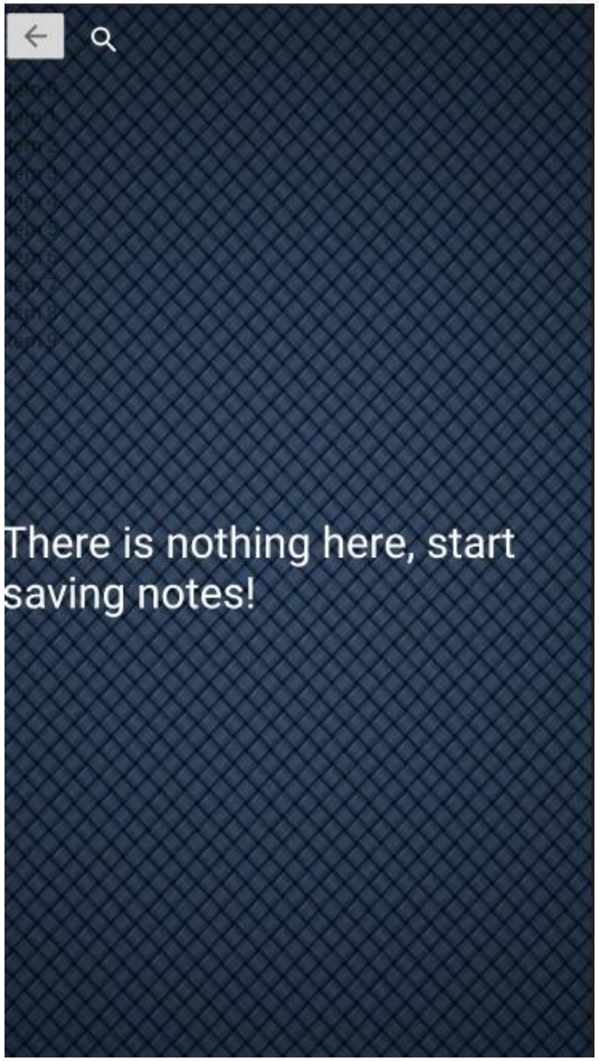
FragmentAllFiles.kt

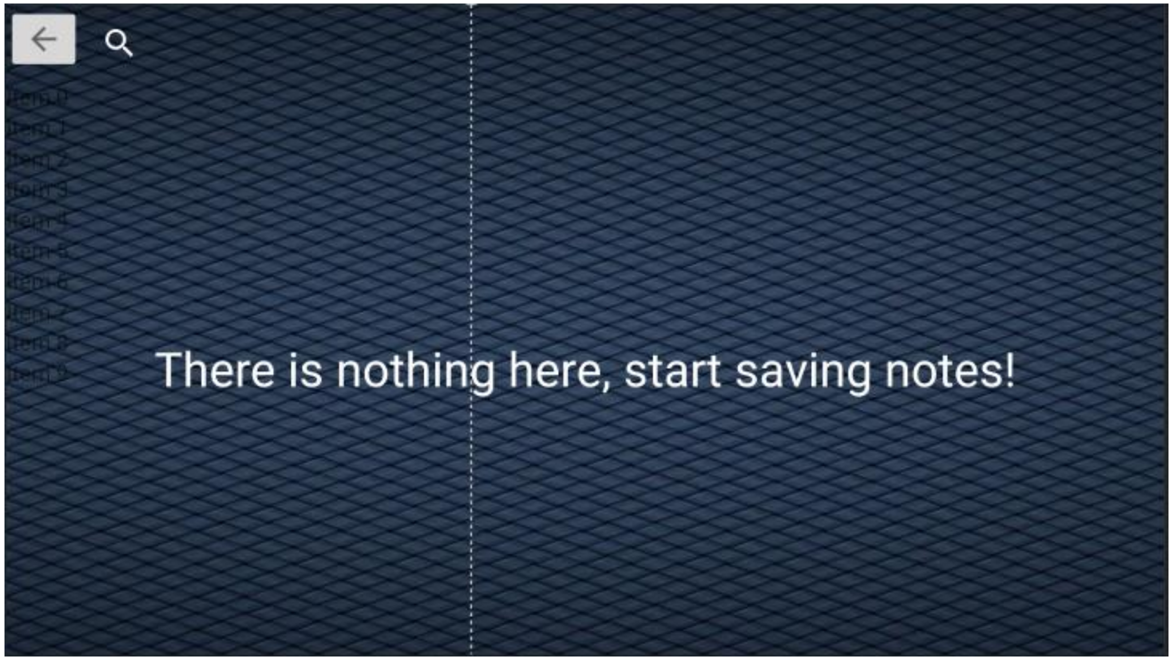
- All’interno di questo fragment vengono mostrate le directories e i files creati/salvati dall’utente
- Esistono due diverse modalità di visualizzazione

