Universidad Tecnológica de León.

Ingeniería en desarrollo y gestión de software.

Tema: Examen Segundo Parcial.

PTC. Roberto Cardiel Rodríguez.

Materia:
Desarrollo para dispositivos inteligentes.

Presenta:

Pedroza Moctezuma José Rubén

19002243 - IDGS904

Fecha:

03 de julio del 2025

MainActivity:

```
package com.ruben.zoodiacochino

// Importaciones necesarias para la actividad principal, Jetpack Compose y Navegación.
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.padding // Para aplicar padding.
import androidx.compose.material3.Scaffold // Componente Scaffold para la estructura básica de la app Material3.
import androidx.compose.ui.Modifier
import androidx.navigation.NavType // Para definir tipos de argumentos de navegación.
import androidx.navigation.compose.* // Funciones de Navegación para Compose (NavHost, composable, rememberNavController).
import androidx.navigation.navArgument // Para definir argumentos de navegación.
// Importaciones de las diferentes pantallas (Composables) de la aplicación.
import com.ruben.zoodiacochino.iu.pantallas.Ventana1
import com.ruben.zoodiacochino.iu.pantallas.Ventana2
import com.ruben.zoodiacochino.iu.pantallas.Ventana3

/**
 * Actividad principal de la aplicación.
 * Configura el contenido de la UI usando Jetpack Compose y establece el sistema de navegación.
 */
class MainActivity : ComponentActivity() {
    /**
     * Método llamado cuando la actividad es creada por primera vez.
     * Aquí se inicializa la UI.
     */
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState) // Llama a la implementación de la superclase.

        // Establece el contenido de la actividad usando Jetpack Compose.
        setContent {
            // Crea y recuerda un NavController. Este controlador es responsable de
            // gestionar la pila de backstack de la aplicación y las operaciones de navegación.
            val navController = rememberNavController()
```

```
        // Scaffold proporciona una estructura básica para pantallas Material Design.
        // `innerPadding` se usa para asegurar que el contenido no se superponga con
        // elementos de Scaffold como barras de aplicación o botones flotantes (si se usaran).
        Scaffold { innerPadding ->
            // NavHost es el contenedor donde se muestran los diferentes destinos (pantallas) de navegación.
            NavHost(
                navController = navController, // El controlador de navegación a usar.
                startDestination = "ventana1", // La ruta del destino inicial que se mostrará al iniciar.
                modifier = Modifier.padding(innerPadding) // Aplica el padding interno de Scaffold.
            ) {
                // Define la primera pantalla (ruta "ventana1").
                composable("ventana1") {
                    // Muestra el Composable Ventana1, pasándole el navController.
                    Ventana1(navController)
                }

                // Define la segunda pantalla (ruta "ventana2/{userId}").
                // "{userId}" define un argumento de ruta que se pasará a esta pantalla.
                composable(
                    route = "ventana2/{userId}", // La plantilla de la ruta.
                    arguments = listOf(navArgument("userId") { type = NavType.StringType }) // Define el tipo del argumento "userId".
                ) { backStackEntry -> // `backStackEntry` contiene los argumentos pasados a esta ruta.
                    // Extrae el argumento "userId" de la pila de backstack.
                    // Proporciona un valor por defecto ("") si el argumento no se encuentra.
                    val userId = backStackEntry.arguments?.getString("userId") ?: ""
                    // Muestra el Composable Ventana2, pasándole el navController y el userId.
                    Ventana2(navController, userId)
                }

                // Define la tercera pantalla (ruta
```

```
"ventana3/{userId}/{calificacion}").
                // Esta ruta tiene dos argumentos: "userId" y "calificacion".
                composable(
                    route = "ventana3/{userId}/{calificacion}", // Plantilla de la
ruta.
                    arguments = listOf( // Define los tipos de los argumentos.
                        navArgument("userId") { type = NavType.StringType },
                        navArgument("calificacion") { type = NavType.StringType }
// Nota: La calificación se pasa como String y luego se convierte.
                    )
                ) { backStackEntry ->
                    // Extrae el argumento "userId".
                    val userId = backStackEntry.arguments?.getString("userId")
?: ""
                    // Extrae el argumento "calificacion", lo convierte a Int,
                    // o usa 0 si la conversión falla o el argumento no está
presente.
                    val calificacion =

backStackEntry.arguments?.getString("calificacion")?.toIntOrNull() ?: 0
                    // Muestra el Composable Ventana3, pasándole el
navController, userId y calificacion.
                    Ventana3(navController, userId, calificacion)
                }
            }
        }
    }
}
```

