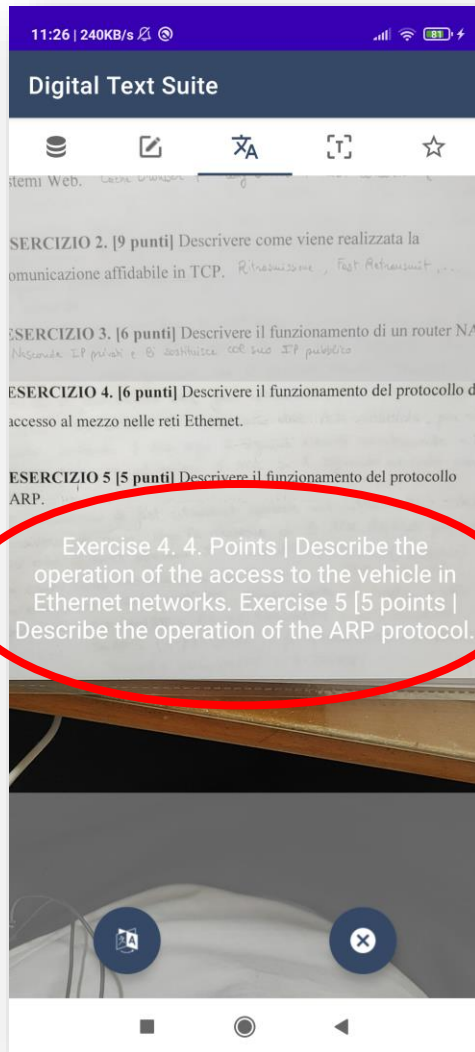




Traduzione

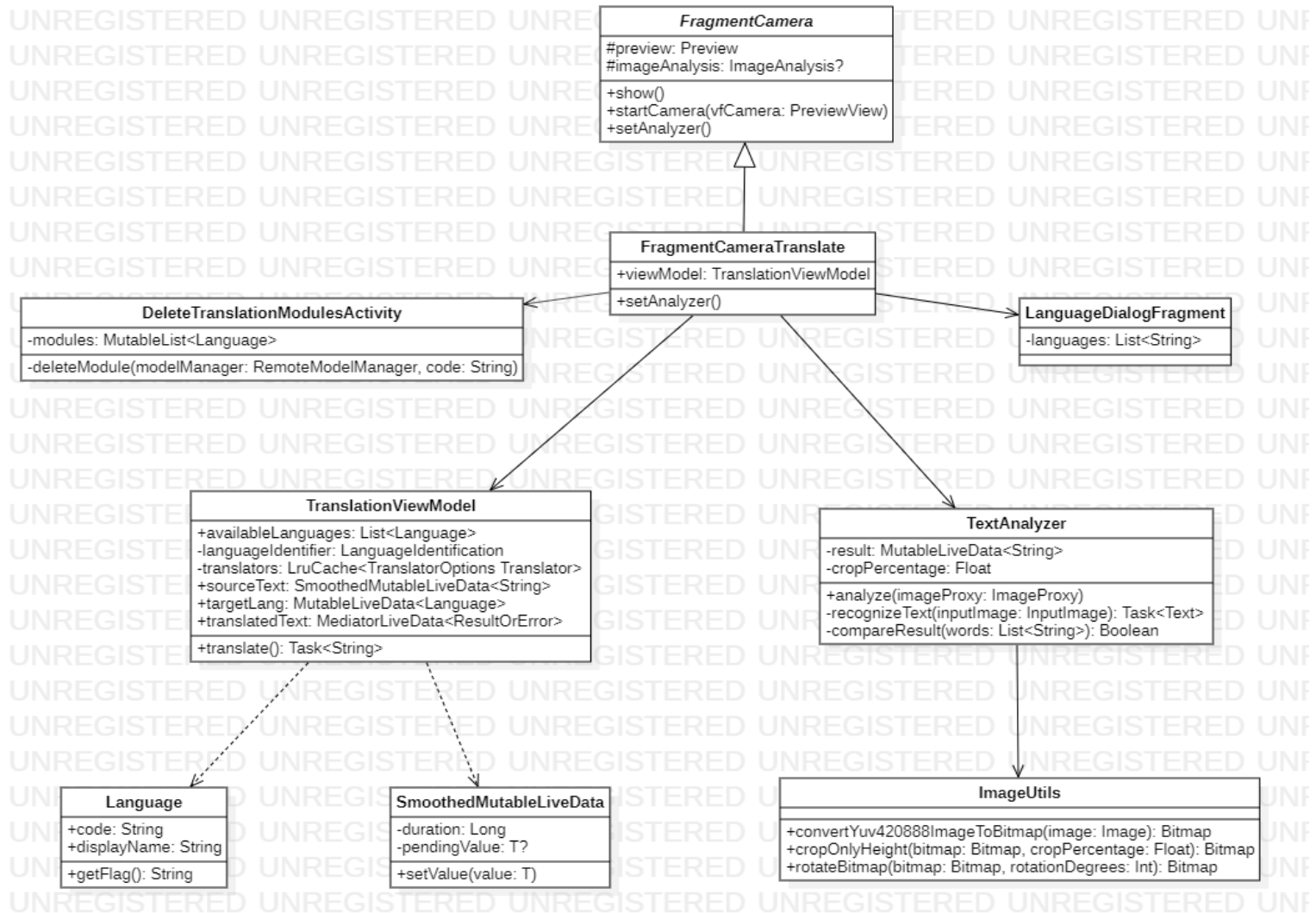
- Tra le features offerte da «Digital Text Suite», c'è la traduzione di testo
- Questa funzionalità viene offerta in due diverse modalità:
 - Traduzione Real-Time
 - Traduzione statica

