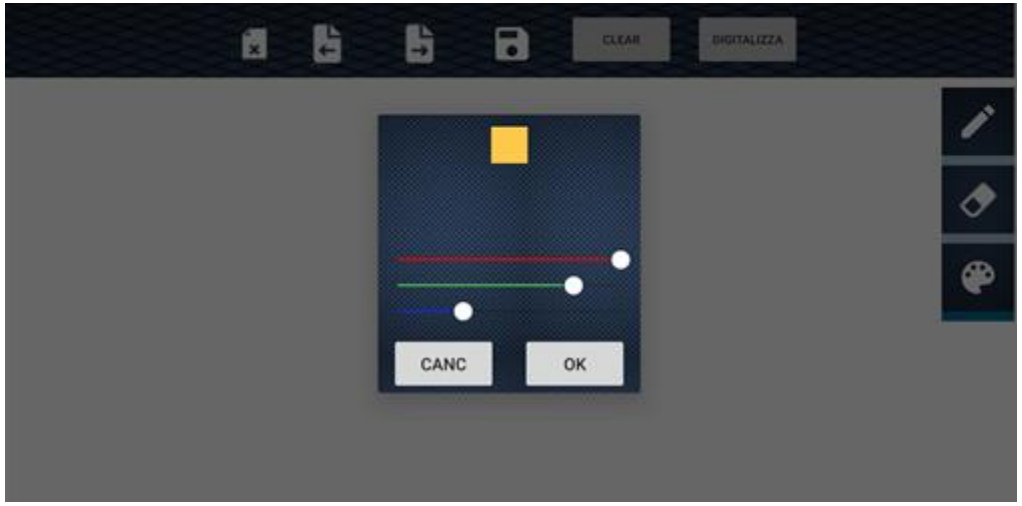
DialogColor -1

- Questo è un Fragment che estende la classe DialogFragment
- DialogColor permette all'utente di scegliere il colore della penna per disegnare sulla lavagna, tramite seekbar, combinando la tabella rgb



DialogColor -2

```
class ColorDialog : DialogFragment() {
    private var colorImageView: ImageView? = null
    private var redValue : Int = 0
    private var greenValue : Int = 0
    private var blueValue : Int = 0
    private var colors : Int = Color.BLACK
    private var colorListener: (colors : Int) -> Unit = {

    }
    private var cancelListener: () -> Boolean ={
        true
    }
    private lateinit var binding: ChooseColorDialogBinding

    companion object{
        fun getInstance(): ColorDialog{
            var instance : ColorDialog? = null
            if(instance == null){
                instance = ColorDialog()
            }
            return instance
        }
    }
```

DialogColor -3

```
override fun onCreateView( inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View {
   binding = ChooseColorDialogBinding.inflate(inflater)
   colorImageView = binding.ColorPicker
   colorImageView!!.setBackgroundColor(Color.BLACK)
   isCancelable = false
   setSeek()
   return binding.root
}
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
   binding.ColorPicker
   binding.seekBarB
   binding.seekBarG
   binding.seekBarR
   setSeek()
   binding.btnOk.setOnClickListener{
      colorListener(colors)
      dismiss()
   }
   binding.btnCanc.setOnClickListener{
      if(cancelListener())
         dismiss()
   }
}
fun setOnColorSelected(listener: (color: Int)-> Unit){
   this.colorListener=listener
}
fun setOnCancelSelected(listener: () -> Boolean){
   cancelListener = listener
}
```

Gestione della seekbar -1

- Per gestire il cambiamento di valore della seekbar bisogna eseguire override dei metodi:
  - onProgressChanged(

    seekBar: SeekBar,

    progress: Int,

    fromUser: Boolean)//segnala il cambiamento del livelli
  - onStartTrackingTouch(

    seekBar: SeekBar?) //segnala inizio del tocco da parte dell’utente
  - onStopTrackingTouch(

    seekBar: SeekBar?) //segnala fine del tocco da parte dell’utente

Gestione della seekbar -2

```
private fun setSeek(){
   val seekBarR = binding.seekBarR
   val seekBarG = binding.seekBarG
   val seekBarB = binding.seekBarB
   seekBarR.setOnSeekBarChangeListener(mChangeListener)
   seekBarG.setOnSeekBarChangeListener(mChangeListener)
   seekBarB.setOnSeekBarChangeListener(mChangeListener)
}
private val mChangeListener: SeekBar.OnSeekBarChangeListener = object :
   SeekBar.OnSeekBarChangeListener {
   override fun onProgressChanged(
      seekBar: SeekBar,
      progress: Int,
      fromUser: Boolean
   ) {
      val viewId = seekBar
      when (viewId) {
         binding.seekBarR -> redValue = progress
         binding.seekBarG -> greenValue = progress
         binding.seekBarB -> blueValue = progress
      }
      colors = Color.rgb(redValue, greenValue, blueValue)
      colorImageView!!.setBackgroundColor(colors)
   }
   override fun onStartTrackingTouch(seekBar: SeekBar?) { }

   override fun onStopTrackingTouch(seekBar: SeekBar?) { }
}
```