Fragment_all_files.xml

- La recycler view mostra le directories oppure i files all’interno della directory selezionata dall’utente.
- L’ ImageButton viene mostrato all’utente solamente se si stanno mostrando i files nella directory, altrimenti è invisibile.
- La textView è visibile solamente se non sono presenti directories oppure non sono presenti files all’interno della directory selezionata.


There is nothing here, start saving notes!

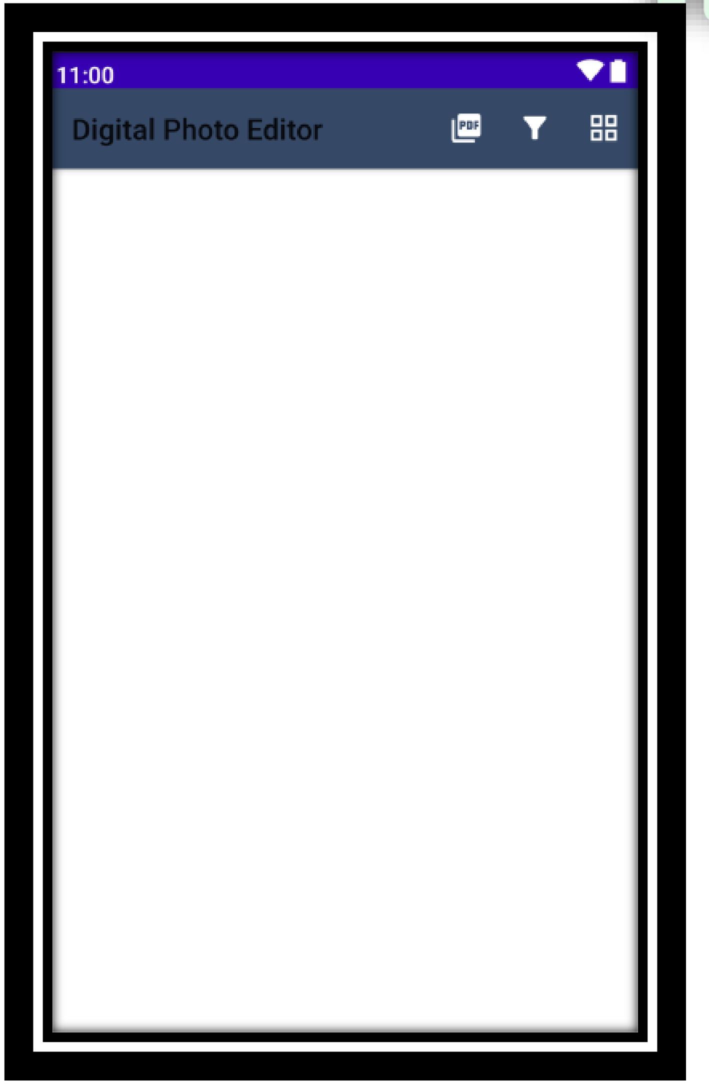
Fragment_all_files.xml (land)



There is nothing here, start saving notes!

