



**Universidad Tecnológica
de Chihuahua**

TECNOLOGIAS DE LA INFORMACIÓN

DISEÑO DE APPS

**PROGRAMA INFORMÁTICO DE CLASES,
CONSIDERANDO EL PATRÓN DE DISEÑO:
MODELO, VISTA, CONTROLADOR.**

TID71D

LUIS ENRIQUE MASCOTE CANO

JORGE LUIS ROMAN MONTAÑO

02 OCTUBRE DEL 2025

1123150136

Contenido

Descripción de la aplicación	2
Funcionalidades principales:	2
Diagrama de Clases (MVC)	2
Explicación de cada componente	3
◆ Modelo.....	3
◆ Vista.....	5
◆ Controlador.....	15
Capturas de pantalla de la aplicación funcionando.....	20
Análisis de ventajas del patrón MVC implementado	24
Conclusiones	25

Descripción de la aplicación

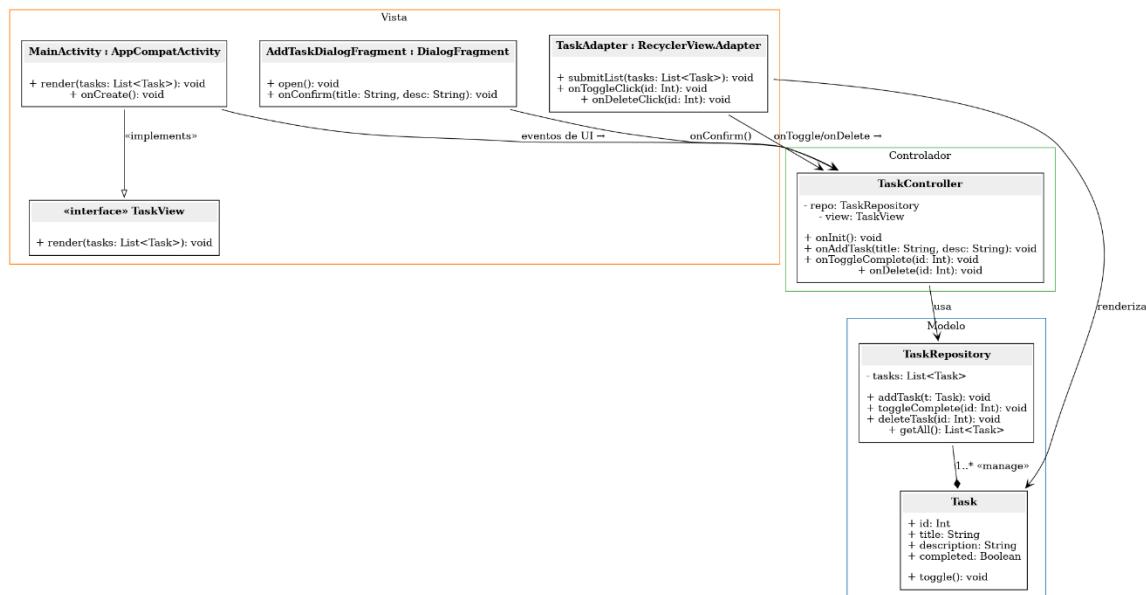
La aplicación **Lista de Tareas** es un sistema sencillo de gestión personal que permite al usuario organizar actividades pendientes.

Funcionalidades principales:

- **Agregar tareas:** el usuario puede registrar nuevas tareas indicando un título y una descripción.
- **Marcar tareas como completadas:** las tareas pueden cambiar de estado (pendiente → completada).
- **Eliminar tareas:** se permite eliminar permanentemente una tarea.
- **Visualizar la lista de tareas:** muestra todas las tareas registradas en un listado.

La aplicación fue desarrollada siguiendo el patrón de diseño **Modelo-Vista-Controlador (MVC)**, con el objetivo de separar responsabilidades, mejorar el mantenimiento del código y permitir escalabilidad.

Diagrama de Clases (MVC)



Explicación de cada componente

◊ Modelo

El **Modelo** representa los datos y la lógica de negocio.

- Clase **Tarea**: contiene atributos como id, título, descripción y estado.
- Métodos para actualizar el estado de la tarea y consultar sus valores.

```
1 package com.utch.tareasapp.modelo
2
3 /**
4  * CLASE MODELO: Tarea
5  * Esta clase representa una tarea individual usando un data class de Kotlin.
6  * Contiene todos los datos que necesita una tarea.
7  * En el patrón MVC, esta es parte del MODELO.
8 */
9 33 Usages
10 data class Tarea(
11     // ===== PROPIEDADES (Variables que guardan la información) =====
12     val id: Int,                                // Número Único para identificar la tarea
13     var titulo: String,                          // El nombre/título de la tarea
14     var descripcion: String,                    // Descripción detallada de la tarea
15     var completada: Boolean = false,           // true = completada, false = pendiente
16     val fechaCreacion: Long = System.currentTimeMillis() // Cuándo se creó la tarea
17 )
18 // ===== MÉTODOS ADICIONALES =====
19
20 /**
21  * Cambia el estado de completada (alterna entre true/false)
22 */
23 1 Usage
24 fun alternarCompletada() {
25     completada = !completada
26 }
27 /**
28  * Verifica si la tarea fue creada hoy
29  * @return true si la tarea se creó hoy
30 */
31 1 Usage
32 fun esDeHoy(): Boolean {
33     val hoy = System.currentTimeMillis()
34     val unDia = 24 * 60 * 60 * 1000 // milisegundos en un día
35     return (hoy - fechaCreacion) < unDia
36 }
```

```

9     data class Tarea(
10         fun esDeHoy(): Boolean {
11             val unDia = 24 * 60 * 60 * 1000 // milisegundos en un día
12             return (hoy - fechaCreacion) < unDia
13         }
14
15         /**
16          * Obtiene un resumen de la tarea para mostrar en la lista
17          * @return texto con formato para mostrar
18          */
19         fun getResumen(): String {
20             val estado = if (completada) "✓" else "✗"
21             return "$estado $titulo"
22         }
23
24         /**
25          * Verifica si el título contiene una palabra específica
26          * @param palabra - palabra a buscar
27          * @return true si contiene la palabra (sin importar mayúsculas/minúsculas)
28          */
29         Usage
30         fun contienePalabra(palabra: String): Boolean {
31             return titulo.contains(other = palabra, ignoreCase = true) ||
32                 descripcion.contains(other = palabra, ignoreCase = true)
33         }
34
35         /**
36          * Método toString personalizado para debug y logging
37          * @return representación en string de la tarea
38          */
39         override fun toString(): String {
40             return "Tarea(id=$id, titulo='$titulo', completada=$completada)"
41         }
42     }

