

28-01-2024

Desarrollo de una aplicación en Android

Estudiante: **HORACIO NELSON RAMÍREZ SILVÁN**

País de Emisión: México

Estudio: Plataformas de Desarrollo de Software

Período Académico: 2023-2024

INDICE

No.	Tema	# Pág.
1.	Introducción.....	2
2.	Objetivos	2
3.	Ventana de bienvenida	3
4.	Ventana para agregar eventos	4
5.	Reglas de validación para poder agregar un evento	5
6.	Vista para visualizar la lista de eventos.....	6
7.	Conclusiones finales.....	8

1. Introducción

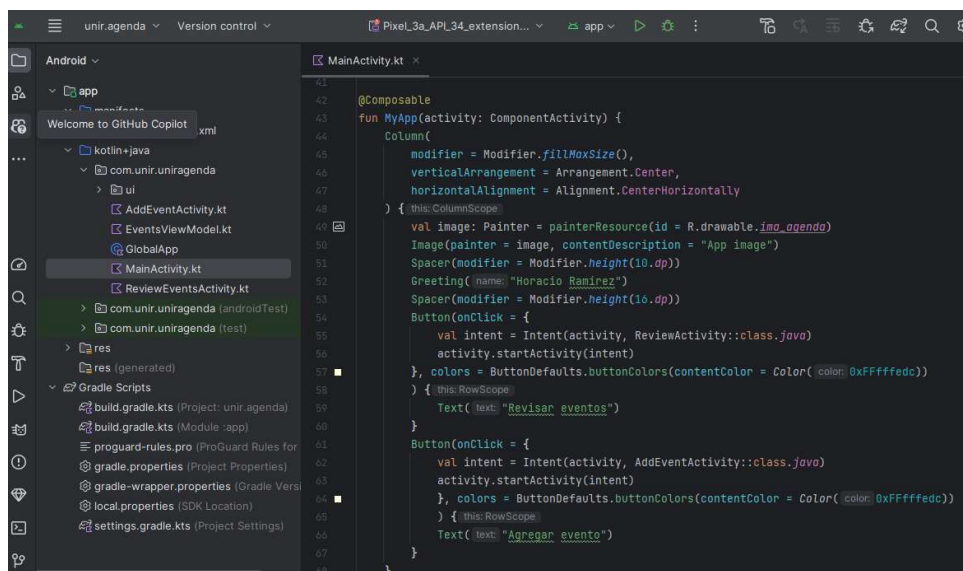
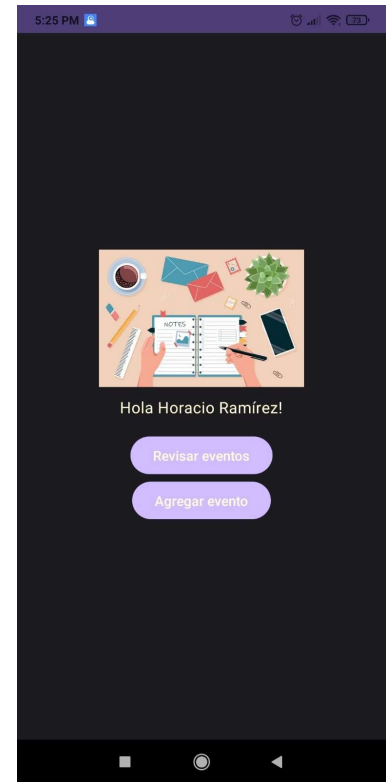
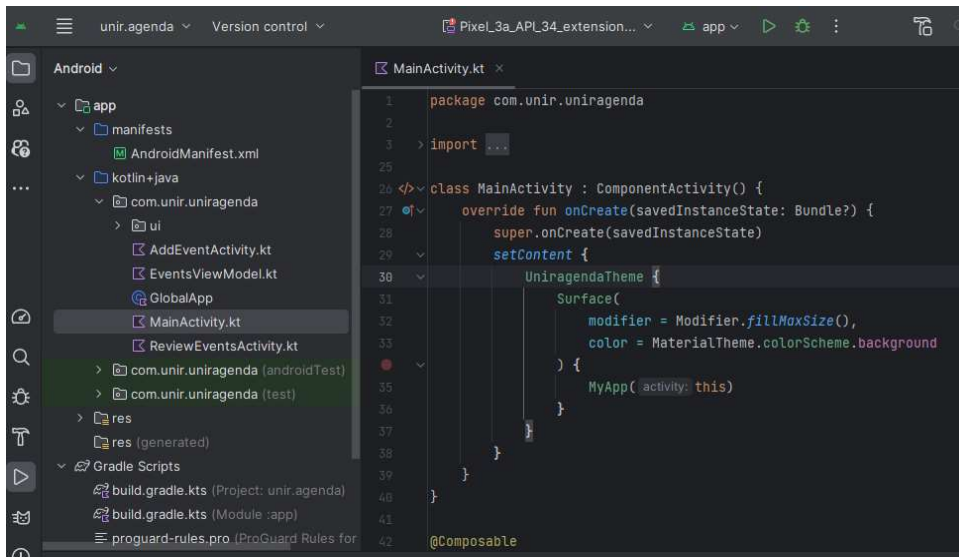
En la presente memoria se documentará de forma práctica el desarrollo de una pequeña aplicación móvil. La aplicación Android desarrollada en Android Studio consiste en la creación de una agenda que permita añadir nuevos eventos al calendario. El alcance de la solución quedará acotado únicamente a la creación de las pantallas y la lógica de validación, sin necesidad de incorporar la persistencia de la información o su integración con servicios de calendario del teléfono o de cuentas en la nube.

