Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Розрахунково-графічна робота

з дисципліни
«Об'єктно орієнтоване програмування»

Виконав:
Студент групи ІМ-21
Сірик Максим Олександрович
номер у списку групи: 22

Перевірив:
Порєв Віктор Миколайович

Київ 2023

Зміст

1 Завдання:

Розробити програму для побудови графіків довільних функцій y = f(x)

2 Обгрунтування проектного рішення

Проектне рішення для створення програми для побудови графіків довільних функцій Ethereal Plot було прийняте на основі важливих факторів:

1. Простота
   Проект "Ethereal Plot"було розроблено для створення високоефективного та інтуїтивно зрозумілого інструменту для побудови графіків довільних функцій на платформі Android з використанням Jetpack Compose.
2. Функціональність
   (а) Масштабування та Зсув
   Зручний інтерфейс для зміни масштабу графіків та їх зсув для детальнішого вивчення областей інтересу.
   (б) Колірна Кодифікація
   Різні функції мають різний колір для зручності візуального розрізнення на графіку.
   (в) Збереження та Обмін
   Можливість зберігання графіків для майбутнього використання.

3 Аналіз можливих варіантів рішення для проекту

3.1 Графічний Двигун:

- *OpenGL ES:* Використання графічного двигуна, такого як OpenGL ES, може забезпечити високу ефективність у побудові графіків та взаємодії з ними. Однак це може призвести до складнішого коду та вищого рівня абстракції.

3.2 Альтернативні бібліотеки для побудови графіків:

- *MPAndroidChart:* Використання сторонньої бібліотеки, такої як MPAndroidChart, може надати широкий функціонал для побудови графіків з мінімальними зусиллями.

3.3 Мова програмування:

- *Kotlin:* Kotlin є офіційною мовою для розробки Android, і використання її підвищує читабельність коду та гарантує високий рівень безпеки.

3.4 Моделювання даних:

- *Flow ma LiveData:* Використання Flow та LiveData для реактивного програмування може спростити оновлення графіків під час зміни даних.

3.5 5. Збереження та обмін даними:

- *SQLite або Room:* Використання вбудованих інструментів для роботи з базами даних сприяє зручному збереженню та обміну збереженими графіками.

3.6 Тестування:

- *JUnit ma Espresso:* Використання JUnit для юніт-тестування та Espresso для інтеграційного тестування допомагає забезпечити стабільність та коректність функціоналу.

UI та UX:

- *Material Design:* Використання принципів Material Design дозволяє створити сучасний та інтуїтивно зрозумілий інтерфейс для користувачів.

Документація та Підтримка:

- *Dokka:* Використання Dokka для генерації документації сприяє кращому розумінню коду та полегшує процес розробки.

4 Пояснювальна записка

Програма Ethereal Plot, андроїд дотаток для побудови графіків.

4.1 Основні функції

Додаток може малювати різноманітні функції, окрім тих, що мають вертикальні асимптоти.
Додаток має можливість зміни кольору, відключення та видалення функцій.
Додаток підтримує material ui 3, і є зручним та інтуїтивним

5 Теоретичні положення

5.1 Jetpack Compose

Посилання на документацію, в якій описані всі положення

5.2 Мова Програмування Kotlin

Посилання на документацію, в якій описані всі положення

5.3 Реактивне Програмування

Посилання на документацію, в якій описані всі положення

5.4 Архітектурні Підходи

Посилання на документацію, в якій описані всі положення

6 Текст програми:

6.1 Module: com.github.erotourtes.utils

Лістинг 1: MathParserTest.kt

```
package com.github.erotourtes.utils

import org.junit.jupiter.api.Assertions.*
import org.junit.jupiter.api.DisplayName
import org.junit.jupiter.api.Nested
```

```
import org.junit.jupiter.api.Test

class MathParserTest {
    @Test
    @DisplayName("Test double input")
    fun doubleInput() {
        val parser = MathParser("1.0+2.0")
        assertEquals("1.02.0+", parser.rpn.joinToString(""))
    }

    @Nested
    @DisplayName("Test simple operations")
    inner class SimpleOperations {
        @Test
        @DisplayName("[+, -]")
        fun simple1() {
            val parser = MathParser("1 + 2 - 3")
            assertEquals("12+3-", parser.rpn.joinToString(""))
        }

        @Test
        @DisplayName("[+, -, *, /]")
        fun simple2() {
            val parser = MathParser("1 + 2 - 3 * 4 / 5")
            assertEquals("12+34*5/-", parser.rpn.joinToString(""))
        }

        @Test
        @DisplayName("[+, -, *, /, ^]")
        fun simple3() {
            val parser = MathParser("1 + 2 - 3 * 4 / 5 ^ 6 - 3")
            assertEquals("12+34*56^/-3-", parser.rpn.joinToString(""))
        }

        @Test
        @DisplayName("[+, -, *, /, ^] with brackets")
        fun simple4() {
            val parser = MathParser("(1 + 2 - 3) * 4 / 5 ^ 6 - 3")
            assertEquals("12+3-4*56^/3-", parser.rpn.joinToString(""))
        }
    }

    @Nested
    @DisplayName("Test functions")
    inner class FnOperations {
        @Test
        @DisplayName("simple sin expression")
        fun fn1() {
            val parser = MathParser("sin(30) * 4 + 3")
            assertEquals("30sin4*3+", parser.rpn.joinToString(""))
        }

        @Test
        @DisplayName("all operators")
```

```
        fun fn2() {
            val parser = MathParser("sin(30) * 4 + cos(31) - (3 - 2) / 3 ^ (4 + 5)")
            assertEquals("30sin4*31cos+32-345+^/-", parser.rpn.joinToString(""))
        }
    }

    @Test
    @DisplayName("Test with variables")
    fun variables1() {
        val parser = MathParser("x + 2")
        parser.setVariable("x", 3.0)
        assertEquals("x2+", parser.rpn.joinToString(""))
        assertEquals(5.0, parser.eval())
    }

    @Nested
    @DisplayName("Test errors")
    inner class Errors {
        @Test
        @DisplayName("constructing invalid expression")
        fun wrongExpression() {
            assertDoesNotThrow {
                MathParser("3+")
                MathParser("+3")
                MathParser(")3+")
                MathParser("3 (^ +")
            }
        }

        @Test
        @DisplayName("evaluating invalid expression")
        fun wrongExpression2() {
            val parser = MathParser("3+")

            assertThrows(IllegalStateException::class.java) {
                parser.eval()
            }
        }

        @Test
        @DisplayName("evaluating invalid expression")
        fun wrongExpression3() {
            val parser = MathParser(") 5 + 3")

            assertThrows(IllegalStateException::class.java) {
                parser.eval()
            }
        }
    }
}
```

Лістинг 2: Ext.kt

```
package com.github.erotourtes.utils
```

```
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.Rect
import androidx.compose.ui.graphics.Color
import androidx.core.graphics.withSave
import com.github.erotourtes.model.PlotUIState
import com.github.erotourtes.data.plot.Plot
import kotlin.random.Random

/**
 * Draws text by reflecting it in the X axis.
 * Useful for drawing text in the Cartesian coordinate system.
 */
fun Canvas.drawTextInRightDirection(text: String, x: Float, y: Float, paint: Paint) {
    withSave {
        val textBoundaries = Rect().apply { paint.getTextBounds(text, 0, text.length, this)
        translate(x - textBoundaries.width() / 2, y - textBoundaries.height() / 2)
        scale(1f, -1f)
        drawText(text, 0f, 0f, paint)
    }
}

inline fun Paint.withColor(c: Int, block: Paint.() -> Unit) {
    color = color.also {
        color = c
        block()
    }
}

fun Color.Companion.random(): Color {
    return Color(
        red = Random.nextFloat(),
        green = Random.nextFloat(),
        blue = Random.nextFloat(),
        alpha = 1f
    )
}

fun Plot.toPlotUIState() = PlotUIState(
    color = Color(color),
    function = function,
    isVisible = isVisible,
    isValid = isValid,
    id = id,
)
```