Traduzione Real-Time



- È disponibile nel tab centrale del tab layout principale dell'app
- Utilizza la fotocamera per prendere in input stream di immagini e analizzarle
- Viene riconosciuto il testo presente nelle immagini e la lingua in cui è scritto
- È possibile selezionare una lingua target in cui tradurre il testo riconosciuto

Struttura



FragmentCameraTranslate



- Questa classe è un Fragment che estende la classe astratta FragmentCamera, la quale mette a disposizione tutti i metodi necessari per il corretto funzionamento della fotocamera
- Sono state utilizzate le API della libreria di Android CameraX
- Si fa override del metodo `setAnalyzer()` per settare un TextAnalyzer come analizzatore delle immagini

FragmentCameraTranslate



```
override fun setAnalyzer() {
    try{
        val analyzer = TextAnalyzer(requireContext(), lifecycle,
viewModel.sourceText, CROP_PERCENTAGE)
        if (imageAnalysis != null) {
            imageAnalysis.also { it!!.setAnalyzer(cameraExecutor, analyzer) }
        }else{
            imageAnalysis = ImageAnalysis.Builder()
                .setTargetResolution(Size(1280, 720))
                .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                .build()
                .also { it.setAnalyzer(cameraExecutor, analyzer) }
        }
    }catch(exception: IllegalStateException){
    }
}
```

TextAnalyzer



- Implementa l'interfaccia `ImageAnalysis.Analyzer`
- Utilizza delle API di `MLKit` per ottenere un riconoscitore di testo, con cui processare le immagini
- Dell'immagine in input, viene analizzata solo una parte; infatti viene prima ruotata nella giusta direzione e poi ne viene fatto un crop, tagliando una percentuale della parte superiore e inferiore. Per queste operazioni, si utilizza la classe `ImageUtils`

TextAnalyzer



```
override fun analyze(imageProxy: ImageProxy) {  
    val mediaImage = imageProxy.image ?: return  
    val rotationDegrees = imageProxy.imageInfo.rotationDegrees  
  
    val bitmap = ImageUtils.convertYuv420888ImageToBitmap(mediaImage)  
    val rotatedBitmap = ImageUtils.rotateBitmap(bitmap, rotationDegrees)  
    val croppedBitmap = ImageUtils.cropOnlyHeight(rotatedBitmap, cropPercentage)  
  
    recognizeText(InputImage.fromBitmap(croppedBitmap, 0))  
        .addOnCompleteListener {  
            // close the imageProxy to allow following images to be analyzed  
            imageProxy.close()  
        }  
}
```