2. Objetivos.

- Creación de una ventana a modo de bienvenida con un mensaje y una imagen, y contará con un botón para poder dar de alta un evento.
- Creación de una segunda actividad (pantalla) a modo de formulario en el que el usuario podrá dar de alta un evento.
- Incluir las siguientes reglas de validación para poder agregar un evento:
 - Título no vacío.
 - Tipo de evento no vacío.
 - Fechas de inicio y fin no vacías (o alternativamente fecha de inicio y todo el día activo)
- Creación de una vista con el resumen de los datos introducidos por el usuario.

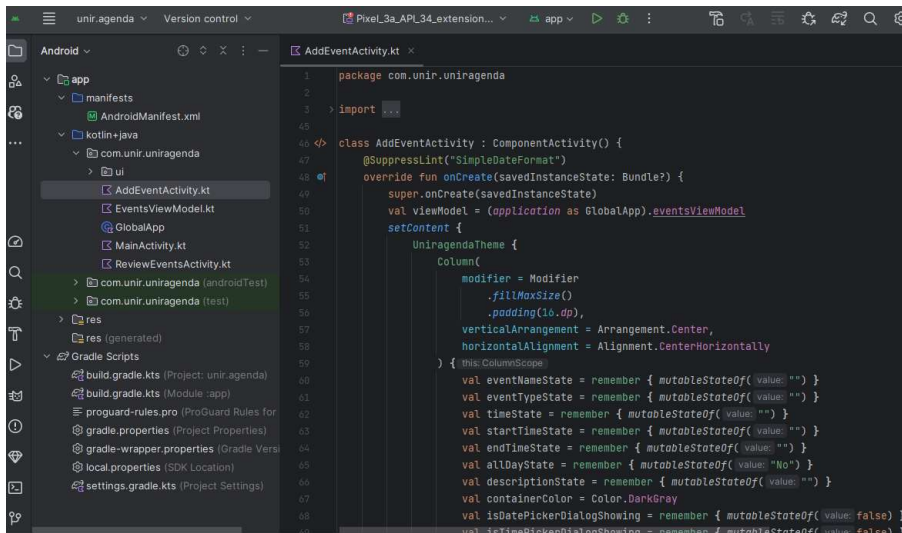
3. Ventana de bienvenida

Para agregar la pantalla de bienvenida se agrega al proyecto de Android Studio un elemento Activity con una clase de tipo `ComponentActivity`. A continuación en las siguientes imágenes se puede visualizar el código escrito y como luce el diseño de la pantalla.