Creazione del dialog

```
binding.btnPickColor.setOnClickListener{
   val colorPick = ColorDialog.getInstance()
   var colore : Int
   colorPick.setOnColorSelected {
      colore = it
      binding.whiteboard.drawingMode = DrawingMode.DRAW
      binding.selected2.setBackgroundColor(ContextCompat.getColor(this,R.color.unselected))
      binding.selected.setBackgroundColor(ContextCompat.getColor(this,R.color.selected_blue))
      binding.selected3.setBackgroundColor(ContextCompat.getColor(this,R.color.unselected))
      setColor(colore)
   }
   colorPick.setOnCancelSelected {
      binding.whiteboard.drawingMode = DrawingMode.DRAW
      binding.selected2.setBackgroundColor(ContextCompat.getColor(this,R.color.unselected))
      binding.selected.setBackgroundColor(ContextCompat.getColor(this,R.color.selected_blue))
      binding.selected3.setBackgroundColor(ContextCompat.getColor(this,R.color.unselected))
      true
   }
   colorPick.show(supportFragmentManager,"ColorDialog")
```

PenDialog

```
penTouch = 0
…
binding.btnPen.setOnClickListener{
   binding.whiteboard.drawingMode= DrawingMode.DRAW
   binding.whiteboard.isEnabled = true
   penTouch++
   …
   val penPick = PenDialog.getInstance()
   var value : Int
   penPick.setOnStrokeSelected {
      penTouch = 1
      value = it
      setStroke(value)
   }
   if(penTouch==2)
      penPick.show(supportFragmentManager,"PenDialog")
}
```

TextResultActivity -1

- Una volta creata una nota abbiamo la possibilità tramite un menu di scegliere se modificarla, salvarla, copiarla, o cancellarla
- Inoltre ci sono 3 diverse features:
  - Traduzione
  - Salvare in formato pdf
  - Metterla tra i preferiti

TextResultActivity -2

```
class TextResultActivity : AppCompatActivity() {
...
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   binding = ActivityTextResultBinding.inflate(layoutInflater)
   setContentView(binding.root)
   note = intent.getParcelableExtra("result") ?: Note("","","","",System.currentTimeMillis(),false)
   textResult = note.text
   language = note.language
   originalText=textResult
   whiteboard = intent.getParcelableExtra("whiteboard") ?: DigitalizedWhiteboards()
   val ordinal = intent.getIntExtra("type",TextResultType.NOT_SAVED.ordinal)
   type = when(ordinal){
      TextResultType.SAVED.ordinal -> TextResultType.SAVED
      TextResultType.EDITABLE.ordinal -> TextResultType.EDITABLE
      else -> TextResultType.NOT_SAVED
   }
   initializeDB()
   setUI()
}
```

Creazione del menu -1

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
   menuInflater.inflate(R.menu.text_result_menu,menu)
   this.menu=menu!!
   setType(menu)
   return super.onCreateOptionsMenu(menu)
}
private fun setType(menu: Menu?){
   var itSave: MenuItem? = null
   var itDelete: MenuItem? = null
   var itUndo: MenuItem? = null
   for (item in menu!!.children){
      when(item.itemId){
         R.id.it_delete -> itDelete=item
         R.id.it_save -> itSave=item
         R.id.it_undo -> itUndo=item
      }
   }
   when(type){
      TextResultType.SAVED -> {
         itDelete!!.isVisible=true
         itSave!!.isVisible=false
         itUndo!!.isVisible=false
      }
      TextResultType.NOT_SAVED -> {
         itDelete!!.isVisible=false
         itSave!!.isVisible=true
         itUndo!!.isVisible=false
      }
      TextResultType.EDITABLE->{
         itDelete!!.isVisible=false
         itSave!!.isVisible=true
         itUndo!!.isVisible=true
      }
```

Creazione del menu -2

