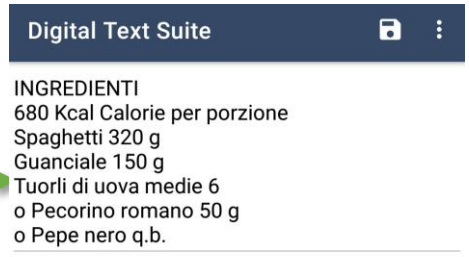
Digitalizzazione di una foto: presentazione caso d'uso

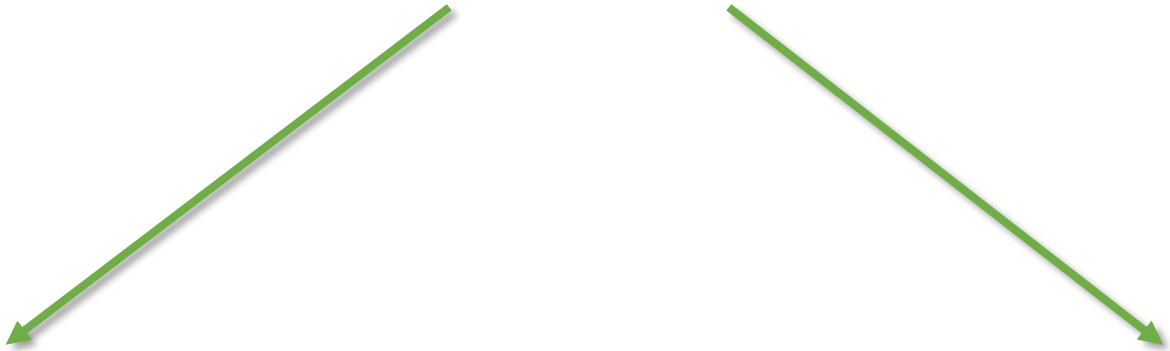
Da un'immagine

Al testo in essa contenuto



Setup

-avvio: disattivazione della modalità notte, gestione sottocartelle di archiviazione per vari casi d'uso, le modalità d'avvio



Normale
(toccando l'icona dell'app)

Condividendo un'immagine (o più) con l'app

Condividendo le immagini con l'app - 1

```
val imageUri = intent.getParcelableExtra<Parcelable>(Intent.EXTRA_STREAM) as? Uri
if (imageUri != null) {
    val intent = Intent(this, RealMainActivity::class.java)
    intent.putExtra("image", imageUri)
    startActivity(intent)
```

Dalla MainActivity: Se condividiamo una sola immagine

Condividendo le immagini con l'app - 2

```
else {
    val listMultiple =
intent.getParcelableArrayListExtra<Parcelable>(Intent.EXTRA_STREA
M)
    if(listMultiple!=null) {
        val listUri = listMultiple.filterIsInstance<Uri>()
        val arrayList = ArrayList<Uri>()
        for(uri in listUri){
            arrayList.add(uri)
        }
        val intent = Intent(this,RealMainActivity::class.java)
        intent.putParcelableArrayListExtra("images",arrayList)
        startActivity(intent)
    }
```

Se ne condividiamo più di una
In tutti e due i casi: intent a RealMainActivity
(vedi di seguito)

Condividendo le immagini con l'app - 3

```
private val REQUIRED_PERMISSIONS = arrayOf(Manifest.permission.CAMERA,
Manifest.permission.READ_EXTERNAL_STORAGE,
Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.INTERNET)
.
.
if(allPermissionsGranted()){
    init()
}else{
    ActivityCompat.requestPermissions(this, REQUIRED_PERMISSIONS,
REQUEST_CODE_PERMISSIONS)
}
```

Nella onCreate: controllo su permessi (per altri casi d'uso)

Condividendo le immagini con l'app - 4

```
val fileUri = intent.getParcelableExtra<Uri>("image")
if(fileUri!=null){
    RecognizerUtil(this).recognize(fileUri,false)
}
else{
    val listUri = intent.getParcelableArrayListExtra<Uri>("images")
    if(listUri!=null){
        RecognizerUtil(this).recognizeAll(listUri,false)
    }
}
```

Successivamente, controlliamo se
effettivamente sono state passate immagini:
Supponiamo di si->La RecognizerUtil analizzerà le immagini

