Instituto Tecnológico Superior de Purísima del Rincón

P1_TasksList

MATERIA:
Desarrollo de aplicaciones para dispositivos móviles

ALUMNOS:
Juan Antonio Pérez Cabrera

DOCENTE:
Jair Emmanuel Ramírez Flores

FECHA: 30 de octubre de 2025 **CIUDAD:** Purísima del Rincón, Gto.

Introducción

En el desarrollo de aplicaciones móviles combinado la experiencia del usuario y la navegación entre pantallas son elementos clave para garantizar la funcionalidad y usabilidad del producto. Esta práctica se enfoca en la creación de una interfaz grafica básica en la que el usuario puede tener una *To Do List*, la cual es una lista de actividades pendientes, en esta lista el usuario puede tachar las actividades para indicar que ha sido realizada, puede editar las tareas y eliminarlas si así lo desea. A través de view model y kotlin, fue posible desarrollar esta aplicación

Desarrollo

1. **Creamos un nuevo proyecto con empty activity**
2. **Creamos un nuevo archivo clase kotlin Task y creamos un data class con los atributos id, isDone, description, incializados.**

```
data class Task (
    val id: Int,
    val description: String,
    val isDone: Boolean = false
)
```

3. **Después creamos una clase kotlin llamada TaskViewModel. En la clase creamos una variable de tipo *Task (*como la data class*)***

```
class TaskViewModel : ViewModel() {
    private var nextID = 0
    var tasks = mutableStateListOf<Task>()
    private set
```

4. **Creamos la función addTask que recibe el parámetro description y lo agrega a task si no esta vacio**

```
fun addTask(description: String) {
    if (description.isNotBlank()) {
        tasks.add(Task(id = nextID++, description = description))
    }
}
```

5. **Creamos la función toogleTaskDone que recibe el parámetro task**

```
fun toggleTaskDone(task: Task) {
    val index = tasks.indexOf(task)
    if (index != -1) {
        val updated = task.copy(isDone = !task.isDone)
        tasks[index] = updated
    }
}
```

6. **Cremoas la function removetask, que recibe el parámetro task y lo elimina de la lista tasks**

```
fun removeTask(task: Task) {
    tasks.remove(element = task)
}
```

7. **Creamos la function editTask que permite editar o corregir una tarea ya agregada en la lista, que recibe el parámetro task y newTask, para saber que tarea se va a corregir y cual será la corrección para actualizar, si no esta vacio, se indexa en la misma ubicación que la tarea a editar**

```
fun editTask(task: Task, newTask: String) {
    val index = tasks.indexOf(task)
    if (index != -1 && newTask.isNotBlank()) {
        tasks[index] = task.copy(description = newTask)
    }
}
```

8. **Creamos un archivo Kotlin y creamos una función composable TaskItem, declaramos las variables *editionIcon* para saber cuando el usuario desea corregir y *correctionTask* que es la variable que guarda la corrección de la tarea.**

```
@Composable
fun TaskItem(
    task: Task,
    onToggle: () -> Unit,
    onDelete: () -> Unit,
    onEdit: (String) -> Unit
) {

    var editionIcon by remember { mutableStateOf(false) }
    var correctionTask by remember { mutableStateOf("") }

    Row(
        modifier = Modifier
            .fillMaxWidth()
```

```
            .padding(8.dp)
            .clickable { onToggle() },
        horizontalArrangement = Arrangement.Center
    )
```

9. **Colocamos una condicional if que detecta si el icono de edicion fue presionado, en tal caso, abre un campo de texto en la tarea que se va a corregir para que se pueda reescribir la tarea, reinicia el estado de editionIcon a false, en caso de que no detecte que fue presionado el botón del icono de edición solo muestra la descripción de la tarea en la lista de tareas.**

```
if (editionIcon) {
    TextField(
        value = correctionTask,
        onValueChange = { correctionTask = it },
        modifier = Modifier.weight(1f),
        singleLine = true
    )
    IconButton(onClick = {
        onEdit(correctionTask)
        editionIcon = false
    }) {
        Icon(Icons.Default.Edit, contentDescription = "Guardar
edición")
    }
} else {
    Text(
        text = task.description,
        style = if (task.isDone)
MaterialTheme.typography.bodyLarge.copy(
            color = MaterialTheme.colorScheme.primary,
            textDecoration = TextDecoration.LineThrough
        )
        else MaterialTheme.typography.bodyLarge
    )

    IconButton(onClick = { editionIcon = true }) {
        Icon(Icons.Default.Edit, contentDescription = "Editar tarea")
    }
}
```

10. Afuera de la condicional agregamos el botón para eliminar una tarea

```
IconButton(onClick = onDelete) {
    Icon(Icons.Default.Delete, contentDescription = "Eliminar
Tarea")
}
```

