5COSC023W - Tutorial 11 Exercises - Sample Solutions

The Cocktails App - Another Web Services Example

Make sure that you include the Internet permission in the manifest file.

File `MainActivity.kt`:

```kotlin
package uk.ac.westminster.cocktailscomposableapp

import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import org.json.JSONObject
import java.io.BufferedReader
import java.io.InputStreamReader
import java.net.URL

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            GUI()
        }
    }
}

@Preview
@Composable
fun GUI() {
    var ingredient by remember { mutableStateOf("") }
    var cocktail_name by remember { mutableStateOf("") }
    var coctails_list: MutableList<String> by remember { mutableStateOf(mutableListOf()) }

    val scope = rememberCoroutineScope()
    val context = LocalContext.current

    Column(
        modifier = Modifier
            .fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        TextField(label = { Text(text = "Ingredient name") },
            value = ingredient, onValueChange = { ingredient = it })

        Button(
            shape = RectangleShape,
            onClick = {
                // get the cocktail names which contain this ingredient
                // in a new coroutine
                scope.launch {
                    coctails_list = getNames(ingredient)
                }
            }) {
            Text(text = "List Cocktails")
        }

        LazyColumn(
            modifier = Modifier
                      .height(500.dp)
                      .padding(5.dp)
        )
        {
            for (i in coctails_list) {
                item {
                    Row(modifier = Modifier
                        .fillMaxWidth()
                        .clickable(onClick = {
                            cocktail_name = i
                        }))
                    {
                        Text(text = i)
                    }
                }
            }
        }

        TextField(label = { Text(text = "Cocktail name") },
            value = cocktail_name, onValueChange = { cocktail_name = it })

        Button(
            shape = RectangleShape,
            onClick = {
                if (cocktail_name.trim() != "")
                    getRecipes(cocktail_name, context)
                else
                Toast.makeText(context, "Requested cocktail name cannot be blank",
                               Toast.LENGTH_SHORT).show()
            }) {
            Text(text = "Get Recipe")
        }
    }
}


suspend fun getNames(ingredient_requested: String): MutableList<String> {
    val stb = StringBuilder("") // contains all json
    val cocktails_list = mutableListOf<String>()

    // run this in a new thread
    withContext(Dispatchers.IO) {
        val url = URL(
            "https://www.thecocktaildb.com/api/json/v1/1/filter.php?i=" +
                    ingredient_requested.trim()
        )
        val con = url.openConnection()
        val bf = BufferedReader(InputStreamReader(con.getInputStream()))

        // read all lines JSON info  in a stringbuilder
        var line = bf.readLine()
        while (line != null) {
            stb.append(line)
            line = bf.readLine()
        }

        // now do the JSON parsing
        if (stb.toString() == "")
            return@withContext

        val json = JSONObject(stb.toString())
        val jsonArray = json.getJSONArray("drinks")

        // find the cocktail names entries and put them in the list
        for (i in 0..jsonArray.length() - 1) {
            val json_drink = jsonArray[i] as JSONObject
            val cocktail_name = json_drink["strDrink"] as String

            cocktails_list.add(cocktail_name)
        }
    }

    return cocktails_list
}


fun getRecipes(cocktail_name: String, context: Context) {
    val i = Intent(context, MainActivity2::class.java)
    i.putExtra("name", cocktail_name.trim())
    context.startActivity(i)
}
```

