Documentación de la pantalla inicial en Jetpack Compose

Descripción:

Se ha implementado una pantalla de bienvenida a la app utilizando **Jetpack Compose** para la interfaz de usuario.

ESTRUCTURA DEL CÓDIGO

```
@Composable
fun contenidoPantallaConInformacionDeLaApp(navController:
NavController) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center,
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        TextoBienvenida()
        Spacer(modifier = Modifier.height(16.dp))
        InstruccionesUso()
        Spacer(modifier = Modifier.height(18.dp))
        PhotoCarousel()
        Spacer(modifier = Modifier.height(16.dp))
        ActionButtons(navController = navController)
    }
}
```

Componentes:

- TextoBienvenida(): Muestra un mensaje de bienvenida.
- InstruccionesUso(): Muestra las instrucciones de uso de la aplicación.
- PhotoCarousel(): Muestra un carrusel de fotos con las pantallas de la app.
- ActionButtons(navController = navController): Muestra los botones de acción(navegar al login o al register)

Comentarios Adicionales:

- Verifica que la navegación con navController sea fluida y que no haya problemas al cambiar de pantalla.
- Realiza pruebas exhaustivas de la pantalla en diferentes dispositivos y tamaños de pantalla para asegurar que se vea y funcione correctamente en todos los casos.

Documentación del Registro en Jetpack Compose y Firestore

Descripción:

Se ha implementado un registro de usuarios utilizando **Jetpack Compose** para la interfaz de usuario y **Firestore** para la base de datos en la nube.

Estructura del Código:

```
@Composable
fun RegistroUsuarioScreen(viewModel: RegistroViewModel) {
    var nombre by remember { mutableStateOf("") }
    var correo by remember { mutableStateOf("") }

    Column(modifier = Modifier.padding(16.dp)) {
        TextField(
            value = nombre,
            onValueChange = { nombre = it },
            label = { Text("Nombre") }
        )
        Spacer(modifier = Modifier.height(8.dp))
        TextField(
            value = correo,
            onValueChange = { correo = it },
            label = { Text("Correo") }
        )
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = { viewModel.registrarUsuario(nombre, correo) }) {
            Text("Registrar")
        }
    }
}
```

Función para Registrar Usuario en Firestore:

```
class RegistroViewModel : ViewModel() {
    private val db = Firebase.firestore

    fun registrarUsuario(nombre: String, correo: String) {
        val usuario = hashMapOf(
            "nombre" to nombre,
            "correo" to correo,
            "fechaRegistro" to FieldValue.serverTimestamp()
        )

        db.collection("usuarios")
            .add(usuario)
            .addOnSuccessListener { Log.d("Firestore", "Usuario registrado exitosamente") }
            .addOnFailureListener { e -> Log.w("Firestore", "Error al registrar usuario", e) }
    }
}
```

Datos Registrados:

- **Nombre:** Juan Pérez
- **Correo:** juan.perez@example.com
- **Fecha de Registro:** 2024-10-30 17:30

Flujo del Registro:

1. **Interfaz de Usuario (Jetpack Compose):**
    - El usuario ingresa su nombre y correo en el formulario.
    - Al presionar el botón "Registrar", se llama a la función del ViewModel.
2. **Registro en Firestore:**
    - Los datos del usuario se guardan en la colección `usuarios`.
    - Se registra un timestamp automático para la fecha del registro.

Errores Conocidos y Soluciones:

1. **Error:** Problema de conexión con Firestore.

   Solución: Verificar la configuración de permisos en Firestore y la conexión a internet.

2. **Error:** Campos vacíos.

   Solución: Validar los campos en el ViewModel antes de enviar los datos.

Puntos de interés:

FUNCIÓN PARA VALIDAR EL CUI

```
                fun isValidCUI(cui: String): Boolean {
                        if (cui.isBlank()) {
                            println("CUI vacío")
                            return false
                        }

                        val cuiRegExp = Regex("^[0-9]{4}\\s?[0-9]{5}\\s?[0-9]{3,4}$")
                        if (!cuiRegExp.matches(cui)) {
                            println("CUI con formato inválido")
                            return false
                        }

                        val cleanCUI = cui.replace("\\s".toRegex(), "")
                        val length = cleanCUI.length

                        if (length != 12 && length != 13) {
                            println("CUI debe tener 12 o 13 dígitos")
                            return false
                        }

                        val numero = cleanCUI.substring(0, 8)
                        val verificador = cleanCUI.substring(8, 9).toIntOrNull() ?: run {
                            println("Verificador inválido")
                            return false
                        }

                        val depto = try {
                            cleanCUI.substring(9, 11).toInt()
                        } catch (e: Exception) {
                            println("Error al extraer departamento: ${e.message}")
                            return false
                        }

                        val muni = try {
                            if (length == 13) {
                                cleanCUI.substring(11, 13).toInt()
                            } else { // length == 12
                                cleanCUI.substring(11, 12).toInt()
                            }
                        } catch (e: Exception) {
                            println("Error al extraer municipio: ${e.message}")
                            return false
                        }

                        val munisPorDepto = listOf(
                            17, 8, 16, 16, 13, 14, 19, 8, 24, 21, 9, 30,
                            32, 21, 8, 17, 14, 5, 11, 11, 7, 17
                        )

                        if (depto == 0 || muni == 0) {
                            println("CUI con código de municipio o departamento inválido.")
                            return false
                        }

                        if (depto > munisPorDepto.size) {
                            println("CUI con código de departamento inválido.")
                            return false
                        }

                        if (muni > munisPorDepto[depto - 1]) {
                            println("CUI con código de municipio inválido.")
                            return false
                        }

                        var total = 0
                        for (i in numero.indices) {
                            val digit = numero[i].toString().toIntOrNull() ?: run {
                                println("Número inválido en posición $i")
                                return false
                            }
                            total += digit * (i + 2)
                        }
                        val modulo = total % 11
                        println("CUI con módulo: $modulo")

                        return modulo == verificador
                    }

            }
```

