RECM

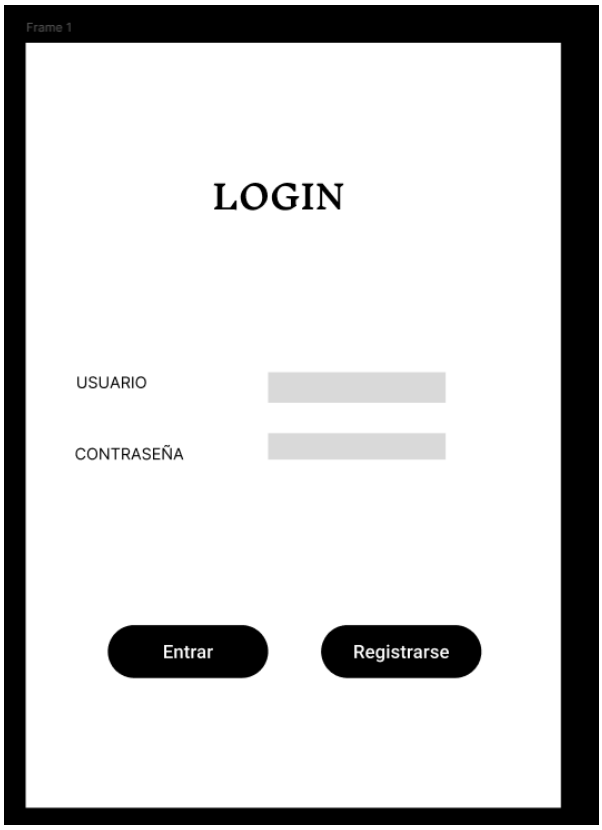
Hecho por Maikel Yoel Romero Loayza 2º DAM

Índice

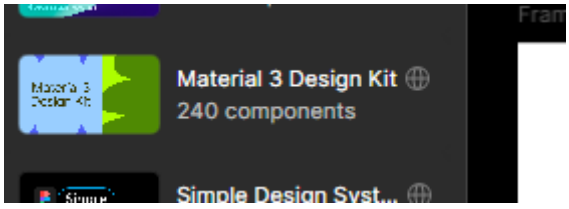
1. VENTANAS

LOGIN

En primer lugar, tenemos la ventana del “Login” en el que contamos con 2 campos para introducir el usuario que en este caso sería un correo electrónico y el otro para introducir la contraseña del usuario:

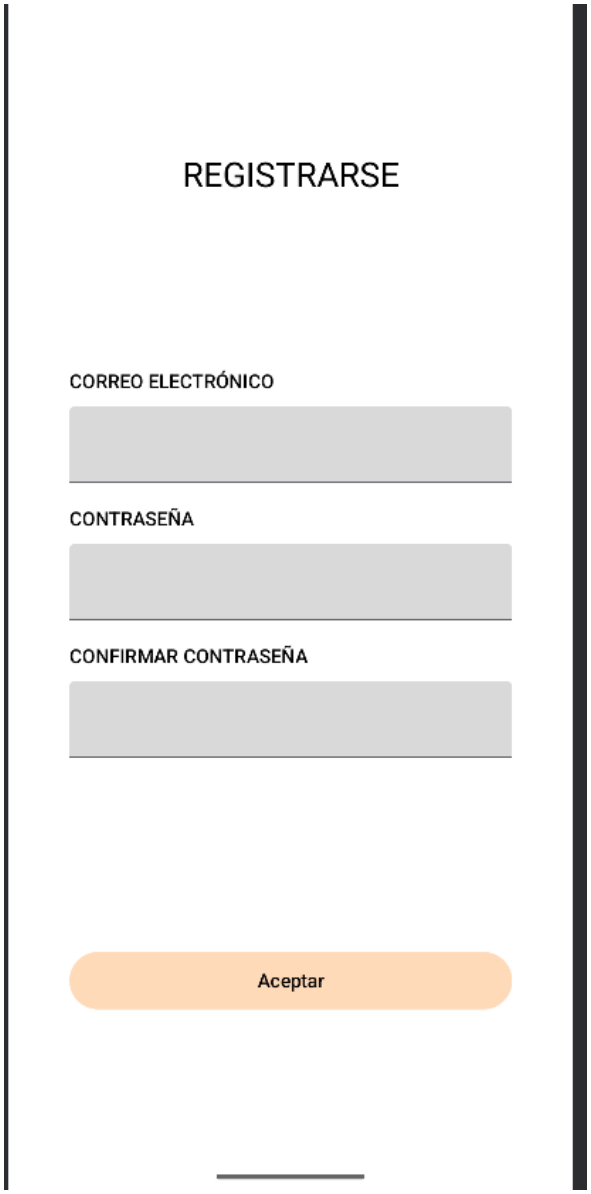


Los botones en este caso los he cogido del apartado “assets” que podemos encontrar en el panel izquierdo, y del siguiente paquete de componentes:



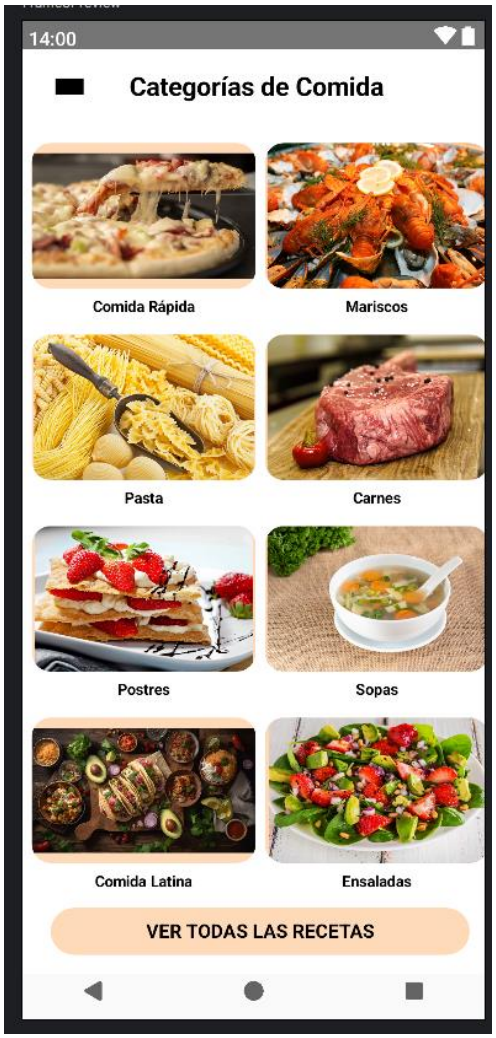
REGISTRO

En esta segunda ventana tenemos tres campos para hacer el registro con un correo y una contraseña, el tercer campo es para confirmar la contraseña. Decido hacerlo así porque puede haber gente que no maneje muy bien los dispositivos inteligentes y es mucho más fácil hacerlo sencillo y claro:



CATEGORIAS

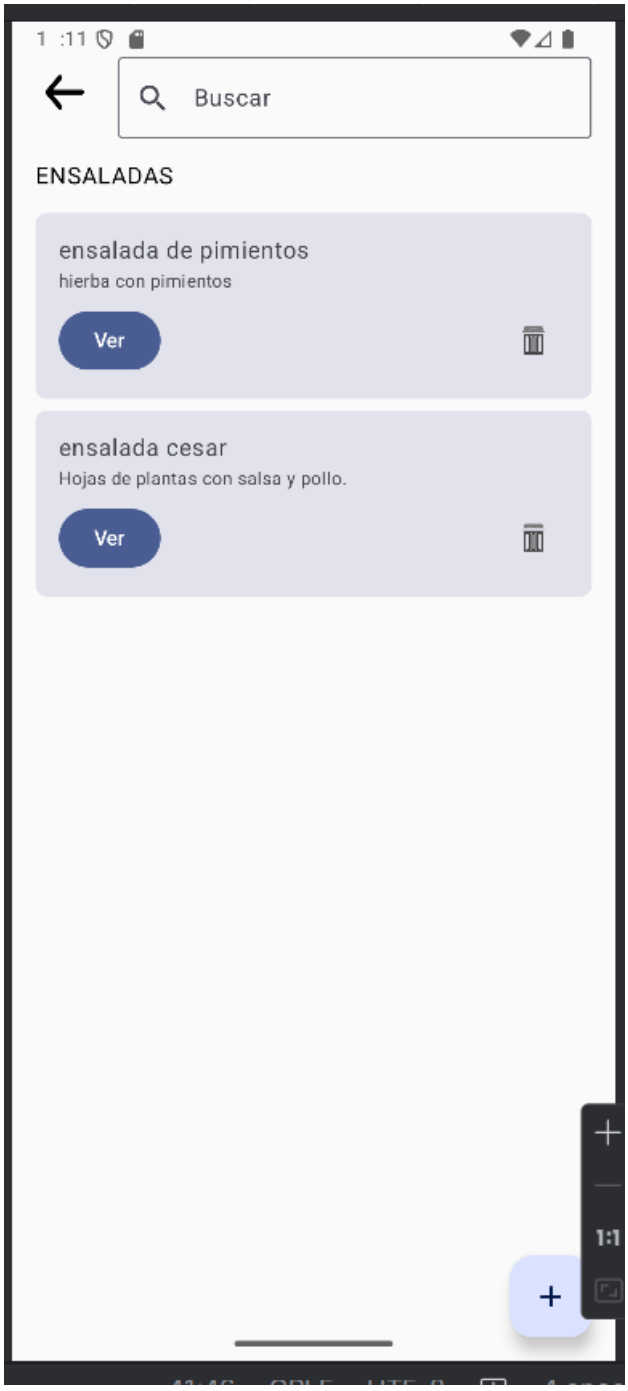
Luego, la siguiente ventana será de categorías de la comida en la que clasificaremos la comida en 8 tipos diferentes y contaremos con un botón que nos permitirá ver todas las recetas que hay en la app sean del tipo que sean, además tenemos el botón de arriba a la izquierda que nos permitirá cerrar sesión:



RECETAS

A continuación, muestro la “ventanaRecetas” en la que muestro las recetas de comida dependiendo de la categoría que se haya seleccionado en la ventana anterior se aplica un filtro por “Categoria” que tenga asignada la receta y se muestran las correspondientesa su categoria:

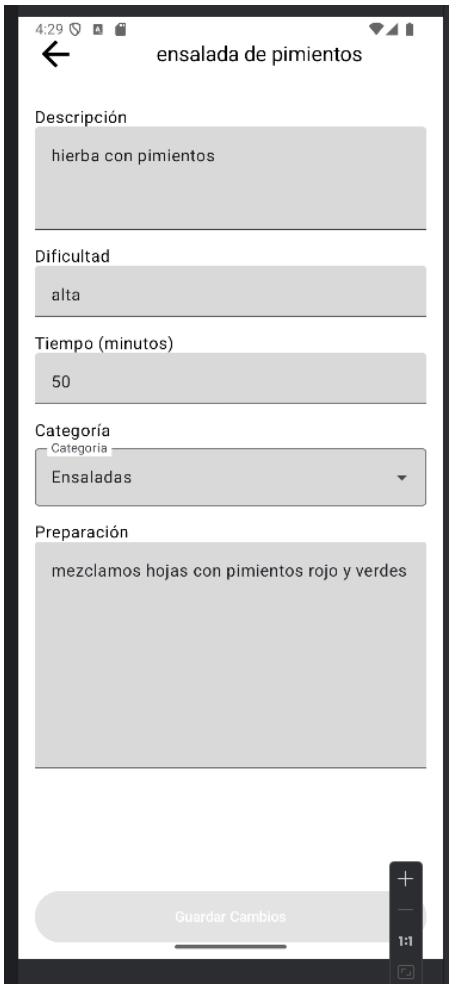
Ventana De Las Recetas



En la ventana podemos observar también que he puesto un “Buscador”, una flecha para volver a las Categorias, y en los card de las recetas se muestra una vista previa de el Nombre de la receta y la Descripción, luego contamos con un icono de una papelera para eliminar la receta y un botón de “Ver” que nos llevará a la ventana que mostraré a continuación. Por último, podemos observar un botón de “+” que sirve para poder agregar una receta nueva en la “ventanaAgregar”.

VER RECETA

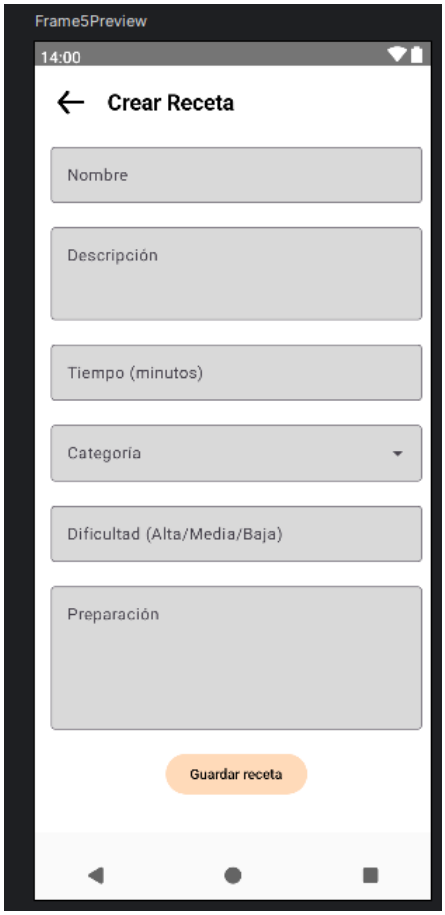
La siguiente imagen corresponde a la ventana de vista y edición de una de las recetas ya creadas:



Aquí tenemos los campos correspondientes y observamos cómo se muestran todos los campos de la receta. Tenemos un botón de “Guardar Cambios” para que en caso que se modifique algo que se active el botón y apliquemos los nuevos cambios a la receta. Si pulsamos la “Flecha” nos lleva de nuevo a la ventana de recetas y si hemos hecho cambios nos permite volver atrás y pulsar el botón para guardar los cambios o descartar directamente que queramos guardar esos cambios.