Лістинг 3: MathParser.kt

```
package com.github.erotourtes.utils

import java.lang.IllegalStateException
import kotlin.math.*

class MathParser(expression: String) {
    private val binaryOperators = mapOf(
```

```
        "+" to Pair(1) { a: Double, b: Double -> a + b },
        "-" to Pair(1) { a: Double, b: Double -> a - b },
        "*" to Pair(2) { a: Double, b: Double -> a * b },
        "/" to Pair(2) { a: Double, b: Double -> a / b },
        "^" to Pair(3) { a: Double, b: Double -> a.pow(b) },
    )

    private val fnOperators = mapOf(
        "sin" to Pair(4) { a: Double -> sin(a) },
        "cos" to Pair(4) { a: Double -> cos(a) },
        "tan" to Pair(4) { a: Double -> tan(a) },
        "ln" to Pair(4) { a: Double -> ln(a) },
        "sqrt" to Pair(4) { a: Double -> sqrt(a) },
    )

    private val brackets = listOf("(", ")")

    val rpn = toRpnOrEmpty(expression)

    private val variables = mutableMapOf("x" to 0.0)

    fun setVariable(name: String, value: Double): MathParser {
        variables[name] = value
        return this
    }

    inline fun evalOrNull(block: (Double) -> Double = { it }): Double? {
        return try {
            block(eval())
        } catch (e: IllegalStateException) {
            null
        }
    }

    fun eval(): Double {
        if (rpn.isEmpty()) throw IllegalStateException("Invalid␣expression")
        try {
            return _eval()
        } catch (e: Exception) {
            throw IllegalStateException("Invalid␣expression")
        }
    }

    @Throws(NoSuchElementException::class)
    private fun _eval(): Double {
        val stack = mutableListOf<Double>()
        for (token in rpn) {
            when (token) {
                in binaryOperators -> {
                    val b = stack.removeLast()
                    val a = stack.removeLast()
                    stack.add(binaryOperators[token]!!.second(a, b))
                }
```

```
                in fnOperators -> {
                    val a = stack.removeLast()
                    stack.add(fnOperators[token]!!.second(a))
                }

                in variables -> stack.add(variables[token]!!)
                else -> stack.add(token.toDouble())
            }
        }
        return stack.removeLast()
    }

    private fun toRpnOrEmpty(exp: String): List<String> = runCatching {
        toRPN(exp)
    }.getOrDefault(emptyList())

    /**
     * Reverse Polish Notation
     * https://en.wikipedia.org/wiki/Reverse_Polish_notation
     * */
    @Throws(NoSuchElementException::class)
    private fun toRPN(exp: String): List<String> {
        val tokens = tokenize(exp)

        val operationStack = mutableListOf<String>()
        val queue = mutableListOf<String>()

        for (token in tokens) {
            when (token) {
                in binaryOperators -> handleBinaryOp(operationStack, token, queue)
                in fnOperators -> operationStack.add(token)
                "(" -> operationStack.add(token)
                ")" -> handleClosingBracket(operationStack, queue)
                else -> queue.add(token)
            }
        }

        while (operationStack.isNotEmpty()) queue.add(operationStack.removeLast())

        return queue
    }

    private fun handleClosingBracket(
        operationStack: MutableList<String>,
        queue: MutableList<String>
    ) {
        while (operationStack.last() != "(") queue.add(operationStack.removeLast())
        operationStack.removeLast() // remove "("

        if (operationStack.lastOrNull() in fnOperators)
            queue.add(operationStack.removeLast())
    }

    private fun handleBinaryOp(
```

```
        operationStack: MutableList<String>,
        token: String,
        queue: MutableList<String>
    ) {
        while (operationStack.lastOrNull() in binaryOperators) {
            val lastPrecedence = binaryOperators[operationStack.last()]!!.first
            val curPrecedence = binaryOperators[token]!!.first
            if (curPrecedence > lastPrecedence) break

            queue.add(operationStack.removeLast())
        }
        operationStack.add(token)
    }

    private fun tokenize(expression: String): List<String> {
        var exp = expression
        // 3+log(2, 3) -> 3 + log (2 , 3)
        for (operator in binaryOperators.keys) exp = exp.replace(operator, " $operator ")

        // sin(2) -> sin ( 2 )
        for (bracket in brackets) exp = exp.replace(bracket, " $bracket ")

        // 3x -> 3 * x
        exp = exp.replace(Regex("(\\d+)([a-zA-Z]+)"), "$1 * $2")

        return exp.split("\\s+".toRegex()).filter { it.isNotBlank() }
    }
}
```

6.2 Module: com.github.erotourtes.data

Лістинг 4: AppContainerImpl.kt

```
package com.github.erotourtes.data

import android.content.Context
import androidx.room.Room
import com.github.erotourtes.data.plot.PlotRepository


interface AppContainer {
    val plotRepository: PlotRepository
}

class AppContainerImpl(private val applicationContext: Context) : AppContainer {
    private val db by lazy {
        Room.databaseBuilder(
            applicationContext, EtherealPlotDatabase::class.java, "plot-database"
        ).build()
    }

    // TODO: use interface
    override val plotRepository: PlotRepository by lazy {
        PlotRepository(plotDao = db.dao)
    }
```

```
}
```

Лістинг 5: EtherealPlotDatabase.kt

```
package com.github.erotourtes.data

import androidx.room.Database
import androidx.room.RoomDatabase
import com.github.erotourtes.data.plot.Plot
import com.github.erotourtes.data.plot.PlotDao

@Database(
    entities = [Plot::class],
    version = 1,
)
abstract class EtherealPlotDatabase : RoomDatabase() {
    abstract val dao: PlotDao
}
```

6.3 Module: com.github.erotourtes.ui

Лістинг 6: EtherealPlotApp.kt

```
package com.github.erotourtes.ui

import androidx.compose.runtime.Composable
import androidx.lifecycle.viewmodel.compose.viewModel
import com.github.erotourtes.data.AppContainer
import com.github.erotourtes.model.PlotViewModel
import com.github.erotourtes.ui.theme.AppTheme

@Composable
fun EtherealPlotApp(
    appContainer: AppContainer,
) {
    val plotViewModel: PlotViewModel = viewModel(factory = PlotViewModel.provideFactory(ap

    AppTheme {
        NavGraph(plotViewModel = plotViewModel)
    }
}
```

Лістинг 7: NavigationGraph.kt

```
package com.github.erotourtes.ui

import androidx.compose.runtime.Composable
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.github.erotourtes.model.PlotViewModel
import com.github.erotourtes.ui.screen.main.MainScreen
import com.github.erotourtes.ui.screen.canvas.CanvasScreen

sealed class Screen(val route: String) {
    object MainScreen : Screen("main")
```

```
    object CanvasScreen : Screen("canvas")
}

@Composable
fun NavGraph(
    plotViewModel: PlotViewModel
) {
    val navController = rememberNavController()

    NavHost(navController = navController, startDestination = Screen.MainScreen.route) {
        composable(Screen.MainScreen.route) {
            MainScreen(
                plotViewModel = plotViewModel,
                navController = navController
            )
        }

        composable(Screen.CanvasScreen.route) {
            CanvasScreen(
                plotViewModel = plotViewModel,
                navController = navController
            )
        }
    }
}
```

6.4 Module: com.github.erotourtes.ui.screen.main

Лістинг 8: Main.kt

```
package com.github.erotourtes.ui.screen.main

import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.navigation.NavController
import com.github.erotourtes.R
import com.github.erotourtes.model.PlotViewModel
import com.github.erotourtes.ui.Screen
import com.github.erotourtes.ui.screen.main.data.QuickFunction


@Composable
fun MainScreen(
    plotViewModel: PlotViewModel, navController: NavController
) {
    MainLayout(
        onGoToPlots = {
            plotViewModel.removeAllPlots()
```

```
            navController.navigate(Screen.CanvasScreen.route)
        },
        onGoToPlotsPreviousSession = {
            plotViewModel.restorePreviousSession()
            navController.navigate(Screen.CanvasScreen.route)
        },
        onGoToPlotsWithPlot = {
            plotViewModel.createNewSync(function = it.formula)
            navController.navigate(Screen.CanvasScreen.route)
        },
    )
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MainLayout(
    onGoToPlots: () -> Unit,
    onGoToPlotsPreviousSession: () -> Unit,
    onGoToPlotsWithPlot: (QuickFunction) -> Unit,
) {
    Scaffold(topBar = {
        CenterAlignedTopAppBar(title = {
            Text(
                text = stringResource(R.string.app_name),
                style = MaterialTheme.typography.headlineLarge,
            )
        })
    }, floatingActionButton = {
        FloatingActionButton(onClick = onGoToPlots, content = {
            Icon(
                imageVector = Icons.Default.Add,
                contentDescription = "Create␣new␣plots",
            )
        })
    }, content = {
        Column(
            horizontalAlignment = Alignment.CenterHorizontally, modifier = Modifier.padding
        ) {
            QuickFunctionAction(
                onQuickFunctionClick = onGoToPlotsWithPlot
            )
            Button(onClick = onGoToPlotsPreviousSession) {
                Text(text = "Restore␣previous␣session")
            }
        }
    })
}

@Preview
@Composable
private fun Preview() {
    MainLayout({}, {}, {})
}
```

Лістинг 9: QuickFunctionAction.kt

```kotlin
package com.github.erotourtes.ui.screen.main

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.material3.Card
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import com.github.erotourtes.ui.screen.main.data.QuickFunction
import com.github.erotourtes.ui.screen.main.data.quickFunctionList
import com.github.erotourtes.ui.theme.spacing

@Composable
fun QuickFunctionAction(
    functionList: List<QuickFunction> = quickFunctionList,
    onQuickFunctionClick: (QuickFunction) -> Unit,
) {
    Column {
        Text(
            text = "Quick Function Action",
            style = MaterialTheme.typography.headlineMedium,
            modifier = Modifier.padding(MaterialTheme.spacing.large)
        )

        LazyVerticalGrid(
            columns = GridCells.Adaptive(128.dp),
            contentPadding = PaddingValues(MaterialTheme.spacing.medium),
            verticalArrangement = Arrangement.spacedBy(10.dp),
            horizontalArrangement = Arrangement.spacedBy(10.dp),
        ) {
            items(functionList.size) { index ->
                val function = functionList[index]
                QuickFunctionItem(function = function, modifier = Modifier.clickable {
                    onQuickFunctionClick(function)
                })
            }
        }
    }
}

@Composable
fun QuickFunctionItem(
    function: QuickFunction,
    modifier: Modifier = Modifier,
) {
    Card(modifier) {
```

```
            Column(
                modifier = Modifier.padding(MaterialTheme.spacing.large)
            ) {
                Row(
                    verticalAlignment = Alignment.CenterVertically,
                ) {
                    Icon(
                        imageVector = function.icon, contentDescription = function.name, modif
                    )
                    Text(
                        text = function.name,
                        style = MaterialTheme.typography.headlineSmall,
                    )
                }
                Text(
                    text = function.formula,
                    modifier = Modifier.fillMaxWidth(),
                    textAlign = TextAlign.Center,
                    style = MaterialTheme.typography.bodyLarge,
                )
            }
        }
}
```