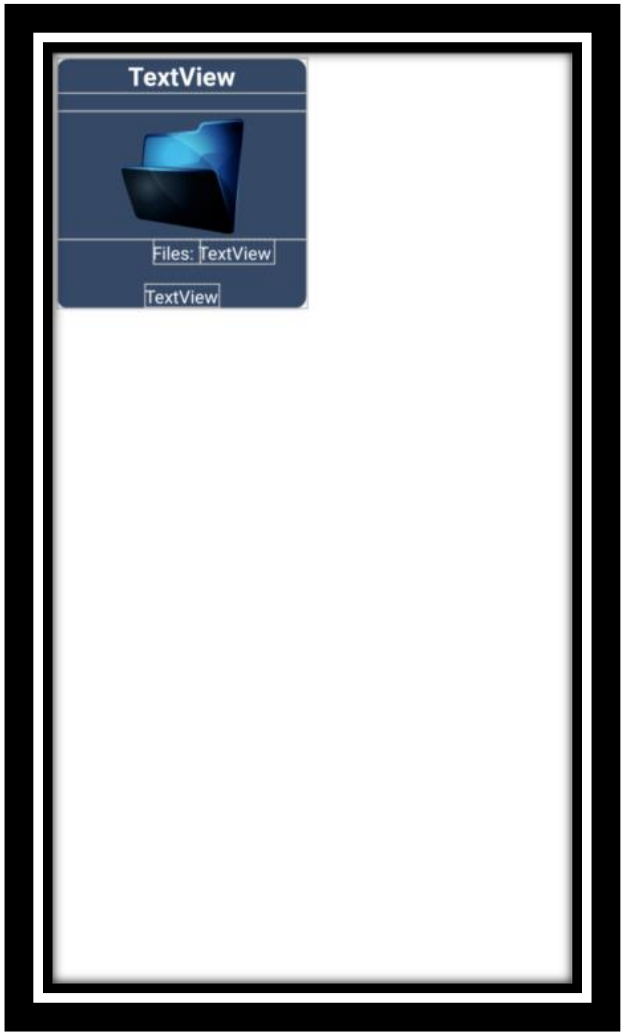
Menu_all_files.xml

- Il menu è strutturato in tre item



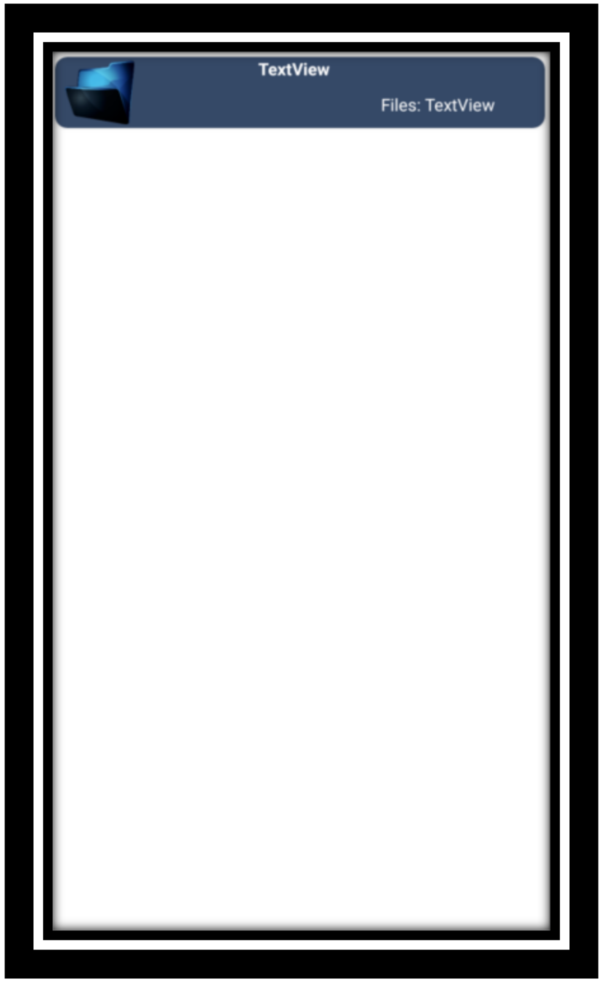
Item_directory_big.xml

- Il layout rappresenta il folder mostrato all'utente se viene scelta la modalità di visualizzazione DIR_BIG