```

◊ Vista

La **Vista** corresponde a la interfaz gráfica presentada al usuario.

- Muestra la lista de tareas en pantalla.
- Incluye formularios para agregar nuevas tareas.
- Actualiza dinámicamente la visualización cuando una tarea se marca como completada o eliminada.

```
1  package com.utch.tareasapp.vista
2
3  > import ...
4
5  /**
6   * CLASE VISTA: MainActivity
7   * Activity principal que implementa una interfaz moderna usando Kotlin.
8   * Utiliza RecyclerView, Material Design y gestión moderna del estado.
9   * En el patrón MVC, esta es la VISTA principal.
10  */
11
12  5 Usages
13  class MainActivity : AppCompatActivity() {
14
15      // ===== PROPIEDADES =====
16      12 Usages
17      private lateinit var controlador: TareaControlador
18      4 Usages
19      private lateinit var adapter: TareaAdapter
20
21      // Views (sin ViewBinding para simplicidad en el tutorial)
22      4 Usages
23      private lateinit var editTextTitulo: com.google.android.material.textfield.TextInputEditText
24      2 Usages
25      private lateinit var buttonAgregar: com.google.android.material.button.MaterialButton
26      3 Usages
27      private lateinit var buttonLimpiar: com.google.android.material.button.MaterialButton
28      4 Usages
29      private lateinit var recyclerViewTareas: RecyclerView
30      2 Usages
31      private lateinit var textViewEstadisticas: android.widget.TextView
32      2 Usages
33      private lateinit var layoutVacio: android.widget.LinearLayout
34
35      // ===== CICLO DE VIDA =====
36
37      /**
38       * Se ejecuta cuando se crea la Activity
39       */
40      override fun onCreate(savedInstanceState: Bundle?) {
```

```
21     class MainActivity : AppCompatActivity() {
22         /*
23         @†      override fun onCreate(savedInstanceState: Bundle?) {
24             super.onCreate(savedInstanceState)
25             setContentView(R.layout.activity_main)
26
27             // Inicialización en orden
28             inicializarControlador()
29             inicializarVistas()
30             configurarRecyclerView()
31             configurarEventos()
32             actualizarInterfaz()
33
34             // Mostrar mensaje de bienvenida
35             mostrarMensaje( mensaje = "¡Bienvenido a tu Lista de Tareas!", TipoMensaje.INFO)
36         }
37
38         /**
39          * Se ejecuta cuando la Activity se reanuda
40          */
41         override fun onResume() {
42             super.onResume()
43             actualizarInterfaz()
44         }
45
46         // ===== MÉTODOS DE INICIALIZACIÓN =====
47
48         /**
49          * Inicializa el controlador MVC
50          */
51         1 Usage
52         private fun inicializarControlador() {
53             controlador = TareaControlador( vista = this)
54
55             // Configurar callback para cambios de datos
56             controlador.onDataChanged = {
57                 actualizarEstadisticas()
58             }
59         }
60     }
```

```
21     class MainActivity : AppCompatActivity() {
22         private fun inicializarControlador() {
23             ...
24
25             /**
26             * Inicializa todas las vistas
27             */
28             1 Usage
29             private fun inicializarVistas() {
30                 editTextTitulo = findViewById( id = R.id.editTextTitulo)
31                 buttonAgregar = findViewById( id = R.id.buttonAgregar)
32                 buttonLimpiar = findViewById( id = R.id.buttonLimpiar)
33                 recyclerViewTareas = findViewById( id = R.id.recyclerViewTareas)
34                 textViewEstadisticas = findViewById( id = R.id.textViewEstadisticas)
35                 layoutVacio = findViewById( id = R.id.layoutVacio)
36             }
37
38             /**
39             * Configura el RecyclerView y su adapter
40             */
41             1 Usage
42             private fun configurarRecyclerView() {
43                 // Crear y configurar adapter
44                 adapter = TareaAdapter().apply {
45                     // Configurar listeners usando SAM (Single Abstract Method)
46                     onTareaClickListener = TareaAdapter.OnTareaClickListener { tarea ->
47                         mostrarDetallesTarea(tarea)
48                     }
49
50                     onCompletadaChangeListener = TareaAdapter.OnCompletadaChangeListener { tareaId ->
51                         manejarCambioCompletada(tareaId)
52                     }
53
54                     onEliminarTareaListener = TareaAdapter.OnEliminarTareaListener { tareaId ->
55                         manejarEliminarTarea(tareaId)
56                     }
57                 }
58
59                 // Configurar RecyclerView
```

```
21 class MainActivity : AppCompatActivity() {
22     private fun configurarRecyclerView() {
23
24         // Configurar RecyclerView
25         recyclerViewTareas.apply {
26             this.adapter = this@MainActivity.adapter
27             layoutManager = LinearLayoutManager(context = this@MainActivity)
28             setHasFixedSize(true)
29         }
30
31         // Agregar ItemTouchHelper para swipe to delete
32         configurarSwipeToDelete()
33     }
34
35     /**
36      * Configura gestos de deslizar para eliminar
37      */
38
39     1 Usage
40     private fun configurarSwipeToDelete() {
41         val itemTouchHelper = ItemTouchHelper(callback = object : ItemTouchHelper.SimpleCallback(
42             dragDirs = 0, swipeDirs = ItemTouchHelper.LEFT or ItemTouchHelper.RIGHT
43         ) {
44             override fun onMove(
45                 recyclerView: RecyclerView,
46                 viewHolder: RecyclerView.ViewHolder,
47                 target: RecyclerView.ViewHolder
48             ): Boolean = false
49
50             override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
51                 val position = viewHolder.adapterPosition
52                 val tarea = adapter.getTareaAt(position)
53                 tarea?.let {
54                     manejarEliminarTarea(tareaId = it.id)
55                 }
56             }
57         })
58
59         itemTouchHelper.attachToRecyclerView(recyclerView = recyclerViewTareas)
60     }
61 }
```

```

21     class MainActivity : AppCompatActivity() {
22         private fun configurarSwipeToDelete() {
23
24             itemTouchHelper.attachToRecyclerView( recyclerView = recyclerViewTareas)
25         }
26
27         /**
28          * Configura todos los eventos de la interfaz
29         */
30         1 Usage
31         private fun configurarEventos() {
32             // ===== BOTÓN AGREGAR =====
33             buttonAgregar.setOnClickListener {
34                 manejarAgregarTarea()
35             }
36
37             // ===== BOTÓN LIMPIAR COMPLETADAS =====
38             buttonLimpiar.setOnClickListener {
39                 manejarLimpiarCompletadas()
40             }
41
42             // ===== ENTER EN CAMPO DE TEXTO =====
43             editTextTitulo.setOnEditorActionListener { _, _, _ ->
44                 manejarAgregarTarea()
45                 true
46             }
47         }
48
49         // ===== MÉTODOS DE MANEJO DE EVENTOS =====
50
51         /**
52          * Maneja la adición de una nueva tarea
53         */
54         2 Usages
55         private fun manejarAgregarTarea() {
56             val titulo = editTextTitulo.text?.toString()?.trim() ?: ""
57
58             when (val resultado = controlador.agregarNuevaTarea(titulo)) {
59                 is TareaControlador.ResultadoOperacion.Exitoso -> {