TextAnalyzer



- Per evitare continui cambiamenti nel testo analizzato, si è cercato di stabilire una threshold dinamica per il cambiamento del testo riconosciuto
- La threshold è calcolata in base alla differenza tra il testo precedentemente riconosciuto e quello attuale
- Tale differenza tiene conto sia del numero di parole che della differenza tra le parole stesse
- Implementata col metodo `compareResult()`

TextAnalyzer



```
private fun compareResult(words: List<String>): Boolean {
    if (result.value == null) {
        return true
    }
    val actualWords = result.value!!.split(" ")
    val size = Math.min(words.size, actualWords.size)
    val maxSize = Math.max(words.size, actualWords.size)
    // reference value of number of words
    val ref = (size * DIFFERENCE_PERCENTAGE)
    val refInt: Int
    if (ref - ref.toInt() >= 0.5f)
        refInt = ref.toInt() + 1
    else
        refInt = ref.toInt()
    var count = 0
    // checking differences among words
    for (i in 0 until size) {
        if (!words[i].equals(actualWords[i]))
            count++
        // if enough different ...
        if (count >= refInt)
            return true
    }
    // else if size is enough different ...
    if (maxSize - size + count >= refInt)
        return true
    return false
}
```



TranslationViewModel

- È la classe in cui si ha il riferimento a tutte le componenti necessarie per la traduzione:
 - Testo sorgente
 - Lingua sorgente
 - Lingua destinazione
- La lingua sorgente viene ottenuta utilizzando un `LanguageIdentifier`, presente nelle librerie di `MLKit`, che analizza il testo sorgente



TranslationViewModel

- Per effettuare la traduzione, vengono utilizzati degli oggetti di tipo `Translator` di `MLKit`
- Sul translator settato correttamente con lingua d'origine e di destinazione, viene invocato il metodo `downloadModelIfNeeded()`
- Tale metodo provvede a scaricare dalla rete i modelli di traduzione della lingua indicata, qualora non siano presenti sul dispositivo



TranslationViewModel

```
val translatorOptions = TranslatorOptions.Builder()
    .setSourceLanguage(sourceLangCode)
    .setTargetLanguage(targetLangCode)
    .build()

// get the translator
val translator = translators[translatorOptions]

modelDownloadTask = translator.downloadModelIfNeeded().addOnCompleteListener {
    modelDownloading.setValue(false)
}

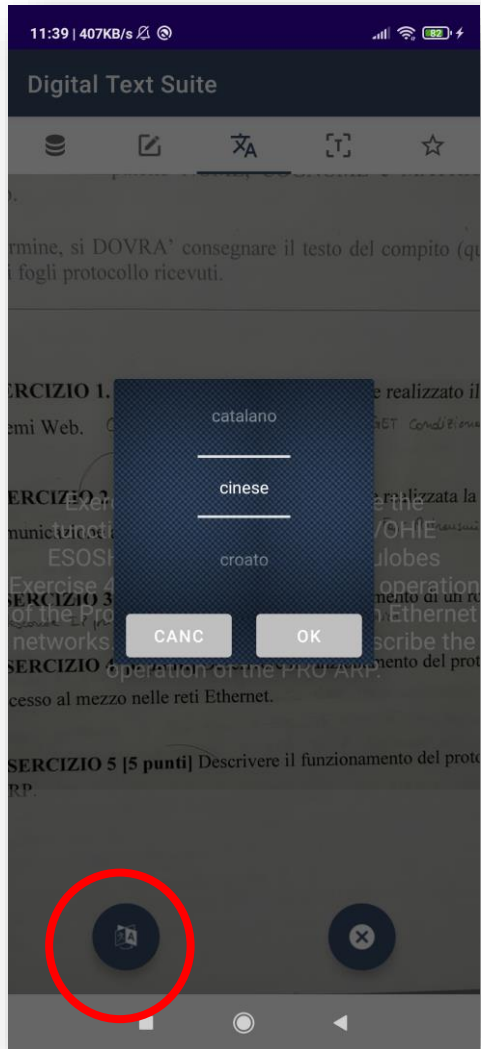
return modelDownloadTask.onSuccessTask {
    // translate the source text
    translator.translate(text)
}.addOnCompleteListener {
    translating.value = false
}
```



