Digital Ink Recognition

- Permette all'utente di poter scrivere a mano su una whiteboard e digitalizzare il testo sotto forma di una Nota all'interno della nostra App.
- Le whiteboard possono essere memorizzate sul dispositivo per essere modificate successivamente
- Sono disponibili più di 300 linguaggi per la digitalizzazione.

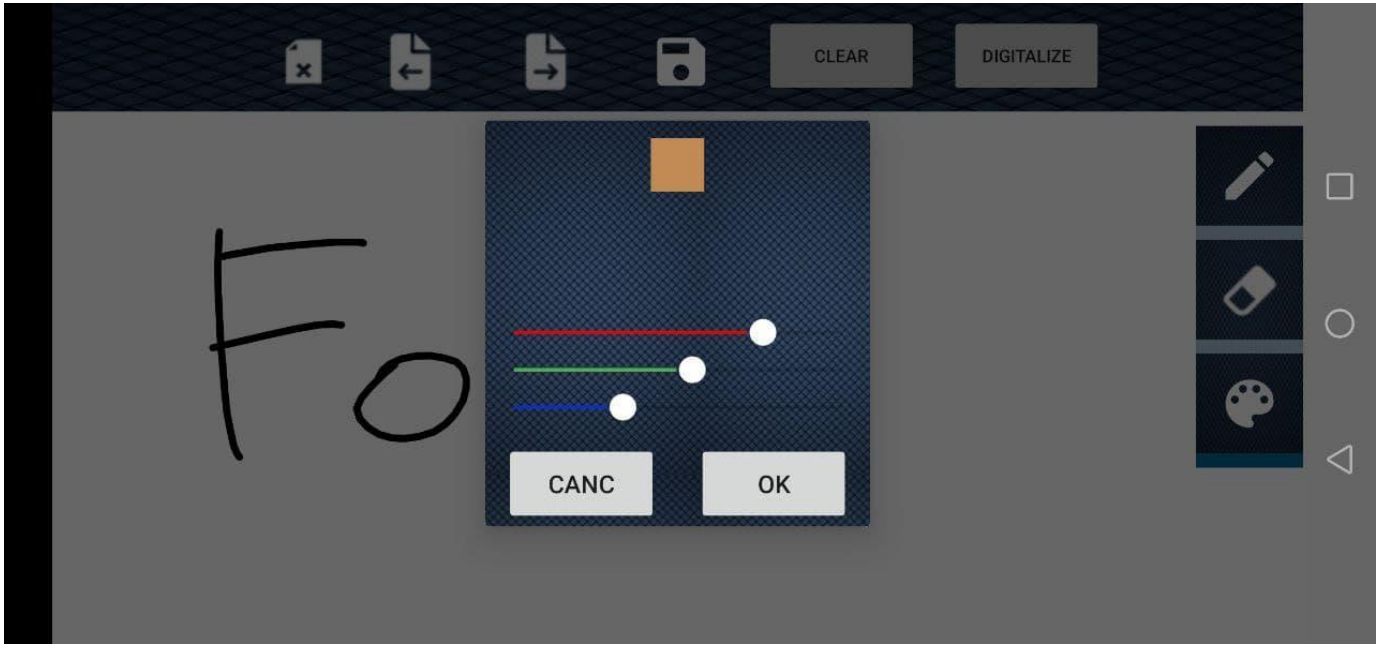
Digital Ink Recognition

- Nel secondo tab vengono mostrate tutte le whiteboard salvate e qui è possibile crearne una nuova.
- Le whiteboard digitalizzate hanno un bottone in basso a destra che mostra la nota associata.