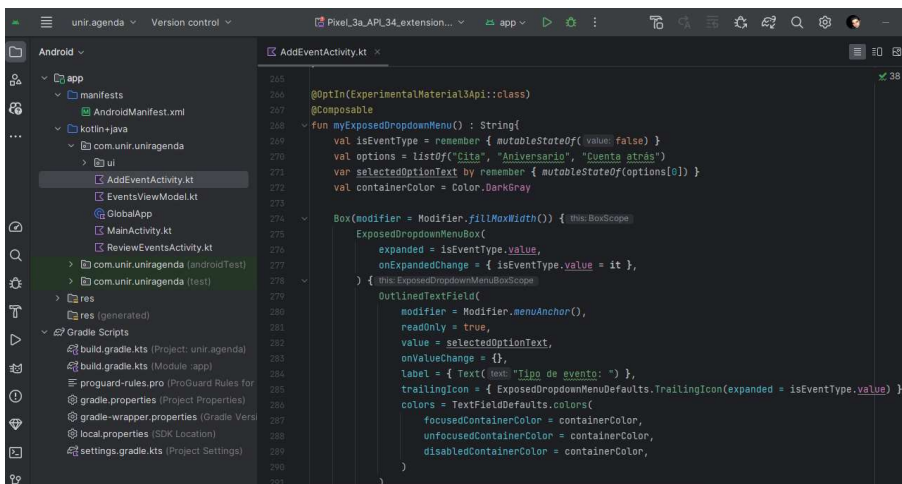


4. Ventana para agregar eventos

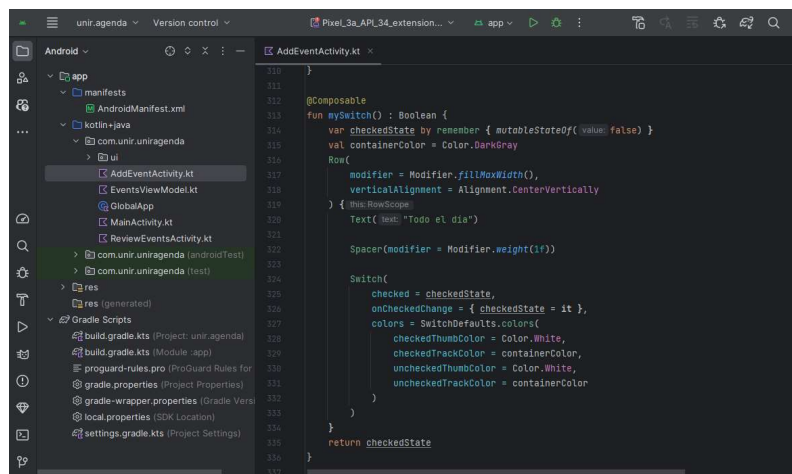
Para la pantalla que sirve para agregar eventos se crea un elemento Activity con una clase de tipo `ComponentActivity`. A continuación en las siguientes imágenes se puede visualizar el código escrito y como luce el diseño de la pantalla.



```
1 package com.unir.uniragenda
2
3 > import ...
4
5 class AddEventActivity : ComponentActivity() {
6     @SuppressLint("SimpleDateFormat")
7     override fun onCreate(savedInstanceState: Bundle?) {
8         super.onCreate(savedInstanceState)
9         val viewModel = (application as GlobalApp).eventsViewModel
10         setContent {
11             UniragendaTheme {
12                 Column(
13                     modifier = Modifier
14                         .fillMaxSize()
15                         .padding(16.dp),
16                     verticalArrangement = Arrangement.Center,
17                     horizontalAlignment = Alignment.CenterHorizontally
18                 ) {
19                     this.ColumnScope
20                     val eventNameState = remember { mutableStateOf<String>("") }
21                     val eventTypeState = remember { mutableStateOf<String>("") }
22                     val timeState = remember { mutableStateOf<String>("") }
23                     val startTimeState = remember { mutableStateOf<String>("") }
24                     val endTimeState = remember { mutableStateOf<String>("") }
25                     val allDayState = remember { mutableStateOf<Boolean>(false) }
26                     val descriptionState = remember { mutableStateOf<String>("") }
27                     val datePickerDialogShowing = remember { mutableStateOf<Boolean>(false) }
28                     val isDatePickerDialogShowing = remember { mutableStateOf<Boolean>(false) }
29                 }
30             }
31         }
32     }
33 }
```



