ДОДАТОК А

Графічний матеріал атестаційної роботи

ДОДАТОК Б

Код застосунку

ExcelFileParser.kt

```
package com.march.nuretests.ui

import android.content.Context
import android.content.Intent
import android.database.Cursor
import android.net.Uri
import android.os.Environment
import android.util.Log
import androidx.core.net.toFile
import com.march.nuretests.model.Test
import org.apache.poi.hssf.usermodel.HSSFWorkbook
import org.apache.poi.ss.usermodel.Workbook
import org.apache.poi.ss.usermodel.WorkbookFactory
import org.apache.poi.xssf.usermodel.XSSFWorkbook
import java.net.URI
import android.provider.MediaStore
import android.os.ParcelFileDescriptor

import android.content.ContentProviderClient

import android.content.ContentResolver
import android.os.FileUtils
import com.march.nuretests.model.Answer
import com.march.nuretests.model.Question
import com.march.nuretests.model.QuestionType
import org.apache.poi.ss.usermodel.Sheet
import org.apache.poi.util.IOUtils
import org.apache.xmlbeans.impl.common.IOUtil
import java.io.*


class ExcelFileParser(private val context: Context) {
```

```
    fun parseExcelTestFile(uri: Uri): Test {
                                                val      inputStream      =
context.contentResolver.openInputStream(Uri.parse(uri.toString()))
                                               val      currentSheet      =
WorkbookFactory.create(inputStream).getSheetAt(0)
        val description = parseTestDescription(currentSheet)
        val topic = parseTestTopic(currentSheet)

        parseQuestions(currentSheet)

        return Test(description, topic, emptyList())
    }

    private fun parseTestDescription(sheet: Sheet): String {
        val firstRow = sheet.getRow(0)
        return firstRow.getCell(1).stringCellValue
    }

    private fun parseTestTopic(sheet: Sheet): String {
        val firstRow = sheet.getRow(0)
        return firstRow.getCell(3).stringCellValue
    }

    private fun parseQuestions(sheet: Sheet) {
        val questions = mutableListOf<Question>()
        var index = 1
        val lastRowNum = sheet.lastRowNum

        while (index != lastRowNum) {
            val currentRow = sheet.getRow(index)
            currentRow ?: break

            val description = currentRow.getCell(1).stringCellValue
            val answers = parseAnswers(sheet, ++index)
                                    questions.add(Question(description,
answers.getQuestionType(), answers, null))

            index += answers.size
        }
    }
```

```
    private fun parseAnswers(sheet: Sheet, rowNum: Int): List<Answer>
{
        var index = rowNum
        var isQuestion: Boolean
        val answers = mutableListOf<Answer>()
        var currentRow = sheet.getRow(index)

        do {
              val isCorrect = currentRow.getCell(1).stringCellValue ==
PLUS_SIGN
            val description = currentRow.getCell(2).stringCellValue

            answers.add(Answer(description, isCorrect))
            currentRow = sheet.getRow(++index)
            isQuestion = currentRow?.let {
                val rowValue = currentRow.getCell(1)?.stringCellValue
                      rowValue.isNullOrEmpty() || (rowValue != "+" &&
rowValue != "-")
            } ?: true
        } while (!isQuestion)

        return answers
    }

    private fun List<Answer>.getQuestionType(): QuestionType {
        val correctAnswers = this.filter { it.isCorrect }.size
         return when {
             correctAnswers > 1 -> QuestionType.CHECKBOX
             correctAnswers == 1 -> QuestionType.RADIO
             else -> QuestionType.TEXT
         }
    }

    companion object {
        private const val PLUS_SIGN = "+"
    }
}
```

HomeFragment.kt

```
package com.march.nuretests.ui

import android.os.Bundle
import android.view.View
import androidx.core.view.GravityCompat
import com.march.nuretests.databinding.FragmentHomeBinding

class HomeFragment : BaseFragment<FragmentHomeBinding>() {

    override fun createBinding(): FragmentHomeBinding {
        return FragmentHomeBinding.inflate(layoutInflater)
    }

        override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding?.menuIV?.setOnClickListener {
            binding?.drawer?.openDrawer(GravityCompat.START, true)
        }
    }

    companion object {
        fun newInstance(): HomeFragment {
            return HomeFragment()
        }
    }
}
```