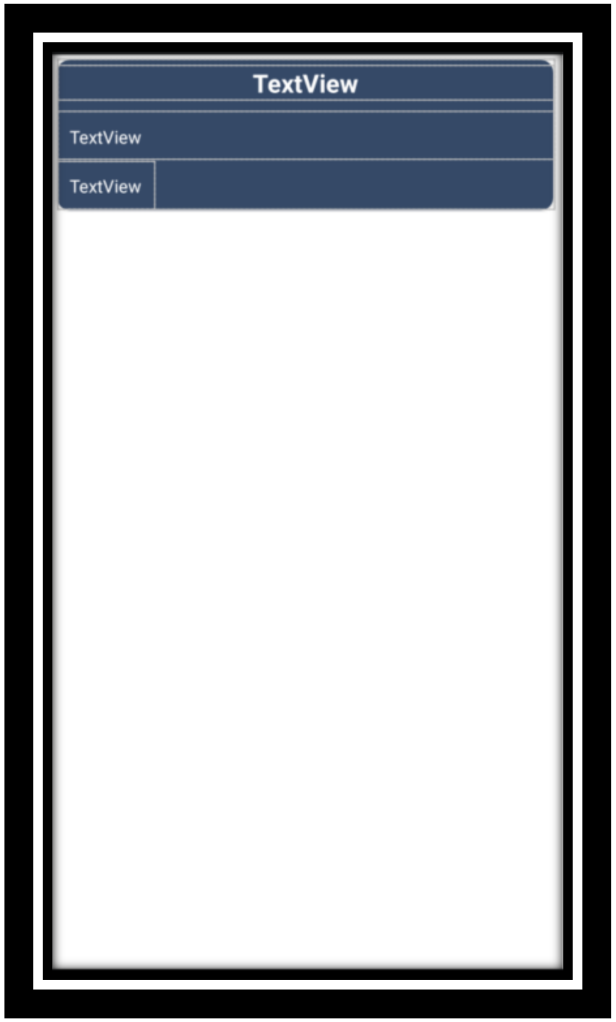
Item_directory_small.xml

- Il layout rappresenta il folder mostrato all’utente se viene scelta la modalità di visualizzazione DIR_SMALL



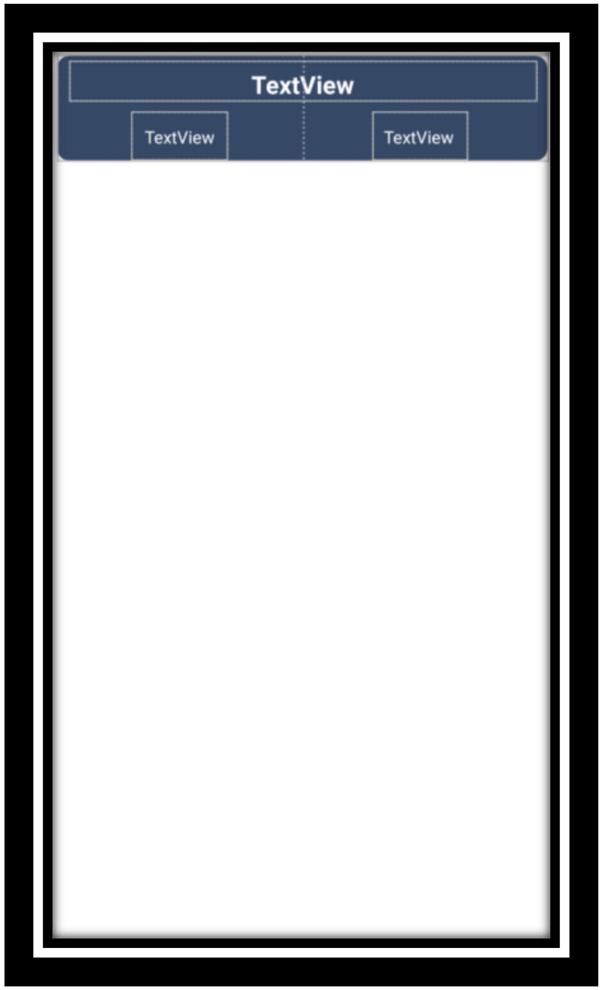
Item_file_big.xml

- Il layout rappresenta come vengono mostrati i files all'interno della directory se viene scelta la modalità di visualizzazione BIG



Item_file_small.xml

- Il layout rappresenta come vengono mostrati i files all'interno della directory se viene scelta la modalità di visualizzazione SMALL



Settaggio degli Observer

```
private fun setLiveData(){
    val directoryObserver = Observer<List<Directory>>(){
        setUI()
        setAdapterDirectory(it)
    }
    val listObserver = Observer<List<Note>>(){
        setUI()
        setAdapterFiles(it)
    }
    viewModel.listDirectory.observe(requireActivity(),directoryObserver)
    viewModel.listNotes.observe(requireActivity(),listObserver)
}
```

