🎨 Camada `Presentation` – Descrição Completa dos Arquivos

📁 Estrutura

```
Presentation/
├── ViewModels/
│   └── TaskListViewModel.swift
└── Views/
    └── TaskListView.swift
```

📄 `Presentation/ViewModels/TaskListViewModel.swift`

📌 Função

Atuar como ponte entre a UI e os casos de uso (UseCases), mantendo o estado da tela e executando ações.

> Segue o padrão MVVM: a ViewModel expõe dados observáveis e métodos de ação.

🧱 Conteúdo

```swift
final class TaskListViewModel: ObservableObject {
    @Published var tasks: [Task] = []

    private let getTasksUseCase: GetTasksUseCase
    private let addTaskUseCase: AddTaskUseCase
    private let updateTaskUseCase: UpdateTaskUseCase
    private let deleteTaskUseCase: DeleteTaskUseCase

    init(
        getTasksUseCase: GetTasksUseCase,
        addTaskUseCase: AddTaskUseCase,
        updateTaskUseCase: UpdateTaskUseCase,
        deleteTaskUseCase: DeleteTaskUseCase
    ) {
        self.getTasksUseCase = getTasksUseCase
        self.addTaskUseCase = addTaskUseCase
```

```
        self.updateTaskUseCase = updateTaskUseCase
        self.deleteTaskUseCase = deleteTaskUseCase
        loadTasks()
    }

    func loadTasks() {
        tasks = getTasksUseCase.execute()
    }

    func addTask(title: String) {
        let newTask = Task(id: UUID(), title: title, isCompleted: false)
        addTaskUseCase.execute(task: newTask)
        loadTasks()
    }

    func toggleTask(_ task: Task) {
        var updated = task
        updated.isCompleted.toggle()
        updateTaskUseCase.execute(task: updated)
        loadTasks()
    }

    func deleteTask(_ task: Task) {
        deleteTaskUseCase.execute(task: task)
        loadTasks()
    }
}
```

🔗 Conexões

- Usa todos os UseCases da camada Domain
- Gerencia o estado de tarefas visível na UI
- Observado pela View usando `@ObservedObject`

📄 `Presentation/Views/TaskListView.swift`

📌 Função

Interface gráfica do app, implementada com SwiftUI, que exibe a lista de tarefas e permite interações.

> A View nunca fala diretamente com os UseCases ou Repositórios.

📦 Conteúdo

```
struct TaskListView: View {
    @ObservedObject var viewModel: TaskListViewModel
    @State private var newTaskTitle: String = ""

    var body: some View {
        NavigationView {
            VStack {
                HStack {
                    TextField("Nova tarefa", text: $newTaskTitle)
                        .textFieldStyle(RoundedBorderTextFieldStyle())

                    Button("Adicionar") {
                        guard !newTaskTitle.isEmpty else { return }
                        viewModel.addTask(title: newTaskTitle)
                        newTaskTitle = ""
                    }
                }
                .padding()

                List {
                    ForEach(viewModel.tasks) { task in
                        HStack {
                            Image(systemName: task.isCompleted ?
"checkmark.circle.fill" : "circle")
                                .onTapGesture {
                                    viewModel.toggleTask(task)
                                }
                            Text(task.title)
                                .strikethrough(task.isCompleted)
                        }
                    }
                    .onDelete { indexSet in
                        indexSet.map { viewModel.tasks[$0] }.forEach { task
in
                            viewModel.deleteTask(task)
                        }
                    }
                }
            }
            .navigationTitle("Minhas Tarefas")
        }
    }
}
```

🔗 Conexões

- Observa o `TaskListViewModel`
- Chama métodos da ViewModel para refletir ações do usuário
- Exibe os dados de forma reativa usando SwiftUI

✅ Resumo

Arquivo	Função	Conhece quem?
`TaskListViewModel.swift`	Orquestra a lógica da tela e chama UseCases	UseCases e entidade `Task`
`TaskListView.swift`	UI SwiftUI reativa para listar e interagir	Apenas a ViewModel (MVVM)