File MainActivity2.kt:

```
package uk.ac.westminster.cocktailscomposableapp

import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.ImageBitmap
import androidx.compose.ui.graphics.asImageBitmap
import androidx.compose.ui.graphics.painter.BitmapPainter
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import org.json.JSONArray
import org.json.JSONObject
import java.io.BufferedInputStream
import java.io.BufferedReader
import java.io.InputStreamReader
import java.net.HttpURLConnection
import java.net.URL


// pointing to the url containing the cocktail photo to be displayed
var picture_url: String? = null

class MainActivity2 : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val cocktail_name = intent.getStringExtra("name")
        setContent {
            displayRecipePhoto(cocktail_name)
        }
    }
}


// display the cocktail recipe and its photo
@Composable
fun displayRecipePhoto(cocktail_name: String?) {
    var recipe by remember{mutableStateOf( "")} // contains the recipe extracted from JSON
    // the bitmap containing the photo of the cocktail
    var cocktail_picture: ImageBitmap? by remember{ mutableStateOf(null) }

    val context = LocalContext.current

    LaunchedEffect(cocktail_name) {
        withContext(Dispatchers.IO) {
            picture_url = null
```

```kotlin
        val url = URL(
            "https://www.thecocktaildb.com/api/json/v1/1/search.php?s=" +
                    cocktail_name?.trim()
        )
        val con = url.openConnection()
        val bf = BufferedReader(InputStreamReader(con.getInputStream()))

        var stb = StringBuilder("")  // contains all JSON

        // read all lines JSON info  in the stb stringbuilder
        var line = bf.readLine()
        while (line != null) {
            stb.append(line + "\n")
            line = bf.readLine()
        }

        /* do the JSON parsing */
        val json = JSONObject(stb.toString())

        //val jsonArray: JSONArray = json.getJSONArray("drinks")
        var jsonArray = json["drinks"]

        if (jsonArray is JSONArray) {
            jsonArray = jsonArray as JSONArray

            // find the matching cocktail name entry and extract recipe
            for (i in 0..jsonArray.length() - 1) {
                val json_drink = jsonArray.getJSONObject(i)
                val drinkName = json_drink.getString("strDrink")
                if (drinkName.lowercase() == cocktail_name?.lowercase()) {
                    recipe = json_drink.getString("strInstructions")
                    picture_url = json_drink.getString("strDrinkThumb")
                }
                else
                    recipe = "Drink not found in DB!"
            }

            if (picture_url != null)
                cocktail_picture = getBitmapPicture()
        }
        else {  // deal with the case that the user requested a non-existing drink
            withContext(Dispatchers.Main) {
                Toast.makeText(context, "Drink not found in DB!",
                                        Toast.LENGTH_SHORT).show()
            }
            return@withContext
        }
    }
    }

    Column (Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally){
        // Cocktail title
        if (cocktail_name != null) {
            Text(text = cocktail_name, fontSize = 30.sp, fontWeight = FontWeight.Bold)

            // the recipe
            Text(text = recipe.toString(), fontSize = 24.sp)

            // the photo of the cocktail
            if (cocktail_picture != null) {
                Image(
                    modifier = Modifier.padding(50.dp),
                    painter = BitmapPainter(cocktail_picture!!),
                    contentDescription = null
                )
            }
        }
    }
}


// retrieve a bitmap image from the URL in JSON
fun getBitmapPicture(): ImageBitmap {
    var bitmap: Bitmap? = null

    val url = URL(picture_url)
    val con = url.openConnection() as HttpURLConnection
    val bfstream = BufferedInputStream(con.inputStream)

    bitmap = BitmapFactory.decodeStream(bfstream)

    return bitmap.asImageBitmap()
}
```