TranslationViewModel

- Per quanto riguarda il testo sorgente, il testo tradotto, le lingue sorgente e target, sono tutti dei `MutableLiveData`
- Tuttavia, sono state usate due varianti particolari di tale tipo di dato, di cui una custom:
 - `SmoothedMutableLiveData` (custom)
 - `MediatorLiveData`, per avere aggiornamenti quando cambia un singolo componente di un gruppo di `MutableLiveData`

Scelta della lingua



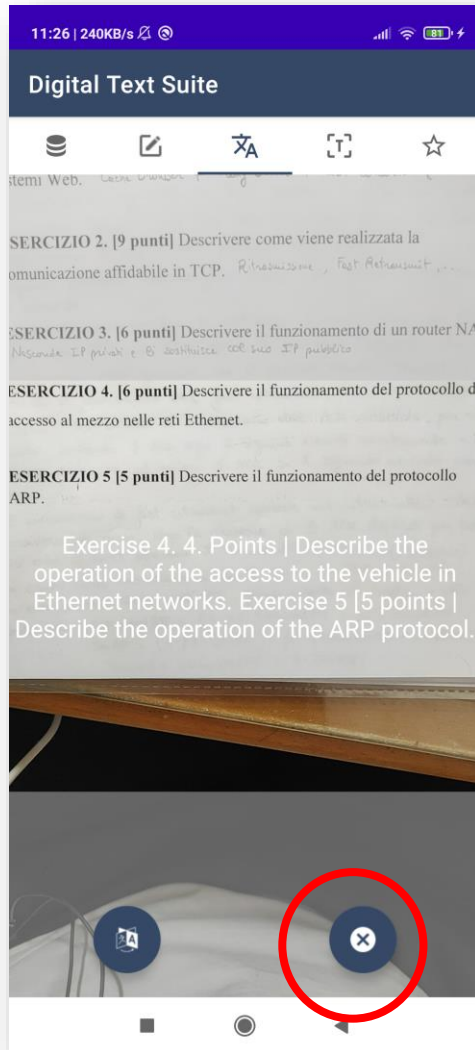
- La lingua di destinazione è selezionabile tramite il bottone in basso a sinistra nel layout della traduzione real-time
- Apparirà il dialog in figura, che permetterà di scegliere tra oltre 50 idiomi disponibili



Scelta della lingua

- Il dialog è stato realizzato estendendo la classe `DialogFragment`
- Presenta un `Number Picker` per la scelta della lingua cui è stato assegnato uno stile personalizzato
- In generale, per la gestione delle lingue, è stato creato un mapping tra gli idiomi riconosciuti da `MLKit` e una classe custom `Language`, cui è stato aggiunto il metodo `getFlag()` per il mapping tra lingue e una bandiera di un paese che meglio la rappresentasse

Rimozione dei modelli di traduzione



- Cliccando sul FloatingActionButton in basso a destra, si aprirà una nuova activity in cui sarà possibile eliminare i modelli per la traduzione precedentemente scaricati
- Ogni modello pesa all'incirca 30MB

DeleteTranslationModulesActivity



- Il layout dell'activity consta sostanzialmente di una RecyclerView, in cui è possibile una selezione multipla degli items da eliminare
- Si usa il meccanismo delle Coroutine per verificare quali modelli sono presenti nel dispositivo; ne viene poi fatto il mapping per costruire una lista di Language da passare all'adapter della RecyclerView per la visualizzazione