LoadFileFragment.kt

```
package com.march.nuretests.ui

import android.Manifest
import android.os.Bundle
import android.util.Log
import android.view.View
import androidx.activity.result.contract.ActivityResultContracts
import com.march.nuretests.databinding.FragmentLoadFileBinding
import org.koin.androidx.viewmodel.ext.android.getViewModel
import android.content.pm.PackageManager

import android.os.Build
```

```
import android.app.Activity

import androidx.core.app.ActivityCompat
import java.lang.Exception


class LoadFileFragment: BaseFragment<FragmentLoadFileBinding>() {

    private lateinit var loadFileViewModel: LoadFileViewModel
                            private       val       openDocument       =
registerForActivityResult(ActivityResultContracts.OpenDocument())
{ uri ->
        Log.e("TAG", "uri: ${uri.path}")
        uri?.let {
            loadFileViewModel.parseFile(uri)
        }
    }

    override fun createBinding(): FragmentLoadFileBinding {
        return FragmentLoadFileBinding.inflate(layoutInflater)
    }

        override  fun  onViewCreated(view:  View,  savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        loadFileViewModel = getViewModel()
        if (!checkPermissionForReadExternalStorage()) {
            requestPermissionForReadExternalStorage()
        }

        binding?.loadButton?.setOnClickListener {
            if (checkPermissionForReadExternalStorage()) {
                openDocument.launch(excelMimes)
            } else {
                requestPermissionForReadExternalStorage()
            }
        }
    }

    private fun checkPermissionForReadExternalStorage(): Boolean {
```

```
        return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                                                    val    result    =
context?.checkSelfPermission(Manifest.permission.READ_EXTERNAL_STORAGE
)
            result == PackageManager.PERMISSION_GRANTED
        } else {
            true
        }
    }

    private fun requestPermissionForReadExternalStorage() {
        ActivityCompat.requestPermissions(
            requireActivity(),
            arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
            READ_EXTERNAL_STORAGE_PERMISSION_CODE
        )
    }

    companion object {
        private const val READ_EXTERNAL_STORAGE_PERMISSION_CODE = 1001

        fun newInstance(): LoadFileFragment {
            return LoadFileFragment()
        }

        val excelMimes = arrayOf(
            "application/vnd.ms-excel",
                                    "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet",
                                    "application/vnd.openxmlformats-
officedocument.spreadsheetml.template",
            "application/vnd.ms-excel.sheet.macroEnabled.12",
            "application/vnd.ms-excel.template.macroEnabled.12",
            "application/vnd.ms-excel.addin.macroEnabled.12",
            "application/vnd.ms-excel.sheet.binary.macroEnabled.12"
        )
    }
}
```