6.5 Module: com.github.erotourtes.ui.screen.canvas

Лістинг 10: CanvasScreen.kt

```
package com.github.erotourtes.ui.screen.canvas

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.CornerSize
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import com.github.erotourtes.ui.screen.canvas.drawing.CanvasView
import com.github.erotourtes.model.PlotUIState
import com.github.erotourtes.model.PlotViewModel
import com.github.erotourtes.model.mockPlots
import com.github.erotourtes.ui.theme.AppTheme
import com.github.erotourtes.ui.theme.spacing


@Composable
```

```
fun CanvasScreen(
    plotViewModel: PlotViewModel,
    navController: NavController,
) {
    // TODO: brainstorm this
    val plotState by plotViewModel.plotUIState.collectAsState()

    CanvasLayout(plotState = plotState,
        onPlotFormulaChange = plotViewModel::changePlotFormulaSync,
        onPlotHideStateChange = plotViewModel::changeHideState,
        onPlotRemove = plotViewModel::removePlotSync,
        onPlotColorChange = plotViewModel::changeColor,
        onPlotNotValid = { plotViewModel.changePlotValidity(it, false) },
        onPlotCreate = { plotViewModel.createNewSync() },
        onBackPressed = { navController.popBackStack() })
}

val BOTTOM_SHEET_SCAFFOLD_HEIGHT = 56.dp

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CanvasLayout(
    plotState: List<PlotUIState>,
    onPlotFormulaChange: (PlotUIState, String) -> Unit,
    onPlotHideStateChange: (PlotUIState, Boolean) -> Unit,
    onPlotRemove: (PlotUIState) -> Unit,
    onPlotColorChange: (PlotUIState, Color) -> Unit,
    onPlotNotValid: (PlotUIState) -> Unit,
    onPlotCreate: () -> Unit,
    onBackPressed: () -> Unit,
    modifier: Modifier = Modifier,
) {
    val scaffoldState = rememberBottomSheetScaffoldState()
    BottomSheetScaffold(
        sheetPeekHeight = BOTTOM_SHEET_SCAFFOLD_HEIGHT,
        scaffoldState = scaffoldState,
        sheetContent = {
            PlotsView(
                fns = plotState,
                onPlotFormulaChange = onPlotFormulaChange,
                onPlotVisibilityChange = onPlotHideStateChange,
                onPlotRemove = onPlotRemove,
                onPlotColorChange = onPlotColorChange,
                onPlotCreate = onPlotCreate,
                modifier = Modifier
                    .padding(MaterialTheme.spacing.medium)
                    .defaultMinSize(minHeight = 600.dp)
            )
        },
        sheetShape = MaterialTheme.shapes.large.copy(bottomStart = CornerSize(0), bottomEnd
        sheetContentColor = MaterialTheme.colorScheme.onSurface,
    ) {
        CanvasView(plotState, onPlotNotValid, modifier.padding(bottom = BOTTOM_SHEET_SCAFF
        Button(
```

```
            onClick = onBackPressed,
            shape = CircleShape,
            contentPadding = PaddingValues(0.dp),
            modifier = Modifier
                .padding(MaterialTheme.spacing.medium)
                .size(48.dp)

        ) {
            Icon(
                imageVector = Icons.Default.ArrowBack,
                contentDescription = "Back to main screen",
            )
        }
    }
}

@Preview(
    showBackground = true, name = "Home Preview"
)
@Composable
private fun CanvasLayoutPreview() {
    AppTheme {
        CanvasLayout(plotState = mockPlots,
            onPlotFormulaChange = { _, _ -> },
            onPlotHideStateChange = { _, _ -> },
            onPlotRemove = { },
            onPlotColorChange = { _, _ -> },
            onPlotNotValid = { _ -> },
            onPlotCreate = {},
            onBackPressed = {})
    }
}
```

Лістинг 11: ColorPickerDialog.kt

```
package com.github.erotourtes.ui.screen.canvas

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.github.erotourtes.ui.theme.AppTheme
import com.github.erotourtes.ui.theme.spacing
import com.github.skydoves.colorpicker.compose.AlphaTile
import com.github.skydoves.colorpicker.compose.ColorEnvelope
import com.github.skydoves.colorpicker.compose.HsvColorPicker
import com.github.skydoves.colorpicker.compose.rememberColorPickerController

private val PICKER_SIZE = 300.dp
```

```
@Composable
fun ColorPickerScreen(
    initialColor: Color,
    onColorChange: (Color) -> Unit,
    modifier: Modifier = Modifier,
    onBackPress: () -> Unit = {},
) {
    val controller = rememberColorPickerController()

    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = modifier
    ) {
        HsvColorPicker(
            modifier = Modifier
                .width(PICKER_SIZE)
                .height(PICKER_SIZE),
            controller = controller,
            onColorChanged = { colorEnvelope: ColorEnvelope ->
                if (colorEnvelope.fromUser)
                    onColorChange(colorEnvelope.color)
            },
            initialColor = initialColor
        )

        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(MaterialTheme.spacing.medium),
            verticalAlignment = Alignment.CenterVertically,
        ) {
            AlphaTile(
                modifier = Modifier
                    .size(80.dp)
                    .clip(RoundedCornerShape(MaterialTheme.spacing.large)),
                controller = controller
            )

            Button(
                onClick = { onBackPress() },
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(MaterialTheme.spacing.medium),
            ) {
                Text(text = "Cancel")
            }
        }
    }
}

@Preview(
```

```
    showBackground = true,
)
@Composable
private fun ColorPickerPreview() {
    AppTheme {
        Surface(color = MaterialTheme.colorScheme.error) {
            ColorPickerScreen(
                initialColor = MaterialTheme.colorScheme.primary,
                onColorChange = { },
            )
        }
    }
}
```

Лістинг 12: PlotView.kt

```
package com.github.erotourtes.ui.screen.canvas

import androidx.compose.animation.core.animateDpAsState
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Close
import androidx.compose.material.icons.filled.KeyboardArrowRight
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.draw.scale
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import com.github.erotourtes.model.PlotUIState
import com.github.erotourtes.model.mockPlots
import com.github.erotourtes.ui.theme.AppTheme
import com.github.erotourtes.ui.theme.spacing
import com.github.erotourtes.ui.utils.ExpandableCard
import com.github.erotourtes.ui.utils.SwapToReveal

@Composable
fun PlotsView(
    fns: List<PlotUIState>,
    onPlotFormulaChange: (PlotUIState, String) -> Unit,
    onPlotVisibilityChange: (PlotUIState, Boolean) -> Unit,
    onPlotRemove: (PlotUIState) -> Unit,
    onPlotColorChange: (PlotUIState, Color) -> Unit,
    onPlotCreate: () -> Unit,
    modifier: Modifier = Modifier,
) {
    Box {
        LazyColumn(modifier = modifier.fillMaxWidth()) {
```

```
            items(fns, PlotUIState::id) { fn ->
                OpenablePlotView(
                    fn = fn,
                    onPlotRemove = onPlotRemove,
                    onPlotVisibilityChange = onPlotVisibilityChange,
                    onPlotFormulaChange = onPlotFormulaChange,
                    onPlotColorChange = onPlotColorChange,
                )
            }

            item {
                Button(onClick = onPlotCreate, modifier = Modifier.fillMaxWidth()) {
                    Text("Add plot")
                }
            }
        }
    }
}

@Composable
private fun OpenablePlotView(
    fn: PlotUIState,
    onPlotRemove: (PlotUIState) -> Unit,
    onPlotVisibilityChange: (PlotUIState, Boolean) -> Unit,
    onPlotFormulaChange: (PlotUIState, String) -> Unit,
    onPlotColorChange: (PlotUIState, Color) -> Unit,
) {
    var isExpanded by remember { mutableStateOf(false) }
    var isRemoving by remember { mutableStateOf(false) }
    var prevColor by remember { mutableStateOf(fn.color) }

    val height by animateDpAsState(
        targetValue = if (isRemoving) 0.dp else Dp.Unspecified,
        label = "Height animation",
        finishedListener = {
            onPlotRemove(fn)
        })

    ExpandableCard(
        modifier = Modifier.height(height),
        expandableContent = {
            ColorPickerScreen(
                initialColor = fn.color,
                onColorChange = {
                    prevColor = fn.color
                    onPlotColorChange(fn, it)
                },
                onBackPress = {
                    onPlotColorChange(fn, prevColor)
                    isExpanded = !isExpanded
                },
                modifier = Modifier.padding(horizontal = MaterialTheme.spacing.medium)
            )
        }, expanded = isExpanded
```

```
    ) {
        SwappablePlotView(
            fn = fn,
            onPlotRemove = { isExpanded = !isExpanded; isRemoving = true },
            onPlotVisibilityChange = onPlotVisibilityChange,
            onPlotFormulaChange = onPlotFormulaChange,
            onPlotColorChangeRequest = { isExpanded = !isExpanded },
        )
    }
    Spacer(modifier = Modifier.height(MaterialTheme.spacing.medium))
}

@Composable
private fun SwappablePlotView(
    fn: PlotUIState,
    onPlotRemove: (PlotUIState) -> Unit,
    onPlotVisibilityChange: (PlotUIState, Boolean) -> Unit,
    onPlotFormulaChange: (PlotUIState, String) -> Unit,
    onPlotColorChangeRequest: () -> Unit,
    modifier: Modifier = Modifier,
) {
    // Another solution would be to use SwipeToDismiss
    SwapToReveal(
        onRemove = { onPlotRemove(fn) },
        hiddenContent = {
            PlotControls(
                isVisible = fn.isVisible,
                onPlotRemove = { onPlotRemove(fn) },
                onPlotVisibilityChange = { onPlotVisibilityChange(fn, it) },
            )
        },
    ) {
        PlotView(
            fn = fn,
            onPlotFormulaChange = { onPlotFormulaChange(fn, it) },
            onPlotColorChangeRequest = onPlotColorChangeRequest,
            modifier = modifier,
        )
    }
}

private val MATERIAL_INPUT_HEIGHT = 50.dp
private val MATERIAL_COLOR_PICKER_WIDTH = 30.dp

@Composable
fun PlotView(
    fn: PlotUIState,
    onPlotFormulaChange: (String) -> Unit,
    onPlotColorChangeRequest: () -> Unit,
    modifier: Modifier = Modifier,
) {
    Row(
        modifier = modifier
            .height(MATERIAL_INPUT_HEIGHT)
```

```
                .fillMaxWidth(),
        ) {
            Box(
                modifier = Modifier
                    .background(MaterialTheme.colorScheme.primary)
                    .size(MATERIAL_INPUT_HEIGHT)
            ) {
                Icon(
                    imageVector = Icons.Default.KeyboardArrowRight,
                    contentDescription = "Swipe to reveal",
                    tint = MaterialTheme.colorScheme.onPrimary,
                )
            }
            TextField(
                value = fn.function,
                onValueChange = onPlotFormulaChange,
                textStyle = MaterialTheme.typography.bodyMedium,
                label = { Text("Function") },
                modifier = Modifier
                    .fillMaxWidth()
                    .weight(1f),
                singleLine = true,
            )
            Box(modifier = Modifier
                .fillMaxHeight()
                .width(MATERIAL_COLOR_PICKER_WIDTH)
                .background(fn.color)
                .clickable { onPlotColorChangeRequest() })
        }
    }

    @Composable
    fun PlotControls(
        isVisible: Boolean,
        onPlotRemove: () -> Unit,
        onPlotVisibilityChange: (Boolean) -> Unit,
    ) {
        Row {
            IconButton(
                onClick = onPlotRemove,
                modifier = Modifier
                    .background(MaterialTheme.colorScheme.primary)
                    .height(MATERIAL_INPUT_HEIGHT)
            ) {
                Icon(
                    imageVector = Icons.Default.Close,
                    contentDescription = "Close",
                    tint = MaterialTheme.colorScheme.onPrimary,
                )
            }
            Spacer(modifier = Modifier.width(MaterialTheme.spacing.small))
            Switch(
                checked = isVisible, onCheckedChange = onPlotVisibilityChange, modifier = Modi
                    .scale(0.8f)
```

```
                .rotate(-90f)
            )
            Spacer(modifier = Modifier.width(MaterialTheme.spacing.small))
        }
    }

    @Preview(
        showBackground = true, name = "PlotView Preview"
    )
    @Composable
    private fun PlotViewPreview() {
        AppTheme {
            PlotsView(
                fns = mockPlots,
                onPlotFormulaChange = { _, _ -> },
                onPlotVisibilityChange = { _, _ -> },
                onPlotRemove = { },
                onPlotColorChange = { _, _ -> },
                onPlotCreate = { },
            )
        }
    }
```