DeleteTranslationModulesActivity



```
val modelManager = RemoteModelManager.getInstance()
// get translation models stored on the device
CoroutineScope(Dispatchers.IO).launch {
    Tasks.await(modelManager.getDownloadedModels(TranslateRemoteModel::class.java)
        .addOnSuccessListener { models ->
            models.forEach {
                modules.add(Language(it.language))
            }
        }
        .addOnFailureListener{
            Log.d("Modules", "Unable to detect downloaded modules")
            finish()
        })

    CoroutineScope(Dispatchers.Default).launch {
        deleteModule(modelManager, it.code)
    }
}
```

DeleteTranslationModulesActivity



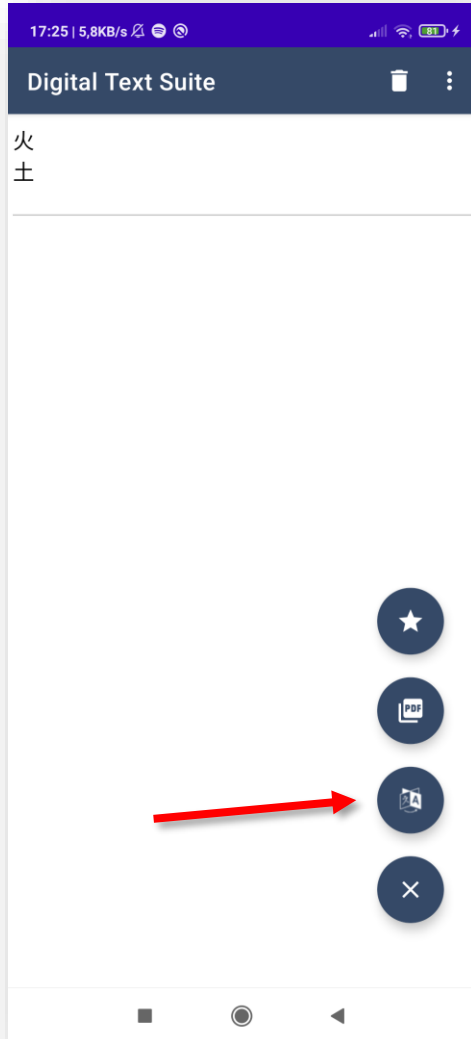
```
private fun deleteModule(modelManager: RemoteModelManager, code: String) {  
    val model = TranslateRemoteModel.Builder(code).build()  
    modelManager.deleteDownloadedModel(model).addOnFailureListener {  
        Toast.makeText(this, "Deletion of model $code failed", Toast.LENGTH_SHORT).show()  
    }  
}
```



Traduzione statica

- È possibile anche effettuare la traduzione del testo in maniera statica, a partire da una nota precedentemente creata
- Questa feature è disponibile nella `TextResultActivity`

Traduzione statica



- Cliccando sul FAB indicato in figura, apparirà lo stesso dialog visto in precedenza per la scelta della lingua target
- Ancora una volta, viene utilizzato il meccanismo delle Coroutines per creare la lista di lingue da passare al dialog



Translator

- Questa volta, per effettuare la traduzione si utilizza la classe `Translator`, in quanto non si ha più bisogno dei `MutableLiveData`
- La traduzione vera e propria avviene sempre tramite le classi `Translator` di `MLKit`, ma stavolta il testo e i codici della lingua sorgente e destinazione vengono direttamente passati come parametri alla funzione `translate()`



Translator

- La funzione `translate()` restituisce un `Task<String>`, il quale, come in precedenza, fa il download dei modelli di traduzione, se necessari, e provvede a tradurre il testo
- Tuttavia, per ottenere il risultato bisogna attendere che il `Task` venga completato, quindi viene bloccata la UI, ma appare una `ProgressBar` accompagnata da una `TextView` per avvisare l'utente che la traduzione è in corso