Ventana1:

```
package com.ruben.zoodiacochino.iu.pantallas

// Importaciones necesarias para UI, estado, navegación y Firebase
import android.util.Log
import androidx.compose.foundation.layout.*
import androidx.compose.material3.Button
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import com.google.firebase.firestore.FirebaseFirestore

/**
 * Composable que representa la pantalla del formulario de registro (Ventana1).
 * Permite al usuario ingresar sus datos personales y los guarda en Firebase Firestore.
 *
 * @param navController Controlador para gestionar la navegación a otras pantallas.
 */
@Composable
fun Ventana1(navController: NavController) {
    // Instancia de FirebaseFirestore para operaciones de base de datos.
    val db = FirebaseFirestore.getInstance()

    // Estados para almacenar los valores de los campos del formulario.
    var nombre by remember { mutableStateOf("") }
    var apellidoP by remember { mutableStateOf("") }
    var apellidoM by remember { mutableStateOf("") }
    var dia by remember { mutableStateOf("") }
    var mes by remember { mutableStateOf("") }
```

```
    var anio by remember { mutableStateOf("") }
    var sexo by remember { mutableStateOf("Masculino") }

    /**
     * Lambda para limpiar todos los campos del formulario a sus valores por
defecto.
     */
    val limpiarCampos = {
        nombre = ""
        apellidoP = ""
        apellidoM = ""
        dia = ""
        mes = ""
        anio = ""
        sexo = "Masculino"
    }

    // Estructura principal de la UI usando una Columna.
    Column(
        modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text("Formulario de Registro", style =
MaterialTheme.typography.titleLarge)

        // Campos de texto para la entrada de datos del usuario.
        OutlinedTextField(value = nombre, onValueChange = { nombre = it },
label = { Text("Nombre") })
        OutlinedTextField(value = apellidoP, onValueChange = { apellidoP = it
}, label = { Text("Apellido Paterno") })
        // ... (otros OutlinedTextFields para apellidoM, dia, mes, anio)

        Text("Sexo:")
        // Selección de sexo mediante RadioButtons.
        Row {
            RadioButton(selected = sexo == "Masculino", onClick = { sexo =
"Masculino" })
            Text("Masculino")
            // ... (RadioButton para Femenino)
```

```
        }

        Spacer(modifier = Modifier.height(16.dp))

        // Botones de acción: Limpiar y Siguiente.
        Row(horizontalArrangement = Arrangement.SpaceEvenly) {
            // Botón para ejecutar la función limpiarCampos.
            Button (onClick = limpiarCampos, modifier = Modifier.padding(8.dp))
{
                Text("Limpiar")
            }

            // Botón para guardar los datos en Firestore y navegar a la siguiente
pantalla.
            Button(onClick = {
                // Prepara los datos del usuario para Firestore.
                val usuario = hashMapOf(
                    "nombre" to nombre,
                    "apellidoP" to apellidoP,
                    "apellidoM" to apellidoM,
                    "dia" to dia.toIntOrNull(),
                    "mes" to mes.toIntOrNull(),
                    "anio" to anio.toIntOrNull(),
                    "sexo" to sexo
                )

                // Guarda el usuario en la colección "usuarios" de Firestore.
                db.collection("usuarios")
                    .add(usuario)
                    .addOnSuccessListener { document ->
                        Log.d("Ventana1", "Usuario guardado con ID:
${document.id}")
                        // Navega a Ventana2 pasando el ID del nuevo usuario.
                        navController.navigate("ventana2/${document.id}")
                    }
                    .addOnFailureListener {
                        Log.e("Ventana1", "Error al guardar: ${it.message}")
                    }
            }) {
                Text("Siguiente")
```

```
            }
        }
    }
}
```