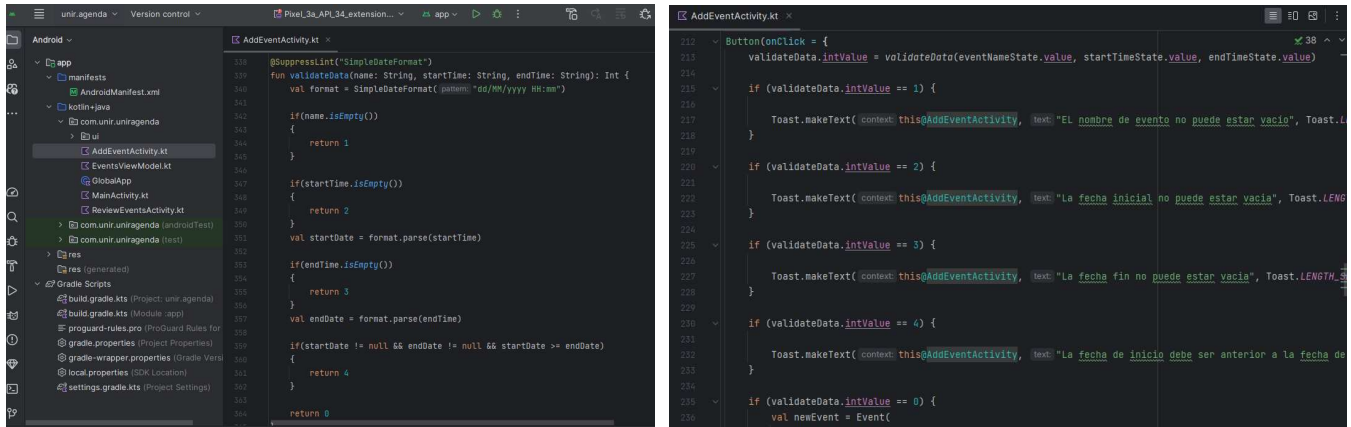
```
265
266 @OptIn(ExperimentalMaterial3Api::class)
267 @Composable
288 fun myExposedDropdownMenu() : String {
289     val isEventType = remember { mutableStateOf<Boolean>(false) }
290     val options = listOf("Gita", "Aniversario", "Cuenta atrás")
291     var selectedOptionText by remember { mutableStateOf<String>(options[0]) }
292     val containerColor = Color.DarkGray
293
294     Box(modifier = Modifier.fillMaxWidth()) {
295         ExposedDropdownMenuBox(
296             expanded = isEventType.value,
297             onExpandedChange = { isEventType.value = it },
298             {
299                 OutlinedTextField(
300                     modifier = Modifier.menuAnchor(),
301                     readOnly = true,
302                     value = selectedOptionText,
303                     onValueChange = {},
304                     label = { Text(text = "Tipo de evento: ") },
305                     trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = isEventType.value) },
306                     colors = TextFieldDefaults.colors(
307                         focusedContainerColor = containerColor,
308                         unfocusedContainerColor = containerColor,
309                         disabledContainerColor = containerColor,
310                     )
311                 )
312             }
313         )
314     }
315 }
```



```
316
317
318 @Composable
319 fun mySwitch() : Boolean {
320     var checkedState by remember { mutableStateOf<Boolean>(false) }
321     val containerColor = Color.DarkGray
322
323     Row(
324         modifier = Modifier.fillMaxWidth(),
325         verticalAlignment = Alignment.CenterVertically
326     ) {
327         this.RowScope
328         Text(text = "Todo el día")
329
330         Spacer(modifier = Modifier.weight(1f))
331
332         Switch(
333             checked = checkedState,
334             onCheckedChange = { checkedState = it },
335             colors = SwitchDefaults.colors(
336                 checkedThumbColor = Color.White,
337                 checkedTrackColor = containerColor,
338                 uncheckedThumbColor = Color.White,
339                 uncheckedTrackColor = containerColor
340             )
341         )
342     }
343     return checkedState
344 }
```