6.6 Module: com.github.erotourtes.ui.screen.main.data

Лістинг 13: QuickFunction.kt

```
package com.github.erotourtes.ui.screen.main.data

import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.rounded.Search
import androidx.compose.ui.graphics.vector.ImageVector
import com.github.erotourtes.model.PlotUIState

data class QuickFunction(
    val name: String,
    val formula: String,
    val icon: ImageVector,
)

val quickFunctionList = listOf(
    QuickFunction(
        name = "Linear",
        formula = "x",
        icon = Icons.Rounded.Search
    ),
    QuickFunction(
        name = "Quadratic",
        formula = "x^2",
        icon = Icons.Rounded.Search
    ),
    QuickFunction(
        name = "Cubic",
        formula = "x^3",
        icon = Icons.Rounded.Search
```

```
        ),
        QuickFunction(
            name = "Sine",
            formula = "sin(x)",
            icon = Icons.Rounded.Search
        ),
        QuickFunction(
            name = "Cosine",
            formula = "cos(x)",
            icon = Icons.Rounded.Search
        ),
        QuickFunction(
            name = "Tangent",
            formula = "tan(x)",
            icon = Icons.Rounded.Search
        ),
        QuickFunction(
            name = "Logarithmic",
            formula = "ln(x)",
            icon = Icons.Rounded.Search
        ),
        QuickFunction(
            name = "Square␣Root",
            formula = "sqrt(x)",
            icon = Icons.Rounded.Search
        ),
        QuickFunction(
            name = "Exponential",
            formula = "x^x",
            icon = Icons.Rounded.Search
        )
)
```

6.7 Module: com.github.erotourtes.ui.screen.canvas.drawing

Лістинг 14: CanvasNativeView.kt

```kotlin
package com.github.erotourtes.ui.screen.canvas.drawing

import android.annotation.SuppressLint
import android.content.Context
import android.graphics.*
import android.util.AttributeSet
import android.view.MotionEvent
import android.view.ScaleGestureDetector
import android.view.ScaleGestureDetector.SimpleOnScaleGestureListener
import android.view.View
import android.view.View.OnTouchListener
import androidx.compose.ui.graphics.toArgb
import androidx.core.graphics.*
import com.github.erotourtes.model.PlotUIState
import com.github.erotourtes.utils.MathParser
import com.github.erotourtes.utils.drawTextInRightDirection
import com.github.erotourtes.utils.withColor
import kotlin.math.absoluteValue
```

```
const val PIXELS_PER_UNIT = 100

data class Colors(
    val axes: Int,
    val bg: Int,
    val text: Int,
)

@SuppressLint("ClickableViewAccessibility")
class CanvasViewNativeView @JvmOverloads constructor(
    context: Context, attrs: AttributeSet? = null, defStyleAttr: Int = 0
) : View(context, attrs, defStyleAttr) {
    private val matrixCamera = Matrix()
    private val matrixCartesian = Matrix()

    private var scaleFactor = 1f

    private var curStepMultiplier = 1f
    private var prevScaleFactor = 1f

    private lateinit var canvas: Canvas

    private lateinit var colors: Colors
    private var fns: List<PlotUIState> = emptyList()
    private val cachedParsers: MutableMap<String, MathParser> = HashMap()

    private var onPlotNotValid: ((PlotUIState) -> Unit)? = null

    // The default color of paint is Colors::axes
    private val paint = Paint().apply {
        isAntiAlias = true
        style = Paint.Style.FILL
        strokeWidth = 10f
        textSize = 50f
    }

    private val scaleGestureDetector = initScaleGestureDetector(context)

    init {
        val listeners = listOf(translateCameraListener(), scaleCameraListener())

        setOnTouchListener { _, event ->
            listeners.forEach { it.onTouch(this, event) }
            true
        }
    }

    override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
        matrixCartesian.apply {
            postScale(1f, -1f) // invert Y axis
            postTranslate(w / 2f, h / 2f) // center the origin
        }
    }
```

```
override fun onDraw(canvas: Canvas) {
    super.onDraw(canvas)
    this.canvas = canvas

    canvas.withSave {
        concat(matrixCamera)
        concat(matrixCartesian)
        scale(scaleFactor, scaleFactor)

        drawBG()
        drawGrid()
        drawAxis()
        drawFns()
    }
}

private fun drawBG() {
    paint.withColor(colors.bg) {
        canvas.drawRect(canvas.clipBounds, paint)
    }
}

private fun drawFns() {
    // TODO: optimise using coroutines
    val invalidatedFns = mutableListOf<PlotUIState>()
    fns.filter { it.isValid && it.isVisible }.forEach {
        val parser = cachedParsers.getOrPut(it.function) {
            MathParser(it.function)
        }
        paint.withColor(it.color.toArgb()) {
            val isSuccess =
                drawFn(parser)/* Can't call onPlotNotValid directly because it will thr
            if (!isSuccess) invalidatedFns.add(it)
        }
    }

    invalidatedFns.forEach { onPlotNotValid?.invoke(it) }
}

/**
 * Draws a function on the canvas.
 * @param fn the function to draw
 * @return true if the function was drawn successfully, false otherwise
 * */
private fun drawFn(fn: MathParser): Boolean {
    var (left, top, right, bottom) = canvas.clipBounds
    top = bottom.also { bottom = top }
    if (top < bottom) throw IllegalArgumentException("top␣<␣bottom")

    val step = 1f * curStepMultiplier
    var xCur = left.toDouble()
    val xEnd = right.toDouble()
```

```
        var yCur = fn.evalInPixels(xCur) ?: return false

        while (xCur < xEnd) {
            val xNext = xCur + step

            val yNext = fn.evalInPixels(xNext) ?: return false
            val yNextNext = fn.evalInPixels(xNext + step) ?: return false

            // check for asymptote
            // TODO: optimise and make it work for all functions
            // Currently it has a bug with tan(x)
            val dy = yNext - yCur
            val tan = dy / step
            val isAsymptote = tan.absoluteValue > 1000

            val tanNext = (yNextNext - yNext) / (step)

            if (isAsymptote && tan * tanNext < 0) {
                xCur = xNext
                yCur = yNext
                continue
            } else if (isAsymptote) {
                val isToDown = tan > tanNext;
                val yAsymptote = if (isToDown) bottom else top

//                Log.i(
//                    "CanvasViewNativeView",
//                    "$tan $tanNext Asymptote at x = ${xCur / PIXELS_PER_UNIT}, y = ${yCur
//                )

                canvas.drawLine(
                    xCur.toFloat(), yCur.toFloat(), xCur.toFloat(), yAsymptote.toFloat(),
                )

                xCur = xNext
                yCur = yNext

                continue
            }

            canvas.drawLine(
                xCur.toFloat(), yCur.toFloat(), xNext.toFloat(), yNext.toFloat(), paint
            )

            xCur = xNext
            yCur = yNext
        }

        return true
    }

    private fun drawGrid() {
        recalculateGridStep()
        val (left, top, right, bottom) = canvas.clipBounds
```

```
        val gridStep = 1 * PIXELS_PER_UNIT

        val gridScale = gridStep * curStepMultiplier
        val mainEvery = 5

        withXGridStep(left, right, gridScale, mainEvery) { x, isMain ->
            paint.forGrid(isMain) {
                canvas.drawLine(x, top.toFloat(), x, bottom.toFloat(), this)
            }
        }

        withYGridStep(top, bottom, gridScale, mainEvery) { y, isMain ->
            paint.forGrid(isMain) {
                canvas.drawLine(left.toFloat(), y, right.toFloat(), y, this)
                if (isMain) writeTextYAxis(y, left, right)
            }
        }

        // Needs to draw separately because otherwise y grid lines are over the text
        withXGridStep(left, right, gridScale, mainEvery) { x, isMain ->
            if (isMain) writeTextXAxis(x, bottom, top) // revere top and bottom because of
        }
    }

    private inline fun withXGridStep(
        left: Int, right: Int, gridScale: Float, mainEvery: Int, block: (Float, Boolean) ->
    ) {
        var x = left - left % gridScale
        while (x < right) {
            val isMain = x.absoluteValue % (gridScale * mainEvery) == 0f
            block(x, isMain)
            x += gridScale
        }
    }

    private inline fun withYGridStep(
        top: Int, bottom: Int, gridScale: Float, mainEvery: Int, block: (Float, Boolean) ->
    ) {
        var y = top - top % gridScale
        while (y < bottom) {
            val isMain = y.absoluteValue % (gridScale * mainEvery) == 0f
            block(y, isMain)
            y += gridScale
        }
    }

    private fun recalculateGridStep() {
        val gridScale = 1 * PIXELS_PER_UNIT * curStepMultiplier

        val prevOrigWidth = gridScale * prevScaleFactor
        val curOrigWidth = gridScale * scaleFactor
        val isZoomedIn = prevScaleFactor < scaleFactor
        val largerTimes = 2
```

```
        if (isZoomedIn && prevOrigWidth * largerTimes < curOrigWidth) {
            curStepMultiplier /= largerTimes
            prevScaleFactor = scaleFactor
        }

        if (!isZoomedIn && prevOrigWidth / largerTimes > curOrigWidth) {
            curStepMultiplier *= largerTimes
            prevScaleFactor = scaleFactor
        }
    }

    private fun formatFloatTextForAxis(text: String): String {
        // if is integer then remove .0
        val isInteger = text.endsWith(".0")
        if (isInteger) return text.substring(0, text.length - 2)

        // if is float with 2 digits leave it
        val digitsAfterDot = text.substringAfter(".").length
        if (digitsAfterDot == 2) return text

        // if is float with more than 2 digits write it in scientific notation
        return text.toDouble().toBigDecimal().toEngineeringString()
    }

    private fun writeTextXAxis(curX: Float, cameraTop: Int, cameraBottom: Int) {
        if (cameraTop < cameraBottom) throw IllegalArgumentException("cameraTop␣<␣cameraBot

        val text = formatFloatTextForAxis((curX / PIXELS_PER_UNIT).toString())
        val textBound = Rect().apply { paint.getTextBounds(text, 0, text.length, this) }
        val textX = curX - textBound.width()
        val textTopCorner = -textBound.height() * 0.1f
        val textBottomCorner = textTopCorner - textBound.height().toFloat()

        var textYOffset = 0f
        if (textTopCorner > cameraTop) textYOffset = -(textTopCorner - cameraTop) - textTop
        if (textBottomCorner < cameraBottom) textYOffset = cameraBottom - textBottomCorner

        if (curX == 0f) textYOffset = 0f // don't move 0 to the left (it is already in the

        paint.withColor(colors.text) {
            canvas.drawTextInRightDirection(text, textX, textTopCorner - textBound.height()
        }
    }

    private fun writeTextYAxis(curY: Float, cameraLeft: Int, cameraRight: Int) {
        if (curY == 0f) return
        val text = formatFloatTextForAxis((curY / PIXELS_PER_UNIT).toString())
        val textBound = Rect().apply { paint.getTextBounds(text, 0, text.length, this) }
        val textY = curY + textBound.height()
        val leftTextCorner = -textBound.width().toFloat()
        val rightTextCorner = 0f

        var textXOffset = 0f
```

```
        if (leftTextCorner < cameraLeft) textXOffset = cameraLeft - leftTextCorner * 1.5f
        if (rightTextCorner > cameraRight) textXOffset = -(rightTextCorner - cameraRight)

        paint.withColor(colors.text) {
            canvas.drawTextInRightDirection(text, leftTextCorner + textXOffset, textY, pai
        }
    }

    private fun Paint.forGrid(isMain: Boolean, block: Paint.() -> Unit) {
        val oldStrokeWidth = this.strokeWidth
        val oldColor = this.color
        this.strokeWidth /= if (isMain) 2f else 4f
        block()
        this.strokeWidth = oldStrokeWidth
        this.color = oldColor
    }

    private fun drawAxis() {
        val (left, top, right, bottom) = canvas.clipBounds

        canvas.drawLine(left.toFloat(), 0f, right.toFloat(), 0f, paint)
        canvas.drawLine(0f, top.toFloat(), 0f, bottom.toFloat(), paint)
    }

    private fun translateCameraListener(): OnTouchListener {
        var startPoint = PointF(0f, 0f)
        var endPoint: PointF

        return OnTouchListener { _, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> startPoint = PointF(event.x, event.y)
                MotionEvent.ACTION_MOVE -> {
                    endPoint = PointF(event.x, event.y)
                    val dx = endPoint.x - startPoint.x
                    val dy = endPoint.y - startPoint.y
                    // update start point for next move if not updated then the camera wil
                    startPoint.set(endPoint)

                    moveCamera(dx, dy)
                }
            }

            true
        }
    }

    private fun moveCamera(dx: Float, dy: Float) {
        matrixCamera.postTranslate(dx, dy)
        invalidate()
    }

    private fun scaleCameraListener(): OnTouchListener = OnTouchListener { _, event ->
        scaleGestureDetector.onTouchEvent(event)
        true
```

```
    }

    private fun initScaleGestureDetector(context: Context): ScaleGestureDetector =
        ScaleGestureDetector(context, object : SimpleOnScaleGestureListener() {
            override fun onScale(detector: ScaleGestureDetector): Boolean {
                val scaleFactor = detector.scaleFactor
//                  matrixCamera.postScale(scaleFactor, scaleFactor, detector.focusX, detecto
                updateScaleFactor(scaleFactor)
                return true
            }
        })

    private fun updateScaleFactor(scaleFactor: Float) {
        val scaled = this.scaleFactor * scaleFactor
        if (scaled > 30f) return // scale when text becomes unreadable, though it is scaled

        this.scaleFactor = scaled
        paint.textSize /= scaleFactor
        paint.strokeWidth /= scaleFactor
        invalidate()
    }

    fun set(c: Colors, newFns: List<PlotUIState>, onPlotNotValid: (PlotUIState) -> Unit) {
        colors = c
        paint.color = c.axes

        fns = newFns

        this.onPlotNotValid = onPlotNotValid

        invalidate()
    }

    private fun MathParser.evalInPixels(x: Double): Double? =
        setVariable("x", x / PIXELS_PER_UNIT).evalOrNull { it * PIXELS_PER_UNIT }
}
```