AGREGAR RECETA

Por último, la ventana de agregar una receta nueva en la que podemos observar que se nos piden que rellenemos todos los campos para agregar la receta, sino nos salta un mensaje y no se agregaría la receta, y sólo si somos el dueño de la receta:



2. FIREBASE

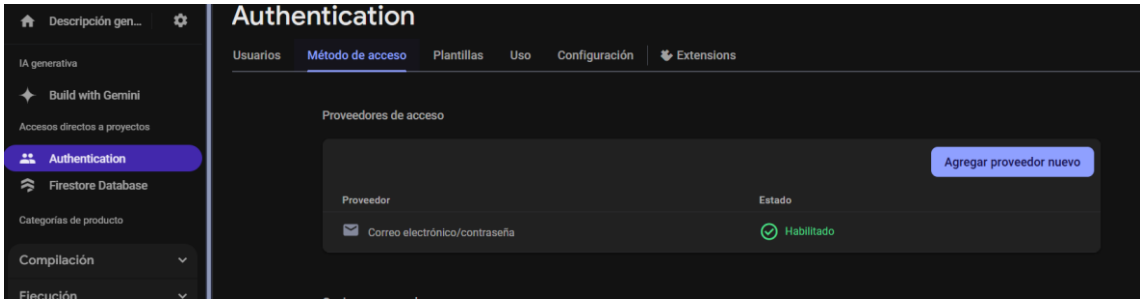
AUTENTICACIÓN

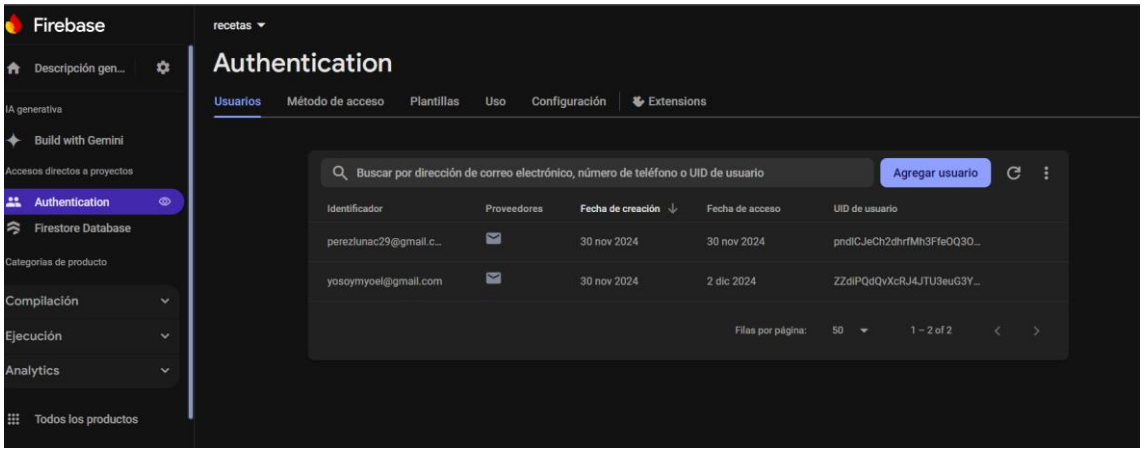
Con la opción de Authentication de Firebase, he hecho la autenticación con el correo y la contraseña del usuario que se registre para la aplicación:

Usuario prueba:

Correo: prueba@gmail.com

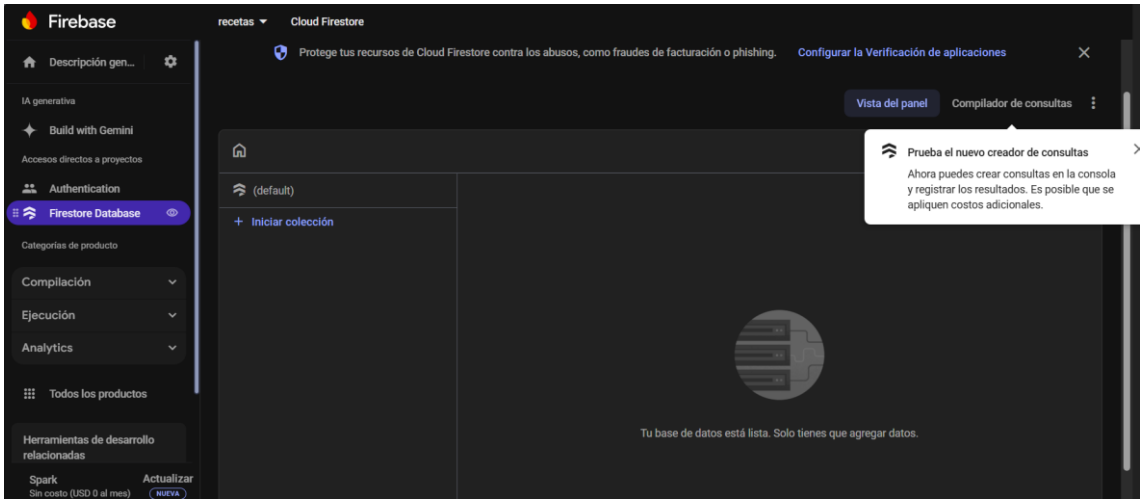
Contraseña: 123456



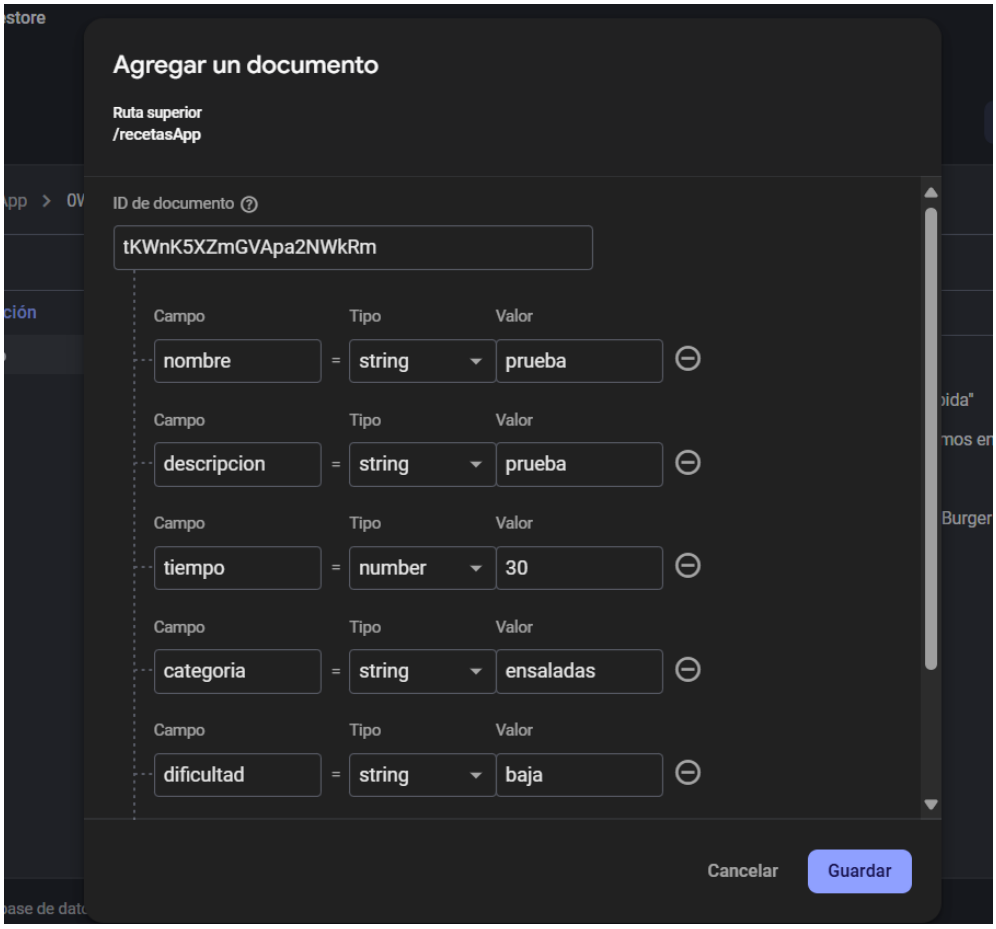


FIRESTORE

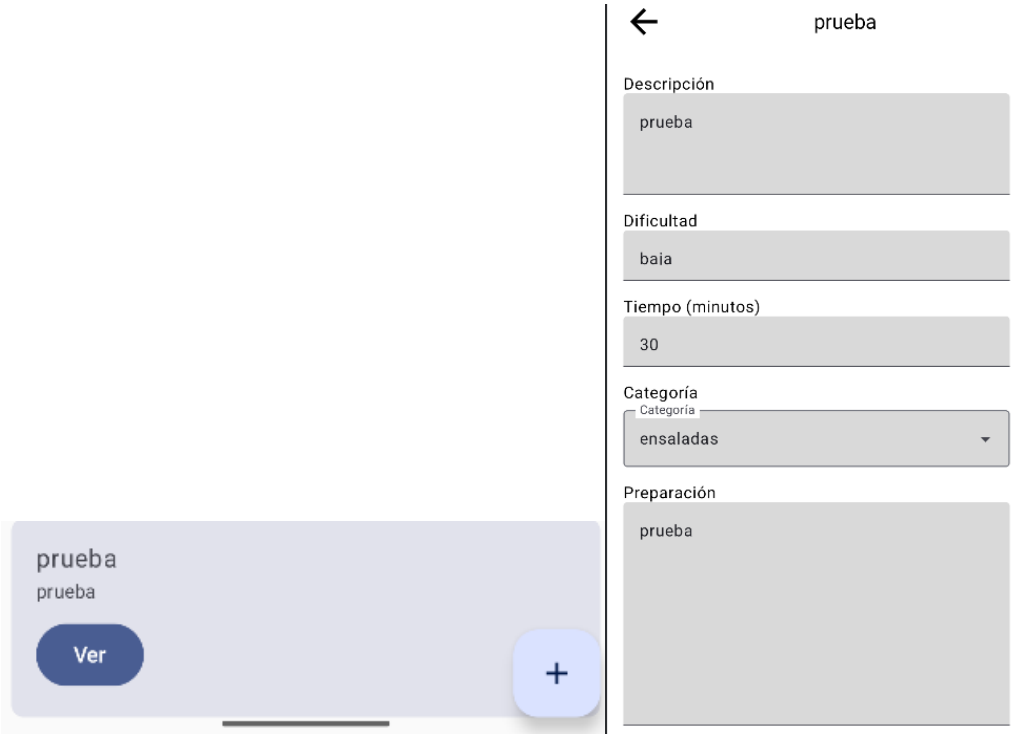
Y para ingresar las recetas usaré el Firestore, y para ello creo una colección para las recetas:



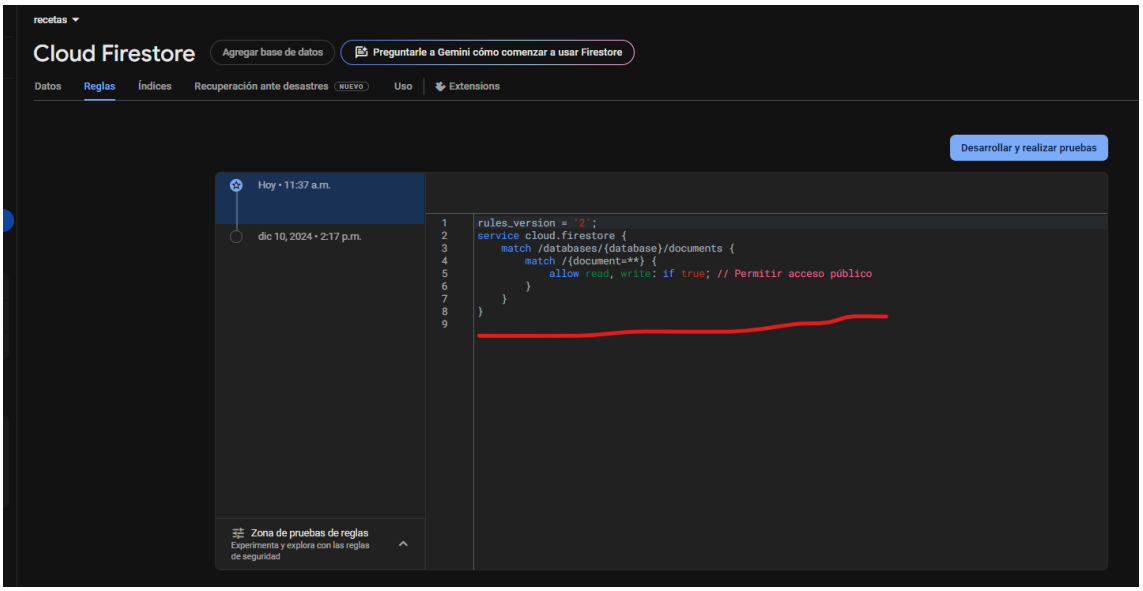
A continuación, creo una receta de prueba en la que pongo el nombre de los campos y el valor que sería los datos de la receta que voy a agregar:



Al pulsar **guardar** la podremos ver ya en la app:



Me había surgido un problema que no me dejaba mostrar las recetas del Firestore ni agregar recetas, y el problema era que las reglas de Firestore no me daba ese permiso, entonces tuve que cambiar y dejar la regla como se ve a continuación:



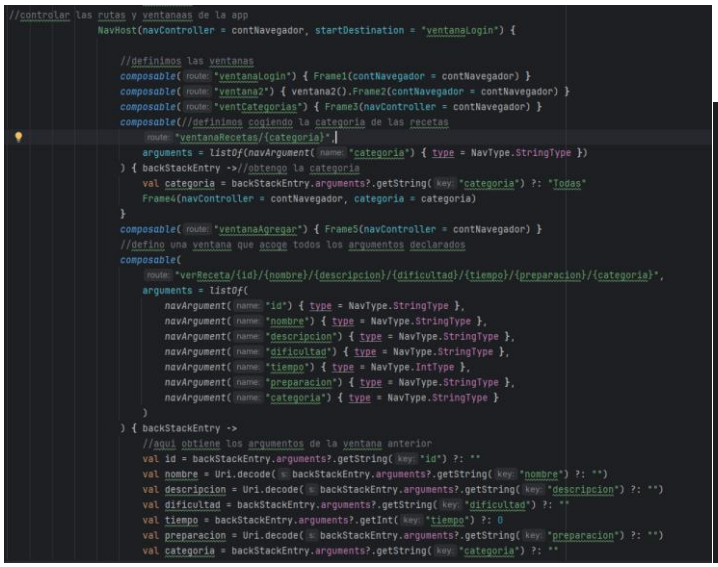
Y ahora podemos observar cómo si que muestra todas las recetas(son ejemplos rápidos):