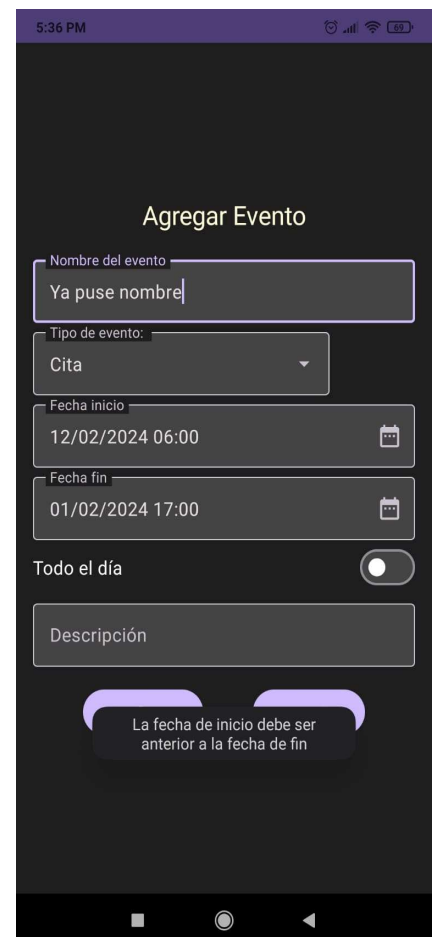
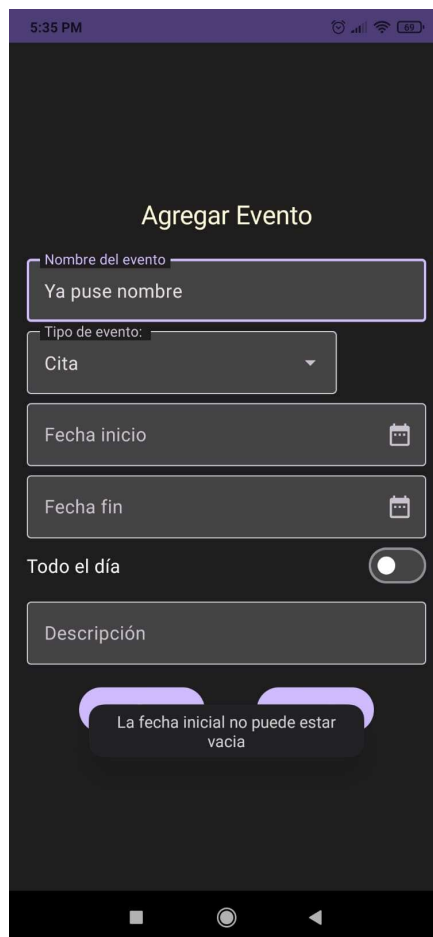
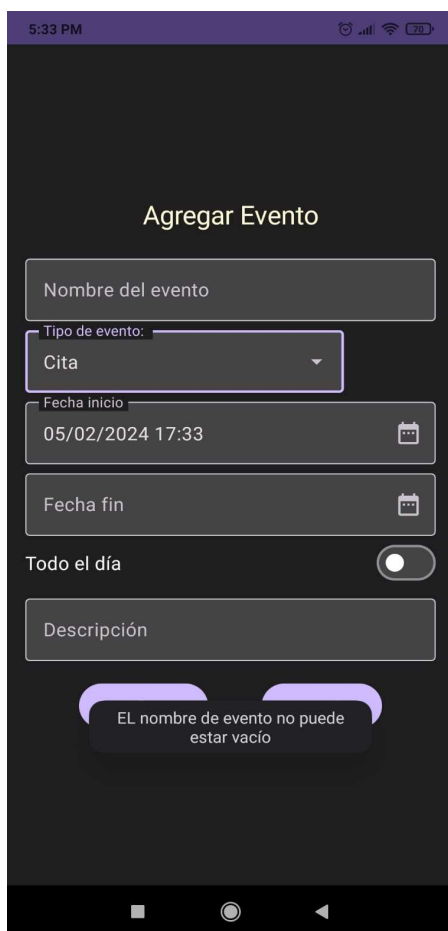
5. Reglas de validación para poder agregar un evento