Formato de Entrada:

- La función verifica que el CUI no esté vacío y que cumpla con un formato específico, utilizando una expresión regular. Este formato permite espacios opcionales y requiere 12 o 13 dígitos en total.
- Si el formato es inválido, se devuelve false y se imprime un mensaje de error.

Validación de Departamento y Municipio:

- La función extrae los códigos de departamento y municipio. Luego, comprueba que ambos códigos sean válidos para Guatemala.
- Se verifica que los códigos de municipio y departamento no sean cero, y que correspondan a los valores permitidos según la lista munisPorDepto, la cual contiene la cantidad de municipios en cada departamento.
- Si los valores extraídos no son válidos, se muestra un mensaje de error y se retorna false.

Cálculo del Módulo y Verificación:

- La función utiliza los primeros 8 dígitos del CUI para calcular un valor de módulo y lo compara con el dígito verificador, ubicado en la posición 9 del CUI.
- Si el módulo coincide con el verificador, el CUI es válido y la función retorna true. En caso contrario, retorna false.

Comentarios Adicionales:

- Se recomienda implementar validación adicional para evitar registros
- duplicados. Futuras mejoras podrían incluir una notificación visual al usuario tras el registro exitoso.

Autor:

Cristina Nájera

Documentación de las Notificaciones en Jetpack Compose

Descripción:

Esta pantalla muestra las notificaciones relacionadas con el estado de las alertas del usuario, utilizando **Jetpack Compose** para la interfaz de usuario. La pantalla incluye un título y una lista de notificaciones.

Estructura del Código:

```
@Composable
fun PantallaDeNotificacion(
    navController: NavController
) {
    val context = LocalContext.current
    val alertService = AlertService(context)

    LaunchedEffect(Unit) {
        alertService.listenChangesOnAlerts()
    }
    val userId = FirebaseAuth.getInstance().currentUser?.uid ?: ""
    val notificaciones = NotificacionRepository.obtenerNotificaciones(userId)
    contenidoPantallaDeNotificacion(navController, notificaciones)
}
```

Propósito General de la Función:

- PantallaDeNotificacion es un composable en Jetpack Compose que muestra la pantalla de notificaciones. Se encarga de gestionar la lista de notificaciones del usuario en función de su userId.
- val userId = FirebaseAuth.getInstance().currentUser?.uid ?: "": obtiene el ID del usuario autenticado actual mediante Firebase. Si no hay usuario autenticado, se asigna una cadena vacía.
- val notificaciones = NotificacionRepository.obtenerNotificaciones(userId): llama al repositorio de notificaciones (NotificacionRepository) para obtener la lista de notificaciones del usuario con el userId especificado.

FUNCIÓN PANTALLA NOTIFICACIÓN:

```
@Composable
fun contenidoPantallaDeNotificacion(
    navController: NavController,
    notificaciones: List<Notificacion>
) {
    Box(modifier = Modifier.fillMaxWidth()) {
        Column(
            modifier = Modifier.fillMaxWidth(),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Top
        ) {
            Spacer(modifier = Modifier.height(130.dp))
            Text(
                text = "Notificaciones del Estado de tus Alertas",
                style = MaterialTheme.typography.titleLarge,
                fontFamily = fontFamily
            )

            LazyColumn(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(16.dp),
                verticalArrangement = Arrangement.spacedBy(8.dp)
            ) {
                items(notificaciones) { notificacion ->
                    NotificacionItem(notificacion)
                }
            }
        }
    }
}
```

Propósito General de la Función:

- La función muestra el contenido de la pantalla de notificaciones, que incluye el título de la sección y una lista de notificaciones.
- notificaciones: List<Notificacion>: lista de notificaciones del usuario que se mostrará en la pantalla.

Función NotificacionItem:

```
@Composable
fun NotificacionItem(notificacion: Notificacion) {
    Card(
        modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth()
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(text = "Alerta ID: ${notificacion.alertId}", fontWeight =
FontWeight.Bold)
            Text(text = "Estado: ${notificacion.estado}")
            Text(text = "Detalles: ${notificacion.message}")
            Text(text = "Tipo de Alerta: ${notificacion.alertType}")
            Text(text = "Latitud: ${notificacion.latitud}")
            Text(text = "Longitud: ${notificacion.longitud}")
        }
    }
}
```

Descripción: Muestra los detalles de una notificación en una tarjeta (Card).

Autor:

Cristina Nájera