Ventana2:

```
package com.ruben.zoodiacochino.iu.pantallas

// Importaciones necesarias para UI, estado, navegación y Firebase
import android.util.Log // No se usa explícitamente, pero es común para debugging.
// import android.widget.Button // Importación de View System, no usada aquí.
import androidx.compose.foundation.layout.*
// import androidx.compose.material.* // Importación de Material (versión anterior), no usada si se usa Material3.
import androidx.compose.material3.Button // Botón de Material Design 3.
import androidx.compose.material3.MaterialTheme // Tema de Material Design 3.
// import androidx.compose.material3.OutlinedTextField // No se usa en este Composable.
import androidx.compose.material3.RadioButton // RadioButton de Material Design 3.
import androidx.compose.material3.Text // Componente Text de Material Design 3.
import androidx.compose.runtime.* // Para `remember` y `mutableStateListOf`.
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController // Para la navegación.
import com.google.firebase.firestore.FirebaseFirestore // Para interactuar con Firestore.

/**
 * Composable que representa la pantalla del examen (Ventana2).
 * Muestra una lista de preguntas con opciones de respuesta.
 * Guarda los resultados en Firebase Firestore y navega a la pantalla de resultados.
 *
 * @param navController Controlador para gestionar la navegación a otras pantallas.
 * @param userId ID del usuario que está realizando el examen, recibido de la pantalla anterior.
 */
```

```
@Composable
fun Ventana2(navController: NavController, userId: String) {
    // Instancia de FirebaseFirestore para operaciones de base de datos.
    val db = FirebaseFirestore.getInstance()

    // Lista de preguntas. Cada elemento es un Triple: (texto de la pregunta, lista de opciones, índice de la respuesta correcta).
    val preguntas = listOf(
        Triple("¿Cuál es la suma de 2 + 2?", listOf("8", "6", "4", "3"), 2),
        Triple("¿Cuál es la capital de Francia?", listOf("Londres", "Madrid", "París", "Roma"), 2),
        Triple("¿Qué color resulta de mezclar azul y amarillo?", listOf("Verde", "Naranja", "Rosa", "Rojo"), 0),
        Triple("¿Cuántos días tiene una semana?", listOf("5", "6", "7", "8"), 2),
        Triple("¿Quién escribió Don Quijote?", listOf("García Márquez", "Cervantes", "Shakespeare", "Neruda"), 1),
        Triple("¿Cuál es el planeta más grande?", listOf("Marte", "Venus", "Júpiter", "Tierra"), 2)
    )

    // Lista mutable para almacenar las respuestas seleccionadas por el usuario (índice de la opción).
    // Se inicializa con `null` para cada pregunta, indicando que no se ha seleccionado ninguna respuesta.
    val respuestasUsuario = remember { mutableStateListOf<Int?>(null, null, null, null, null, null) }

    // Estructura principal de la UI usando una Columna.
    Column(Modifier.padding(16.dp)) {
        Text("Examen", style = MaterialTheme.typography.titleLarge) // Título de la pantalla.

        // Itera sobre la lista de preguntas para mostrar cada una.
        preguntas.forEachIndexed { index, (pregunta, opciones, _) ->
            Text("${index + 1}. $pregunta") // Muestra el número y el texto de la pregunta.

            // Fila para mostrar las opciones de respuesta horizontalmente.
            Row(
                modifier = Modifier
```

```
                .fillMaxWidth()
                .padding(vertical = 8.dp),
            horizontalArrangement = Arrangement.SpaceEvenly // Distribuye
el espacio entre las opciones.
        ) {
            // Itera sobre las opciones de la pregunta actual.
            opciones.forEachIndexed { i, opcion ->
                // Fila para cada RadioButton y su texto.
                Row(
                    modifier = Modifier.weight(1f), // Asigna el mismo espacio a
cada opción.
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    RadioButton(
                        selected = respuestasUsuario[index] == i, // Marca como
seleccionado si es la respuesta actual del usuario.
                        onClick = { respuestasUsuario[index] = i } // Actualiza la
respuesta del usuario al hacer clic.
                    )
                    Text(opcion) // Muestra el texto de la opción.
                }
            }
        }
        Spacer(modifier = Modifier.height(16.dp)) // Espacio entre
preguntas.
    }


    // Botón para finalizar el examen.
    Button(onClick = {
        // Calcula el número de respuestas correctas.
        val aciertos = preguntas.indices.count {
            respuestasUsuario[it] == preguntas[it].third // Compara la
respuesta del usuario con la respuesta correcta.
        }

        // Guarda los resultados (ID del usuario y número de aciertos) en la
colección "resultados" de Firestore.
        db.collection("resultados")
            .add(mapOf("userId" to userId, "aciertos" to aciertos))
```

```
                .addOnSuccessListener {
                    // Si se guarda con éxito, navega a Ventana3 pasando el
userId y el número de aciertos.
                    navController.navigate("ventana3/$userId/$aciertos")
                }
            // .addOnFailureListener { /* Opcional: Manejar error al guardar */ }
        }) {
            Text("Terminar")
        }
    }
}
```