Dentro del código de la clase AddEventActivity se agrega la siguiente función encargada de validar los datos y los mensajes que se muestran en pantalla. A continuación en las siguientes imágenes se puede visualizar el código escrito y como luce el diseño de la pantalla.



```
538 @SuppressLint("SimpleDateFormat")
539 fun validateData(name: String, startTime: String, endTime: String): Int {
540     val format = SimpleDateFormat("dd/MM/yyyy HH:mm")
541     if (name.isEmpty()) {
542         return 1
543     }
544     if (startTime.isEmpty()) {
545         return 2
546     }
547     val startDate = format.parse(startTime)
548     if (endTime.isEmpty()) {
549         return 3
550     }
551     val endDate = format.parse(endTime)
552     if (startDate != null && endDate != null && startDate > endDate) {
553         return 4
554     }
555     return 0
556 }
```

```
212 Button(onClick = {
213     validateData(intValue = validateData(eventNameState.value, startTimeState.value, endTimeState.value))
214 }
215 if (validateData.intValue == 1) {
216     Toast.makeText(context, "El nombre de evento no puede estar vacío", Toast.LENGTH_SHORT)
217 }
218 if (validateData.intValue == 2) {
219     Toast.makeText(context, "La fecha inicial no puede estar vacía", Toast.LENGTH_SHORT)
220 }
221 if (validateData.intValue == 3) {
222     Toast.makeText(context, "La fecha fin no puede estar vacía", Toast.LENGTH_SHORT)
223 }
224 if (validateData.intValue == 4) {
225     Toast.makeText(context, "La fecha de inicio debe ser anterior a la fecha de fin", Toast.LENGTH_SHORT)
226 }
227 if (validateData.intValue == 0) {
228     val newEvent = Event(
229         eventNameState.value,
230         startTimeState.value,
231         endTimeState.value,
232         locationState.value,
233         descriptionState.value,
234         isAllDayState.value
235     )
236     viewModel.addEvent(newEvent)
237 }
```