También he agregado a la aplicación que las recetas que crea un usuario no las pueda **eliminar** o **editar** otro distinto al que la ha creado, es decir, que pueda solo visualizarla. Para ello he modificado de nuevo las reglas en Firestore dando permiso de vista y crear nuevas recetas a todos los usuarios autentificados, y permisos de edición y eliminación a cada usuario de lo que haya creado:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /recetasApp/{recetaId} {
      allow read: if true;
      allow create: if request.auth != null;
      allow delete: if request.auth != null && request.auth.uid == resource.data.creatorId;
      allow update: if request.auth != null && request.auth.uid == resource.data.creatorId;
    }
  }
}
```

3. CÓDIGO

-**MainActivity**: aquí controlamos las ventanas de navegación, también podemos observar que en una de las ventanas lo que hacemos es pasar los campos de la receta como argumentos a otra ventana, esto para que en la ventana “verReceta” pueda visualizarse la receta con sus campos. Y en categorías hacemos igual, obtenemos la categoría y según cuál es la seleccionada la “ventanaRecetas” te muestra unas u otras recetas.



```
//controlar las rutas y ventanaas de la app
NavHost(navController = contNavegador, startDestination = "ventanaLogin") {

    //definimos las ventanas
    composable( route: "ventanaLogin") { Frame1(contNavegador = contNavegador) }
    composable( route: "ventana2") { ventana2().Frame2(contNavegador = contNavegador) }
    composable( route: "ventCategorias") { Frame3(navController = contNavegador) }
    composable(//definimos cogiendo la categoria de las recetas
        route: "ventanaRecetas/{categoria}",
        arguments = listOf(navArgument( name: "categoria") { type = NavType.StringType })
    ) { backStackEntry ->//obtengo la categoria
        val categoria = backStackEntry.arguments?.getString( key: "categoria") ?: "Todas"
        Frame4(navController = contNavegador, categoria = categoria)
    }
    composable( route: "ventanaAgregar") { Frame5(navController = contNavegador) }
    //defino una ventana que acoge todos los argumentos declarados
    composable(
        route: "verReceta/{id}/{nombre}/{descripcion}/{dificultad}/{tiempo}/{preparacion}/{categoria}",
        arguments = listOf(
            navArgument( name: "id") { type = NavType.StringType },
            navArgument( name: "nombre") { type = NavType.StringType },
            navArgument( name: "descripcion") { type = NavType.StringType },
            navArgument( name: "dificultad") { type = NavType.StringType },
            navArgument( name: "tiempo") { type = NavType.IntType },
            navArgument( name: "preparacion") { type = NavType.StringType },
            navArgument( name: "categoria") { type = NavType.StringType }
        )
    ) { backStackEntry ->
        //aqui obtiene los argumentos de la ventana anterior
        val id = backStackEntry.arguments?.getString( key: "id") ?: ""
        val nombre = Uri.decode( s: backStackEntry.arguments?.getString( key: "nombre") ?: "")
        val descripcion = Uri.decode( s: backStackEntry.arguments?.getString( key: "descripcion") ?: "")
        val dificultad = backStackEntry.arguments?.getString( key: "dificultad") ?: ""
        val tiempo = backStackEntry.arguments?.getInt( key: "tiempo") ?: 0
        val preparacion = Uri.decode( s: backStackEntry.arguments?.getString( key: "preparacion") ?: "")
        val categoria = backStackEntry.arguments?.getString( key: "categoria") ?: ""
```

```
                Frame6(
                    navController = contNavegador,
                    recetaId = id,
                    nombre = nombre,
                    descripcionInicial = descripcion,
                    dificultadInicial = dificultad,
                    tiempoInicial = tiempo,
                    preparacionInicial = preparacion,
                    categoriaInicial = categoria
                )
            }
        }
    }
}

@Preview(showSystemUi = true)
@Composable
fun PreviewMainActivity() {
    Proyecto_after_AplicacionTheme {
        val navController = rememberNavController()
        Frame1(contNavegador = navController)
    }
}
```

-**ventanaLogin**: En esta ventana manejamos la autenticación del usuario para poder acceder a la aplicación con un correo y contraseña, tambien disponemos de un botón que nos llevaría a la ventana de registro para crear un nuevo usuario y poder acceder. También manejamos los mensajes de error con el siguiente trozo de código, para lanzarlo con un “Toast”. Ademas muestro el cómo se maneja la autenticación de las credenciales si esos campos existen y coinciden con los de FirebaseAuthentication, se navega a la ventana de categorías.

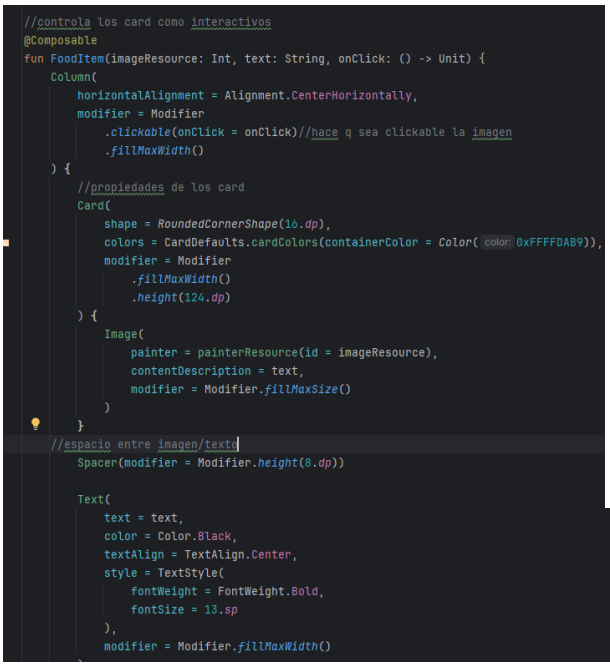
```
//manejar los mensajes de error
val contexto = LocalContext.current
LaunchedEffect(errorMensaje.value) {
    errorMensaje.value?.let { message ->
        //se muestra el mensaje  con un toast
        Toast.makeText(
            contexto,
            message,
            Toast.LENGTH_SHORT
        ).show()
        errorMensaje.value = null
    }
}
}
}
```

