## Guided Learning (QLC‑3): Integrating the UI

### **Objective:**

We’ve written and tested our TaskListViewModel. Now it’s time to integrate it with a SwiftUI view using @StateObjectso users can add and remove tasks through the UI.

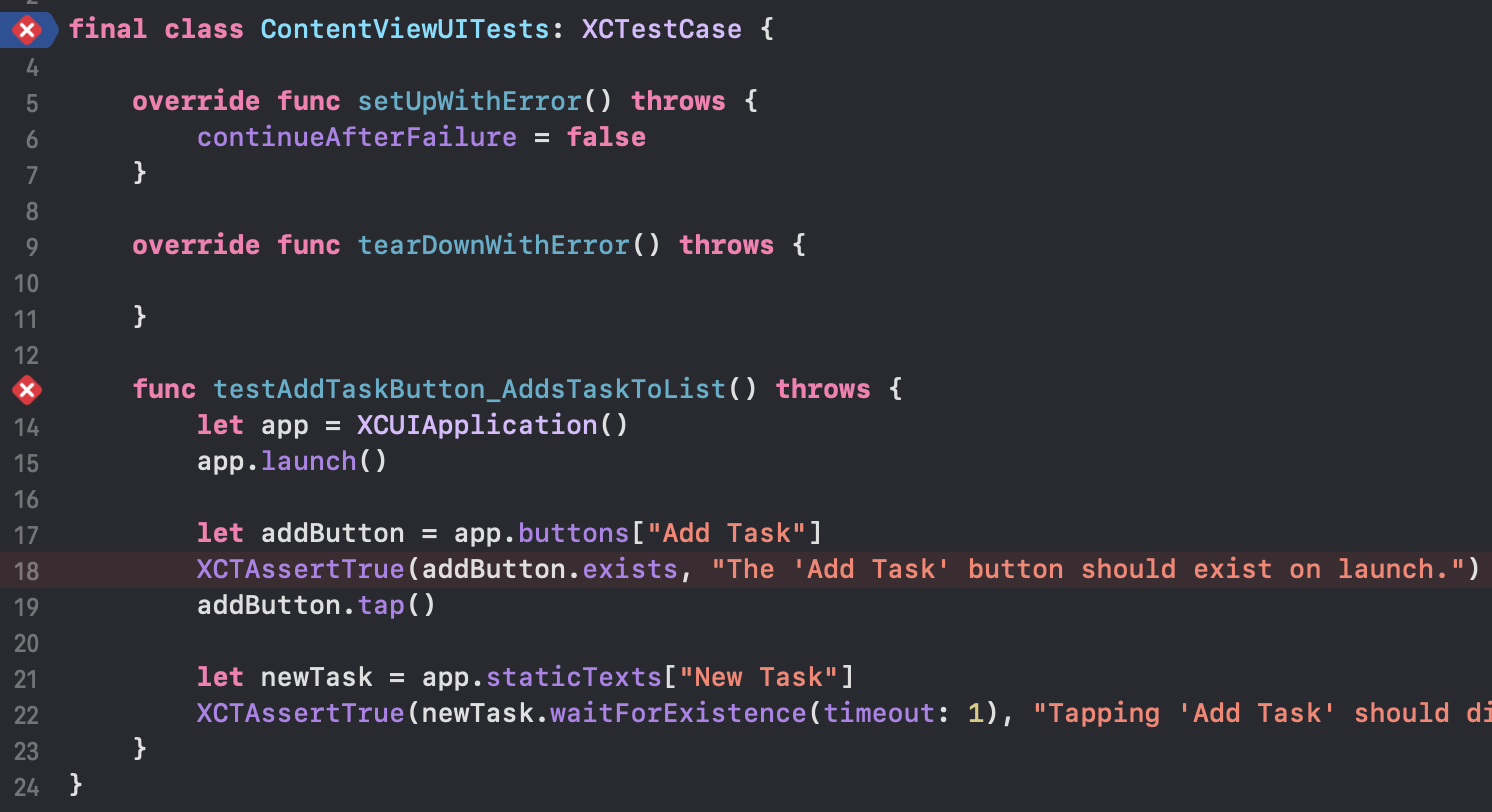
### Step 1 — Define the UI Requirement

We want a simple SwiftUI interface that:

* Displays the list of tasks
* Lets users add a new task with a button tap
* Lets users remove a task from the list

### Step 2 — Define the UI Behavior with a Failing UI Test (RED)

* Open your **UITests target**.
* Create a new test that launches the app, taps **“Add Task”**, and asserts that a task appears in the list.

Example:

**Why Use setUpWithError() and tearDownWithError()?**

* setUp runs **before each test**, great for resetting app state or configuring the environment.
* tearDown runs **after each test**, useful for cleanup or logging failures.
* continueAfterFailure = false stops execution immediately on failure, which is recommended for UI tests.

## QLC‑3.1: Adding Tasks to the UI with TDD

**Objective:**Ensure that tapping the **Add Task** button in the UI correctly adds a task to the list.

**Steps:**

1. Implement **@StateObject** for TaskListViewModel in **ContentView**.
2. Bind viewModel.tasks to a List view that displays each task’s title.
3. Add a button labeled **“Add Task”** that calls viewModel.addTask(title:) when tapped.
4. Add an .accessibilityIdentifier("Add Task") to the button so the UI test can locate it.
5. Run your UI test and confirm it **passes**.
6. Verify that all other tests still pass.

Check [https://github.com/Edrzapi/IOSTDD/tree/QLC-3.1-ANS](https://github.com/Edrzapi/IOSTDD/tree/QLC-1-ANS) for the solution

Why We Use .accessibilityIdentifier("")

In SwiftUI, **UI tests rely on accessibility identifiers to locate elements on screen.**Unlike visible labels or images (which may change), accessibility identifiers provide a stable reference point for automated tests.

## QLC‑3.2: Removing Tasks from the UI with TDD

**Objective:**

Ensure that tapping a delete button in the UI removes the task from the list.

**Steps:**

1. Write a **UI test** that verifies tapping a delete button next to a task removes it from the list.
2. Run the test and confirm it **fails**.
3. Update **ContentView** to include a delete button inside each list row that calls viewModel.removeTask(id:).
4. Run the test again and confirm it **passes**.
5. Ensure all previous **UI tests and unit tests still pass**.

Check [https://github.com/Edrzapi/IOSTDD/tree/QLC-3.2-ANS](https://github.com/Edrzapi/IOSTDD/tree/QLC-1-ANS) for the solution

## QLC‑3.3: Refactoring the UI and Project Structure

**Objective:**Now that your add and remove functionality is working and tested, it’s time to refactor your UI and project setup to improve maintainability and prepare for future features.

**Steps:**

1. **Rename ContentView to TaskListView**
   * Reflects its purpose more clearly as the main task list screen.
2. **Move TaskListView into a dedicated Views/ folder**
   * This keeps your project structure clean and organized.
3. **(Optional for later) Prepare TaskListView for task details feature**
   * You may add a placeholder NavigationLink or stubbed method for viewing/editing task details.
4. **Run all unit tests and UI tests after refactoring to confirm everything still works**

Be careful of unique identifiers, we need to implement something to keep our tests working, and identifiers like these will help us manage multiple icons of the same id.   
  
​​