Condividendo le immagini con l'app - 5

```
fun recognize . .
.
val stream =
context.contentResolver.openInputStream(fileUri)
val bitmap = BitmapFactory.decodeStream(stream)
stream!!.close()
val image = InputImage.fromBitmap(bitmap,0)
val recognizer =
TextRecognition.getClient(TextRecognizerOptions.
DEFAULT_OPTIONS)
```

Se riceviamo una sola immagine:
Dall'Uri ricaviamo prima la bitmap dell'immagine,
poi l'oggetto InputImage su cui usare il recognizer
(uso della libreria MLKit)

Condividendo le immagini con l'app - 6

```
recognizer.process(image).addOnSuccessListener { text ->
            if(text.text.isNotEmpty()) {
                val intent = Intent(context, TextResultActivity::class.java)
                val language = text.textBlocks[0].recognizedLanguage
                val note = Note(text.text,"","",
                language,System.currentTimeMillis(),false)
                intent.putExtra("result",note)
                intent.putExtra("type", TextResultType.NOT_SAVED.ordinal)
                if(temp)
                    fileUri.toFile().delete()
                    context.startActivity(intent)
            }
```

Intent all' Activity che mostrerà il testo contenuto nell'immagine

Creazione oggetto Nota: servirà poi per un' eventuale inserimento in DB

Se l'immagine è temporanea allora la cancelliamo (dipende dal caso d'uso)

Aggiunta la nota e il suo tipo all'intent

Funzionamento a blocchi di testo

Condividendo le immagini con l'app - 7

```
else{
    CoroutineScope(Dispatchers.Main).launch{
        Toast.makeText(context,
context.getString(R.string.empty_text),
Toast.LENGTH_LONG).show()
        if(temp)
            fileUri.toFile().delete()
    }
}
```

In caso di testo vuoto oppure di errore nell' elaborazione, lo notifichiamo all'utente senza fare altro

```
.addOnFailureListener{
    CoroutineScope(Dispatchers.Main).launch{
        Toast.makeText(context,context.getString
        (R.string.error_recognition),
        Toast.LENGTH_LONG).show()
        if(temp)
            fileUri.toFile().delete()
    }
}
```

Condividendo le immagini con l'app - 8

```
private fun recognizeRecursive(listUri: List<Uri>, temp: Boolean, count: Int){
        .
        .
        recognizer.process(image).addOnSuccessListener {
            text ->
            if(text.text.isNotEmpty()){
                result.append(text.text)
                result.append("\n")
                if(count == listUri.size-1){
                    .
                    .
                    context.startActivity(intent)
                }
                else{
                    if(temp)
                        listUri[count].toFile().delete()
                    recognizeRecursive(listUri,temp,count+1)
                }
            }
        }.addOnFailureListener{
            .
            .
            }
        }
    }
}
```

Se riceviamo più immagini: quasi stessa funzione, ma ricorsiva, aggiungendo man mano le note ricavate da ogni singola immagine

Avviando regolarmente - 1

```
else {
    val sharedPref = getPreferences(Context.MODE_PRIVATE)
    val isFirst = sharedPref.getBoolean("isFirst",true)
    if(isFirst){
        val intent = Intent(this,TutorialActivity::class.java)
        sharedPref.edit().putBoolean("isFirst",false).apply()
        startActivity(intent)
    }
    else {
        Handler(Looper.getMainLooper()).postDelayed({
            val intent = Intent(this, RealMainActivity::class.java)
            startActivity(intent)
            finish()
        }, SPLASH_SCREEN_DURATION)
    }
}
```

Se è la prima volta che avviamo l'app: tutorial

Altrimenti: splash screen di 1,5 secondi

Se avviamo regolarmente, ovviamente verrà eseguito l'altro ramo nella Main activity

Avviando regolarmente - 2

```
override fun onPageSelected(position: Int) {
    super.onPageSelected(position)
    requestedOrientation = if(position==2 || position==3){
        ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
    } else{
        ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR
    }
}
```

Dettagli sul tab layout: controllo pagina selezionata. Problema grafico con la fotocamera.