Adapters

- Sono stati implementati quattro diversi adapter: due adapter per quanto riguarda i files e due adapter per le directories
- Il motivo di ciò è perché esistono quattro modalità di visualizzazione, due per i files e due per le directories

setAdapterDirectory ()

```kotlin
private fun setAdapterDirectory(listDirectory: List<Directory>){
    binding.rvFiles.invalidate()
    Log.d("DIR_SIZE",listDirectory.size.toString())
    if(showMode == FilesShowMode.DIR_BIG){
        adapterDirectoryBig = DirectoryBigAdapter(listDirectory, requireContext(),this,spanCount)
        binding.rvFiles.adapter = adapterDirectoryBig
    }
    else if (showMode == FilesShowMode.DIR_SMALL){
        adapterDirectorySmall = DirectorySmallAdapter(listDirectory,requireContext(),this)
        binding.rvFiles.adapter = adapterDirectorySmall
    }
}
```

setAdapterFiles ()

```
private fun setAdapterFiles(listFiles: List<Note>){
   if(showMode == FilesShowMode.BIG){
      adapter = AllFilesBigAdapter(listFiles,requireContext(),this)
      binding.rvFiles.adapter=adapter
   }
   else{
      adapterSmall = AllFilesSmallAdapter(listFiles,requireContext(),this,this)
      binding.rvFiles.adapter=adapterSmall
   }
}
```

setUI()

```
val note = Note()
note.id = -1
viewModel.selectedNote.value = note
binding.rvFiles.invalidate()
binding.rvFiles.invalidateItemDecorations()
binding.rvFiles.adapter = null
binding.rvFiles.layoutManager = null
val layoutManager: RecyclerView.LayoutManager
for(i in 0 until binding.rvFiles.itemDecorationCount){
    binding.rvFiles.removeItemDecorationAt(i)
}
```

setUI() (2)

```
val orientation = requireContext().resources.configuration.orientation
if(orientation == Configuration.ORIENTATION_LANDSCAPE && showMode == FilesShowMode.SMALL){
    binding.previewLayout!!.isVisible = true
    binding.guidelineAllFiles!!.setGuidelinePercent(0.4f)
   childFragmentManager.beginTransaction()
        .replace(binding.previewLayout!!.id, FragmentNoteDetails(), DETAILS_FRAGMENT_TAG )
        .commit()
}
else{
    if(orientation == Configuration.ORIENTATION_LANDSCAPE){
        binding.previewLayout!!.isVisible = false
        binding.guidelineAllFiles!!.setGuidelinePercent(1f)
    }
}
```

setUI() (3)

```
when(showMode) {
   FilesShowMode.BIG -> {
      spanCount = if(orientation == Configuration.ORIENTATION_LANDSCAPE)
         4
      else
         2
      layoutManager =
         StaggeredGridLayoutManager(spanCount, StaggeredGridLayoutManager.VERTICAL)
      (layoutManager as StaggeredGridLayoutManager).gapStrategy=StaggeredGridLayoutManager.GAP_HANDLING_NONE
   }
   FilesShowMode.DIR_BIG -> {
      spanCount =  if(orientation == Configuration.ORIENTATION_LANDSCAPE)
         5
      else
         3
      layoutManager =
         StaggeredGridLayoutManager(spanCount, StaggeredGridLayoutManager.VERTICAL)
      (layoutManager as StaggeredGridLayoutManager).gapStrategy=StaggeredGridLayoutManager.GAP_HANDLING_NONE
   }
   FilesShowMode.DIR_SMALL -> {
      spanCount = 1
      layoutManager = LinearLayoutManager(requireContext())
   }
   FilesShowMode.SMALL -> {
      spanCount = 1
      layoutManager = LinearLayoutManager(requireContext())
   }
}
```

setUI() (4)

```
      binding.rvFiles.layoutManager = layoutManager
      if(showMode==FilesShowMode.DIR_BIG || showMode == FilesShowMode.BIG) {
        if (binding.rvFiles.itemDecorationCount == 0) {
          binding.rvFiles.addItemDecoration(
            GridSpacingDecorator(
              resources.getDimensionPixelSize(R.dimen.cardView_margin),
              spanCount
            )
          )
        }
      }
       else{
          if(binding.rvFiles.itemDecorationCount == 0){
                binding.rvFiles.addItemDecoration(
LinearSpacingDecorator(resources.getDimensionPixelSize(R.dimen.cardView_margin),
      spanCount))
          }
      }
```

setUI() (5)

```
    if(showMode == FilesShowMode.DIR_BIG || showMode == FilesShowMode.DIR_SMALL){
        binding.imageButton.visibility=View.GONE
    }
    else{
        if(viewModel.listNotes.value != null)
        setAdapterFiles(viewModel.listNotes.value!!)
        binding.imageButton.visibility=View.VISIBLE
    }
    binding.imageButton.setOnClickListener{
        showMode = if(showMode == FilesShowMode.BIG)
            FilesShowMode.DIR_BIG
        else
            FilesShowMode.DIR_SMALL
        mActionMode = null
        selectedItem = ArrayList()
        setUI()
        loadData()
    }
    binding.sv.setOnQueryTextListener(MyQueryListener())

}
```

