UNIVERSIDAD TECNOLÓGICA DE LEÓN

Ingeniería en Desarrollo y gestión de software

Desarrollo para dispositivos inteligentes

Examen 2do Parcial

presenta:

Martínez Ruano Marco Esteban

IDGS901

Fecha: 02/Jun /2025

Main Activity

```kotlin
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Examen2doParcialTheme {
                val navController = rememberNavController()
                NavHost(navController = navController, startDestination = "datos_personales") {
                    composable( route: "datos_personales") {
                        DatosPersonalesScreen(
                            onCancelar = { finish() },
                            onSiguiente = { datos ->
                                val datosJson = Gson().toJson(datos)
                                navController.navigate( route: "preguntas/$datosJson")
                            }
                        )
                    }
                    composable(
                        route: "preguntas/{datosJson}",
                        arguments = listOf(navArgument( name: "datosJson") { type = NavType.StringType })
                    ) { backStackEntry ->
                        val datosJson = backStackEntry.arguments?.getString( key: "datosJson") ?: ""
                        val datos = Gson().fromJson(datosJson, DatosPersonales::class.java)
                        PreguntasScreen(
                            onVolver = { navController.popBackStack() },
                            onFinalizar = { calificacion, totalPreguntas ->
                                val datosJson2 = Gson().toJson(datos)
                                navController.navigate( route: "resultados/$datosJson2/$calificacion/$totalPreguntas")
                            }
                        )
                    }

                    composable(
                        route: "resultados/{datosJson}/{calificacion}/{totalPreguntas}",
                        arguments = listOf(
                            navArgument( name: "datosJson") { type = NavType.StringType },
                            navArgument( name: "calificacion") { type = NavType.IntType },
                            navArgument( name: "totalPreguntas") { type = NavType.IntType }
                        )
                    ) { backStackEntry ->
                        val datosJson = backStackEntry.arguments?.getString( key: "datosJson") ?: ""
                        val datos = Gson().fromJson(datosJson, DatosPersonales::class.java)
                        val calificacion = backStackEntry.arguments?.getInt( key: "calificacion") ?: 0
                        val totalPreguntas = backStackEntry.arguments?.getInt( key: "totalPreguntas") ?: 0
                        ResultadosScreen(
                            datos = datos,
                            calificacion = calificacion,
                            totalPreguntas = totalPreguntas,
                            onFinalizar = { finish() }
                        )
                    }
                }
            }
        }
    }
}
```