TextResultActivity



```
setNormalLayoutEnable(false)
binding.grpTranslation.visibility = View.VISIBLE
CoroutineScope(Dispatchers.Default).launch {
    val translator = Translator()

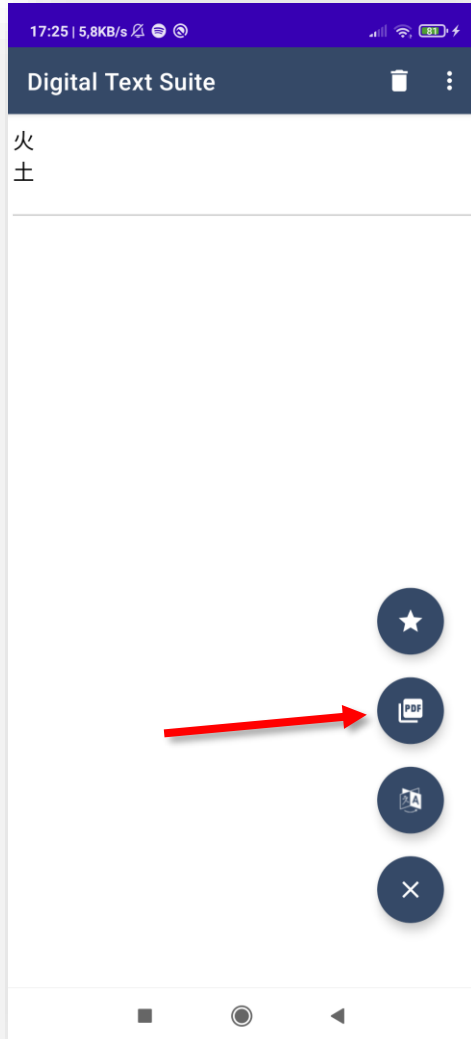
    // waiting the translation, blocking UI
    val newRes = Tasks.await(translator.translate(textResult, language, targetLang))
    CoroutineScope(Dispatchers.Main).launch {
        binding.grpTranslation.visibility = View.GONE
        val intent = Intent(this@TextResultActivity, TextResultActivity::class.java)
        val newNote = Note(newRes, "", "", targetLang, System.currentTimeMillis(), false)
        intent.putExtra("result", newNote)
        intent.putExtra("type", TextResultType.NOT_SAVED)
        startActivity(intent)
        setNormalLayoutEnable(true)
    }
}
```




TextResultActivity

- Una volta ottenuto il testo tradotto, viene creata una nuova nota (classe `Note`)
- Dopodiché viene lanciata una `Intent` verso un'altra istanza di `TextResultActivity`, passando la nuova nota creata come dato

Salvataggio in PDF



- Un'altra feature offerta dalla applicazione è il salvataggio del testo di una nota in formato PDF
- Per la creazione e la scrittura su un documento PDF si è fatto uso della libreria «itextpdf»



Salvataggio in PDF

- Per consentire il salvataggio di files nell'external storage, nella RealMainActivity, vengono create delle cartelle
- Si ottiene dapprima una cartella di root dell'applicazione con il nome dell'app stessa
- Nella cartella di root viene creata una cartella «pdf», al cui interno verrà inserita la cartella di «Default», in cui si potranno salvare i files
- All'utente viene comunque data la possibilità di creare altre sottocartelle di «pdf», in cui salvare i propri files



Creazione cartelle per PDF

```
lateinit var rootDir : File
lateinit var pdfDir : File
lateinit var pdfDefaultDir : File
```

```
rootDir = getOutputDirectory()
pdfDir = File(rootDir, getString(R.string.pdf_dir)).apply { mkdir() }
pdfDefaultDir = File(pdfDir, getString(R.string.default_dir)).apply { mkdir() }
```

```
private fun getOutputDirectory(): File {
    val mediaDir = externalMediaDirs.firstOrNull()?.let {
        File(it, resources.getString(R.string.app_name)).apply { mkdirs() }
    }
    return if (mediaDir != null && mediaDir.exists())
        mediaDir else filesDir
}
```



Creazione del file PDF

- Nella `TextResultActivity`, alla pressione del FAB per il PDF, verrà mostrato un Dialog per la scelta della cartella di destinazione
- Si usa ancora il meccanismo delle Coroutines per andare a leggere dall'external storage quali sono le cartelle precedentemente create, così da passarle al Dialog
- Per la creazione del file PDF è stata creata una apposita classe: `PdfManager`