loadData()

```
private fun loadData(){
    Log.d("RESUME","RESUME")
    val dao = DbDigitalPhotoEditor.getInstance(requireContext()).digitalPhotoEditorDAO()
    CoroutineScope(Dispatchers.IO).launch{
        val results = dao.loadDirectories()
        val listDirectory = mutableListOf<Directory>()
        var size = 0
        var lastModify: Long
        for(res in results){
            size = dao.loadDirectorySize(res)
            lastModify = dao.getLastModifyDir(res)
            listDirectory.add(Directory(res,size, Date(lastModify)))
        }
        CoroutineScope(Dispatchers.Main).launch {
            viewModel.listDirectory.value = listDirectory
            binding.textView7.isVisible = listDirectory.isEmpty()
        }
    }
}
```

Metodi chiamati negli adapter

- All’interno delle classi adapter vengono chiamati i seguenti metodi della classe FragmentAllFiles:

selectedHandler

```kotlin
override fun selectedHandler(element: Any): Boolean {
   var isFirst = false
   if(selectedItem.isEmpty()){
      mActionMode = requireActivity().startActionMode(AllFilesActionModeCallback())
      isFirst = true
   }
   if(selectedItem.contains(element)) {
      selectedItem.remove(element)
      if(selectedItem.size==0)
         mActionMode!!.finish()
      return false
   }
   selectedItem.add(element)
   if(isFirst) {
      val note = Note()
      note.id = -1
      viewModel.selectedNote.value = note
      //Invoke reset only if the fragment exists
      (if (childFragmentManager.findFragmentByTag(DETAILS_FRAGMENT_TAG) == null) null
      else
         (childFragmentManager.findFragmentByTag(DETAILS_FRAGMENT_TAG) as FragmentNoteDetails))?.reset()
   }
   return true
}
```

```
fun isSelected(note: Note): Boolean{
    for(any in selectedItem){
        if(any is Note){
            if(any.id == note.id)
                return true
        }
    }
    return false
}
override fun isAnItemSelected(): Boolean {
    return selectedItem.isNotEmpty()
}
```

changeToFiles

```kotlin
fun changeToFiles(directory: String){
    CoroutineScope(Dispatchers.IO).launch{
        val files = DbDigitalPhotoEditor.getInstance(requireContext()).digitalPhotoEditorDAO().loadAllByDirectory(directory)
        CoroutineScope(Dispatchers.Main).launch{
            if(showMode == FilesShowMode.DIR_BIG)
                showMode = FilesShowMode.BIG
            else if(showMode == FilesShowMode.DIR_SMALL)
                showMode = FilesShowMode.SMALL
            setUI()
            actualDirectory = directory
            viewModel.listNotes.value=files.toMutableList()
            viewModel.sort(sortingType)
            selectedItem = ArrayList()
            mActionMode = null
        }
    }
}
```