11. Creamos un nuevo archivo kotlin TaskListScreen

```
@Composable
fun TaskListScreen(viewModel: TaskViewModel){
    var input by remember { mutableStateOf("")}

    Column(
        modifier = Modifier
            .padding(16.dp)
    ) {
```

12. Creamos un Row con algunos spacer y el titulo “Mis Tareas” y con modifier hacemos que ocupe todo el espacio

```
Spacer(modifier = Modifier.height(30.dp))
Text(text = "Mis Tareas", style =
MaterialTheme.typography.headlineMedium)
Spacer(modifier = Modifier.height(15.dp))
Row(
    modifier = Modifier
        .fillMaxWidth(),
    horizontalArrangement = Arrangement.SpaceBetween
) {
```

13. Agregamos un TextField para que se ingrese la tarea y agregamos un botón de “Agregar” para que se agregue la tarea a la lista de tareas

```
TextField(
        value = input,
        onValueChange = { input = it },
        label = { Text("Nueva Tarea") },
        modifier = Modifier.weight(1f).padding(8.dp)
```

```
    )
    Spacer(modifier = Modifier.height(8.dp))

    Button(onClick = {
        viewModel.addTask(input)
        input = ""
    }) {
        Text(text = "Agregar")
    }
}//row

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

14. Con un LazyColumn

```
LazyColumn(
    modifier = Modifier.fillMaxWidth(),
    horizontalAlignment = Alignment.CenterHorizontally
) {
    items(viewModel.tasks) { task ->
        TaskItem(
            task = task,
            onToggle = { viewModel.toggleTaskDone(task) },
            onDelete = { viewModel.removeTask(task) },
            onEdit = { newDesc -> viewModel.editTask(task,
newDesc) }
        )
    }
}
```

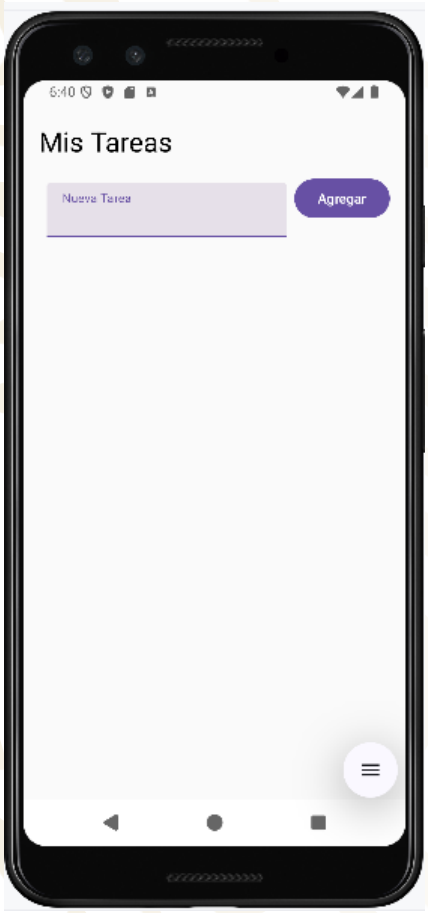
15. En el Activity creamos las variables con las que mandaremos llamar las funciones

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
          val characterViewModel: CharacterViewModel =
viewModel()
```

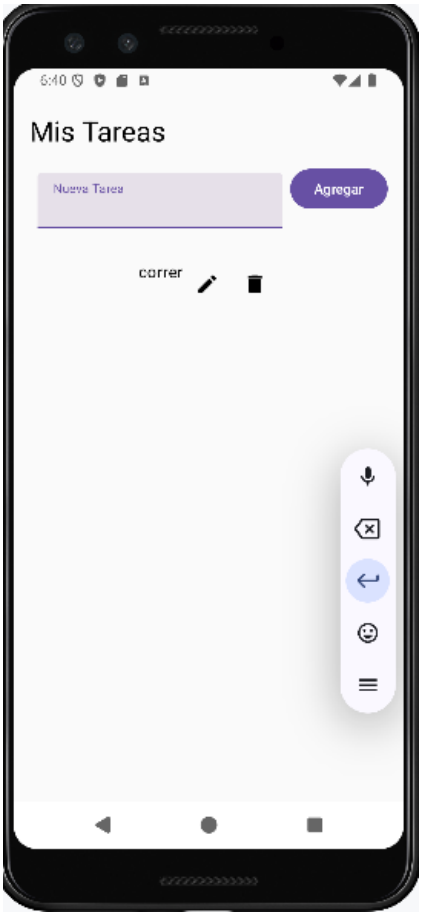
```
            CharacterListScreen(viewModel = characterViewModel)
        }
    }
}
```