```

```
21     class MainActivity : AppCompatActivity() {
171         private fun manejarAgregarTarea() {
175             is TareaControlador.ResultadoOperacion.Exitoso -> {
176                 editTextTitulo.text?.clear()
177                 mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.EXITO)
178                 actualizarInterfaz()
179             }
180             is TareaControlador.ResultadoOperacion.Error -> {
181                 mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.ERROR)
182             }
183             is TareaControlador.ResultadoOperacion.Info -> {
184                 mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.INFO)
185             }
186         }
187     }
188
189     /**
190      * Maneja el cambio de estado completada/pendiente
191      */
192     1 Usage
193     private fun manejarCambioCompletada(tareaId: Int) {
194         when (val resultado = controlador.alternarCompletada(tareaId)) {
195             is TareaControlador.ResultadoOperacion.Exitoso -> {
196                 mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.EXITO)
197                 actualizarInterfaz()
198             }
199             is TareaControlador.ResultadoOperacion.Error -> {
200                 mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.ERROR)
201                 // Revertir cambio en la interfaz
202                 actualizarLista()
203             }
204             else -> {
205                 actualizarInterfaz()
206             }
207         }
208     }
209
210     /**
211      * Maneja la eliminación de una tarea
212      */
213 }
```

```
21     class MainActivity : AppCompatActivity() {
209
210         /**
211          * Maneja la eliminación de una tarea
212          */
213
214         2 Usages
215         private fun manejarEliminarTarea(tareaId: Int) {
216             // Obtener datos de la tarea antes de eliminar para el undo
217             val tarea = controlador.obtenerDetalleTarea(tareaId)
218
219             when (val resultado = controlador.eliminarTarea(tareaId)) {
220                 is TareaControlador.ResultadoOperacion.Exitoso -> {
221                     actualizarInterfaz()
222
223                     // Mostrar Snackbar con opción de deshacer
224                     tarea?.let { tareaEliminada ->
225                         mostrarSnackbarUndo( mensaje = resultado.mensaje, tareaEliminada)
226                     }
227
228                 is TareaControlador.ResultadoOperacion.Error -> {
229                     mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.ERROR)
230                 }
231                 else -> {
232                     actualizarInterfaz()
233                 }
234             }
235
236             /**
237              * Maneja la limpieza de tareas completadas
238              */
239
240             1 Usage
241             private fun manejarLimpiarCompletadas() {
242                 when (val resultado = controlador.limpiarCompletadas()) {
243                     is TareaControlador.ResultadoOperacion.Exitoso -> {
244                         mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.EXITO)
245                         actualizarInterfaz()
246                     }
247                     is TareaControlador.ResultadoOperacion.Info -> {
248                         mostrarMensaje( mensaje = resultado.mensaje, TipoMensaje.INFO)
```

```

21     class MainActivity : AppCompatActivity() {
237     private fun manejarLimpiarCompletadas() {
244         if (resultado is TareaControlador.ResultadoOperacion.Error) {
245             mostrarMensaje(mensaje = resultado.mensaje, TipoMensaje.INFO)
246         } else if (resultado is TareaControlador.ResultadoOperacion.Error) {
247             mostrarMensaje(mensaje = resultado.mensaje, TipoMensaje.ERROR)
248         }
249     }
250 }
251
252 /**
253 * Muestra los detalles de una tarea en un Toast
254 */
255 1 Usage
256 private fun mostrarDetallesTarea(tarea: Tarea) {
257     val detalles = buildString {
258         appendLine(value = "■ ${tarea.titulo}")
259         if (tarea.descripcion.isNotBlank()) {
260             appendLine(value = "▣ ${tarea.descripcion}")
261         }
262         appendLine(value = "▢ ID: ${tarea.id}")
263         appendLine(value = "▢ ${if (tarea.esDeHoy()) "Creada hoy" else "Creada anteriormente"})
264         append("▢ ${if (tarea.completada) "Completada" else "Pendiente"})
265     }
266     Toast.makeText(context = this, text = detalles, duration = Toast.LENGTH_LONG).show()
267 }
268
269 // ===== MÉTODOS PÚBLICOS (LLAMADOS DESDE EL CONTROLADOR) =====
270
271 /**
272 * Actualiza la lista de tareas (llamado desde el controlador)
273 */
274 3 Usages
275 fun actualizarLista() {
276     val tareas = controlador.obtenerListaTareas()
277     adapter.submitList(list = tareas.toList()) // Crear nueva lista para trigger DiffUtil
278 }
```

```
21 class MainActivity : AppCompatActivity() {
22
23     fun actualizarLista() {
24         val tareas = controlador.obtenerListaTareas()
25         adapter.submitList( list = tareas.toList() ) // Crear nueva lista para trigger DiffUtil
26     }
27
28
29     /**
30      * Actualiza toda la interfaz
31      */
32
33     8 Usages
34     private fun actualizarInterfaz() {
35         actualizarLista()
36         actualizarEstadisticas()
37         actualizarVisibilidadVistas()
38     }
39
40
41     /**
42      * Actualiza las estadísticas mostradas
43      */
44
45     2 Usages
46     private fun actualizarEstadisticas() {
47         textViewEstadisticas.text = controlador.obtenerEstadisticas()
48     }
49
50
51     /**
52      * Actualiza la visibilidad de vistas según el estado
53      */
54
55     1 Usage
56     private fun actualizarVisibilidadVistas() {
57         val hayTareas = controlador.hayTareas()
58
59             recyclerViewTareas.visibility = if (hayTareas) View.VISIBLE else View.GONE
60             layoutVacio.visibility = if (hayTareas) View.GONE else View.VISIBLE
61             buttonLimpiar.isEnabled = controlador.obtenerTareasCompletadas().isNotEmpty()
62     }
63
64
65     // ===== MÉTODOS DE UTILIDAD =====
66
67     /**
68
```

```
21     class MainActivity : AppCompatActivity() {
204     }
205
206     // ===== MÉTODOS DE UTILIDAD =====
207
208     /**
209      * Muestra un mensaje usando diferentes métodos según el tipo
210     */
211     11 Usages
212     private fun mostrarMensaje(mensaje: String, tipo: TipoMensaje) {
213         when (tipo) {
214             TipoMensaje.EXITO -> {
215                 Toast.makeText(context = this, text = "✅ $mensaje", duration = Toast.LENGTH_SHORT).show()
216             }
217             TipoMensaje.ERROR -> {
218                 Toast.makeText(context = this, text = "❌ $mensaje", duration = Toast.LENGTH_LONG).show()
219             }
220             TipoMensaje.INFO -> {
221                 Toast.makeText(context = this, text = "ℹ️ $mensaje", duration = Toast.LENGTH_SHORT).show()
222             }
223         }
224     }
225
226     /**
227      * Muestra Snackbar con opción de deshacer
228     */
229     1 Usage
230     private fun mostrarSnackbarUndo(mensaje: String, tareaEliminada: Tarea) {
231         Snackbar.make(view = findViewById(id = android.R.id.content), text = mensaje, duration = Snackbar.LENGTH_LONG)
232             .setAction(text = "DESHACER") {
233                 // Funcionalidad para restaurar tarea (implementación básica)
234                 controlador.agregarNuevaTarea(titulo = tareaEliminada.titulo, descripcion = tareaEliminada.descripcion)
235                 mostrarMensaje(mensaje = "Tarea restaurada", TipoMensaje.INFO)
236             }
237             .show()
238     }
239
240     // ===== ENUM PARA TIPOS DE MENSAJE =====
```