File Edit View Navigate Code Refactor Build Run Tools Git Window Help

Examen2doParcial ∨   master ∨

MainActivity.kt   DatosPersonalesScreen.kt ×   ResultadosScreen.kt   PreguntasScreen.kt   DatosPersonalesViewModel.kt   DatosPersonales.kt   Pregunta.kt

```kotlin
@Composable
fun DatosPersonalesScreen(
    viewModel: DatosPersonalesViewModel = viewModel(),
    onCancelar: () -> Unit,
    onSiguiente: (DatosPersonales) -> Unit
) {
    val nombre by viewModel.nombre
    val apellidoPaterno by viewModel.apellidoPaterno
    val apellidoMaterno by viewModel.apellidoMaterno
    val dia by viewModel.dia
    val mes by viewModel.mes
    val anio by viewModel.anio
    val sexo by viewModel.sexo

    var mostrarError by remember { mutableStateOf( value: false) }

    Surface(
        modifier = Modifier.fillMaxSize(),
        color = Color.White
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(24.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                text = "Datos Personales",
                fontSize = 24.sp,
                modifier = Modifier.padding(bottom = 24.dp)
            )

            OutlinedTextField(
                value = nombre,
                onValueChange = { viewModel.updateNombre(it) },
                label = { Text( text: "Nombre(s)") },
                modifier = Modifier.fillMaxWidth()
            )
            Spacer(modifier = Modifier.height(8.dp))

            OutlinedTextField(
                value = apellidoPaterno,
                onValueChange = { viewModel.updateApellidoPaterno(it) },
                label = { Text( text: "Apellido Paterno") },
                modifier = Modifier.fillMaxWidth()
            )
            Spacer(modifier = Modifier.height(8.dp))

            OutlinedTextField(
                value = apellidoMaterno,
                onValueChange = { viewModel.updateApellidoMaterno(it) },
                label = { Text( text: "Apellido Materno") },
                modifier = Modifier.fillMaxWidth()
            )
            Spacer(modifier = Modifier.height(16.dp))

            Row(
                verticalAlignment = Alignment.CenterVertically,
                modifier = Modifier.fillMaxWidth()
            ) {
                OutlinedTextField(
                    value = dia,
                    onValueChange = { if (it.length <= 2) viewModel.updateDia(it) },
                    label = { Text( text: "Día") },
                    keyboardOptions = KeyboardOptions.Default.copy(keyboardType = androidx.compose.ui.text.input.KeyboardType.Number),
                    modifier = Modifier.weight(1f)
                )
                Spacer(modifier = Modifier.width(8.dp))

                OutlinedTextField(
                    value = mes,
                    onValueChange = { if (it.length <= 2) viewModel.updateMes(it) },
                    label = { Text( text: "Mes") },
                    keyboardOptions = KeyboardOptions.Default.copy(keyboardType = androidx.compose.ui.text.input.KeyboardType.Number),
                    modifier = Modifier.weight(1f)
                )
                Spacer(modifier = Modifier.width(8.dp))

                OutlinedTextField(
                    value = anio,
                    onValueChange = { if (it.length <= 4) viewModel.updateAnio(it) },
                    label = { Text( text: "Año") },
                    keyboardOptions = KeyboardOptions.Default.copy(keyboardType = androidx.compose.ui.text.input.KeyboardType.Number),
                    modifier = Modifier.weight(1f)
                )
            }
        }
```