Digital Ink Recognition

- La barra a destra permette di selezionare gli strumenti di disegno. La barra in alto fornisce strumenti di gestione della whiteboard

Digital Ink Recognition



- E’ possibile cambiare colore della penna con un simpatico Dialog dove l’utente può costruire il colore combinando l’rgb

Digital Ink Recognition

- Entriamo nel merito del codice…
- Per realizzare questo caso d'uso abbiamo utilizzato il servizio di API offerto da MLKit.
- Le api di MLKit operano l'analisi su oggetti di tipo Ink e restituiscono il testo riconosciuto.
- Per consentire all'utente di scrivere sullo schermo abbiamo realizzato un oggetto di view: Whiteboard

Digital Ink Recognition

- Per la visualizzazione delle whiteboard memorizzate abbiamo utilizzato un Fragment il DigitalInkFragment
- Le whiteboard vengono memorizzate nel database mentre i metadati sono memorizzati in un json nello storage del dispositivo

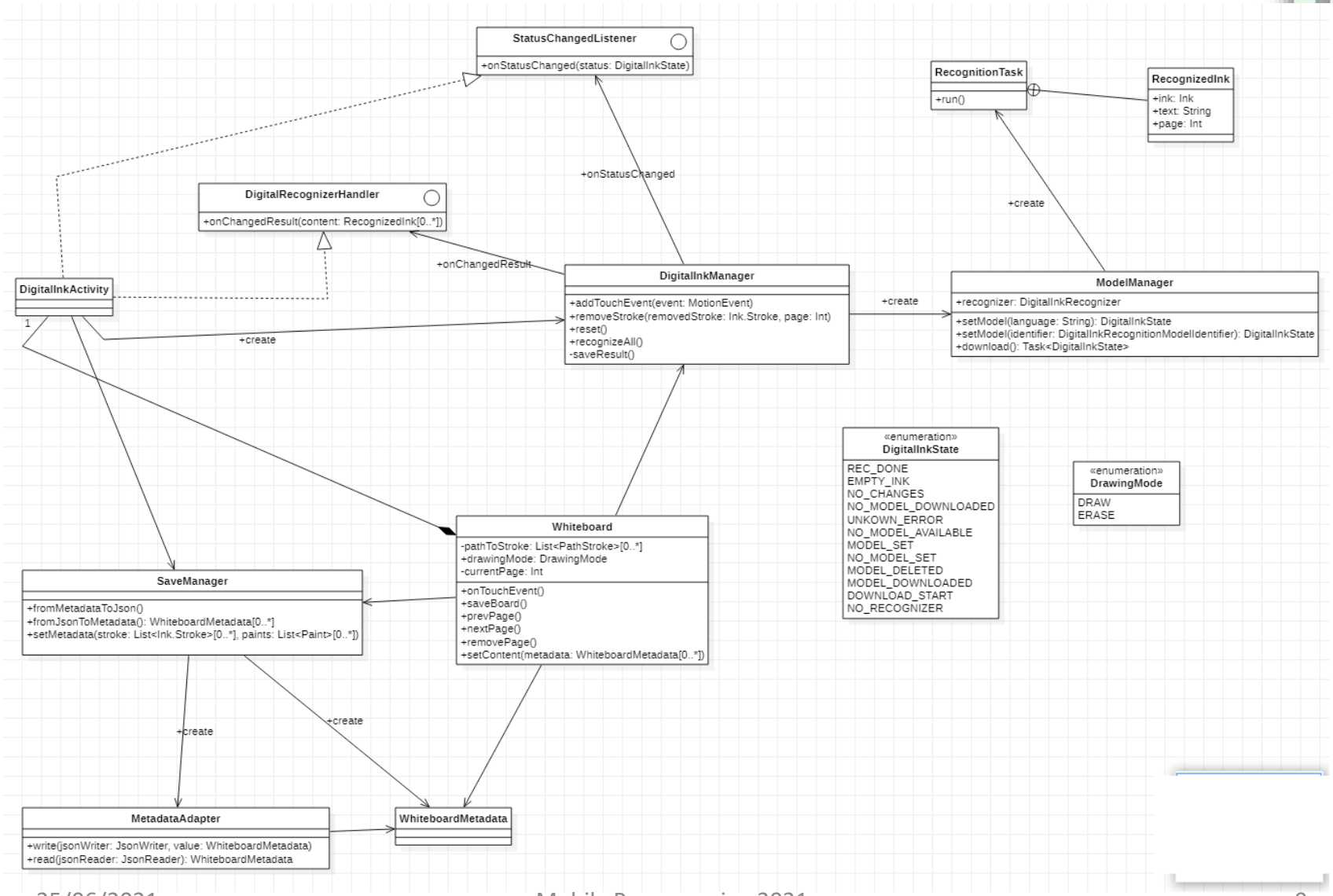
Digital Ink Recognition

- E’ stato realizzato anche un meccanismo di selezione multipla e rimozione per le whiteboard sfruttando le ActionMode
- Le whiteboard cambiano colore grazie ad un selector impostato come background

Digital Ink Recognition

- Per consentire l'interazione con la whiteboard è stata realizzata una attività apposita DigitalInkActivity
- Per una esperienza migliore questa activity ha una screen orientation in LANDSCAPE
- Per l'interazione con il manager l'activity implementa le interfacce StatusChangedListener e DigitalRecognizerHandler

DigitalInkRecognition



Whiteboard

- Whiteboard è una classe che estende View. Offre uno spazio bianco dove l'utente può disegnare.
- Una Whiteboard è autosufficiente e gestisce automaticamente la comunicazione con le classi responsabili della recognition
- Offre dei metodi per il salvataggio sullo storage del device e ripristino di dati salvati.

Whiteboard

- Per operare una Whiteboard ha bisogno di un DigitalInkManager.

```
fun setDigitalInkManager(digInkManager: DigitalInkManager){
    manager = digInkManager
}
```

- Per consentire all'utente di disegnare viene fatto l'override del metodo onTouchEvent

Whiteboard

```
override fun onTouchEvent(event: MotionEvent): Boolean {
    performClick()
    val action = event.actionMasked