◊ Controlador

El **Controlador** actúa como intermediario.

- Recibe la interacción del usuario desde la vista (botones, formularios).
- Modifica el modelo según la acción (crear, actualizar, eliminar).
- Envía los cambios a la vista para que se actualice la interfaz.

```
1 package com.utch.tareasapp.controlador
2
3 > import ...
4
5 /**
6  * CLASE CONTROLADOR: TareaControlador
7  * Esta clase es el "intermediario" entre la Vista (pantalla) y el Modelo (datos).
8  * Usa características modernas de Kotlin como funciones lambda y propiedades.
9  * En el patrón MVC, esta es el CONTROLADOR.
10 */
11
12
13 class TareaControlador(private val vista: MainActivity) {
14     // ===== PROPIEDADES =====
15     private val modelo = TareaManager()    // Instancia del administrador de datos
16
17     // Callback para notificar cambios a la vista
18     var onDataChange: (() -> Unit)? = null
19
20     // ===== MÉTODOS PÚBLICOS (API del Controlador) =====
21
22     /**
23      * OBTENER la lista de todas las tareas
24      * @return lista inmutable de tareas
25      */
26     fun obtenerListaTareas(): List<Tarea> = modelo.obtenerTareas()
27
28     /**
29      * AGREGAR una nueva tarea con validación
30      * @param titulo - Título que escribió el usuario
31      * @param descripción - Descripción que escribió el usuario (opcional)
32      * @return resultado de la operación
33      */
34     fun agregarNuevaTarea(titulo: String, descripción: String = ""): ResultadoOperacion {
35         return when {
36             titulo.isBlank() -> {
```

```

13  class TareaControlador(private val vista: MainActivity) {
14      fun agregarNuevaTarea(titulo: String, descripcion: String = ""): ResultadoOperacion {
15          if (titulo.isBlank()) -> {
16              return ResultadoOperacion.Error(mensaje = "El titulo no puede estar vacío")
17          }
18          if (titulo.length > 100) -> {
19              return ResultadoOperacion.Error(mensaje = "El titulo es demasiado largo (máximo 100 caracteres)")
20          }
21          else -> {
22              try {
23                  val nuevaTarea = modelo.agregarTarea(titulo, descripcion)
24                  notificarCambio()
25                  return ResultadoOperacion.Exitoso(mensaje = "Tarea agregada: ${nuevaTarea.titulo}")
26              } catch (e: Exception) {
27                  return ResultadoOperacion.Error(mensaje = "Error al agregar tarea: ${e.message}")
28              }
29          }
30      }
31  }
32
33
34  /**
35   * CAMBIAR el estado de una tarea (completada/pendiente)
36   * @param id - ID de la tarea que se quiere cambiar
37   * @return resultado de la operación
38   */
39  1 Usage
40  fun alternarCompletada(id: Int): ResultadoOperacion {
41      return if (modelo.alternarCompletada(id)) {
42          val tarea = modelo.obtenerTareaPorId(id)
43          val estado = if (tarea?.completada == true) "completada" else "pendiente"
44          notificarCambio()
45          return ResultadoOperacion.Exitoso(mensaje = "Tarea marcada como $estado")
46      } else {
47          return ResultadoOperacion.Error(mensaje = "No se encontró la tarea")
48      }
49  }
50
51  /**
52   * ELIMINAR una tarea
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

```
13 class TareaControlador(private val vista: MainActivity) {
14
15     /**
16      * ELIMINAR una tarea
17      * @param id - ID de la tarea que se quiere eliminar
18      * @return resultado de la operación
19      */
20
21     1 Usage
22     fun eliminarTarea(id: Int): ResultadoOperacion {
23         val tarea = modelo.obtenerTareaPorId(id)
24         return if (modelo.eliminarTarea(id)) {
25             notificarCambio()
26             ResultadoOperacion.Exitoso( mensaje = "Tarea eliminada: ${tarea?.titulo ?: "Desconocida"}")
27         } else {
28             ResultadoOperacion.Error( mensaje = "No se pudo eliminar la tarea")
29         }
30     }
31
32
33     /**
34      * OBTENER los detalles de una tarea específica
35      * @param id - ID de la tarea que queremos ver
36      * @return la tarea con sus detalles, o null si no existe
37      */
38
39     1 Usage
40     fun obtenerDetalleTarea(id: Int): Tarea? = modelo.obtenerTareaPorId(id)
41
42
43     /**
44      * OBTENER estadísticas formateadas
45      * @return string con estadísticas actuales
46      */
47
48     1 Usage
49     fun obtenerEstadisticas(): String = modelo.obtenerEstadisticas()
50
51
52     /**
53      * BUSCAR tareas por palabra clave
54      * @param busqueda - texto a buscar
55      * @return lista de tareas que coinciden con la búsqueda
56      */
57
58     fun buscarTareas(busqueda: String): List<Tarea> = modelo.buscarTareas( palabra = busqueda)
```