MainActivity.kt   DatosPersonalesScreen.kt ×   ResultadosScreen.kt   PreguntasScreen.kt   DatosPersonalesViewModel.kt   DatosPersonales.kt   Pregunta.kt

```kotlin
15      fun DatosPersonalesScreen(
33          ) {
39          ) {
131                 Row(
132                     horizontalArrangement = Arrangement.SpaceBetween,
133                     modifier = Modifier.fillMaxWidth()
134                 ) {
135                     Button(
136                         onClick = onCancelar,
137                         modifier = Modifier.weight(1f)
138                     ) {
139                         Text( text: "Cancelar")
140                     }
141                     Spacer(modifier = Modifier.width(16.dp))
142                     Button(
143                         onClick = {
144                             if (nombre.isNotBlank() && apellidoPaterno.isNotBlank() && dia.isNotBlank() && mes.isNotBlank() && anio.isNotBlank()) {
145                                 onSiguiente(
146                                     DatosPersonales(
147                                         nombre,
148                                         apellidoPaterno,
149                                         apellidoMaterno,
150                                         dia,
151                                         mes,
152                                         anio,
153                                         sexo
154                                     )
155                                 )
156                             } else {
157                                 mostrarError = true
158                             }
159                         },
160                         modifier = Modifier.weight(1f)
161                     ) {
162                         Text( text: "Siguiente")
163                     }
164                 }
165             }
166         }
167     }
```

```kotlin
16
17  fun guardarDatosEnFirebase(datos: DatosPersonales, calificacion: Int) {
18      val db = FirebaseFirestore.getInstance()
19      val registro = hashMapOf(
20          "nombre" to datos.nombre,
21          "apellidoPaterno" to datos.apellidoPaterno,
22          "apellidoMaterno" to datos.apellidoMaterno,
23          "dia" to datos.dia,
24          "mes" to datos.mes,
25          "anio" to datos.anio,
26          "sexo" to datos.sexo,
27          "calificacion" to calificacion
28      )
29      db.collection( collectionPath: "resultados")
30          .add(registro)
31  }
32
33  @Composable
34  fun ResultadosScreen(
35      datos: DatosPersonales,
36      calificacion: Int,
37      totalPreguntas: Int,
38      onFinalizar: () -> Unit
39  ) {
40      // Cálculo de edad
41      val birthYear = datos.anio.toIntOrNull() ?: 2000
42      val birthMonth = datos.mes.toIntOrNull() ?: 1
43      val birthDay = datos.dia.toIntOrNull() ?: 1
44      val today = LocalDate.now()
45      var edad = today.year - birthYear
46      if (today.monthValue < birthMonth || (today.monthValue == birthMonth && today.dayOfMonth < birthDay)) {
47          edad -= 1
48      }
49
50      // Zodiaco chino
51      val animalesZodiaco = listOf(
52          "Rata", "Buey", "Tigre", "Conejo", "Dragón", "Serpiente",
53          "Caballo", "Cabra", "Mono", "Gallo", "Perro", "Cerdo"
54      )
55      val signoIndex = if (datos.anio.length == 4) (birthYear - 4) % 12 else 0
56      val signoZodiaco = animalesZodiaco[if (signoIndex >= 0) signoIndex else (signoIndex + 12) % 12]
57
58      // Imagen del signo
59      val context = LocalContext.current
```

```kotlin
        val imageName = signoZodiaco.lowercase()
        val imageId = context.resources.getIdentifier(imageName, defType: "drawable", context.packageName)

        // Guardar en Firebase una sola vez
        LaunchedEffect(Unit) {
            guardarDatosEnFirebase(datos, calificacion)
        }

        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(24.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(text = "Hola ${datos.nombre}", fontSize = 28.sp, modifier = Modifier.padding(bottom = 16.dp))
            Text(text = "Tienes $edad años y tu signo zodiacal es:", fontSize = 20.sp)
            if (imageId != 0) {
                Image(
                    painter = painterResource(id = imageId),
                    contentDescription = signoZodiaco,
                    modifier = Modifier.size(120.dp)
                )
            }
            Text(text = signoZodiaco, fontSize = 32.sp, modifier = Modifier.padding(vertical = 16.dp))
            Divider(modifier = Modifier.padding(vertical = 16.dp))
            Text(
                text = "Calificación del examen:",
                fontSize = 20.sp,
                modifier = Modifier.padding(bottom = 8.dp)
            )
            Text(
                text = "$calificacion / $totalPreguntas",
                fontSize = 26.sp,
                color = MaterialTheme.colorScheme.primary
            )
            Spacer(modifier = Modifier.weight(1f))
            Button(
                onClick = onFinalizar,
                modifier = Modifier.fillMaxWidth()
            ) {
                Text( text: "Finalizar")
            }
        }
    }
```

```kotlin
fun PreguntasScreen(
        modifier = Modifier.fillMaxSize(),
        color = Color.White
) {
    LazyColumn(
        modifier = Modifier
            .fillMaxSize()
            .padding(24.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        item {
            Text(
                text = "Cuestionario",
                fontSize = 24.sp,
                modifier = Modifier.padding(bottom = 24.dp)
            )
        }

        preguntas.forEachIndexed { index, pregunta ->
            item {
                Text(