Лістинг 15: CavnasView.kt

```
package com.github.erotourtes.ui.screen.canvas.drawing

import android.util.Log
import androidx.compose.foundation.background
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clipToBounds
import androidx.compose.ui.graphics.toArgb
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.viewinterop.AndroidView
import com.github.erotourtes.model.PlotUIState

@Composable
fun CanvasView(
    plotState: List<PlotUIState>, onPlotNotValid: (PlotUIState) -> Unit, modifier: Modifier
```

```
) {
    val colorScheme = Colors(
        MaterialTheme.colorScheme.primary.toArgb(),
        MaterialTheme.colorScheme.background.toArgb(),
        MaterialTheme.colorScheme.onBackground.toArgb(),
    )
    AndroidView(
        factory = { context -> Log.i("CanvasView", "Creating new native view"); CanvasView
        modifier = modifier.clipToBounds()
    ) { view ->
        view.set(colorScheme, plotState, onPlotNotValid)
    }
}

@Preview(
    showBackground = true
)
@Composable
private fun Preview() {
    CanvasView(plotState = listOf(
        PlotUIState(
            color = androidx.compose.ui.graphics.Color.Red,
            function = "sin(x)",
            isVisible = true,
            isValid = true,
            id = 0,
        ),
        PlotUIState(
            color = androidx.compose.ui.graphics.Color.Blue,
            function = "cos(x)",
            isVisible = true,
            isValid = true,
            id = 1,
        ),
    ), {})
}
```

6.8 Module: com.github.erotourtes

Лістинг 16: ExampleUnitTest.kt

```
package com.github.erotourtes

import org.junit.Test

import org.junit.Assert.*

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
```

```
        assertEquals(4, 2 + 2)
    }
}
```

Лістинг 17: ExampleInstrumentedTest.kt

```
package com.github.erotourtes

import androidx.test.platform.app.InstrumentationRegistry
import androidx.test.ext.junit.runners.AndroidJUnit4

import org.junit.Test
import org.junit.runner.RunWith

import org.junit.Assert.*

/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.github.erotourtes", appContext.packageName)
    }
}
```

Лістинг 18: MainActivity.kt

```
package com.github.erotourtes

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import com.github.erotourtes.ui.EtherealPlotApp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val appContainer = (application as EtherealPlotApplication).container

        setContent { EtherealPlotApp(appContainer) }
    }
}
```

Лістинг 19: MyApplication.kt

```
package com.github.erotourtes

import android.app.Application
```

```kotlin
import com.github.erotourtes.data.AppContainer
import com.github.erotourtes.data.AppContainerImpl

class EtherealPlotApplication : Application() {

    // AppContainer is a dependency container that holds single instances of repositories,
    lateinit var container: AppContainer

    override fun onCreate() {
        super.onCreate()
        container = AppContainerImpl(this)
    }
}
```

6.9 Module: com.github.erotourtes.model

Лістинг 20: MockPlots.kt

```kotlin
package com.github.erotourtes.model

import androidx.compose.ui.graphics.Color

val mockPlots = listOf(
    PlotUIState(
        color = Color.Red,
        function = "sin(x)",
        isVisible = true,
        isValid = true,
        id = 0,
    ),
    PlotUIState(
        color = Color.Blue,
        function = "cos(x)",
        isVisible = true,
        isValid = true,
        id = 1,
    ),
)
```