    val x = event.x
    val y = event.y
    if(drawingMode == DrawingMode.DRAW) {
        when (action) {
            MotionEvent.ACTION_DOWN -> currentStroke.moveTo(x, y)
            MotionEvent.ACTION_MOVE -> currentStroke.lineTo(x, y)
            MotionEvent.ACTION_UP -> {
                currentStroke.lineTo(x, y)
                drawCanvas.drawPath(currentStroke, currentStrokePaint)
                paths[currentPage].add(currentStroke)
                lastStroke = currentStroke
                currentStroke = Path()
            }
            else -> {

            }
        }
        /**
         * Inform the manager of the event. The method returns true if and only if a new stroke has been added.
         * In this case the stroke is stored with the associated path
         */
        if(manager.addTouchEvent(event,currentPage)){
            pathToStroke[currentPage].add(PathStroke(lastStroke,manager.lastStroke,currentStrokePaint))
        }
        invalidate()
        return true
    }
```

Whiteboard

```
else{
   /**
    * Erase mode: Search among all the paths the one that contains the point corresponding to the touch event by
    * building a rectangle around it for each path and verifying that the rectangle contains the point
    */
   currentStroke = Path()
   val rect = RectF()
   for (i in 0 until pathToStroke[currentPage].size){
      pathToStroke[currentPage][i].path.computeBounds(rect,false)
      if(rect.contains(x,y)){
         /**
          * Notify the manager of the removal of the detected stroke
          */
         manager.removeStroke(pathToStroke[currentPage][i].stroke,currentPage)
         erase(i)
         break
      }
   }
   return false
}
```

Whiteboard

- E’ possibile aggiungere più di una pagina in una whiteboard, muoversi tra le pagine e rimuoverle tramite questi bottoni

Whiteboard

- Per aggiungere una pagina o spostarsi in avanti:

```
fun nextPage(){
   if(currentPage+1>=paths.size){
      paths.add(ArrayList())
      pathToStroke.add(ArrayList())
      manager.newPage()
   }
   currentStroke = Path()
   currentPage++
   onSizeChanged(
      canvasBitmap!!.width,
      canvasBitmap!!.height,
      canvasBitmap!!.width,
      canvasBitmap!!.height
   )
}
```