Avviando regolarmente - 3

Dettagli su view pager: caricamento dei fragment

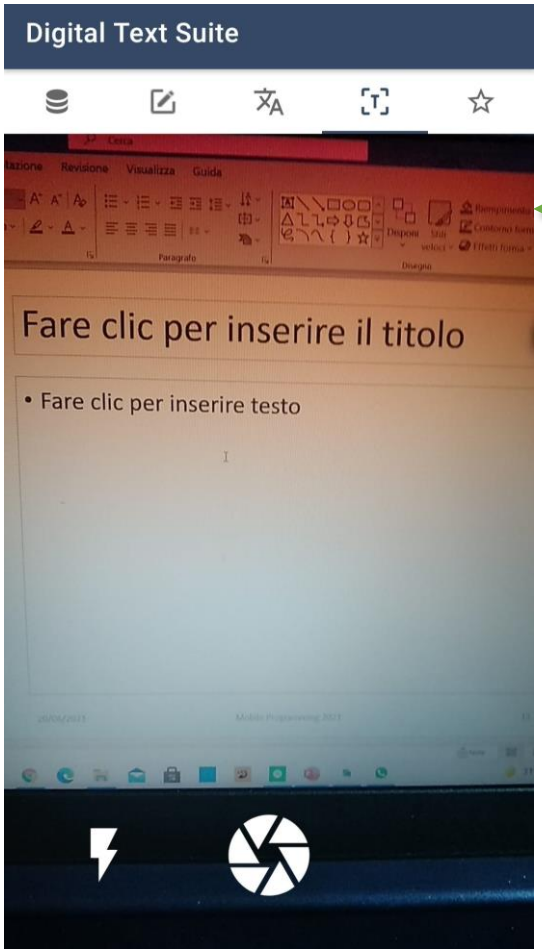
Cliccando su un tab: vengono inizializzati (onCreate, onViewCreated…) tutti i fragment dal primo a quello successivo rispetto al selezionato, poi: onResume su quello interessato

Nostro problema: uso di requireContext() che può generare un'eccezione (faceva crashare l'app)

Avviando regolarmente - 4

La RealMainActivity, in questo caso, non chiamerà ancora nessun metodo di RecognizerUtil.
Cosa possiamo fare?
Clicchiamo qui:

Avviando regolarmente - 5



Il fragment che ci si presenterà.
Consente di scattare una foto, da cui poi verrà estratto il testo.
Come funziona?
Fragment Astratto

Avviando regolarmente - 6

```
abstract class CameraFragment : Fragment(), CaptureHandler {
    protected var cameraExecutor: ExecutorService =
    Executors.newSingleThreadExecutor()
    protected var imageAnalysis: ImageAnalysis? = null
    protected var lensFacing = CameraSelector.DEFAULT_BACK_CAMERA
    protected lateinit var captureHandler: CaptureHandler
    protected var imageCapture : ImageCapture? = null
    protected lateinit var preview : Preview
    protected lateinit var cameraProvider: ProcessCameraProvider
    protected var camera : Camera? = null
```

Executor: oggetto diverso dalle coroutine che permette la programmazione concorrente, vive finché non viene ucciso da programma. Il funzionamento di CameraX prevede il loro utilizzo.

Le altre varie variabili: riguardano CameraX, per esempio la preview o l'oggetto che gestisce le impostazioni di cattura (es. flash)

CaptureHandler: interfaccia che le classi concrete devono implementare, riuso (vedi in seguito)

Avviando regolarmente - 7

```
override fun onAttach(context: Context) {
    if(context is CaptureHandler){
        captureHandler = context
    }
    else
        captureHandler = this
    super.onAttach(context)
}
```

Quando ci sarà il binding tra fragment e ciclo di vita.

Distruzione dell'executor

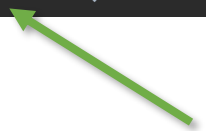
```
override fun onDestroy() {
    super.onDestroy()
    cameraExecutor.shutdown()
}
```

```
fun setFlash(mode: Int){
    imageCapture!!.flashMode=mode
}
```

Impostazione modalità flash

Avviando regolarmente - 8

```
fun takePhoto(save: Boolean){
    val outputDirectory = if(save)
    //store the image
        requireActivity().getExternalFilesDir
        (Environment.DIRECTORY_DCIM+File.separator+"effectscam")
    else
    //cache the image
        requireContext().cacheDir
    val filename = "Image_" + System.currentTimeMillis()
    val extension = ".jpg"
    val photoFile = if(save)
        File(outputDirectory, filename+extension)
    else
        File.createTempFile(filename,extension,outputDirectory)
```



L'immagine è da salvare in memoria o meno?