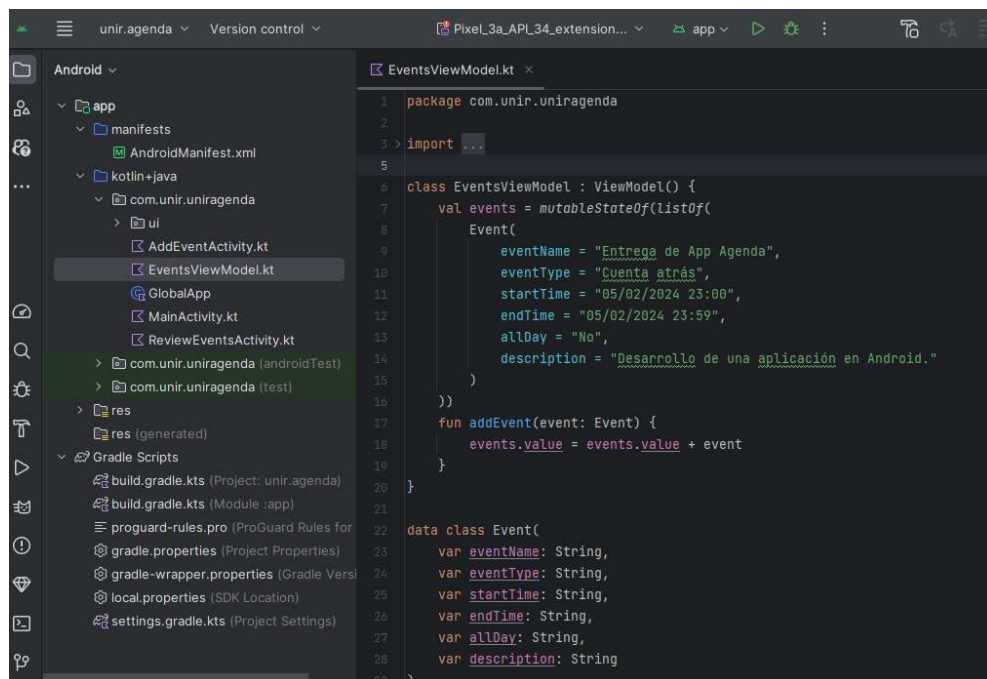


6. Vista para visualizar la lista de eventos

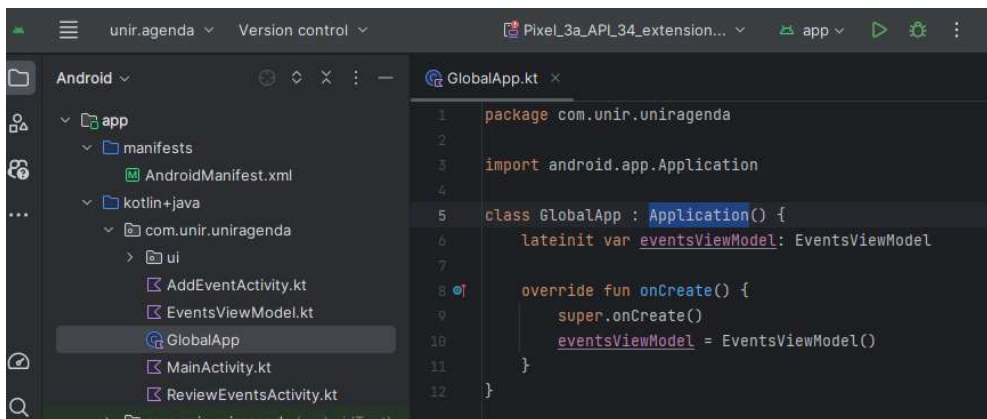
Para poder visualizar los eventos creados por el usuario se tuvo que crear las siguientes clases:

- Clase de tipo ViewModel llamada EventsViewModel que tiene la definición de una lista de tipo POCOEntity con los atributos de un evento y un método para agregar un evento a la lista.
- Clase de tipo Application llamada GlobalApp para exponer el ViewModel y sea accesibles desde otras clases Activity.
- Clase ComponentActivity llamada ReviewActivity que contiene una llamada a la instancia de la ViewModel que imprime en pantalla todos los eventos y un botón llamado “Regresar” que te devuelve a la pantalla de bienvenida.