```

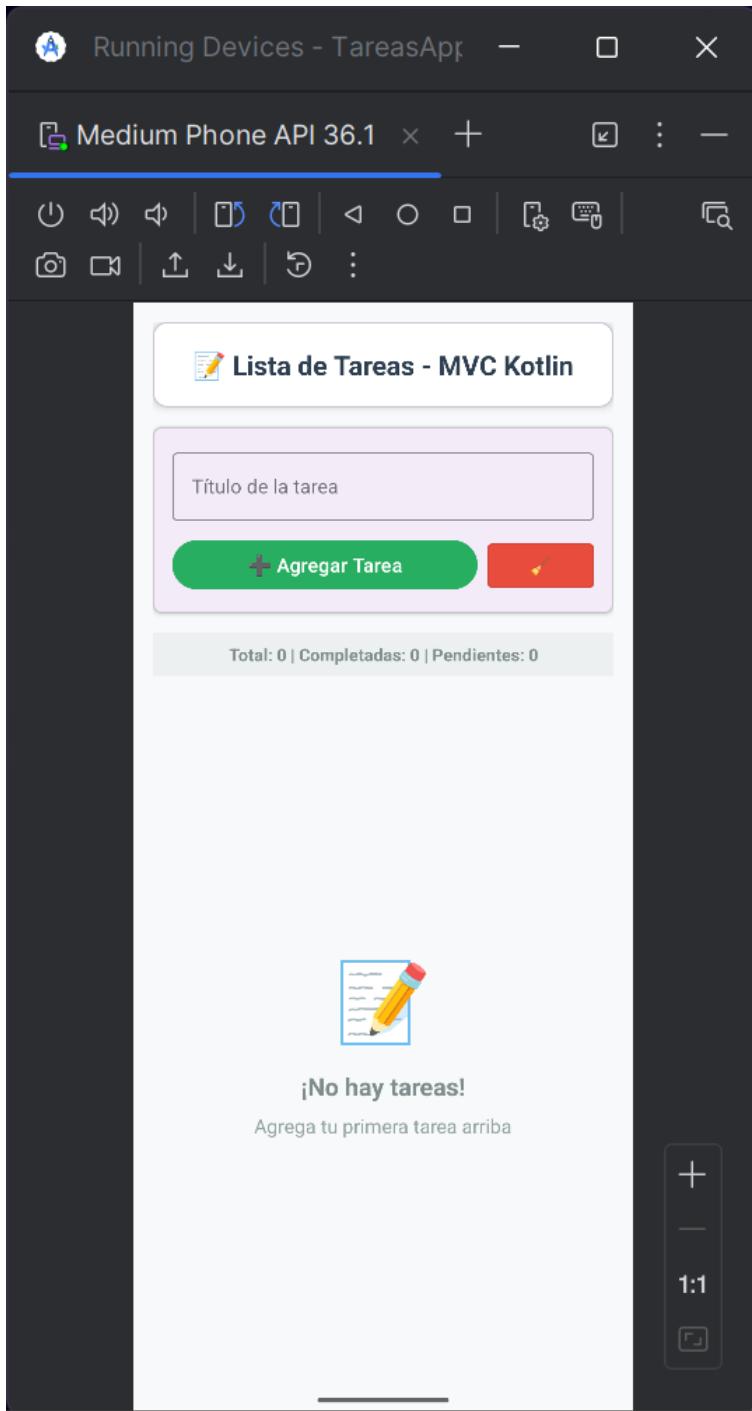
13  class TareaControlador(private val vista: MainActivity) {
103     fun buscarTareas(busqueda: String): List<Tarea> = modelo.buscarTareas(palabra = busqueda)
104
105    /**
106     * LIMPIAR todas las tareas completadas
107     * @return resultado de la operación con cantidad eliminada
108     */
109    1 Usage
110    fun limpiarCompletadas(): ResultadoOperacion {
111        val cantidad = modelo.limpiarCompletadas()
112        return if (cantidad > 0) {
113            notificarCambio()
114            ResultadoOperacion.Exitoso(mensaje = "$cantidad tareas completadas eliminadas")
115        } else {
116            ResultadoOperacion.Info(mensaje = "No hay tareas completadas para eliminar")
117        }
118    }
119    /**
120     * OBTENER solo tareas completadas
121     */
122    1 Usage
123    fun obtenerTareasCompletadas(): List<Tarea> = modelo.obtenerTareasCompletadas()
124
125    /**
126     * OBTENER solo tareas pendientes
127     */