Ventana3:

```
package com.ruben.zoodiacochino.iu.pantallas

// Importaciones necesarias para UI, estado, Firebase, recursos y utilidades de fecha.
import android.util.Log
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext // Para acceder a recursos.
import androidx.compose.ui.res.painterResource // Para cargar imágenes.
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import com.google.firebase.firestore.FirebaseFirestore
import java.util.Calendar
import java.util.GregorianCalendar

/**
 * Composable que representa la pantalla de resultados y perfil del usuario (Ventana3).
 * Muestra el nombre, edad, signo zodiacal (con imagen) y la calificación obtenida en el examen.
 * Los datos del usuario se cargan desde Firebase Firestore.
 *
 * @param navController Controlador para gestionar la navegación (no se usa activamente para navegar desde aquí).
 * @param userId ID del usuario cuyos datos se van a mostrar.
 * @param calificacion Calificación obtenida por el usuario en el examen.
 */
@Composable
fun Ventana3(navController: NavController, userId: String, calificacion: Int) {
    // Instancia de FirebaseFirestore.
```

```
    val db = FirebaseFirestore.getInstance()

    // Estados para almacenar la información del usuario y el estado de
carga/error.
    var nombreCompleto by remember { mutableStateOf<String?>(null) }
    var edad by remember { mutableStateOf<Int?>(null) }
    var signo by remember { mutableStateOf<String?>(null) }
    var isLoading by remember { mutableStateOf(true) } // Indica si los datos
se están cargando.
    var errorMessage by remember { mutableStateOf<String?>(null) } //
Almacena mensajes de error.

    // Contexto local para acceder a recursos como drawables.
    val contexto = LocalContext.current

    // Efecto lanzado cuando `userId` cambia, para cargar los datos del
usuario.
    LaunchedEffect(userId) {
        isLoading = true
        errorMessage = null // Resetea el mensaje de error.
        // Resetea los datos del usuario.
        nombreCompleto = null
        edad = null
        signo = null

        if (userId.isNotEmpty()) {
            // Obtiene el documento del usuario desde la colección "usuarios"
en Firestore.
            db.collection("usuarios").document(userId)
                .get()
                .addOnSuccessListener { documentSnapshot ->
                    if (documentSnapshot.exists()) {
                        // Extrae los datos del documento.
                        val nombre = documentSnapshot.getString("nombre") ?: ""
                        val apP = documentSnapshot.getString("apellidoP") ?: ""
                        val apM = documentSnapshot.getString("apellidoM") ?: ""
                        val anio = (documentSnapshot.getLong("anio")?.toInt())
                        val mes = (documentSnapshot.getLong("mes")?.toInt())
                        val dia = (documentSnapshot.getLong("dia")?.toInt())
```

```
                    if (anio != null && mes != null && dia != null) {
                        // Construye el nombre completo y calcula edad y signo.
                        nombreCompleto = "$nombre $apP $apM".trim().replace("
", " ")
                        if (nombreCompleto!!.isBlank()) nombreCompleto =
"Usuario Desconocido"

                        edad = calcularEdad(dia, mes, anio)
                        signo = calcularSigno(anio)
                    } else {
                        // Manejo de error si los datos de fecha son incompletos.
                        Log.e("Ventana3", "Datos de fecha incompletos para el
usuario: $userId. Año: $anio, Mes: $mes, Día: $dia")
                        errorMessage = "No se pudieron cargar los datos de fecha
del usuario."
                    }
                } else {
                    // Manejo de error si el usuario no se encuentra.
                    Log.e("Ventana3", "No se encontró el documento del usuario
con ID: $userId")
                    errorMessage = "Usuario no encontrado."
                }
                isLoading = false // Finaliza el estado de carga.
            }
            .addOnFailureListener { exception ->
                // Manejo de error si falla la obtención de datos.
                Log.e("Ventana3", "Error al obtener usuario con ID $userId:
${exception.message}", exception)
                errorMessage = "Error al cargar los datos del usuario."
                isLoading = false
            }
        } else {
            // Manejo de caso donde el userId está vacío.