A continuación en las siguientes imágenes se puede visualizar el código escrito.



```
1 package com.unir.uniragenda
2
3 import ...
4
5
6 class EventsViewModel : ViewModel() {
7     val events = mutableStateOf(listOf(
8         Event(
9             eventName = "Entrega de App Agenda",
10            eventType = "Cuenta atrás",
11            startTime = "05/02/2024 23:00",
12            endTime = "05/02/2024 23:59",
13            allDay = "No",
14            description = "Desarrollo de una aplicación en Android."
15        )
16    ))
17
18     fun addEvent(event: Event) {
19         events.value = events.value + event
20     }
21 }
22
23 data class Event(
24     var eventName: String,
25     var eventType: String,
26     var startTime: String,
27     var endTime: String,
28     var allDay: String,
29     var description: String
30 )
```



```
1 package com.unir.uniragenda
2
3 import android.app.Application
4
5 class GlobalApp : Application() {
6     lateinit var eventsViewModel: EventsViewModel
7
8     override fun onCreate() {
9         super.onCreate()
10         eventsViewModel = EventsViewModel()
11     }
12 }
```



```

25 class ReviewActivity : AppCompatActivity() {
26     override fun onCreate(savedInstanceState: Bundle?) {
27         super.onCreate(savedInstanceState)
28
29         val viewModel = (application as GlobalApp).eventsViewModel
30         setContent {
31             UniragendaTheme {
32                 Column {
33                     modifier = Modifier
34                         .fillMaxSize()
35                         .padding(16.dp),
36                     verticalArrangement = Arrangement.Top,
37                     horizontalAlignment = Alignment.Start
38                 } { this: ColumnScope
39                     ShowEvents(viewModel)
40                     Spacer(modifier = Modifier.height(16.dp))
41                     Box {
42                         modifier = Modifier
43                             .fillMaxWidth()
44                             .contentAlign = Alignment.Center
45                     } { this: BoxScope
46                         Button(onClick = { finish() },
47                             colors = ButtonDefaults.buttonColors(contentColor = Color(0xFFFFF5DC)))
48                     } { this: RowScope
49                         Text(text = "Regresar")
50                     }
51                 }
52             }
53         }
54     }
55 }

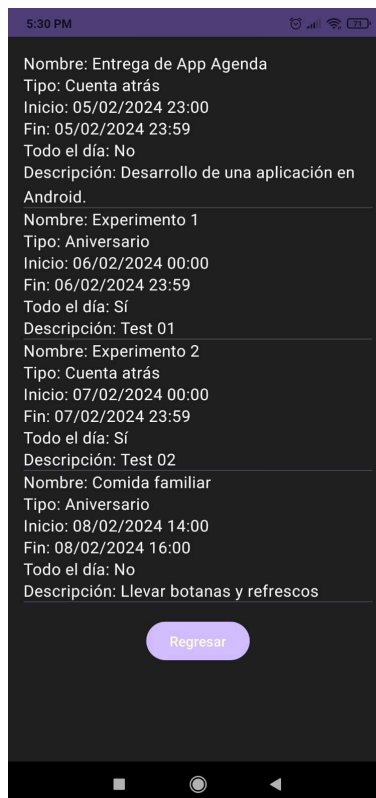
```

```

57 class EventsViewModel : ViewModel() {
58     @Composable
59     fun ShowEvents(viewModel: EventsViewModel) {
60         val events = viewModel.events.value
61         for (event in events) {
62             Text(text = "Nombre: ${event.eventName}")
63             Text(text = "Tipo: ${event.eventType}")
64             Text(text = "Inicio: ${event.startTime}")
65             Text(text = "Fin: ${event.endTime}")
66             Text(text = "Todo el día: ${event.allDay}")
67             Text(text = "Descripción: ${event.description}")
68             Divider()
69         }
70     }
71 }

```

A continuación se puede visualizar como luce el diseño de la vista con los eventos capturados por el usuario.



7. Conclusiones finales

El desarrollo de esta aplicación de agenda para Android ha sido un proceso educativo y desafiante. A través de este proyecto, se han aplicado conceptos fundamentales de la programación Android, como la creación de múltiples actividades, la transición entre estas actividades y la validación de la entrada del usuario.

La aplicación cumple con los objetivos establecidos al inicio del proyecto. Proporciona una interfaz de usuario amigable que incluye una pantalla de bienvenida y un formulario para agregar nuevos eventos al calendario. Las reglas de validación aseguran que los campos críticos del evento, como el título, el tipo de evento y las fechas de inicio y fin, estén correctamente llenados antes de que el usuario pueda agregar el evento.

Aunque la aplicación es funcional y cumple con los requisitos básicos, hay varias áreas de mejora y expansión para futuros trabajos. Por ejemplo, la persistencia de datos podría ser implementada para permitir a los usuarios guardar y recuperar sus eventos. Además, la integración con los servicios de calendario del teléfono o de cuentas en la nube podría proporcionar una funcionalidad adicional y hacer que la aplicación sea más útil.

En general, este proyecto ha proporcionado una valiosa experiencia práctica en el desarrollo de aplicaciones Android. Ha demostrado la importancia de una planificación cuidadosa, una buena organización del código y una atención meticulosa a los detalles. Aunque el desarrollo de aplicaciones puede ser un proceso complejo y desafiante, los resultados finales son gratificantes.