Лістинг 21: PlotModel.kt

```kotlin
package com.github.erotourtes.model

import android.util.Log
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.toArgb
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.viewModelScope
import com.github.erotourtes.data.plot.Plot
import com.github.erotourtes.data.plot.PlotRepository
import com.github.erotourtes.utils.random
import com.github.erotourtes.utils.toPlotUIState
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.update
```

```
import kotlinx.coroutines.launch

data class PlotUIState(
    val color: Color,
    val function: String,
    val isVisible: Boolean = true,
    val isValid: Boolean = true,
    val id: Long,
) {

    fun toPlot(): Plot {
        return Plot(
            color = color.toArgb(),
            function = function,
            isVisible = isVisible,
            isValid = isValid,
        ).apply {
            id = this@PlotUIState.id
        }
    }
}

class PlotViewModel(
    private val plotRepo: PlotRepository
) : ViewModel() {
    // Jetpack compose doesn't detect changes if list is modified in place
    private val _plotUIState = MutableStateFlow<List<PlotUIState>>(listOf())
    val plotUIState = _plotUIState.asStateFlow()

    fun loadStateSync() {
        if (_plotUIState.value.isNotEmpty()) return

        viewModelScope.launch {
            _plotUIState.value = plotRepo.getPreviousPlots().map { it.toPlotUIState() }
            Log.i("PlotViewModel", "loadState in plotViewModel ${_plotUIState.value}")
        }
    }

    fun saveStateSync() {
        viewModelScope.launch {
            Log.i("PlotViewModel", "saveState")
            plotRepo.savePlots(_plotUIState.value.map { it.toPlot() })
        }
    }

    fun changePlotFormulaSync(oldState: PlotUIState, newValue: String) {
        Log.i("PlotViewModel", "changePlotFormula: $newValue")
        val updated = oldState.copy(function = newValue, isValid = true)
        viewModelScope.launch {
            plotRepo.savePlot(updated.toPlot())
        }

        _plotUIState.value = _plotUIState.value.toMutableList().apply {
            set(indexOfFirst { it.id == oldState.id }, updated)
```

```
    }
    Log.i("PlotViewModel", "changePlotFormula: ${_plotUIState.value}")
}

fun removePlotSync(plotUIState: PlotUIState) {
    viewModelScope.launch {
        val plot = plotUIState.toPlot()
        Log.i("PlotViewModel", "removePlot: ${plot.function} ${plot.id}")
        plotRepo.deletePlot(plot)
    }

    _plotUIState.value = _plotUIState.value.toMutableList().apply { remove(plotUIState)
}

fun changeColor(plotUIState: PlotUIState, color: Color) {
    Log.i("PlotViewModel", "changeColor: $color")
    updateProperty(plotUIState) {
        copy(color = color)
    }
}

fun changeHideState(plotUIState: PlotUIState, isVisible: Boolean) {
    Log.i("PlotViewModel", "changeHideState: $isVisible")
    updateProperty(plotUIState) {
        copy(isVisible = isVisible)
    }
}

fun changePlotValidity(plotUIState: PlotUIState, isValid: Boolean) {
    Log.i("PlotViewModel", "changePlotValidity: $isValid ${Thread.currentThread().name}
    updateProperty(plotUIState) {
        copy(isValid = isValid)
    }
}

fun createNewSync(color: Color = Color.random(), function: String = "") {
    Log.i("PlotViewModel", "createNew: $color")
    viewModelScope.launch {
        val plot = Plot(
            color = color.toArgb(),
            function = function,
            isVisible = true,
            isValid = true,
        )
        val id = plotRepo.savePlot(plot)
        Log.i("PlotViewModel", "createNew: $id")
        _plotUIState.update { list ->
            val updated = list.toMutableList()
            updated.add(
                PlotUIState(
                    color = color, function = function, isVisible = true, isValid = tru
                )
            )
            updated
```

```
            }
        }
    }

    private fun isInvalidIndex(index: Int): Boolean {
        if (index == -1) {
            Log.e(
                "PlotViewModel", "Severe: changePlotFormula: index == -1; happens on rapid
            )
            return true
        }
        return false
    }

    private fun updateProperty(
        plotUIState: PlotUIState, update: PlotUIState.() -> PlotUIState
    ) {
        val index = _plotUIState.value.indexOfFirst { it.id == plotUIState.id }
        if (isInvalidIndex(index)) return
        _plotUIState.update { list ->
            val updated = list.toMutableList()
            updated[index] = updated[index].update()
            updated
        }
    }

    fun restorePreviousSession() {
        viewModelScope.launch {
            _plotUIState.value = plotRepo.getPreviousPlots().map { it.toPlotUIState() }
            Log.i("PlotViewModel", "restorePreviousSession: ${_plotUIState.value.size}")
        }
    }

    fun removeAllPlots() {
        _plotUIState.value = listOf()
        viewModelScope.launch {
            plotRepo.deleteAllPlots()
        }
    }

    companion object {
        fun provideFactory(plotRepo: PlotRepository): ViewModelProvider.Factory = object :
            @Suppress("UNCHECKED_CAST")
            override fun <T : ViewModel> create(modelClass: Class<T>): T {
                return PlotViewModel(plotRepo) as T
            }
        }
    }
}
```

6.10 Module: com.github.erotourtes.data.plot

Лістинг 22: Plot.kt

```
package com.github.erotourtes.data.plot
```

```
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "Plot")
data class Plot(
    val color: Int,
    val function: String,
    val isVisible: Boolean = true,
    val isValid: Boolean = true,

    ) {
    @PrimaryKey(autoGenerate = true)
    var id: Long = 0
}
```

Лістинг 23: PlotDao.kt

```
package com.github.erotourtes.data.plot

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Query
import androidx.room.Upsert

@Dao
interface PlotDao {
    @Upsert
    suspend fun savePlot(plot: Plot): Long

    @Upsert // Upsert is a combination of insert and update
    suspend fun savePlots(plots: List<Plot>): List<Long>

    @Delete
    suspend fun deletePlot(plot: Plot)


    @Query("SELECT * FROM Plot LIMIT 10")
    suspend fun getPreviousPlots(): List<Plot>

    @Query("DELETE FROM Plot")
    suspend fun deletePlots()
}
```

Лістинг 24: PlotRepository.kt

```
package com.github.erotourtes.data.plot

class PlotRepository(
    private val plotDao: PlotDao
) {

    suspend fun savePlot(plot: Plot): Long {
        return plotDao.savePlot(plot)
    }
```

```
    suspend fun savePlots(plots: List<Plot>): List<Long> {
        return plotDao.savePlots(plots)
    }

    suspend fun deletePlot(plot: Plot) {
        plotDao.deletePlot(plot)
    }

    suspend fun deleteAllPlots() {
        plotDao.deletePlots()
    }

    suspend fun getPreviousPlots(): List<Plot> {
        return plotDao.getPreviousPlots()
    }
}
```

6.11 Module: com.github.erotourtes.ui.utils

Лістинг 25: ExpandableCard.kt

```
package com.github.erotourtes.ui.utils

import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowDropDown
import androidx.compose.material3.*
import android.annotation.SuppressLint
import androidx.compose.animation.*
import androidx.compose.animation.core.*
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.Dp
import com.github.erotourtes.ui.theme.spacing

const val EXPANSTION_TRANSITION_DURATION = 450

@SuppressLint("UnusedTransitionTargetStateParameter")
@Composable
fun ExpandableCard(
    expandableContent: @Composable () -> Unit,
    expanded: Boolean,
    modifier: Modifier = Modifier,
    paddingExpanded: Dp = MaterialTheme.spacing.large,
    paddingCollapsed: Dp = MaterialTheme.spacing.default,
    content: @Composable () -> Unit,
) {
```

```
    val transitionState = remember {
        MutableTransitionState(expanded).apply {
            targetState = !expanded
        }
    }
    val transition = updateTransition(transitionState, label = "transition")
    val cardPaddingHorizontal by transition.animateDp({
        tween(durationMillis = EXPANSTION_TRANSITION_DURATION)
    }, label = "paddingTransition") { if (expanded) paddingExpanded else paddingCollapsed }
    val cardRoundedCorners by transition.animateDp({
        tween(
            durationMillis = EXPANSTION_TRANSITION_DURATION, easing = FastOutSlowInEasing
        )
    }, label = "cornersTransition") {
        if (expanded) MaterialTheme.spacing.small else MaterialTheme.spacing.medium
    }

    Card(
        shape = RoundedCornerShape(cardRoundedCorners), modifier = modifier
            .fillMaxWidth()
            .padding(
                horizontal = cardPaddingHorizontal,
            )
    ) {
        Column {
            content()
            ExpandableContent(visible = expanded, content = expandableContent)
        }
    }
}

@Composable
fun ExpandableContent(
    visible: Boolean = true, content: @Composable () -> Unit
) {
    val enterTransition = remember {
        expandVertically(
            expandFrom = Alignment.Top, animationSpec = tween(EXPANSTION_TRANSITION_DURATIO
        ) + fadeIn(
            initialAlpha = 0.3f, animationSpec = tween(EXPANSTION_TRANSITION_DURATION)
        )
    }
    val exitTransition = remember {
        shrinkVertically(
            // Expand from the top.
            shrinkTowards = Alignment.Top, animationSpec = tween(EXPANSTION_TRANSITION_DURA
        ) + fadeOut(
            // Fade in with the initial alpha of 0.3f.
            animationSpec = tween(EXPANSTION_TRANSITION_DURATION)
        )
    }

    AnimatedVisibility(
        visible = visible, enter = enterTransition, exit = exitTransition
```

```
    ) {
        content()
    }
}

@Preview
@Composable
private fun PreviewExpandableCard() {
    var expanded by remember { mutableStateOf(false) }
    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(Color.White)
    ) {
        ExpandableCard(
            expandableContent = {
                Text(
                    text = "Expandable content",
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(16.dp),
                    textAlign = TextAlign.Center
                )
            },
            expanded = expanded,
        ) {
            Text(
                text = "Card content", modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp), textAlign = TextAlign.Center
            )
        }
    }
}
```