```
 override fun onOptionsItemSelected(item: MenuItem): Boolean
{
       when(item.itemId){
          R.id.it_save -> { }
          R.id.it_editable->{ }
          R.id.it_undo -> { }
          R.id.it_delete -> { }
          R.id.it_copy -> { }
   return super.onOptionsItemSelected(item)
}
```

Salvare la nota

```
CoroutineScope(Dispatchers.IO).launch {
    val directoryList = dao.loadDirectories()
    CoroutineScope(Dispatchers.Main).launch {
        val dialog = MakeDirectoryDialog.getInstance()
        dialog.setDirectoryList(directoryList)
        if (type == TextResultType.NOT_SAVED) {
            dialog.setOnDirectorySelected { directory: String, title: String ->
                note.text = textResult
                note.language = language
                note.directory = directory
                note.title = title
                CoroutineScope(Dispatchers.IO).launch {
                    dao.insertNote(note)
                    …
                  }
              }
        }
}
```

Editare nota

- Prima di poter modificare la nota bisogna salvarla

```
if(type==TextResultType.NOT_SAVED){
        Toast.makeText(this,getString(R.string.not_saved_edit),Toast.LENGTH_LONG).show()
    }
        ....
if (editable) {
    type = TextResultType.EDITABLE
    binding.editTextTextMultiLine.isEnabled = true
    setType(menu)
}
```

Ritornare alla nota originale

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

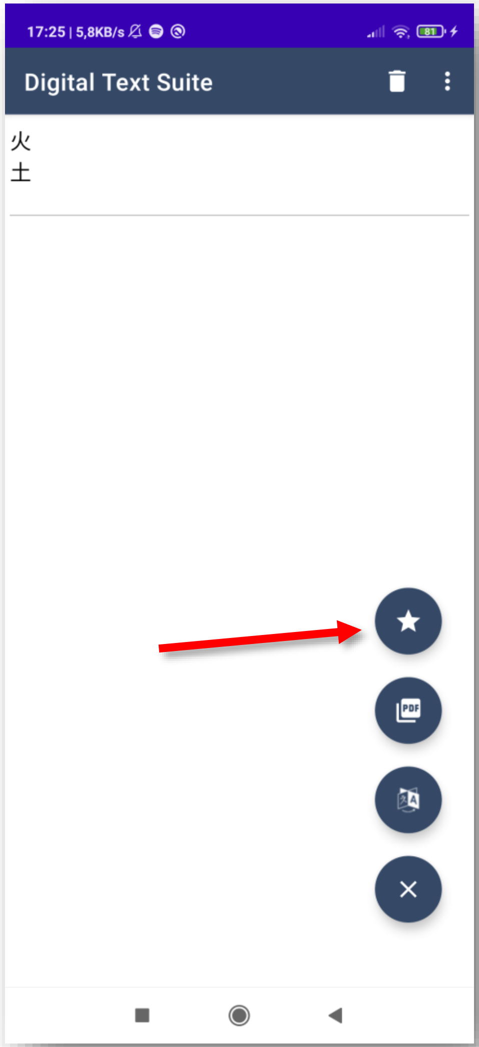
Eliminare la nota

```
CoroutineScope(Dispatchers.IO).launch {
dao.deleteNote(note)
    CoroutineScope(Dispatchers.Main).launch {
        dialogInterface.cancel()
        finish()
    }
}
```

Copiare la nota

```
R.id.it_copy -> {
   val clipboard = getSystemService(Context.CLIPBOARD_SERVICE) as ClipboardManager
   val clip = ClipData.newPlainText("note",textResult)
   clipboard.setPrimaryClip(clip)
```

Mettere la nota tra i preferiti



- Una feature che troviamo è quella di salvare le nostre note preferite per trovarle più facilmente nella sezione dedicata
- Nel bottone indicato apparirà la stella piena (salvata tra i preferiti), altrimenti solo contorno della stella

Salvare la nota tra i preferiti

```
binding.fabFavourite.setOnClickListener{
    CoroutineScope(Dispatchers.IO).launch{
        note.preferito = !note.preferito
        dao.updateNote(note)
        CoroutineScope(Dispatchers.Main).launch{
            binding.fabFavourite.setImageDrawable(if(note.preferito)
                ContextCompat.getDrawable(this@TextResultActivity,R.drawable.ic_baseline_star_24)
            else
                ContextCompat.getDrawable(this@TextResultActivity,R.drawable.favourite_icon_24))
        }
    }
}
```