            Log.w("Ventana3", "userId está vacío, no se puede cargar la
información del usuario.")
            errorMessage = "No se proporcionó un ID de usuario válido."
            isLoading = false
        }
    }
```

```
    // Estructura principal de la UI usando una Columna.
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(20.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        // Muestra diferentes UI según el estado (cargando, error, datos
cargados).
        when {
            isLoading -> {
                CircularProgressIndicator() // Muestra un indicador de progreso.
                Spacer(modifier = Modifier.height(16.dp))
                Text("Cargando datos del usuario...")
            }
            errorMessage != null -> {
                Text("Error: $errorMessage", color =
MaterialTheme.colorScheme.error) // Muestra el mensaje de error.
            }
            // Si los datos se cargaron correctamente:
            nombreCompleto != null && edad != null && signo != null -> {
                Text("Hola ${nombreCompleto ?: "Usuario"}", style =
MaterialTheme.typography.titleLarge)
                Spacer(modifier = Modifier.height(8.dp))
                Text("Tienes ${edad ?: "desconocida"} años y tu signo zodiacal
es:")
                Spacer(modifier = Modifier.height(4.dp))
                Text(signo?.replaceFirstChar { it.uppercase() } ?: "Desconocido",
style = MaterialTheme.typography.headlineMedium)

                Spacer(modifier = Modifier.height(16.dp))
                // Prepara el nombre del signo para buscar el recurso drawable.
                val nombreSignoParaRecurso = signo?.lowercase()?.filter {
it.isLetter() } ?: ""
                if (nombreSignoParaRecurso.isNotEmpty()) {
                    // Obtiene el ID del recurso drawable basado en el nombre del
signo.
                    val imagenId =
contexto.resources.getIdentifier(nombreSignoParaRecurso, "drawable",
```

```
contexto.packageName)
                if (imagenId != 0) {
                    Image( // Muestra la imagen del signo zodiacal.
                        painter = painterResource(id = imagenId),
                        contentDescription = "Imagen del signo $signo",
                        modifier = Modifier.size(120.dp)
                    )
                } else {
                    Text("Imagen para '$signo' no encontrada (recurso:
'$nombreSignoParaRecurso').")
                }
            } else {
                Text("Nombre de signo inválido para buscar imagen.")
            }

            Spacer(modifier = Modifier.height(24.dp))
            // Muestra la calificación obtenida.
            Text("Calificación: $calificacion / 6", style =
MaterialTheme.typography.titleMedium)
        }
        // Caso por defecto si los datos no se pudieron cargar y no hay error
específico.
        else -> {
            Text("No se pudo cargar la información del usuario. Verifica la
conexión o el ID del usuario.")
        }
    }
  }
}

/**
 * Calcula la edad de una persona a partir de su fecha de nacimiento.
 *
 * @param dia Día de nacimiento.
 * @param mes Mes de nacimiento (1-12).
 * @param anio Año de nacimiento.
 * @return La edad calculada.
 */
fun calcularEdad(dia: Int, mes: Int, anio: Int): Int {
    val hoy = Calendar.getInstance()
```

```
    val nacimiento = GregorianCalendar(anio, mes - 1, dia) // Mes en
Calendar es 0-indexado.

    var edadCalculada = hoy.get(Calendar.YEAR) -
nacimiento.get(Calendar.YEAR)

    // Ajusta la edad si aún no ha cumplido años este año.
    if (hoy.get(Calendar.MONTH) < nacimiento.get(Calendar.MONTH) ||
        (hoy.get(Calendar.MONTH) == nacimiento.get(Calendar.MONTH) &&
                hoy.get(Calendar.DAY_OF_MONTH) <
nacimiento.get(Calendar.DAY_OF_MONTH))
    ) {
        edadCalculada--
    }
    return edadCalculada
}

/**
 * Calcula el signo del zodiaco chino basado en el año de nacimiento.
 *
 * @param anioNacimiento Año de nacimiento.
 * @return El nombre del signo zodiacal chino en minúsculas.
 */
fun calcularSigno(anioNacimiento: Int): String {
    val signos = listOf(
        "rata", "buey", "tigre", "conejo", "dragon", "serpiente",
        "caballo", "cabra", "mono", "gallo", "perro", "cerdo"
    )
    // El ciclo del zodiaco chino se basa en un ciclo de 12 años, comenzando
desde 1900 (Rata).
    var indice = (anioNacimiento - 1900) % 12
    if (indice < 0) { // Ajuste para años anteriores a 1900.
        indice = (indice + 12) % 12
    }
    return signos[indice]
}
```