Лістинг 26: SwapToReveal.kt

```
package com.github.erotourtes.ui.utils

import androidx.compose.animation.core.*
import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.gestures.*
import androidx.compose.foundation.layout.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.onGloballyPositioned
import androidx.compose.ui.platform.LocalDensity
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.IntOffset
import androidx.compose.ui.unit.dp
import kotlin.math.roundToInt

enum class DragAnchors(val fraction: Float) {
```

```
    Start(0f), Half(.5f), // recalculating this value on size change to fit hidden content
    End(1f),
)

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun SwapToReveal(
    hiddenContent: @Composable () -> Unit,
    modifier: Modifier = Modifier,
    onRemove: () -> Unit = {},
    content: @Composable () -> Unit,
) {
    var hiddenWidth by remember { mutableFloatStateOf(0f) }
    var fullWidth by remember { mutableFloatStateOf(0f) }

    val minVisibleFraction = 0.1f
    val density = LocalDensity.current

    val anchoredDraggableState = remember {
        AnchoredDraggableState(
            initialValue = DragAnchors.Start,
            positionalThreshold = { distance: Float -> distance * 0.5f },
            velocityThreshold = { with(density) { 100.dp.toPx() } },
            animationSpec = tween(),
            confirmValueChange = { newValue ->
                if (newValue == DragAnchors.End) {
                    onRemove()
                    false
                } else true
            },
        ).apply {
            updateAnchors(DraggableAnchors {
                DragAnchors.entries.forEach { anchor -> anchor at 0f }
            })
        }
    }

    LaunchedEffect(fullWidth, hiddenWidth) {
        val dragEndPoint = fullWidth * (1 - minVisibleFraction)
        anchoredDraggableState.updateAnchors(DraggableAnchors {
            DragAnchors.entries.forEach { anchor ->
                if (anchor == DragAnchors.Half) anchor at hiddenWidth
                else anchor at dragEndPoint * anchor.fraction
            }
        })
    }

    Box(modifier = modifier.onGloballyPositioned {
        fullWidth = it.size.width.toFloat()
    }) {
        Box(modifier = Modifier
            .fillMaxHeight()
            .onGloballyPositioned {
                hiddenWidth = it.size.width
```

```
                    .toFloat()
                    .coerceIn(0f, fullWidth)
            }) {
            hiddenContent()
        }

        Box(modifier = Modifier
            .fillMaxHeight()
            .offset {
                IntOffset(
                    x = anchoredDraggableState
                        .requireOffset()
                        .roundToInt(), y = 0
                )
            }
            .anchoredDraggable(anchoredDraggableState, Orientation.Horizontal)) {
            content()
        }
    }
}

@Preview
@Composable
private fun SwapPreview() {
    val height by remember {
        mutableStateOf(150.dp)
    }
    SwapToReveal(
        hiddenContent = {
            Box(
                modifier = Modifier
                    .background(color = Color.Green)
                    .width(100.dp)
                    .height(height),
            )
        },
        modifier = Modifier
            .background(color = Color.Blue)
            .fillMaxWidth()
            .height(height)
            .padding(10.dp),
    ) {
        Box(
            modifier = Modifier
                .background(color = Color.Red)
                .fillMaxWidth()
                .height(height),
        )
    }
}
```

6.12 Module: com.github.erotourtes.ui.theme

Лістинг 27: Color.kt

```kotlin
package com.github.erotourtes.ui.theme

import androidx.compose.ui.graphics.Color

val TextWhite = Color(0xffeeeeee)
val DeepBlue = Color(0xff06164c)
val ButtonBlue = Color(0xff505cf3)
val DarkerButtonBlue = Color(0xff566894)
val LightRed = Color(0xfffc879a)
val AquaBlue = Color(0xff9aa5c4)
val OrangeYellow1 = Color(0xfff0bd28)
val OrangeYellow2 = Color(0xfff1c746)
val OrangeYellow3 = Color(0xfff4cf65)
val Beige1 = Color(0xfffdbda1)
val Beige2 = Color(0xfffcaf90)
val Beige3 = Color(0xfff9a27b)
val LightGreen1 = Color(0xff54e1b6)
val LightGreen2 = Color(0xff36ddab)
val LightGreen3 = Color(0xff11d79b)
val BlueViolet1 = Color(0xffaeb4fd)
val BlueViolet2 = Color(0xff9fa5fe)
val BlueViolet3 = Color(0xff8f98fd)
```

Лістинг 28: Spacing.kt

```kotlin
package com.github.erotourtes.ui.theme

import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.runtime.Immutable
import androidx.compose.runtime.ReadOnlyComposable
import androidx.compose.runtime.compositionLocalOf
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp

@Immutable
data class Spacing(
    val default: Dp = 2.dp,
    val small: Dp = 4.dp,
    val medium: Dp = 8.dp,
    val large: Dp = 16.dp,
    val xLarge: Dp = 32.dp,
    val xxLarge: Dp = 64.dp,
)

val LocalSpacing = compositionLocalOf { Spacing() }

val MaterialTheme.spacing: Spacing
    @Composable
    @ReadOnlyComposable
    get() = LocalSpacing.current
```

Лістинг 29: Theme.kt

```kotlin
package com.github.erotourtes.ui.theme
```

```
import android.app.Activity
import android.os.Build
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.dynamicDarkColorScheme
import androidx.compose.material3.dynamicLightColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.runtime.CompositionLocalProvider
import androidx.compose.runtime.SideEffect
import androidx.compose.ui.graphics.toArgb
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalView
import androidx.core.view.WindowCompat

private val DarkColorScheme = darkColorScheme(
    primary = DeepBlue,
    secondary = LightGreen1,
    tertiary = BlueViolet1,
)

private val LightColorScheme = lightColorScheme(
    primary = OrangeYellow1,
    secondary = LightGreen1,
    tertiary = BlueViolet1

    /* Other default colors to override
    background = Color(0xFFFFFBFE),
    surface = Color(0xFFFFFBFE),
    onPrimary = Color.White,
    onSecondary = Color.White,
    onTertiary = Color.White,
    onBackground = Color(0xFF1C1B1F),
    onSurface = Color(0xFF1C1B1F),
    */
)

@Composable
fun AppTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
//        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
//            val context = LocalContext.current
//            if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(c
//        }
//
        darkTheme -> DarkColorScheme
        else -> LightColorScheme
```

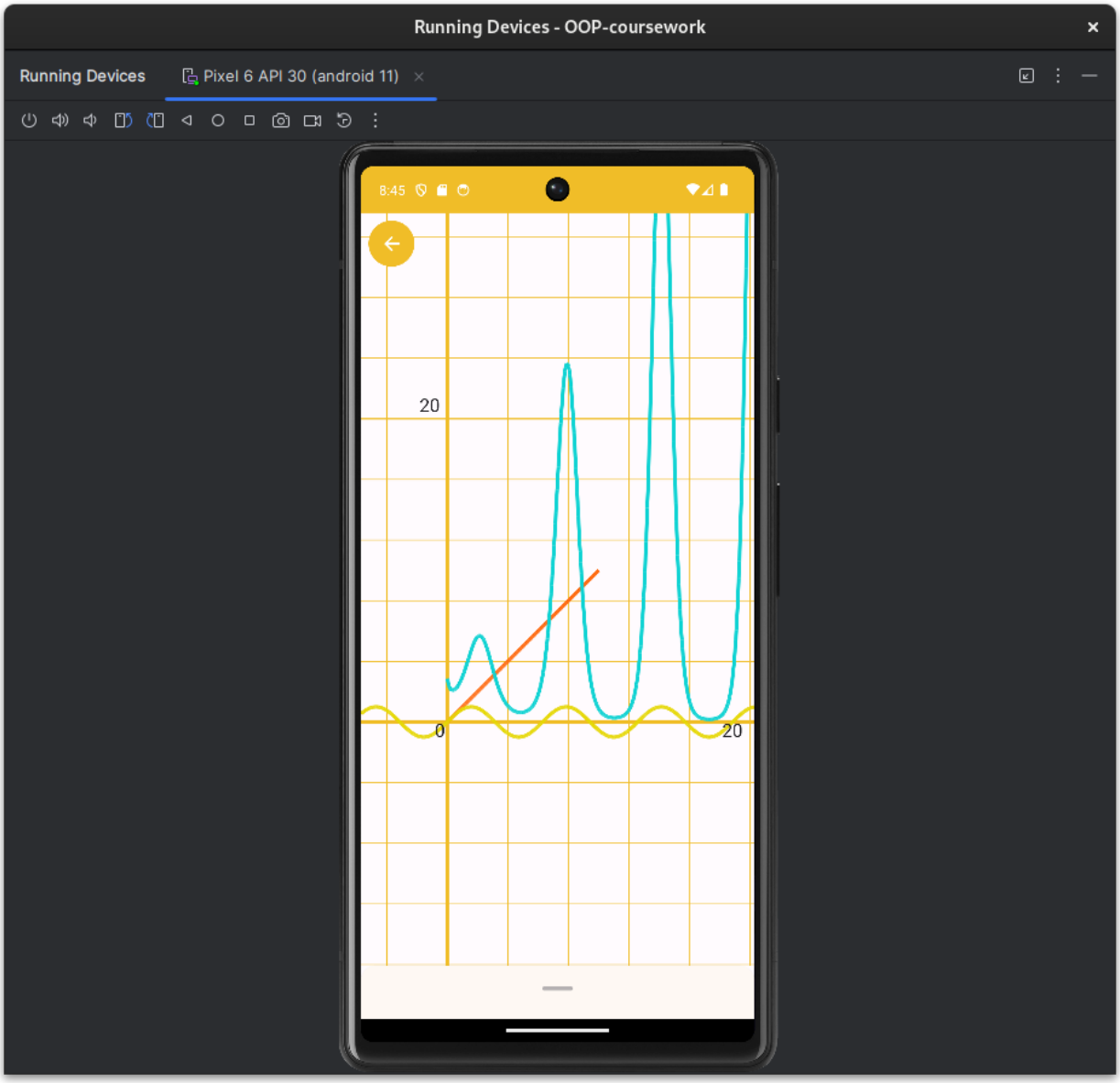
```
    }
    val view = LocalView.current
    if (!view.isInEditMode) {
        SideEffect {
            val window = (view.context as Activity).window
            window.statusBarColor = colorScheme.primary.toArgb()
            WindowCompat.getInsetsController(window, view).isAppearanceLightStatusBars = da
        }
    }

    CompositionLocalProvider(
        LocalSpacing provides Spacing()
    ) {
        MaterialTheme(
            colorScheme = colorScheme,
            typography = Typography,
            content = content
        )
    }
}
```