                    text = pregunta.texto,
                    fontSize = 18.sp,
                    modifier = Modifier.padding(vertical = 8.dp)
                )
                pregunta.opciones.forEach { opcion ->
                    Row(
                        verticalAlignment = Alignment.CenterVertically,
                        modifier = Modifier
                            .fillMaxWidth()
                            .padding(vertical = 4.dp)
                    ) {
                        RadioButton(
                            selected = respuestas[index] == opcion,
                            onClick = { respuestas[index] = opcion }
                        )
                        Text(opcion, modifier = Modifier.padding(start = 8.dp))
                    }
                }
                Spacer(modifier = Modifier.height(16.dp))
            }
        }

        item {
            if (mostrarError) {
```

```kotlin
@Composable
fun PreguntasScreen(
    onVolver: () -> Unit,
    onFinalizar: (calificacion: Int, totalPreguntas: Int) -> Unit
) {
    val preguntas = listOf(
        Pregunta(
            texto = "¿Cuánto es 7 + 5?",
            opciones = listOf("11", "12", "13", "14"),
            respuestaCorrecta = "12"
        ),
        Pregunta(
            texto = "¿Cuál es el resultado de 9 x 3?",
            opciones = listOf("27", "21", "18", "24"),
            respuestaCorrecta = "27"
        ),
        Pregunta(
            texto = "¿Quién escribió 'Cien años de soledad'?",
            opciones = listOf("Gabriel García Márquez", "Octavio Paz", "Mario Vargas Llosa", "Jorge Luis Borges"),
            respuestaCorrecta = "Gabriel García Márquez"
        ),
        Pregunta(
            texto = "¿Cuál es la raíz cuadrada de 81?",
            opciones = listOf("9", "8", "7", "6"),
            respuestaCorrecta = "9"
        ),
        Pregunta(
            texto = "¿Quién es el autor de 'Don Quijote de la Mancha'?",
            opciones = listOf("Miguel de Cervantes", "Lope de Vega", "Benito Pérez Galdós", "Federico García Lorca"),
            respuestaCorrecta = "Miguel de Cervantes"
        ),
        Pregunta(
            texto = "¿Cuánto es 15 dividido entre 3?",
            opciones = listOf("5", "4", "6", "3"),
            respuestaCorrecta = "5"
        )
    )

    val respuestas = remember { mutableStateMapOf<Int, String>() }
    var mostrarError by remember { mutableStateOf( value  false) }

    Surface(
        modifier = Modifier.fillMaxSize(),
        color = Color.White
```

```kotlin
fun PreguntasScreen(
    ) {
        ) {

        item {
            if (mostrarError) {
                Text(
                    text = "Por favor, responde todas las preguntas",
                    color = Color.Red,
                    modifier = Modifier.padding(bottom = 8.dp)
                )
            }
            Row(
                horizontalArrangement = Arrangement.SpaceBetween,
                modifier = Modifier.fillMaxWidth()
            ) {
                Button(
                    onClick = onVolver,
                    modifier = Modifier.weight(1f)
                ) {
                    Text( text "Volver")
                }
                Spacer(modifier = Modifier.width(16.dp))
                Button(
                    onClick = {
                        if (respuestas.size == preguntas.size) {
                            val correctas = preguntas.indices.count { i ->
                                respuestas[i] == preguntas[i].respuestaCorrecta
                            }
                            onFinalizar(correctas, preguntas.size)
                        } else {
                            mostrarError = true
                        }
                    },
                    modifier = Modifier.weight(1f)
                ) {
                    Text( text "Finalizar")
                }
            }
        }
    }
}
```



```kotlin
package com.idgs.examen2doparcial

data class DatosPersonales{
    val nombre: String,
    val apellidoPaterno: String,
    val apellidoMaterno: String,
    val dia: String,
    val mes: String,
    val anio: String,
    val sexo: String
}
```

```kotlin
package com.idgs.examen2doparcial

import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.State
import androidx.lifecycle.ViewModel

class DatosPersonalesViewModel : ViewModel() {
    private val _nombre = mutableStateOf( value "")
    val nombre: State<String> get() = _nombre

    private val _apellidoPaterno = mutableStateOf( value "")
    val apellidoPaterno: State<String> get() = _apellidoPaterno

    private val _apellidoMaterno = mutableStateOf( value "")
    val apellidoMaterno: State<String> get() = _apellidoMaterno

    private val _dia = mutableStateOf( value "")
    val dia: State<String> get() = _dia

    private val _mes = mutableStateOf( value "")
    val mes: State<String> get() = _mes

    private val _anio = mutableStateOf( value "")
    val anio: State<String> get() = _anio

    private val _sexo = mutableStateOf( value "Masculino")
    val sexo: State<String> get() = _sexo

    fun updateNombre(value: String) { _nombre.value = value }
    fun updateApellidoPaterno(value: String) { _apellidoPaterno.value = value }
    fun updateApellidoMaterno(value: String) { _apellidoMaterno.value = value }
    fun updateDia(value: String) { _dia.value = value }
    fun updateMes(value: String) { _mes.value = value }
    fun updateAnio(value: String) { _anio.value = value }
    fun updateSexo(value: String) { _sexo.value = value }
}
```

```kotlin
package com.idgs.examen2doparcial

data class Pregunta(
    val texto: String,
    val opciones: List<String>,
    val respuestaCorrecta: String
)
```