Avviando regolarmente - 9

```
val outputFileOptions: ImageCapture.OutputFileOptions
if(lensFacing== CameraSelector.DEFAULT_FRONT_CAMERA){
    val metadata = ImageCapture.Metadata()
    //Reverse image
    metadata.isReversedHorizontal=true
    outputFileOptions = ImageCapture.OutputFileOptions.
    Builder(photoFile).setMetadata(metadata).build()
}
else
    outputFileOptions = ImageCapture.OutputFileOptions.Builder(photoFile).build()
```

Impostazioni per la cattura dell'immagine: riuso dall'esonero

Avviando regolarmente - 10

```
imageCapture!!.takePicture(outputFileOptions, cameraExecutor,
  object : ImageCapture.OnImageSavedCallback {
    override fun onImageSaved(outputFileResults: ImageCapture.OutputFileResults) {
        captureHandler.saveImage(Uri.fromFile(photoFile),!save)
    }

    override fun onError(exception: ImageCaptureException) {
        captureHandler.errorHandler(exception)
    }
})
```

Metodo messo a disposizione delle API di cameraX.
Parametri e funzioni sovrascritte.

Avviando regolarmente - 11

Fragment concreto di questo caso d'uso: FragmentCameraDigitalize

```
private lateinit var mediaPlayer: MediaPlayer
private var flashMode = ImageCapture.FLASH_MODE_ON
```

Effetto audio

Modalità flash

Le funzioni dell'interfaccia CaptureHandler implementate: se si cattura l'immagine, verrà inviata all'oggetto RecognizerUtil

```
override fun saveImage(fileUri: Uri, temp: Boolean) {
    RecognizerUtil(requireContext()).recognize(fileUri,true)
}

override fun errorHandler(exception: ImageCaptureException) {
    CoroutineScope(Dispatchers.Main).launch{
        Toast.makeText(requireContext(),
        getString(R.string.error_camera),
        Toast.LENGTH_LONG).show()
    }
}
```

Avviando regolarmente - 12

```
. . .
    object : OrientationEventListener(requireContext()) {
        override fun onOrientationChanged(orientation: Int) {
            if (orientation == UNKNOWN_ORIENTATION) {
                return
            }

            val rotation = when (orientation) {
                in 45 until 135 -> Surface.ROTATION_270
                in 135 until 225 -> Surface.ROTATION_180
                in 225 until 315 -> Surface.ROTATION_90
                else -> Surface.ROTATION_0
            }
            if(imageCapture!=null)
                imageCapture!!.targetRotation = rotation
        }
    }
}
```

Gestione orientamento immagine per non avere problemi con l'analizzatore dopo aver catturato l'immagine: come funziona?

Avviando regolarmente - 13

```
mediaPlayer = MediaPlayer.create(requireContext(), R.raw.camera_shutter_click)
binding.imgBtnCamera.setOnClickListener{
    takePhoto(false)

    mediaPlayer.start()

    val zoomOut = AnimationUtils.loadAnimation(requireContext(),
      R.anim.zoom_out)
    zoomOut.setAnimationListener(ZoomAnimationListener
      (binding.imgBtnCamera,requireContext()))

    binding.imgBtnCamera.startAnimation(zoomOut)
}
```

Stessa cosa per l'animazione

L'effetto audio viene preso dai file in resources e viene fatto partire

Avviando regolarmente - 14

```
binding.imgBtnFlash.setOnClickListener{
    when(flashMode){
        ImageCapture.FLASH_MODE_ON -> {
            flashMode = ImageCapture.FLASH_MODE_AUTO
        }
        ImageCapture.FLASH_MODE_AUTO -> {
            flashMode = ImageCapture.FLASH_MODE_OFF
        }
        ImageCapture.FLASH_MODE_OFF -> {
            flashMode = ImageCapture.FLASH_MODE_ON
        }
    }
    setIconFlash()
    setFlash(flashMode)
    val sharedPreferences = requireActivity().
    getPreferences(Context.MODE_PRIVATE)
    val edit = sharedPreferences.edit()
    edit.putInt("flash",flashMode)
    edit.apply()
}
```