128    fun obtenerTareasPendientes(): List<Tarea> = modelo.obtenerTareasPendientes()
129
130    /**
131     * VERIFICAR si hay tareas
132     */
133    1 Usage
134    fun hayTareas(): Boolean = !modelo.estaVacia()
135
136    /**
137     * Notifica a la vista que los datos han cambiado

```

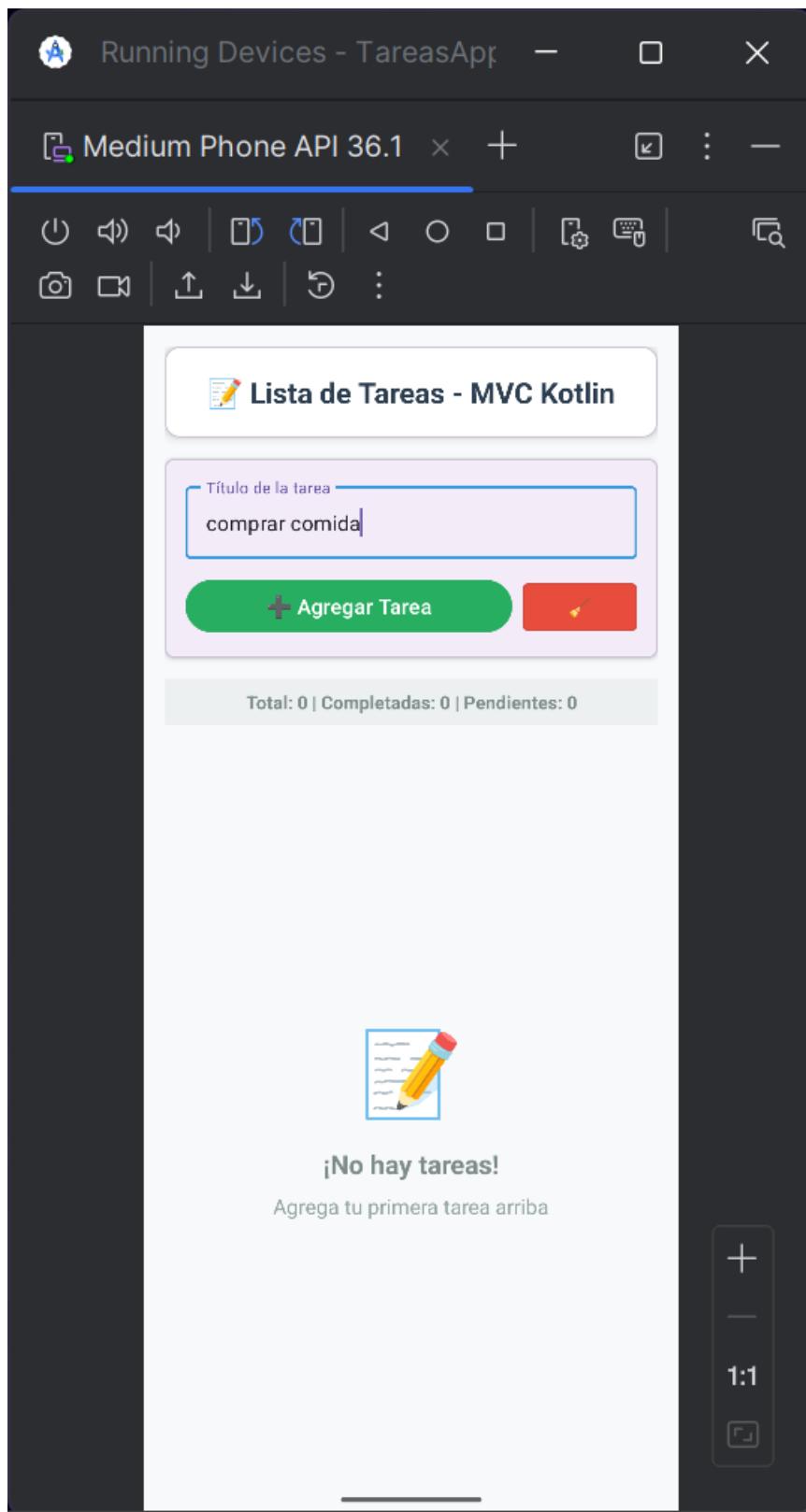
```
13     class TareaControlador(private val vista: MainActivity) {
130         * VERIFICAR si hay tareas
131         */
132         1 Usage
133         fun hayTareas(): Boolean = !modelo.estaVacia()
134         // ===== MÉTODOS PRIVADOS =====
135
136         /**
137         * Notifica a la vista que los datos han cambiado
138         */
139         4 Usages
140         private fun notificarCambio() {
141             vista.actualizarLista()
142             onDataChanged?.invoke()
143         }
144         // ===== CLASE SELLADA para resultados de operaciones =====
145         /**
146         * Representa el resultado de una operación del controlador
147         */
148         @ 27 Usages 3 Inheritors