```
Button(//boton para iniciar sesion si las credenciales son correctas
    onClick = {
        if (email.value.isEmpty() || contrasena.value.isEmpty()) {//si esta vacio
            errorMensaje.value = "Por favor ingrese usuario y contraseña"
            return@Button
        }
        conrrectas nos lleva a la ventana de las categorias
        autentificacion.signInWithEmailAndPassword(email.value, contrasena.value)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    contNavegador.navigate( route: "ventCategorias")
                } else {//sino salta un mensaje de error de inicio de sesión
                    errorMensaje.value =
                        "Error de inicio de sesión: ${task.exception?.message}"
```

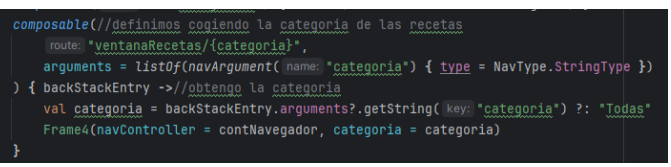
-**Ventana2**: esta es la ventana de registro como ya dije anteriormente, lo hice básico con un email y contraseña para aquellas personas que no manejen mucho los dispositivos inteligentes, me parecía algo excesivo pedir muchos más datos. Se revisa que el correo use una estructura de correo válida, luego pido un mínimo de 6 caracteres de contraseña y hago una comparación de "contrasena" y "confContrasena", si no coinciden salta un mensaje y si coincide lo registra en firebase:

```
Button(
    onClick = {
        if (contrasena.value != confContrasena.value) {
            errorMensaje.value = "Las contraseñas no coinciden"
            return@Button
        }

        val emailRegex = Regex( pattern: "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\$")
        if (!emailRegex.matches(email.value)) {
            errorMensaje.value = "El correo no es válido"
            return@Button
        }
        if (contrasena.value.length < 6) {
            errorMensaje.value = "La contraseña debe tener al menos 6 caracteres"
            return@Button
        }

        auth.createUserWithEmailAndPassword(email.value, contrasena.value)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    contNavegador.navigate( route: "ventanaLogin") // ir al login
                } else {
                    errorMensaje.value =
                        "Error de registro: ${task.exception?.localizedMessage ?: "Desconocido"}"
                }
            }
```

-**ventCategorias**: en esta ventana ya lo he comentado antes tenemos card de las distintas categorías, un botón de todas las recetas y el botón de cerrar sesión. Y en la imagen podemos observar la función para hacer interactivo los ítem/card y las características de tamaño, color, etc… La imagen de la derecha es del "MainActivity" que nos filtra según la categoría seleccionada en la siguiente "ventanaRecetas":



```
//controla los card como interactivos
@Composable
fun FoodItem(imageResource: Int, text: String, onClick: () -> Unit) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .clickable(onClick = onClick)//hace q sea clickable la imagen
            .fillMaxWidth()
    ) {
        //propiedades de los card
        Card(
            shape = RoundedCornerShape(16.dp),
            colors = CardDefaults.cardColors(containerColor = Color( color: 0xFFFFDAB9)),
            modifier = Modifier
                .fillMaxWidth()
                .height(124.dp)
        ) {
            Image(
                painter = painterResource(id = imageResource),
                contentDescription = text,
                modifier = Modifier.fillMaxSize()
            )
        }
    //espacio entre imagen/texto
        Spacer(modifier = Modifier.height(8.dp))

        Text(
            text = text,
            color = Color.Black,
            textAlign = TextAlign.Center,
            style = TextStyle(
                fontWeight = FontWeight.Bold,
                fontSize = 13.sp
            ),
            modifier = Modifier.fillMaxWidth()
```



```
composable(//definimos cogiendo la categoria de las recetas
    route: "ventanaRecetas/{categoria}",
    arguments = listOf(navArgument( name: "categoria") { type = NavType.StringType })
) { backStackEntry ->//obtengo la categoria
    val categoria = backStackEntry.arguments?.getString( key: "categoria") ?: "Todas"
    Frame4(navController = contNavegador, categoria = categoria)
}
```

-**ventanaRecetas**: aquí muestro todas las recetas filtradas por la categoría seleccionada, o sin filtros con el botón. También tenemos la opción de búsqueda, volver atrás, eliminar, visualizar completa la receta pulsando un botón y crear una nueva receta. Controlo la eliminación que la haga solo el creador de la propia receta y si es el que está en ese momento con la cuenta iniciada. Y a la hora de mandar los datos a "verReceta" los mando encoded para que no me de errores a la hora de navegar a la siguiente ventana. Por último tenemos el botón flotante para agregar una nueva receta.

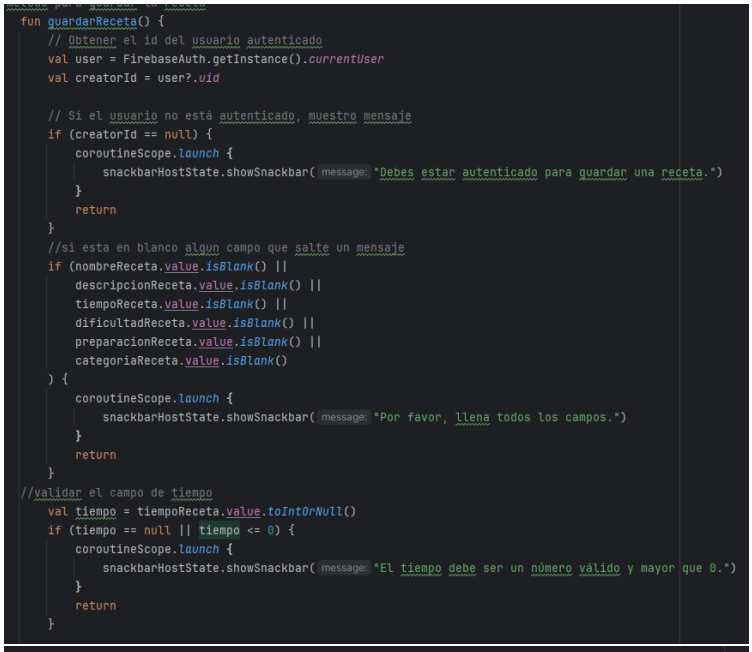
```
//filtro para la búsqueda
    val recetasFiltradas = recetas.value.filter { receta ->
        val nombre = (receta["nombre"] as? String)?.lowercase() ?: ""
        val descripcion = (receta["descripcion"] as? String)?.lowercase() ?: ""
        val query = searchQuery.value.lowercase()
        nombre.contains(query) || descripcion.contains(query)
    }
```

```
//función para eliminar la receta, solo quien crea la receta la puede eliminar
fun eliminarReceta() {
    recetaAEliminar.value?.let { (id, receta) ->
        val creatorId = receta["creatorId"] as? String
        val currentUserId = FirebaseAuth.getInstance().currentUser?.uid

        if (creatorId == null) {
            println("ERROR: La receta no tiene un campo 'creatorId'.")
            return
        }

        if (currentUserId == null) {
            println("ERROR: El usuario no está autenticado.")
            return
        }
//si el usuario es el qeu creo la receta se puede eliminar
        if (currentUserId == creatorId) {

            db.collection( collectionPath: "recetasApp").document(id)
                .delete()
                .addOnSuccessListener {
                    println("Receta eliminada correctamente en Firestore: $id")
                    recetas.value = recetas.value.filter { it["id"] != id } // Actualiza la lista local.
                }
                .addOnFailureListener { exception ->
                    println("Error al eliminar la receta en Firestore: ${exception.message}")
                }
        } else {
            //sino salta un mensaje de que no tienes permiso para eliminarla
            coroutineScope.launch {
                snackbarHostState.showSnackbar( message: "No tienes permiso para eliminar esta receta.")
            }
            println("ERROR: El usuario autenticado no es el creador de la receta.")
        }
    } ?: run {
        println("ERROR: No se seleccionó ninguna receta para eliminar.")
    }