- Per spostarsi indietro:

```
fun prevPage(): Boolean{
   if(currentPage-1>=0)
      currentPage--
   else
      return false
   currentStroke = Path()
   onSizeChanged(
      canvasBitmap!!.width,
      canvasBitmap!!.height,
      canvasBitmap!!.width,
      canvasBitmap!!.height
   )
   return currentPage>0
}
```

Whiteboard

- Per rimuovere una pagina:

```
/**
 * This method allows you to remove the current page. If it is the only one present then a new one is created
 */
fun removePage(){
    paths.removeAt(currentPage)
    pathToStroke.removeAt(currentPage)
    if(currentPage == 0 && paths.isEmpty() && pathToStroke.isEmpty()){
        paths.add(ArrayList())
        pathToStroke.add(ArrayList())
    }
    else{
        if(currentPage>0)
            currentPage--
    }
    manager.deletePage(currentPage)
    currentStroke = Path()
    onSizeChanged(
        canvasBitmap!!.width,
        canvasBitmap!!.height,
        canvasBitmap!!.width,
        canvasBitmap!!.height
    )
    invalidate()
}
```

Whiteboard

- E’ possibile salvare una Whiteboard o ripristinarne una sfruttando i WhiteboardMetadata e la classe SaveManager. Una Whiteboard viene salvata in formato json.
- Offre un metodo saveBoard per il salvataggio e un metodo setContent per il ripristino

Whiteboard - saveBoard

```
/**
 * This method allows you to save the current whiteboard (and all its pages)
 * @param path is the file that must contain the metadata used to restore the whiteboard
 * @param imagePath is the file that must contain the whiteboard preview (Usually a front page image)
 */
fun saveBoard(path: File,imagePath: File){
    val listOfStrokes = mutableListOf<MutableList<Ink.Stroke>>()
    val listPaints = mutableListOf<MutableList<Paint>>()
    for(page in pathToStroke.indices) {
        listOfStrokes.add(mutableListOf())
        listPaints.add(mutableListOf())
        Log.d("PAGESTROKESIZE",pathToStroke[page].size.toString())
        for (i in pathToStroke[page].indices) {
            listOfStrokes[page].add(pathToStroke[page][i].stroke)
            listPaints[page].add(pathToStroke[page][i].paint)
        }
    }
    val saveManager = SaveManager()
    saveManager.path = path.absolutePath
    saveManager.setMetadata(listOfStrokes,listPaints)
    saveManager.fromMetadataToJson()
    val fileOutputStream = FileOutputStream(imagePath)
    val bitmap = Bitmap.createBitmap(canvasBitmap!!.width,canvasBitmap!!.height,Bitmap.Config.ARGB_8888)
    val canvas = Canvas()
    canvas.setBitmap(bitmap)
    for(p in pathToStroke[0]){
        canvas.drawPath(p.path,p.paint)
    }
    bitmap.compress(Bitmap.CompressFormat.PNG,100,fileOutputStream)
}
```

Whiteboard – setContent (1)

```
/**
 * This method allows you to restore a whiteboard instance
 * @param metadata The stored WhiteboardMetadata
 */
fun setContent(metadata: MutableList<WhiteboardMetadata>){
    metadata.sortBy{
        it.id
    }
    manager.setStartNumberPage(metadata.size)
    currentStroke = Path()
    paths = mutableListOf()
    pathToStroke = mutableListOf()
    currentPage = -1
    for(whiteboard in metadata) {
        currentPage ++
        paths.add(ArrayList())
        pathToStroke.add(ArrayList())
        val decoded = whiteboard.decodeMetadata()
        val listStroke = mutableListOf<Ink.Stroke>()
        for(elem in decoded){
            listStroke.add(elem.stroke)
        }
        manager.setInkStrokes(listStroke,currentPage)
```