Resultados

Pantalla sin tareas



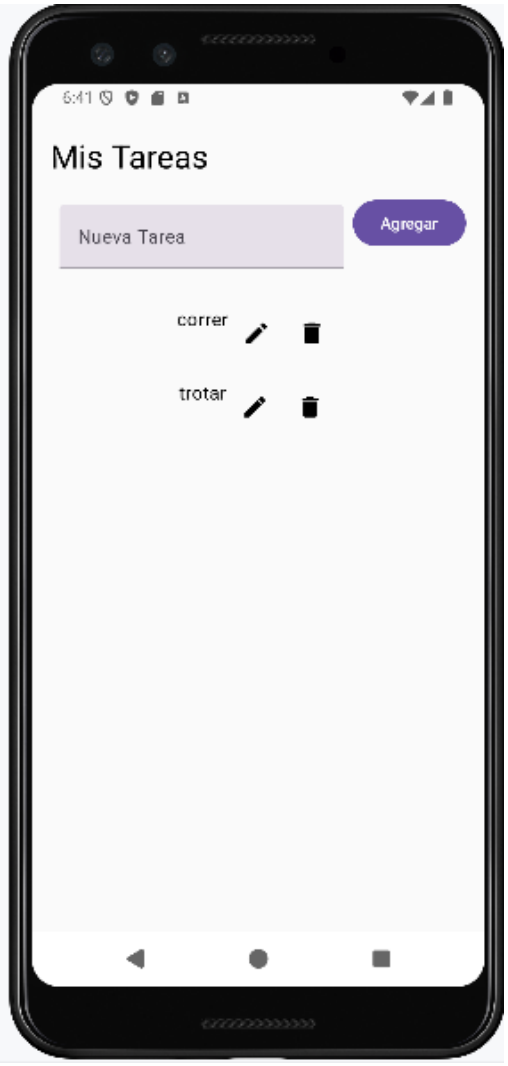
Agregando una tarea



Agregando una segunda tarea

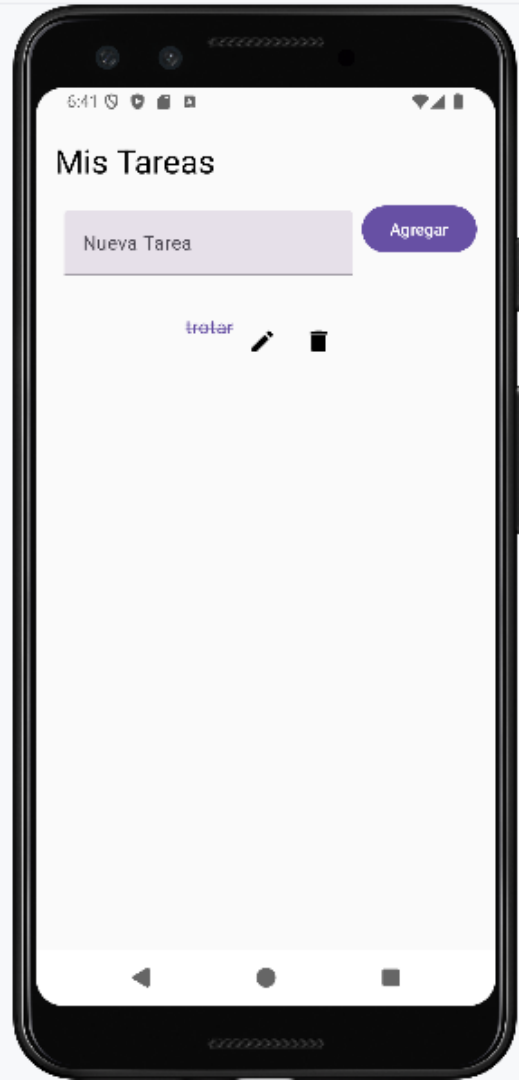
Al dar clic en el icono del lápiz (editar) se abre el campo de texto para introducir la tarea corregida

Escribimos la corrección y al dar clic en el icono de lápiz, se guardan los cambios y se visualizan en la lista de tareas



Al dar clic sobre el texto de una tarea, se tacha automáticamente indicando que ha sido completada, así mismo se puedes deshacer el cambio volviendo a darle clic a la tarea

Al darle clic al botón de eliminar se elimina la tarea



Conclusión

La implementación de las funciones de editar y eliminar tareas en esta practica fueron importantes ya que eso le da un valor agregado a la aplicación, pues de lo contrario solo sería una lista de actividades interminable y quizá hasta errónea. El uso de los viewmodel también fue clave para lograr el desarrollo de esta app, ya que evita tener códigos largos y que no se puedan reciclar, pero gracias a la herencia es posible. El refuerzo de conocimientos sobre viewmodel y clases es crucial para poder codificar una aplicación como esta.