The Memory Game

```
package com.example.memorygamecomposable

import android.os.Bundle
import android.os.CountDownTimer
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
```

```kotlin
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.runtime.toMutableStateList
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.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlin.random.Random

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            // render the screen
            GUI()
        }
    }
}


/* Which buttons have been presented to the user as green in the current game round.
   Buttons are identified with pairs of ints corresonding to (row, col) pais,
   e.g. starting with (1,1), (1,2), (1,3) for the first row, etc. */
var presented_buttons: MutableList<List<Int>> = mutableListOf()

// how many green cells will be presented to the user in the start of each game
val random_cells = 6

var number_of_clicks  = 0

var score_total = 0  // total answers given
var score_correct = 0  // total correct answers given by the user

// Returns a list with the rows and columns of the grid
fun initialise(): List<Int> {
    // 1. Decide on the number of rows and columns of the grid
    var rows = 3 + Random.nextInt(3)
    var cols =  3 + Random.nextInt(3)
    // clear presented button from the previous game
    presented_buttons.clear()

    number_of_clicks = 0

    // 2. Choose some (6) random cells to become green
    var count = 0
    while (count < random_cells) {
        var r = 1 + Random.nextInt(rows)
        val c = 1 + Random.nextInt(cols)
        val new_pair = listOf(r, c)

        // check the uniqueness of highlighted cells
        if (new_pair !in presented_buttons) {
            presented_buttons.add(new_pair)
            ++count
        }
    }

    return listOf(rows, cols)
}


// Switch off green cells by making them to their original colour again
@Composable
fun disableGreenCells(onCompletion: () -> Unit) {
    // 4. Present some cells to the user with green colour - for 3 secs
    val timer = object: CountDownTimer(3000, 1000) {
        override fun onTick(millisUntilFinished: Long) { }

        override fun onFinish() {
            onCompletion()
        }
    }
    timer.start()
}

@Preview
@Composable
fun GUI() {
    // false or true depending on whether the timer displaying some green cells
    // has run previously - it needs to be run only once per game
    var run_timer_before by remember{ mutableStateOf(false) }

    var rows by remember{ mutableStateOf(5) }
    var cols by remember{ mutableStateOf(5) }

    var selected_green: MutableList<Boolean> =
                    remember{MutableList(rows*cols) {i -> false}.toMutableStateList() }
    var selected_red: MutableList<Boolean> =
                    remember{MutableList(rows*cols) {i -> false}.toMutableStateList() }

    // true when a new game starts
    var new_game by remember{ mutableStateOf(true) }

    if (new_game) {
        // determine the grid size and the green cells to remember
        var dimensions = initialise()
        rows = dimensions[0]
        cols = dimensions[1]

        // reset the values of the cell colours in the 2 lists
        for (i in 0..selected_green.size-1) {
            selected_green[i] = false
            selected_red[i] = false
        }

        // the timer needs to run again to highlight green cells for some secs
        run_timer_before = false

        new_game = false
    }

    // make green the selected cells and keep them green until after
    // the timer runs to completion
    if (!run_timer_before) {
        for (cell in presented_buttons) {
            val r = cell[0]
            val c = cell[1]

            selected_green[(r - 1) * cols + c - 1] = true
        }

    }

    Column() {
        // Render the score
        Text("Score: $score_correct/$score_total", modifier = Modifier.fillMaxWidth(),
              fontSize = 32.sp, textAlign = TextAlign.End)

        // 3. Draw the grid using rows of buttons
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(top = 100.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
        ) {
            for (r in 1..rows) {
                Row() {
                    for (c in 1..cols) {
                        // index of the button in a linearly 1-dimensional calculated way
                        val button_index = (r - 1) * cols + c - 1

                        var background_colour =
                            if (selected_red[button_index] == true)
                                Color.Red
                            else if (selected_green[button_index] == false)
                                Color.Gray
                            else
                                Color.Green

                        Button(
                            colors = ButtonDefaults.buttonColors(
                                         containerColor = background_colour
                            ),
                            onClick = {
                                if (number_of_clicks < presented_buttons.size) {
                                    if (listOf(r, c) in presented_buttons) {
                                        selected_green[button_index] = true
                                        updateScore(1)
                                    }
                                    else {
                                        selected_red[button_index] = true
                                        updateScore(0)
                                    }
                                    ++number_of_clicks
                                } else {
                                    new_game = true
                                }
                            }) {
                            if (background_colour == Color.Red)
                                Text("X")
                        }
                    }
                }
            }
        }
    }

    // run the count down timer only once
    if (!run_timer_before) {
        // pick up all the cells that were determined to be displayed as green
        disableGreenCells({
            // pick up all the cells that were determined to be displayed as green
            // and make them as gray
            for (coords in presented_buttons) {
                val r = coords[0]
                val c = coords[1]
                selected_green[(r-1)*cols + c - 1] = false
            }
        })

        //disable the timer for the current game
        run_timer_before = true
    }
}


fun updateScore(correct: Int) {
        ++score_total
        score_correct = score_correct + correct
}

/* Converts a pair of row (r) and column (c) arguments coordinates (1 to maxRow or maxCol)
   to a single index 0 to max_elements, based on row traversing - The function is not used
   here but it could make things easier for calculations */
fun RowCol2Index(r: Int, c: Int, cols: Int): Int {
    val index = (r-1)*cols + c - 1

    return index
}
```