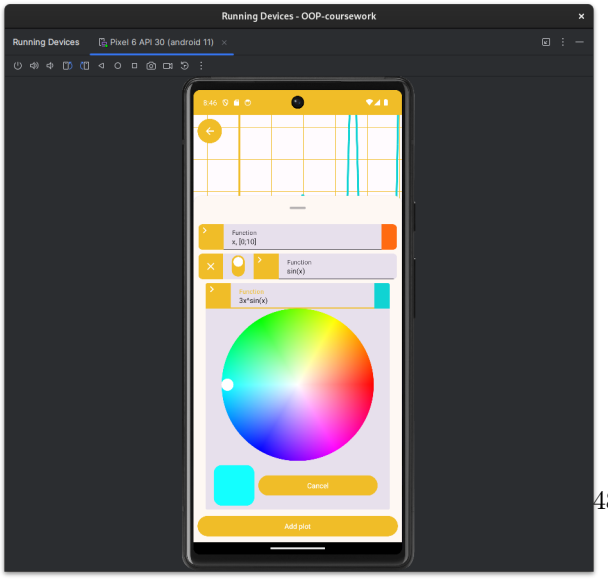
Лістинг 30: Type.kt

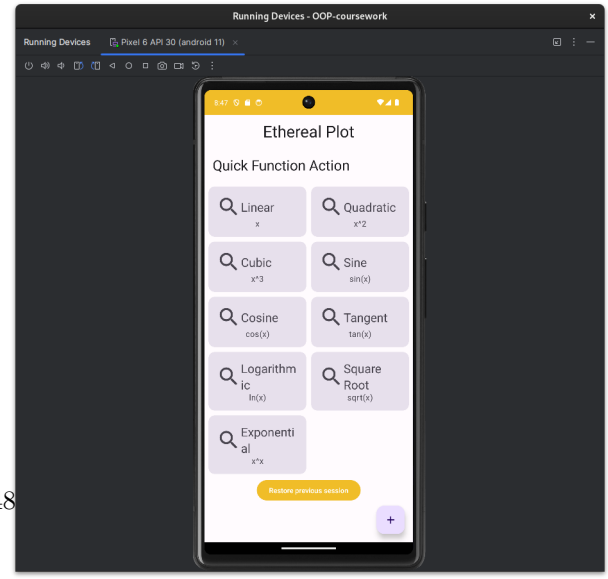
```
package com.github.erotourtes.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
    /* Other default text styles to override
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
```

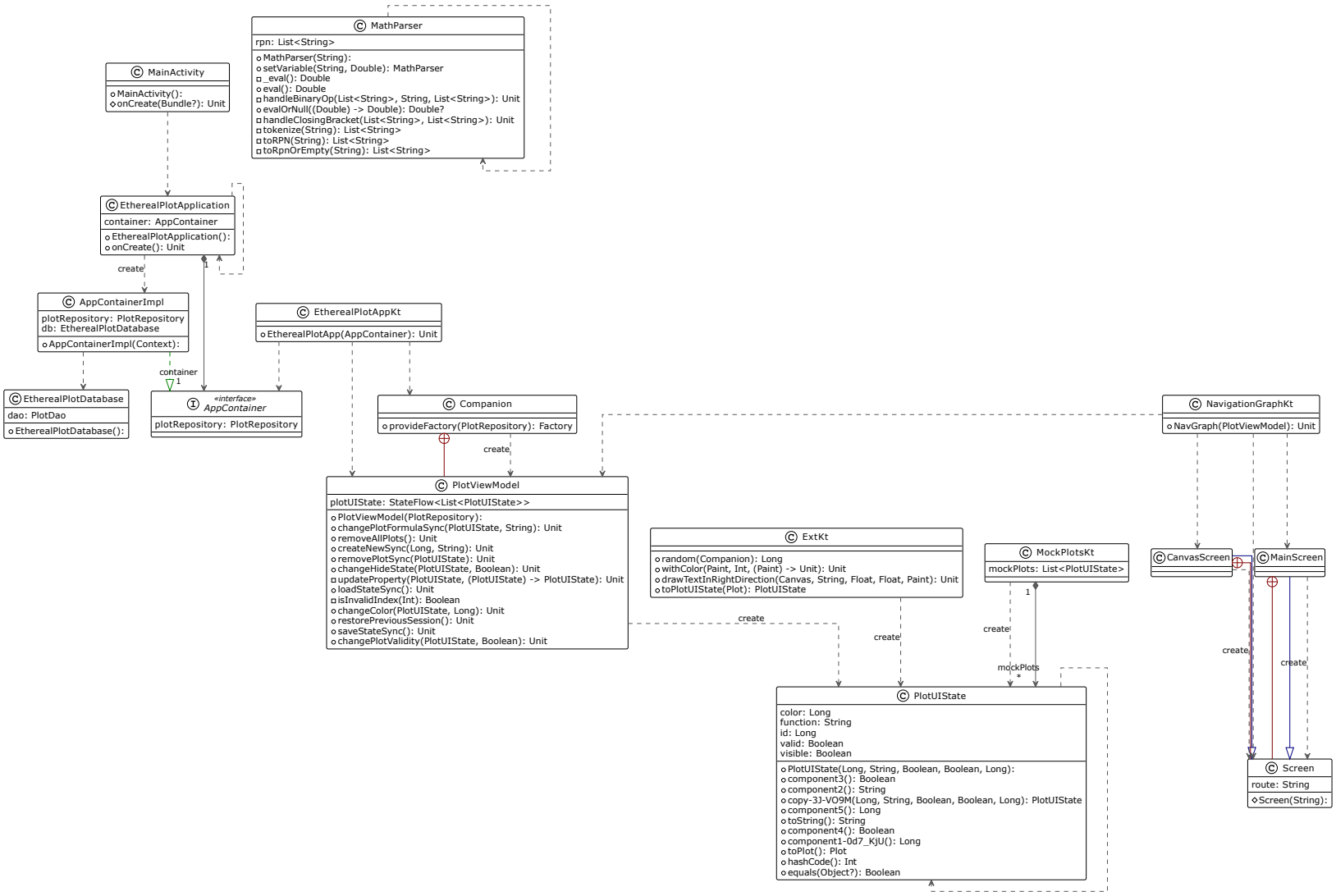
```
    */
)
```

7 Ілюстрації:








MathParser
rpn: List<String>
MathParser(String):
setVariable(String, Double): MathParser
_eval(): Double
eval(): Double
handleBinaryOp(List<String>, String, List<String>): Unit
evalOrNull((Double) -> Double): Double?
handleClosingBracket(List<String>, List<String>): Unit
tokenize(String): List<String>
toRPN(String): List<String>
toRpnOrEmpty(String): List<String>
MainActivity
MainActivity():
onCreate(Bundle?): Unit
EtherealPlotApplication
container: AppContainer
EtherealPlotApplication():
onCreate(): Unit
create
1
AppContainerImpl
plotRepository: PlotRepository
db: EtherealPlotDatabase
AppContainerImpl(Context):
container
1
EtherealPlotDatabase
dao: PlotDao
EtherealPlotDatabase():
«interface»
AppContainer
plotRepository: PlotRepository
EtherealPlotAppKt
EtherealPlotApp(AppContainer): Unit
Companion
provideFactory(PlotRepository): Factory
create
PlotViewModel
plotUIState: StateFlow<List<PlotUIState>>
PlotViewModel(PlotRepository):
changePlotFormulaSync(PlotUIState, String): Unit
removeAllPlots(): Unit
createNewSync(Long, String): Unit
removePlotSync(PlotUIState): Unit
changeHideState(PlotUIState, Boolean): Unit
updateProperty(PlotUIState, (PlotUIState) -> PlotUIState): Unit
loadStateSync(): Unit
isInvalidIndex(Int): Boolean
changeColor(PlotUIState, Long): Unit
restorePreviousSession(): Unit
saveStateSync(): Unit
changePlotValidity(PlotUIState, Boolean): Unit
ExtKt
random(Companion): Long
withColor(Paint, Int, (Paint) -> Unit): Unit
drawTextInRightDirection(Canvas, String, Float, Float, Paint): Unit
toPlotUIState(Plot): PlotUIState
create
create
MockPlotsKt
mockPlots: List<PlotUIState>
1
create
mockPlots
*
NavigationGraphKt
NavGraph(PlotViewModel): Unit
CanvasScreen
MainScreen
create
create
create
PlotUIState
color: Long
function: String
id: Long
valid: Boolean
visible: Boolean
PlotUIState(Long, String, Boolean, Boolean, Long):
component3(): Boolean
component2(): String
copy-3J-VO9M(Long, String, Boolean, Boolean, Long): PlotUIState
component5(): Long
toString(): String
component4(): Boolean
component1-0d7_KjU(): Long
toPlot(): Plot
hashCode(): Int
equals(Object?): Boolean
Screen
route: String
Screen(String):

8 Висновки:

У даній теоретичній частині було розглянуто ключові аспекти, пов'язані з розробкою програми для побудови графіків довільних функцій на платформі Android з використанням Jetpack Compose. Основні висновки можна сформулювати наступним чином:

1. **Графічні Двигуни та Бібліотеки:** Було проведено аналіз різних графічних двигунів та бібліотек для побудови графіків, і Jetpack Compose виявився потужним та сучасним інструментом з багатим функціоналом.

2. **Jetpack Compose та Kotlin:** Вибір Jetpack Compose та мови програмування Kotlin був обґрунтований їх сучасністю, продуктивністю та підтримкою Android. Це сприяє ефективній розробці та забезпечує чистий та читабельний код.

3. **Реактивне Програмування:** Використання концепцій реактивного програмування дозволяє ефективно взаємодіяти з графічним інтерфейсом, роблячи його більш відзивчивим та динамічним.

4. **Архітектурні Підходи:** Використання архітектурного підходу MVVM дозволяє ефективно розділити логіку програми та відображення, полегшуючи підтримку та розширення.

5. **Тестування в Android:** Стратегії тестування, такі як юніт-тестування та інтеграційне тестування, є необхідним елементом для забезпечення стабільності та надійності програми.

В цілому, теоретичний аналіз дозволяє визначити оптимальний напрям для подальшого розвитку проекту "Ethereal Plot"та забезпечує підґрунтя для успішної реалізації практичної частини.