Whiteboard – setContent (2)

```
for (stroke in whiteboard.decodeMetadata()) {
        for (i in stroke.stroke.points.indices) {
           if (i == 0) {
              currentStroke.moveTo(stroke.stroke.points[i].x, stroke.stroke.points[i].y)
           } else {
              currentStroke.lineTo(stroke.stroke.points[i].x, stroke.stroke.points[i].y)
           }
        }
        paths[currentPage].add(currentStroke)
        lastStroke = currentStroke
        currentStroke = Path()
        color = stroke.paint.color
        this.stroke = (stroke.paint.strokeWidth/(resources.displayMetrics.densityDpi/ DisplayMetrics.DENSITY_DEFAULT)).toInt()
        resetPaint()
        pathToStroke[currentPage].add(PathStroke(lastStroke, stroke.stroke,currentStrokePaint))
      }
   }
   currentPage = 0
   currentStroke = Path()
}
```

Whiteboard

- Una Whiteboard offre anche un metodo temporarySave per il salvataggio in un file in cache da utilizzare per una bufferizzazione temporanea della whiteboard.
- Il metodo utilizza gli stessi meccanismi descritti precedentemente per il salvataggio ma crea l'uri nella cache del dispositivo e restituisce l'uri corrispondente al file creato

DigitalInkManager

- DigitalInkManager è una classe che gestisce le operazioni di digital ink recognize
- Costruisce gli Ink da analizzare utilizzando le api di MLKit e le informazioni fornite dalla Whiteboard
- E’ responsabile della creazione e gestione del ModelManager per il download dei modelli
- Offre un metodo recognizeAll per la digitalizzazione degli Ink

DigitalInkManager

- Un oggetto di DigitalInkManager possiede uno stato che può variare nel tempo!
- Per gestire le variazioni di stato l'activity deve implementare l'interfaccia StatusChangedListener

```
interface StatusChangedListener {
   /** This method is called when the recognized content changes.  */
   fun onStatusChanged(status: DigitalInkState)
}
```

- Lo stato del manager è uno dei valori dell'enum DigitalInkState

DigitalInkManager

- Per ricevere i risultati di una digital ink recognition l'activity deve implementare l'interfaccia DigitalRecognizerHandler

```
interface DigitalRecognizerHandler {
   /**
    * This method is called by the DigitalInkManager when a new result is ready
    */
   fun onChangedResult(content: MutableList<RecognitionTask.RecognizedInk>)
}
```

- L'activity deve inoltre registrarsi come handler usando gli appositi metodi esposti dal DigitalInkManager

ModelManager

- Questa classe è responsabile della gestione dei modelli di MLKit per la digital ink recognition
- E’ anche responsabile della creazione del recognizer di MLKit