    // Cerrar el cuadro de diálogo
    showDialog.value = false
}
```

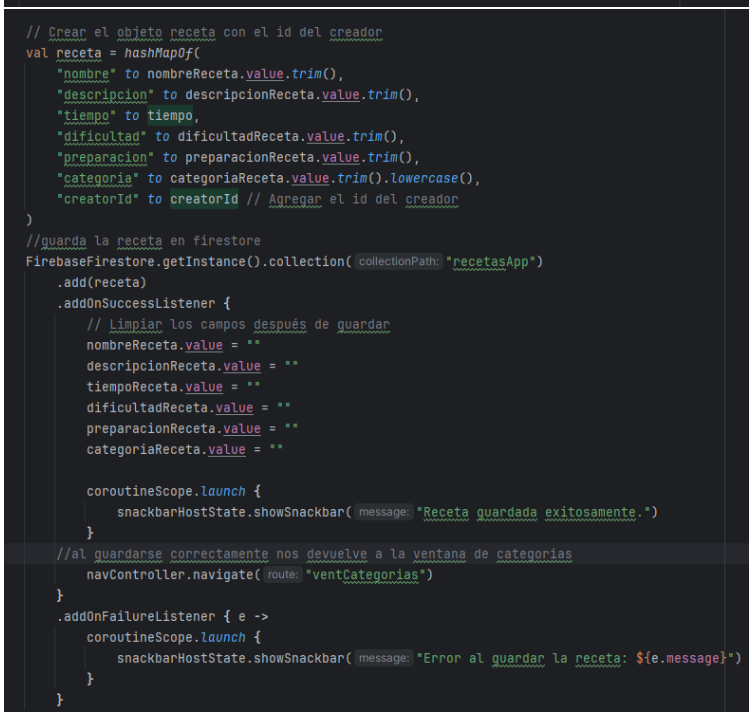
```
//boton para ver la receta con todos sus campos
Button(
    onClick = {
        //uso encoded para pasar los parametros con mayor seguridad
        val encodedNombre = Uri.encode(nombre)
        val encodedDescripcion = Uri.encode(descripcion)
        val encodedPreparacion = Uri.encode(preparacion)

        navController.navigate(
            route: "verReceta/$id/$encodedNombre/$encodedDescripcion/$dificultad/$tiempo/$encodedPreparacion/$categoriaReceta"
        )
    },
    colors = ButtonDefaults.buttonColors(containerColor = Color( color: 0xFFE3C79A)),
    modifier = Modifier.padding(end = 8.dp)
) {
    Text( text: "Ver")
}
```

```
botón flotante para que nos lleve a agregar una nueva receta
    FloatingActionButton(
        onClick = {
            navController.navigate( route: "ventanaAgregar")
        },
        modifier = Modifier
            .align(Alignment.BottomEnd)
            .padding(16.dp),
        containerColor = Color( color: 0xFFE3C79A)
    ) {
        Icon(
            imageVector = Icons.Default.Add,
            contentDescription = "Agregar receta"
        )
    }
```

-**ventanaAgregar**: En esta ventana simplemente es para agregar una nueva receta, rellenamos todos los campos sin dejar ninguno en blanco y se agregaría cogiendo el id del usuario que lo crea para que luego este sea el único que pueda editar y eliminar la receta. Pulsando el botón se llama a la función de la imagen que realizaría lo dicho anteriormente.

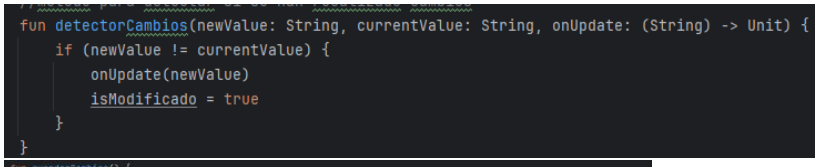


```
fun guardarReceta() {
    // Obtener el id del usuario autenticado
    val user = FirebaseAuth.getInstance().currentUser
    val creatorId = user?.uid

    // Si el usuario no está autenticado, muestro mensaje
    if (creatorId == null) {
        coroutineScope.launch {
            snackbarHostState.showSnackbar( message: "Debes estar autenticado para guardar una receta.")
        }
        return
    }
    //si esta en blanco algun campo que salte un mensaje
    if (nombreReceta.value.isBlank() ||
        descripcionReceta.value.isBlank() ||
        tiempoReceta.value.isBlank() ||
        dificultadReceta.value.isBlank() ||
        preparacionReceta.value.isBlank() ||
        categoriaReceta.value.isBlank()
    ) {
        coroutineScope.launch {
            snackbarHostState.showSnackbar( message: "Por favor, llena todos los campos.")
        }
        return
    }
//validar el campo de tiempo
    val tiempo = tiempoReceta.value.toIntOrNull()
    if (tiempo == null || tiempo <= 0) {
        coroutineScope.launch {
            snackbarHostState.showSnackbar( message: "El tiempo debe ser un número válido y mayor que 0.")
        }
        return
    }
```

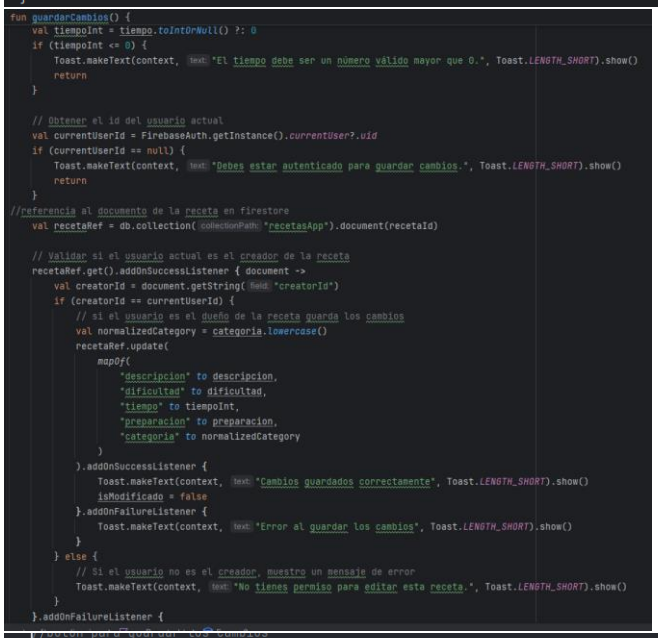