Variazione modalità flash

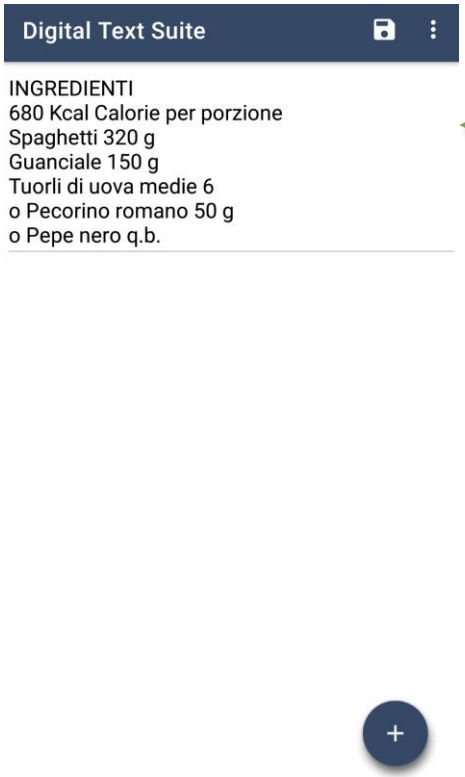
Ricordiamo inoltre la modalità scelta per la prossima volta

Avviando regolarmente - 15

```
private fun setIconFlash(){
    binding.imgBtnFlash.setImageDrawable(when(flashMode){
        ImageCapture.FLASH_MODE_ON ->
            ContextCompat.getDrawable(requireContext(),
            R.drawable.flash_on_48)
        ImageCapture.FLASH_MODE_OFF ->
            ContextCompat.getDrawable(requireContext(),
            R.drawable.flash_off_48)
        else -> ContextCompat.getDrawable(requireContext(),
            R.drawable.flash_auto_48)
    })
}
```

Ovviamente in base alla configurazione del flash, cambiamo anche l'icona

Risultato finale - 1



Schermata generale, vari elementi

Interazioni: elaborazione della nota, previo salvataggio

Risultato finale - 2

```
R.id.it_editable->{
    if(type==TextResultType.NOT_SAVED){
        Toast.makeText(this,getString(R.string.not_saved_edit),
        Toast.LENGTH_LONG).show()
    }
    else {
        .
        .
        if (editable) {
            type = TextResultType.EDITABLE
            binding.editTextTextMultiLine.isEnabled = true
            setType(menu)
        } else {
            if (originalText.equals(textResult)) {
                type = TextResultType.SAVED
                setType(menu)
            }
            binding.editTextTextMultiLine.isEnabled = false
        }
    }
}
```

Se vogliamo editare dobbiamo prima salvare il testo

Abilitiamo o disabilitiamo la possibilità di modificare il testo digitalizzato…

Risultato finale - 3

```
R.id.it_undo -> {
    binding.editTextTextMultiLine.text.clear()
    binding.editTextTextMultiLine.text.append(originalText)
    textResult=originalText
}
```

Possibilità di
tornare indietro
con le modifiche

Risultato finale - 4

```
R.id.it_delete -> {
    val alert = AlertDialog.Builder(this)
    alert.setTitle(R.string.alert_delete_title)
    alert.setMessage(R.string.alert_delete_message)
    alert.setPositiveButton(R.string.yes)
    { dialogInterface: DialogInterface, _:
      Int ->
        CoroutineScope(Dispatchers.IO).launch {
        dao.deleteNote(note)
            CoroutineScope(Dispatchers.Main).launch {
                dialogInterface.cancel()
                finish()
            }
    }
    }
    alert.setNegativeButton(R.string.no) { dialogInterface: DialogInterface, _:
        Int ->
        dialogInterface.cancel()
    }
    alert.show()
```

Possibilità di cancellare del tutto la nota: alert

Risultato finale - 5

```
R.id.it_copy -> {
    val clipboard = getSystemService(Context.CLIPBOARD_SERVICE) as ClipboardManager
    val clip = ClipData.newPlainText("note",textResult)
    clipboard.setPrimaryClip(clip)
    Toast.makeText(this,getString(R.string.copied),Toast.LENGTH_SHORT).show()
}
```

Possibilità di copiare la nota per intero

Extra: animazione tasti