Menu Contestuale

- Tenendo premuto su una directory o sul un file è possibile abilitare la selezione multipla degli elementi per poi dare all’utente la possibilità di eseguire determinate azioni

```
private inner class AllFilesActionModeCallback() : ActionMode.Callback{
    override fun onCreateActionMode(mode: ActionMode?, menu: Menu?): Boolean {
        if(showMode==FilesShowMode.DIR_BIG || showMode == FilesShowMode.DIR_SMALL)
            mode!!.menuInflater.inflate(R.menu.menu_onlong_digital,menu)
        else
            mode!!.menuInflater.inflate(R.menu.menu_onlong_files,menu)
        return true
    }

    override fun onPrepareActionMode(mode: ActionMode?, menu: Menu?): Boolean {
        return false
    }

    override fun onActionItemClicked(mode: ActionMode?, item: MenuItem?): Boolean {
        if(showMode==FilesShowMode.BIG || showMode == FilesShowMode.SMALL){
            //Files case
            when(item!!.itemId){
                R.id.it_files_delete -> deleteFiles(mode)
                R.id.it_files_merge -> mergeFiles(mode)
                else -> deleteFiles(mode)
            }
        }
        else{
            //Directory case
            deleteDirectories(mode)
        }
        return true
    }

    override fun onDestroyActionMode(mode: ActionMode?) {
        selectedItem = ArrayList()
        mActionMode=null
    }
```

```
private fun deleteDirectories(mode: ActionMode?){
   val alertDialog = AlertDialog.Builder(requireContext())
   alertDialog.setTitle(R.string.delete_directories_alert_title)
   alertDialog.setMessage(R.string.delete_directories_alert_message)
   alertDialog.setPositiveButton(R.string.yes){ dialogInterface: DialogInterface, i: Int ->
      CoroutineScope(Dispatchers.IO).launch{
         val dao = DbDigitalPhotoEditor.getInstance(requireContext()).digitalPhotoEditorDAO()
         val listDir = ArrayList<Directory>()
         for(elem in selectedItem){
            if(elem is Directory){
               dao.deleteDirectory(elem.name)
               listDir.add(elem)
            }
         }
         CoroutineScope(Dispatchers.Main).launch{
            val newList = viewModel.listDirectory.value!!
            newList.removeAll(listDir)
            viewModel.listDirectory.value = newList
            dialogInterface.dismiss()
            mode!!.finish()
         }
      }
   }
   alertDialog.setNegativeButton(R.string.no){ dialogInterface: DialogInterface, _: Int ->
      dialogInterface.dismiss()
      mode!!.finish()
   }
   alertDialog.show()
}
```

```kotlin
private fun deleteFiles(mode: ActionMode?){
   val alertDialog = AlertDialog.Builder(requireContext())
   alertDialog.setTitle(R.string.delete_files_alert_title)
   alertDialog.setTitle(R.string.delete_files_alert_message)
   alertDialog.setPositiveButton(R.string.yes){ dialogInterface: DialogInterface, _: Int ->
      val dao = DbDigitalPhotoEditor.getInstance(requireContext()).digitalPhotoEditorDAO()
      CoroutineScope(Dispatchers.IO).launch{
         val listNote = ArrayList<Note>()
         for(elem in selectedItem){
            if(elem is Note)
               listNote.add(elem)
         }
         dao.deleteAll(listNote)
         CoroutineScope(Dispatchers.Main).launch{
            val list = viewModel.listNotes.value!!
            list.removeAll(listNote)
            viewModel.listNotes.value = list
            dialogInterface.dismiss()
            mode!!.finish()
         }
      }
   }
   alertDialog.setNegativeButton(R.string.no){ dialogInterface: DialogInterface, _: Int ->
      dialogInterface.dismiss()
      mode!!.finish()
   }
   alertDialog.show()
}
```

```kotlin
private fun mergeFiles(mode: ActionMode?){
    val stringBuilder = StringBuilder()
    for(elem in selectedItem){
        if(elem is Note){
            stringBuilder.append(elem.text)
            stringBuilder.append("\n")
        }
    }
    val note = Note(stringBuilder.toString(),"","","und",System.currentTimeMillis(),false)
    val intent = Intent(requireContext(),TextResultActivity::class.java)
    intent.putExtra("result",note)
    intent.putExtra("type", TextResultType.NOT_SAVED.ordinal)
    requireContext().startActivity(intent)
    mode!!.finish()
}
```

Cambiare la modalità visualizzazione dei dati (1)

```
R.id.it_preview -> {
   selectedItem = ArrayList()
   mActionMode = null
   if (showMode == FilesShowMode.DIR_BIG || showMode == FilesShowMode.BIG) {
      Log.d("HERE", "HERE")
      item.icon = ContextCompat.getDrawable(
         requireContext(),
         R.drawable.ic_baseline_grid_view_24
      )
      if (showMode == FilesShowMode.DIR_BIG) {
         showMode = FilesShowMode.DIR_SMALL
         setUI()
         setAdapterDirectory(viewModel.listDirectory.value!!)
      } else {
         showMode = FilesShowMode.SMALL
         setUI()
         setAdapterFiles(viewModel.listNotes.value!!)
      }
}
```