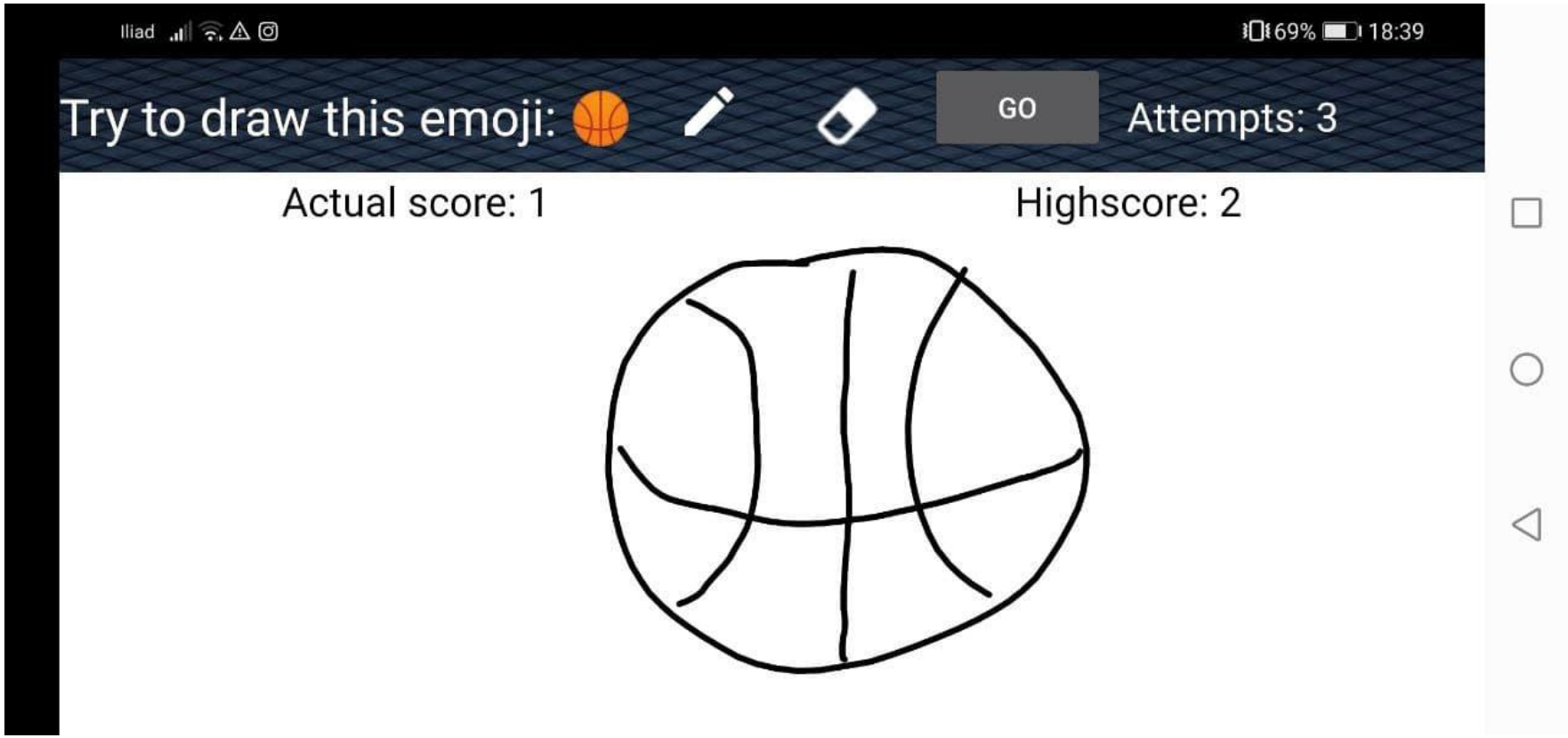
SaveManager

- Questa classe è responsabile delle operazioni di IO per una Whiteboard.
- Usa le api offerte da Gson per la lettura e scrittura di file Json
- E’ stata creata una classe MetadataAdapter per fornire un TypeAdapter a Gson per la scrittura in memoria di oggetti di tipo WhiteboardMetadata in formato Json

Play With Emoji

- Abbiamo realizzato un minigame chiamato Play With Emoji accessibile tramite Widget oppure tramite notifica. Il minigame consiste nel riuscire a disegnare una emoji suggerita dall'app
- Sfrutta l'oggetto Whiteboard e le classi relative alla digital ink recognition già descritte in precedenza
- E' stata realizzata una apposita Activity chiamata PlayWithEmojiActivity

PlayWithEmoji



Notifiche

- L’app invia una notifica all’utente invitandolo a giocare al minigame PlayWithEmoji dopo 24 ore dall’ultima volta che ha aperto il gioco
- La notifica viene inviata con intervallo regolare di 24 ore a meno che l’utente non apra il minigame
- A tale scopo sono state realizzate 2 classi :
  - NotificationBuilder responsabile della creazione della notifica
  - NotificationReceiver responsabile del lancio della notifica ogni 24 ore