To connect the model to the UI in MVVM, you can use the following steps:

1. Create a view model class. The view model class should expose the data from the model to the view in a way that is easy to understand and use. It should also handle user input and update the model accordingly.
2. Bind the view to the view model. This can be done using a data binding library, such as SwiftUI's `@ObservedObject` property wrapper.
3. In the view, update the UI based on the data in the view model.

Here is an example of how to connect the model to the UI in SwiftUI:

Swift

```swift
// Model
struct Todo {
    let id: UUID
    let title: String
    let isCompleted: Bool
}

// View Model
class TodoViewModel: ObservableObject {

    @Published private(set) var todos: [Todo] = []

    func addTodo(title: String) {
        let todo = Todo(id: UUID(), title: title, isCompleted: false)
        todos.append(todo)
    }

    func removeTodo(at index: Int) {
        todos.remove(at: index)
    }

    func toggleTodo(at index: Int) {
        todos[index].isCompleted.toggle()
    }
}

// View
struct TodoListView: View {

    @ObservedObject var viewModel = TodoViewModel()

    var body: some View {
        List {
            ForEach(viewModel.todos, id: \.id) { todo in
                Text(todo.title)
                    .strikethrough(todo.isCompleted)
```

```
                    .onTapGesture {
                        viewModel.toggleTodo(at:
viewModel.todos.firstIndex(where: { $0.id == todo.id })!)
                    }
            }
        }
    }
}
```

In this example, the `TodoViewModel` class exposes the `todos` array to the `TodoListView`. The `TodoListView` then binds to the `todos` array using the `@ObservedObject` property wrapper. This means that the UI will be automatically updated whenever the `todos` array changes.

When the user taps on a to-do item, the `TodoListView` calls the `toggleTodo()` method on the view model. The view model then updates the `isCompleted` property of the to-do item. The UI will then be automatically updated to reflect the new state of the to-do item.

This is just a simple example of how to connect the model to the UI in MVVM. The specific steps involved will vary depending on the framework or toolkit that you are using. However, the general principles remain the same.