149         sealed class ResultadoOperacion(val mensaje: String) {
150             9 Usages
151                 class Exitoso(mensaje: String) : ResultadoOperacion(mensaje)
152                 10 Usages
153                 class Error(mensaje: String) : ResultadoOperacion(mensaje)
154                 3 Usages
155                 class Info(mensaje: String) : ResultadoOperacion(mensaje)
156
157                 val esExitoso: Boolean get() = this is Exitoso
158                 val esError: Boolean get() = this is Error
159             }
160         }
```

Capturas de pantalla de la aplicación funcionando

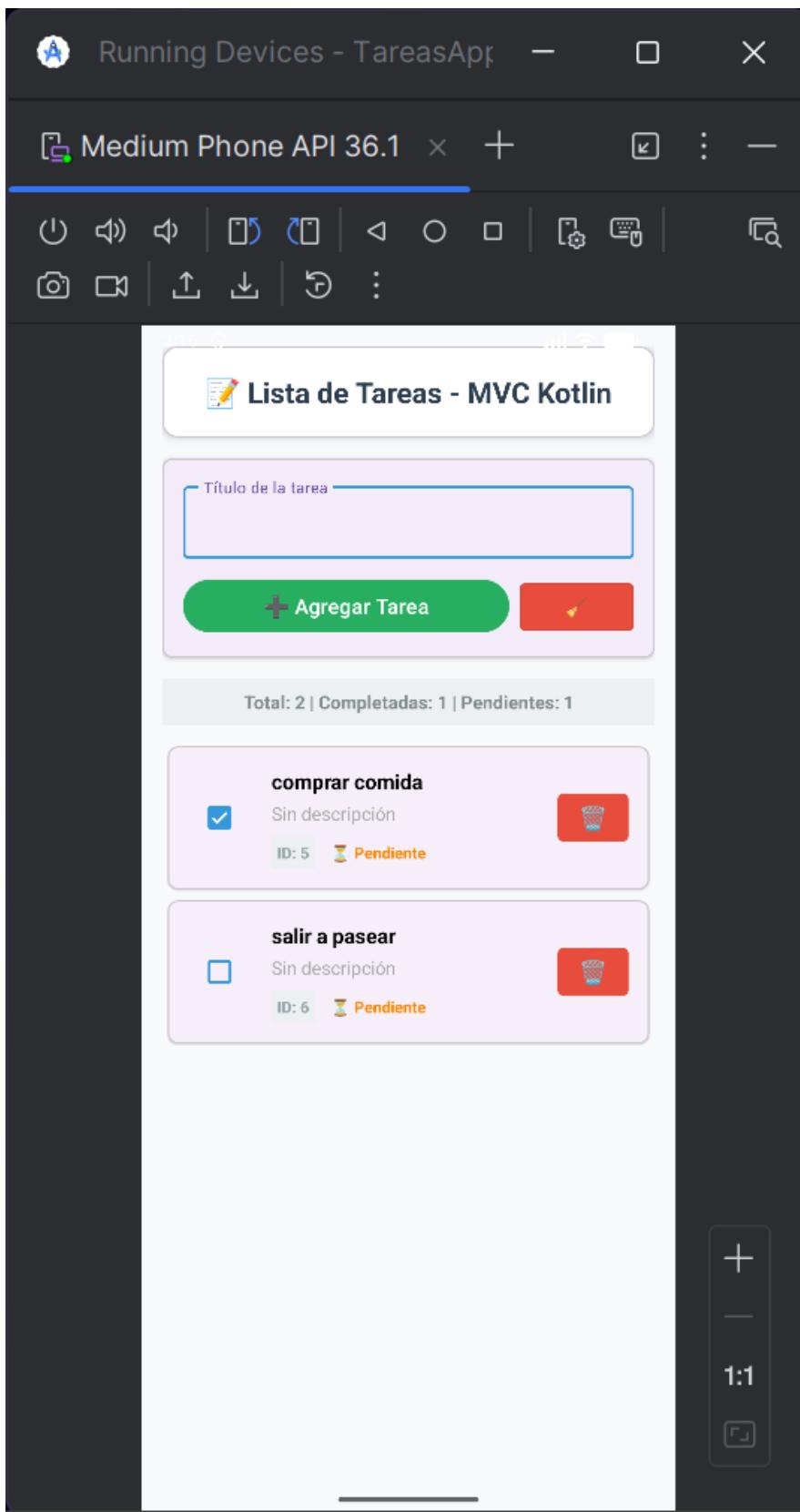
1. Pantalla principal mostrando la lista vacía.



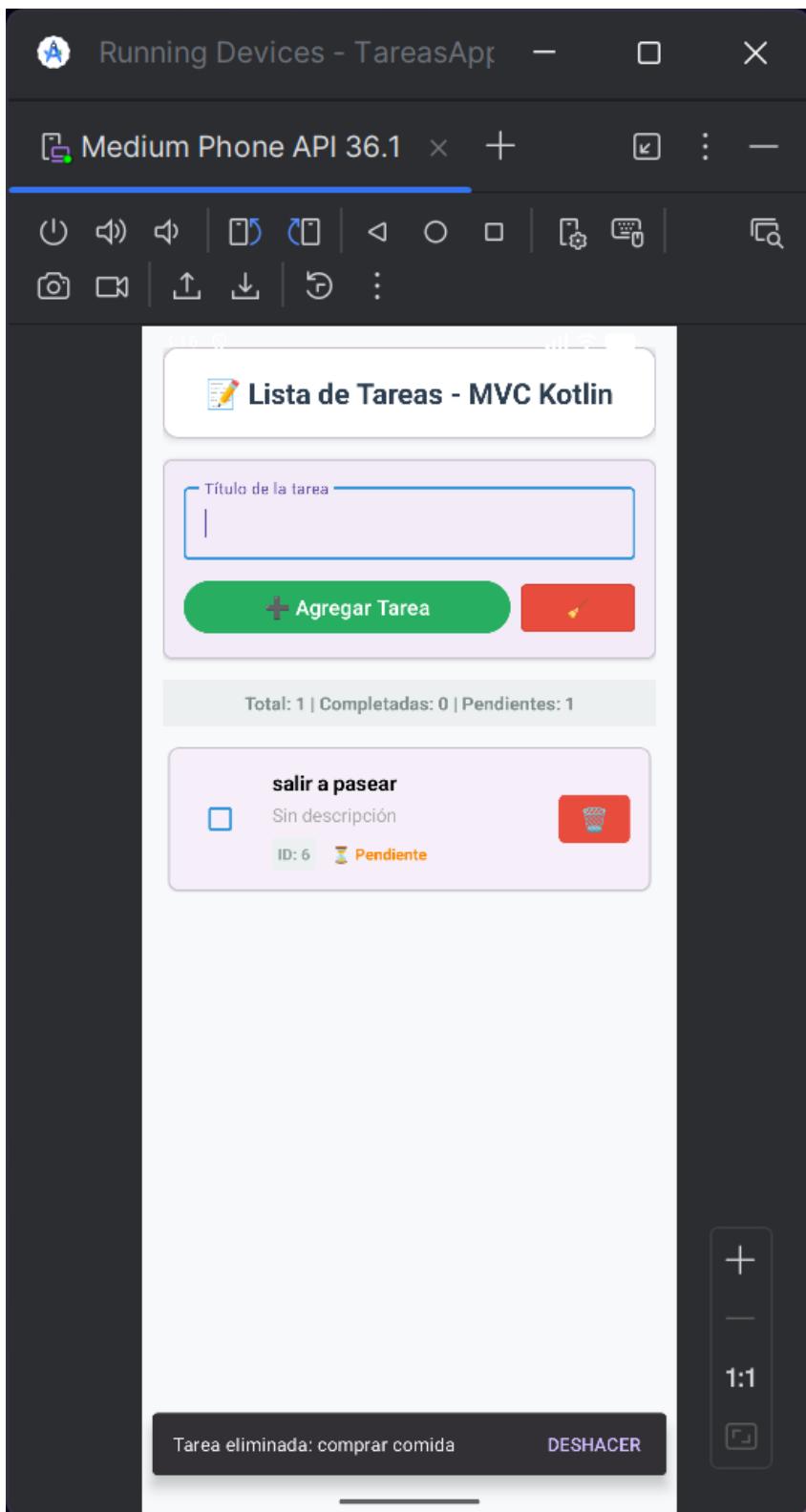
2. Formulario para agregar una nueva tarea.



3. Lista de tareas con una tarea completada.



4. Ejemplo de eliminación de una tarea.



Análisis de ventajas del patrón MVC implementado

- **Separación de responsabilidades:** la lógica de negocio está separada de la interfaz.
- **Facilidad de mantenimiento:** los cambios en la interfaz no afectan la lógica interna.
- **Escalabilidad:** es posible agregar nuevas funciones (ejemplo: categorías de tareas, prioridad) sin reescribir todo el código.
- **Reutilización:** el modelo puede ser utilizado en otras aplicaciones o interfaces.

Conclusiones

El uso del patrón **MVC** en el desarrollo de la aplicación de Lista de Tareas me permitió comprender cómo separar la lógica de negocio de la interfaz de usuario. Esto facilitó la organización del código, lo hizo más entendible y flexible para futuras mejoras.

En comparación con un enfoque sin patrón, noté que MVC aporta claridad en la estructura y permite trabajar en equipo de forma más ordenada, ya que cada integrante puede enfocarse en una capa diferente (Modelo, Vista o Controlador).

Como conclusión, considero que **MVC es una herramienta práctica y eficaz** para proyectos académicos y profesionales, y planeo seguir aplicándola en futuros desarrollos.