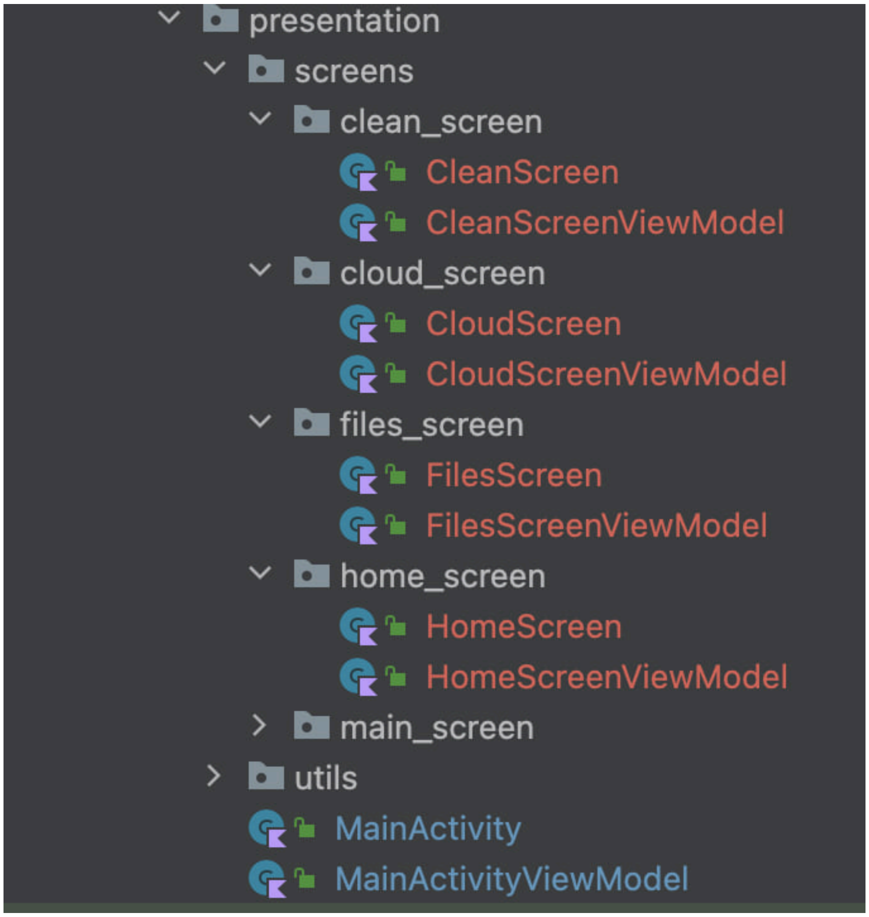
Structure of an app:



```
com
  example
    gigafile
      app
      core
      data
      di
      domain
      presentation
```

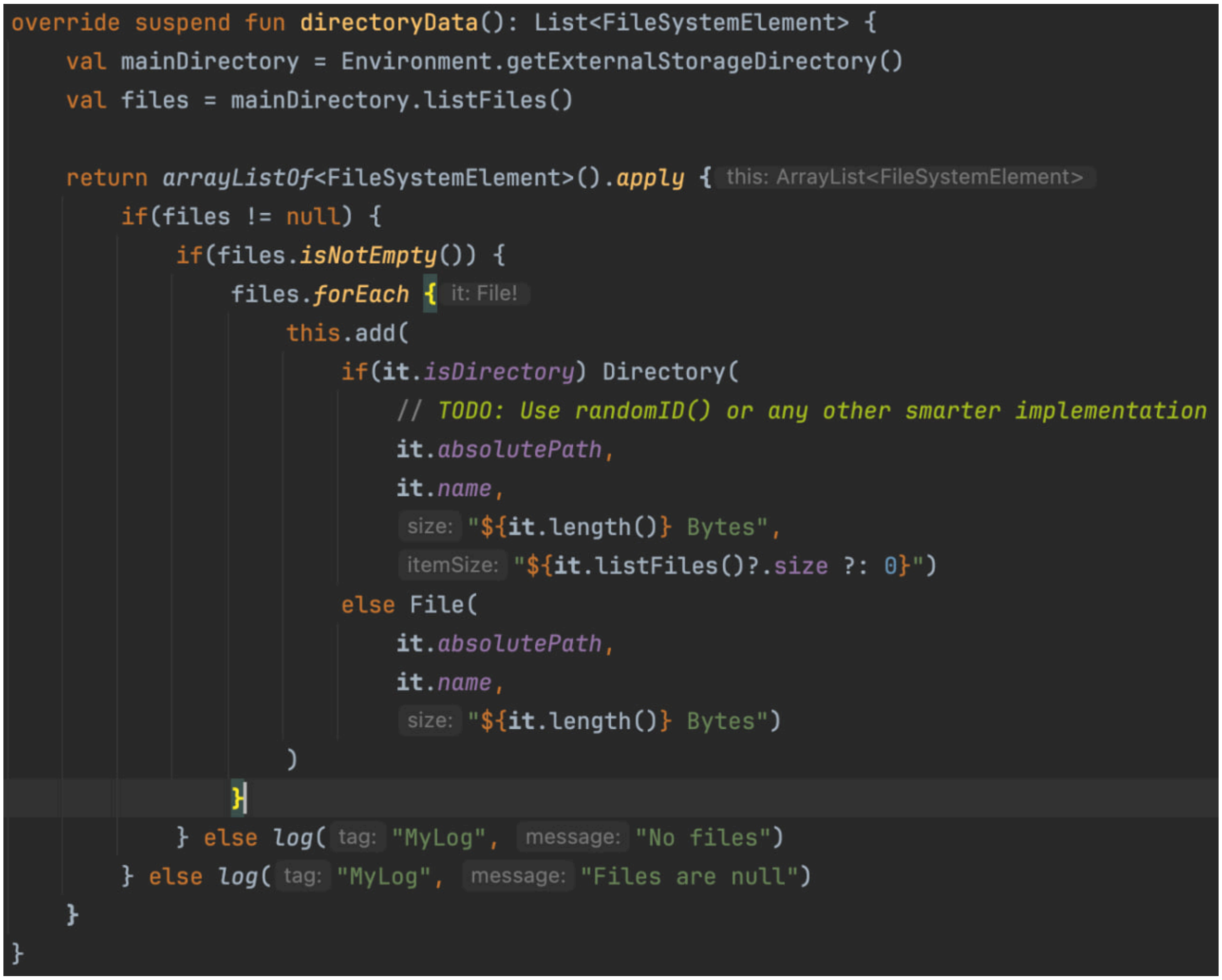


```
com
  example
    gigafile
      app
      core
      data
      di
        modules
          AppModule
          DatabaseModule
          NetworkModule
          RepositoryModule
      domain
      presentation
```



```
presentation
  screens
    clean_screen
      CleanScreen
      CleanScreenViewModel
    cloud_screen
      CloudScreen
      CloudScreenViewModel
    files_screen
      FilesScreen
      FilesScreenViewModel
    home_screen
      HomeScreen
      HomeScreenViewModel
    main_screen
  utils
  MainActivity
  MainActivityViewModel
```

Data retrieval from OS Environment:



```
override suspend fun directoryData(): List<FileSystemElement> {
    val mainDirectory = Environment.getExternalStorageDirectory()
    val files = mainDirectory.listFiles()

    return arrayListOf<FileSystemElement>().apply { this: ArrayList<FileSystemElement>
        if(files != null) {
            if(files.isNotEmpty()) {
                files.forEach { it: File!
                    this.add(
                        if(it.isDirectory) Directory(
                            // TODO: Use randomID() or any other smarter implementation
                            it.absolutePath,
                            it.name,
                            size: "${it.length()} Bytes",
                            itemSize: "${it.listFiles()?.size ?: 0}")
                        else File(
                            it.absolutePath,
                            it.name,
                            size: "${it.length()} Bytes")
                    )
                }
            } else log( tag: "MyLog",  message: "No files")
        } else log( tag: "MyLog",  message: "Files are null")
    }
}
```