```
// Crear el objeto receta con el id del creador
val receta = hashMapOf(
    "nombre" to nombreReceta.value.trim(),
    "descripcion" to descripcionReceta.value.trim(),
    "tiempo" to tiempo,
    "dificultad" to dificultadReceta.value.trim(),
    "preparacion" to preparacionReceta.value.trim(),
    "categoria" to categoriaReceta.value.trim().lowercase(),
    "creatorId" to creatorId // Agregar el id del creador
)
//guarda la receta en firestore
FirebaseFirestore.getInstance().collection( collectionPath: "recetasApp")
    .add(receta)
    .addOnSuccessListener {
        // Limpiar los campos después de guardar
        nombreReceta.value = ""
        descripcionReceta.value = ""
        tiempoReceta.value = ""
        dificultadReceta.value = ""
        preparacionReceta.value = ""
        categoriaReceta.value = ""

        coroutineScope.launch {
            snackbarHostState.showSnackbar( message: "Receta guardada exitosamente.")
        }
    //al guardarse correctamente nos devuelve a la ventana de categorias
        navController.navigate( route: "ventCategorias")
    }
    .addOnFailureListener { e ->
        coroutineScope.launch {
            snackbarHostState.showSnackbar( message: "Error al guardar la receta: ${e.message}")
        }
    }
```

-**verReceta**: la última ventana, es tan básica como que se muestra una ventana con todos los campos de la receta y sus respectivos datos. Si se modifica alguno de estos datos hay una función que detecta los cambios y que pone activo el botón “Guardar Cambios”, sólo puede guardar el dueño de la receta, si no tendríamos que volver a la vista de todas las recetas. Esta ventana es para poder ver la receta completa.

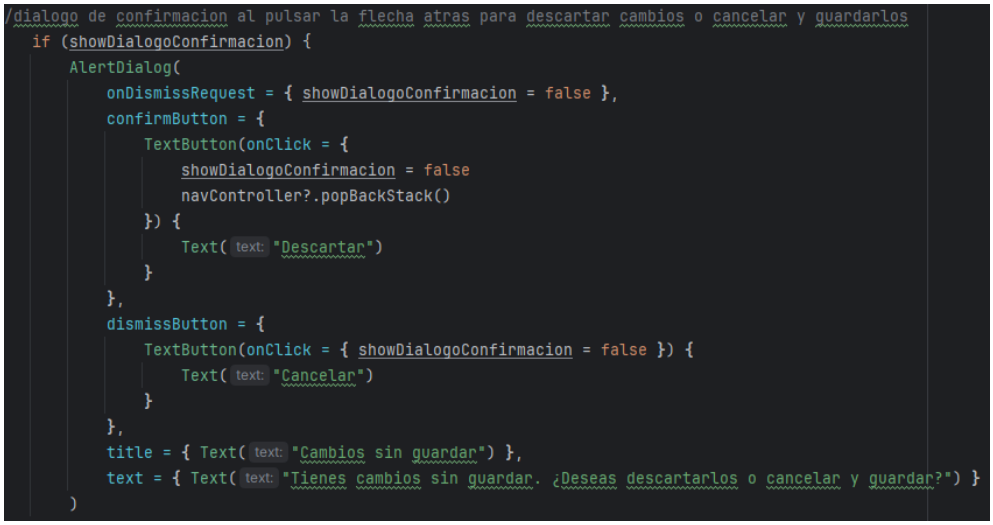


```
fun detectorCambios(newValue: String, currentValue: String, onUpdate: (String) -> Unit) {
    if (newValue != currentValue) {
        onUpdate(newValue)
        isModificado = true
    }
}
```



```
fun guardarCambios() {
    val tiempoInt = tiempo.toIntOrNull() ?: 0
    if (tiempoInt <= 0) {
        Toast.makeText(context, text: "El tiempo debe ser un número válido mayor que 0.", Toast.LENGTH_SHORT).show()
        return
    }

    // Obtener el id del usuario actual
    val currentUserId = FirebaseAuth.getInstance().currentUser?.uid
    if (currentUserId == null) {
        Toast.makeText(context, text: "Debes estar autenticado para guardar cambios.", Toast.LENGTH_SHORT).show()
        return
    }
//referencia al documento de la receta en firestore
    val recetaRef = db.collection( collectionPath: "recetasApp").document(recetaId)

    // Validar si el usuario actual es el creador de la receta
    recetaRef.get().addOnSuccessListener { document ->
        val creatorId = document.getString( field: "creatorId")
        if (creatorId == currentUserId) {
            // si el usuario es el dueño de la receta guarda los cambios
            val normalizedCategory = categoria.lowercase()
            recetaRef.update(
                mapOf(
                    "descripcion" to descripcion,
                    "dificultad" to dificultad,
                    "tiempo" to tiempoInt,
                    "preparacion" to preparacion,
                    "categoria" to normalizedCategory
                )
            ).addOnSuccessListener {
                Toast.makeText(context, text: "Cambios guardados correctamente", Toast.LENGTH_SHORT).show()
                isModificado = false
            }.addOnFailureListener {
                Toast.makeText(context, text: "Error al guardar los cambios", Toast.LENGTH_SHORT).show()
            }
        } else {
            // Si el usuario no es el creador, muestro un mensaje de error
            Toast.makeText(context, text: "No tienes permiso para editar esta receta.", Toast.LENGTH_SHORT).show()
        }
    }.addOnFailureListener {
```

```
    Button(
        onClick = { guardarCambios() },
        enabled = isModificado,
        shape = RoundedCornerShape(100.dp),
        colors = ButtonDefaults.buttonColors(containerColor = if (isModificado) Color( color: 0xFFFFDAB9) else Color.Gray),
        modifier = Modifier
            .fillMaxWidth()
            .height(50.dp)
    ) {
        Text( text: "Guardar Cambios", color = Color.Black, style = MaterialTheme.typography.labelLarge)
    }
```



```
//dialogo de confirmacion al pulsar la flecha atras para descartar cambios o cancelar y guardarlos
    if (showDialogoConfirmacion) {
        AlertDialog(
            onDismissRequest = { showDialogoConfirmacion = false },
            confirmButton = {
                TextButton(onClick = {
                    showDialogoConfirmacion = false
                    navController?.popBackStack()
                }) {
                    Text( text: "Descartar")
                }
            },
            dismissButton = {
                TextButton(onClick = { showDialogoConfirmacion = false }) {
                    Text( text: "Cancelar")
                }
            },
            title = { Text( text: "Cambios sin guardar") },
            text = { Text( text: "Tienes cambios sin guardar. ¿Deseas descartarlos o cancelar y guardar?") }
        )
```

4. Bibliografía

- www.figma.com
- www.firebase.com
- Login y creación de usuarios en Firebase con Android Jetpack Compose: Authentication y FireStore
- Build Food Menu App in Android With Kotlin(de aquí me hice una idea del diseño)
- Más de 1 millón de Imágenes Gratis para Descargar - Pixabay
- CÓMO Navegar entre Pantallas en JETPACK COMPOSE | #5
- Agrega datos a Cloud Firestore | Firebase
- https://youtu.be/PQc4aVuSr48 (de aquí saque lo de los permisos, no lo vi completo)
- Cómo guardar datos simples con SharedPreferences | Android Developers
- Efectos secundarios en Compose | Jetpack Compose | Android Developers