Creazione del file PDF



```
binding.fabPrintPdf.setOnClickListener {
    CoroutineScope(Dispatchers.IO).launch {
        val stringDirectoryList: MutableList<String> = mutableListOf()
        RealMainActivity.pdfDir.listFiles()?.forEach { stringDirectoryList.add(it.name) }
        stringDirectoryList.add(getString(R.string.new_dir))
        CoroutineScope(Dispatchers.Main).launch {
            val pdfDialog = MakeDirectoryDialog.getInstance()
            pdfDialog.setDirectoryList(stringDirectoryList)

            pdfDialog.setOnDirectorySelected { directory: String, title: String ->
                val text = textResult
                val dir = File(RealMainActivity.pdfDir, directory)
                CoroutineScope(Dispatchers.IO).launch {
                    dir.apply { mkdirs() }
                    PdfManager.transformToPdf(title, text, dir)
                    CoroutineScope(Dispatchers.Main).launch {
                        Toast.makeText(
                            this@TextResultActivity,
                            getString(R.string.pdf_saved),
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                }
            }
            pdfDialog.setOnCancelListener {
                true
            }
            pdfDialog.show(supportFragmentManager, "DIALOGPDF")
        }
    }
}
```

PdfManager



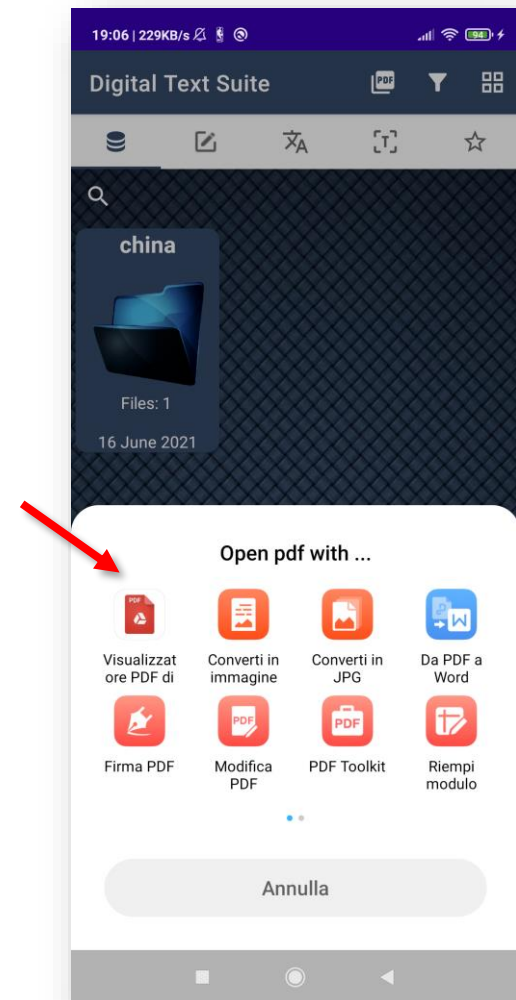
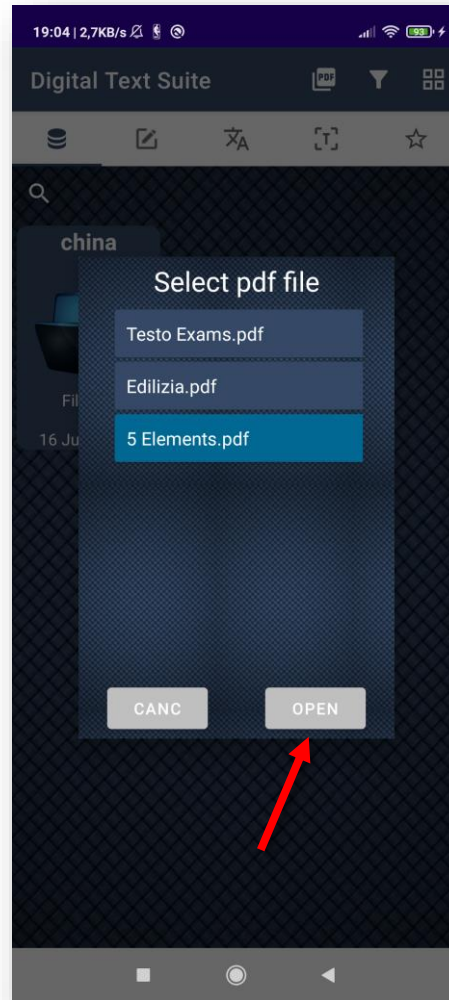
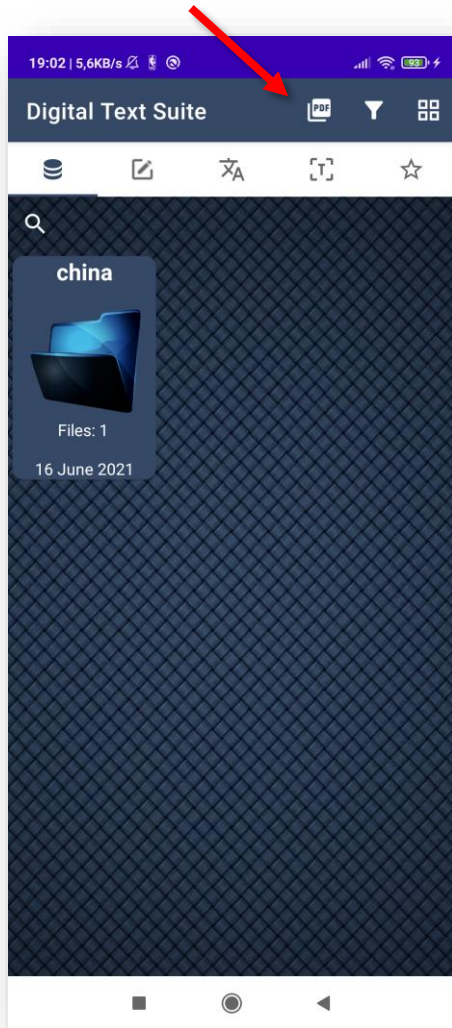
- Questa classe mette a disposizione il metodo `transformToPdf()`, che permette di creare un file pdf con un determinato nome in una determinata cartella, contenente il testo passato come ulteriore parametro
- Utilizza le API della libreria «itextpdf» per costruire un documento a partire da un `PdfDocument` aperto in scrittura
- Il testo viene scritto sul documento creando un oggetto di tipo `Paragraph`, il quale viene aggiunto al documento