NureLogoView.kt

```
package com.march.nuretests.ui
import android.animation.ObjectAnimator
import android.content.Context
import android.graphics.*
import android.util.AttributeSet
import android.view.View
import android.view.animation.AccelerateDecelerateInterpolator
import androidx.core.animation.addListener
import com.march.nuretests.R
import kotlin.math.cos
import kotlin.math.min
import kotlin.math.sin

class NureLogoView : View {
    companion object {
        private const val DEFAULT_STROKE_WIDTH = 10f
        private const val DEFAULT_STROKE_COLOR = Color.WHITE
        private const val DEFAULT_ANIMATION_DURATION = 5000L
    }

    var animationEndListener: (() -> Unit)? = null
    private var radius = 0f
    private var paint = Paint(Paint.ANTI_ALIAS_FLAG)
    private var strokeWidth = DEFAULT_STROKE_WIDTH
    private var pathLength = 0f
    private var isProgressAnimation = false
    private var animationDuration = DEFAULT_ANIMATION_DURATION
    private val currentPoint = PointF()
    private val path = Path()
    private val linesInfo = ArrayList<Pair<Int, Float>?>()
       private val animator by lazy { ObjectAnimator.ofFloat(this,
"phase", 1.0f, 0.0f) }
    constructor(context: Context?) : this(context, null)
          constructor(context: Context?, attrs: AttributeSet?) :
this(context, attrs, 0)
     constructor(context: Context?, attrs: AttributeSet?, defStyleAttr:
Int) : super(
        context,
        attrs,
        defStyleAttr
    ) {
        init(attrs)
    }
    private fun startProgressAnimation() {
        animator.duration = animationDuration
        animator.repeatMode = ObjectAnimator.REVERSE
        animator.repeatCount = ObjectAnimator.INFINITE
        animator.interpolator = AccelerateDecelerateInterpolator()
        animator.start()
    }
    fun startAnimation() {
        animator.duration = animationDuration
```

```
        animator.interpolator = AccelerateDecelerateInterpolator()
        animator.start()
    }
    override fun onDraw(canvas: Canvas?) {
        super.onDraw(canvas)
        canvas?.drawPath(path, paint)
    }
      override fun onLayout(changed: Boolean, left: Int, top: Int,
right: Int, bottom: Int) {
        super.onLayout(changed, left, top, right, bottom)
        fillPath()

        if (isProgressAnimation) {
            startProgressAnimation()
        }
    }

    private fun init(attrs: AttributeSet?) {
        var strokeColor = DEFAULT_STROKE_COLOR

        attrs?.apply {
                val typedArray = context.obtainStyledAttributes(this,
R.styleable.NureLogoView)
            strokeWidth =
                typedArray.getDimension(R.styleable.NureLogoView_strok
e_width, DEFAULT_STROKE_WIDTH)
            isProgressAnimation =
                typedArray.getBoolean(R.styleable.NureLogoView_progres
s_animation, false)
            strokeColor =
                typedArray.getColor(R.styleable.NureLogoView_stroke_co
lor, DEFAULT_STROKE_COLOR)
            animationDuration =
                typedArray.getInteger(R.styleable.NureLogoView_animati
on_duration,
                    DEFAULT_ANIMATION_DURATION.toInt()).toLong()
            typedArray.recycle()
        }

        paint.color = strokeColor
        paint.strokeWidth = strokeWidth
        paint.style = Paint.Style.STROKE

                                        animator.addListener(onEnd     =
{ animationEndListener?.invoke() })

        fillLinesInfo()
    }

    private fun fillLinesInfo() {
        linesInfo.add(Pair(0, 0.36f))
        linesInfo.add(Pair(60, 0.77f))
```

```
        linesInfo.add(Pair(0, 0.6f))
        linesInfo.add(null)
        linesInfo.add(Pair(0, -0.5f))
        linesInfo.add(Pair(-55, -1.1f))
        linesInfo.add(Pair(0, 1.48f))
    }
    private fun fillPath() {
          radius = (min(measuredWidth, measuredHeight) - strokeWidth *
2) / 2f
        val centerPoint = PointF()
        val startRadianAngle = degreeToRadians(214)
        centerPoint.x = measuredWidth / 2f
        centerPoint.y = measuredHeight / 2f
                    currentPoint.x   =   centerPoint.x   +   (radius   *
cos(startRadianAngle))
                    currentPoint.y   =   centerPoint.y   +   (radius   *
sin(startRadianAngle))
        path.moveTo(currentPoint.x, currentPoint.y)
        for (line in linesInfo) {
            if (line != null) {
                updateCurrentPoint(line.first, radius * line.second)
                path.rLineTo(currentPoint.x, currentPoint.y)
            } else {
                drawBezier()
            }
        }

        drawCircle()

        val measure = PathMeasure(path, false)
        measure.setPath(path, true)
        pathLength = measure.length
    }

    private fun setPhase(phase: Float) {
        paint.pathEffect = createPathEffect(pathLength, phase)
        invalidate()
    }

      private fun createPathEffect(pathLength: Float, phase: Float):
PathEffect {
        return DashPathEffect(
            floatArrayOf(pathLength, pathLength),
            phase * pathLength
        )
    }

    private fun drawCircle() {
        val length = (2 * Math.PI * radius).toFloat() / 360f / 2f
        for (i in 0..360) {
            updateCurrentPoint(-60 - i, length)
            path.rLineTo(currentPoint.x, currentPoint.y)
```

```
            path.rLineTo(currentPoint.x, currentPoint.y)
        }
    }

    private fun drawBezier() {
        path.rCubicTo(
            (radius * 0.5f) / 0.75f / 2,
            0f,
            (radius * 0.5f) / 0.75f / 2,
            -radius * 0.5f,
            0f,
            -radius * 0.5f
        )
    }

    private fun updateCurrentPoint(angle: Int, length: Float) {
        val radianAngle = degreeToRadians(angle)
        currentPoint.x = length * cos(radianAngle)
        currentPoint.y = length * sin(radianAngle)
    }

    private fun degreeToRadians(degree: Int): Float {
        return (degree * Math.PI / 180f).toFloat()
    }
}
```

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮІК.505ХХХ.012 ПЗ	Пояснювальна записка	78 с.
		Графічні документи	
2		Слайд-презентація	21 слайд
		Носії інформації	
3		Компакт-диск (CD)	1 шт.

Змін.	Арк.	Номер докум.	Підп.	Дата	ГЮІК.505ХХХ.012 ВД			
Розроб.		Марчук С.С.		22.12	Система мобільного тестування знань на платформі Андроїд	Літ	Аркуш	Аркушів
Перевір.		Шкіль О.С.		22.12		У		
Н.контр.		Шкіль О.С.		22.12		ХНУРЕ Кафедра АПОТ		
Затв.		Чумаченко С.В.		22.12	Відомість кваліфікаційної роботи			