Cambiare la modalità visualizzazione dei dati (2)

```
    } else {
        item.icon =
            ContextCompat.getDrawable(
                requireContext(),
                R.drawable.ic_baseline_format_list_bulleted_24
            )
        if (showMode == FilesShowMode.DIR_SMALL) {
            showMode = FilesShowMode.DIR_BIG
            setUI()
            setAdapterDirectory(viewModel.listDirectory.value!!)
        } else {
            showMode = FilesShowMode.BIG
            setUI()
            setAdapterFiles(viewModel.listNotes.value!!)
        }
    }
}
```

```
R.id.it_filter -> {
    val sharedPreferences = requireActivity().getSharedPreferences("filter_all_files",
        Activity.MODE_PRIVATE)
    var filterMode = sharedPreferences.getInt("filterMode",FilterMode.BY_TEXT.ordinal)
    var sortingType = sharedPreferences.getInt("sortingType",SortingType.ALPHABETIC_ASC.ordinal)
    val dialog = FilterDialog.getInstance(filterMode,sortingType)
    dialog.setOnFilterSelected{ filter: FilterMode, sorting: SortingType ->
        selectedItem = ArrayList()
        mActionMode = null
        filterMode = filter.ordinal
        sortingType = sorting.ordinal
        this.filterMode = filter
        this.sortingType = sorting
        if(showMode == FilesShowMode.DIR_BIG || showMode == FilesShowMode.DIR_SMALL) {
            if(filter == FilterMode.BY_COUNTRY){
                Toast.makeText(requireContext(),requireContext().
                                        getString(R.string.no_country_directory),Toast.LENGTH_LONG).show()
            }
            viewModel.filterDirectory(binding.sv.query.toString())
            viewModel.sortDirectories(sorting)
        }
        else {
            viewModel.filter(binding.sv.query.toString(), filter,actualDirectory)
            viewModel.sort(sorting)
        }
        val editor = sharedPreferences.edit()
        editor.putInt("filterMode",filterMode)
        editor.putInt("sortingType",sortingType)
        editor.apply()
    }
    dialog.show(parentFragmentManager,"FILTERDIALOG")
}
```

PDF Files

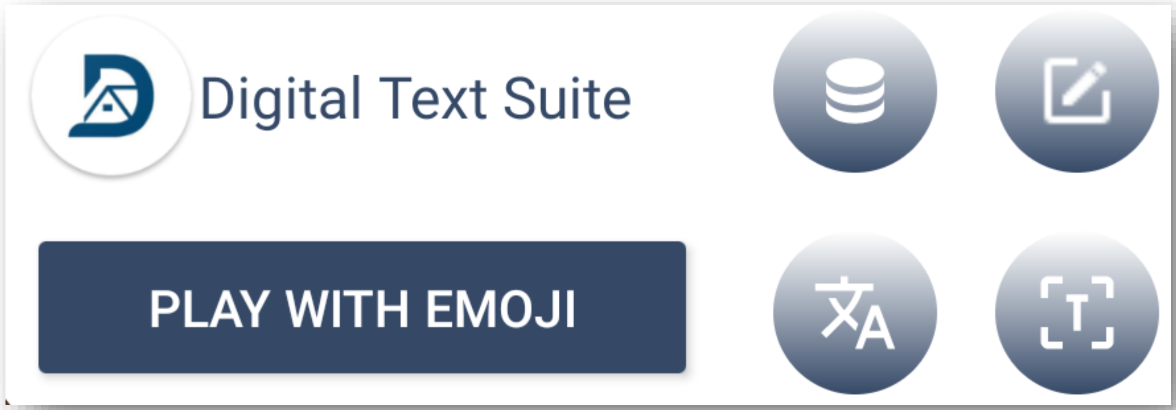
```
R.id.it_pdf -> {
    CoroutineScope(Dispatchers.IO).launch {
        val rootDir = RealMainActivity.rootDir
        val listFiles = getPdfFilesFromRootDir(rootDir)
        CoroutineScope(Dispatchers.Main).launch {
            val dialog = SelectPdfDialog.getInstance(listFiles)
            dialog.show(parentFragmentManager, "SELECTPDFDIALOG")
        }
    }

}
```

PDF Files (1)

```kotlin
private fun getPdfFilesFromRootDir(rootDir: File): List<File> {
    return rootDir.walk().filter{
      it.extension == "pdf"
    }.toList()
}
```

Widget



Widget (2)

- MyAppWidget.kt
- xml/my_app_widget_info.xml
- Layout/layout_my_app_widget.xml

Widget (3): RealMainActivity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityRealMainBinding.inflate(layoutInflater)
    widget_metadata = intent.getIntExtra("fragment",-1)
    setContentView(binding.root)
    if(allPermissionsGranted()){
        init()
    }else{
        ActivityCompat.requestPermissions(this, REQUIRED_PERMISSIONS, REQUEST_CODE_PERMISSIONS)
    }
```

Widget (4): RealMainActivity

```
if(widget_metadata != -1) {
    binding.viewPagerMain.currentItem = widget_metadata
}
```