PdfManager



```
fun transformToPdf(title: String, text: String, folder: File) {  
  
    val fileLocation = File(folder, title + PDF_EXTENSION).canonicalPath  
  
    // create an instance of PdfDocument at fileLocation location  
    val pdfDocument = PdfDocument(PdfWriter(fileLocation))  
    pdfDocument.defaultPageSize = PageSize.A4  
  
    val document = Document(pdfDocument)  
    val paragraph = Paragraph(text)  
    paragraph.setFontSize(16f)  
    paragraph.setTextAlignment(TextAlignment.LEFT)  
    document.add(paragraph)  
  
    // This will create a file at your fileLocation, specified while creating  
    // PdfDocument instance  
    document.close()  
}
```


Come visualizzare i PDF





Come visualizzare i PDF

- Si può selezionare il file da visualizzare tramite il Dialog mostrato nella seconda delle figure precedenti, che al suo interno contiene una RecyclerView con tutti i files PDF creati con «DigitalTextSuite»
- Selezionato il file da aprire, viene lanciata una Intent con un Chooser, il quale permette di scegliere con quale applicazione aprire il file



Come visualizzare i 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")
    }
}

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

```
intent.setDataAndTypeAndNormalize(data, "application/pdf")
intent.flags = Intent.FLAG_ACTIVITY_CLEAR_TOP
intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
startActivity(Intent.createChooser(intent, "Open pdf with ..."))
```