**Implementing Parent-Child Component Communication and Lifecycle Methods**

By the end of this lab, you will be able to implement parent-to-child and child-to-parent communication in Blazor components using parameters and EventCallbacks.

**Step 1: Prepare for the application**

Scenario Overview: You will create a Blazor application where a parent component manages a list of tasks, and child components display individual tasks and allow for status updates. This step involves setting up your Visual Studio Code project and a basic Blazor app structure.

**Instructions:**

1. Create a new Blazor Server App in Visual Studio Code.
2. Name the project TaskManagerApp.
3. Open the TaskManagerApp folder in Visual Studio Code.
4. Create two Razor components:
   * ParentTaskManager.razor
   * ChildTaskDisplay.razor

**Step 2: Implement Parent-to-Child Communication**

**Scenario Overview:** in this step, the parent component passes a task name to the child component using the [Parameter] attribute.

**Instructions:**

1. Open ParentTaskManager.razor and define a list of task names in the @code block.
2. Use a foreach loop to render a ChildTaskDisplay component for each task.
3. Pass each task name as a parameter to ChildTaskDisplay.

**Step 3: Implement Child-to-Parent Communication**

**Scenario Overview:** The child component notifies the parent when a task is marked complete using an EventCallback.

**Instructions:**

1. Open ChildTaskDisplay.razor and define a @code block.
2. Declare an [Parameter] property for the task name.
3. Declare an EventCallback named OnTaskCompleted.
4. Add a button that, when clicked, invokes the OnTaskCompleted callback.

**Step 4: Connect Parent-to-Child and Child-to-Parent Communication**

**Scenario Overview:** Combine the parent and child components to enable full two-way communication.

**Instructions:**

1. In ParentTaskManager.razor, handle the OnTaskCompleted event from each child component.
2. Update the task list to reflect the completed tasks.
3. Display a message in the parent component when all tasks are completed.

**Step 5: Test and Run the Application**

**Scenario Overview:** Ensure that the application runs correctly and that parent-to-child and child-to-parent communication works as expected.

**Instructions:**

1. Run the application using dotnet run in the terminal.
2. Interact with the app by marking tasks as complete and verify the updates in the parent component.

**ParentTaskManager.razor**

<h3 class="mb-3">Task Manager (Parent)</h3>

@if (tasks.All(t => t.Completed))

{

<div class="alert alert-success mb-3">🎉 All tasks are completed!</div>

}

<ul class="list-group">

@foreach (var t in tasks)

{

<li class="list-group-item d-flex justify-content-between align-items-center">

<span class="@(t.Completed ? "text-decoration-line-through text-muted" : null)">

@t.Name

</span>

<ChildTaskDisplay

TaskName="@t.Name"

Completed="@t.Completed"

OnTaskCompleted="HandleTaskCompleted" />

</li>

}

</ul>

<p class="mt-2 text-muted">Completed: @tasks.Count(x => x.Completed) / @tasks.Count</p>

@code {

private readonly List<TaskItem> tasks =

[

new("Complete Blazor lab"),

new("Review EventCallback example"),

new("Push code to GitHub"),

];

private void HandleTaskCompleted(string taskName)

{

var item = tasks.FirstOrDefault(x => x.Name == taskName);

if (item is not null)

{

item.Completed = true;

StateHasChanged();

Console.WriteLine($"Completed: {taskName}");

}

}

private sealed record TaskItem(string Name)

{

public bool Completed { get; set; }

}

}

**ChildTaskDisplay.razor**

<p class="mb-1">Task: <strong>@TaskName</strong></p>

<button type="button"

class="btn btn-sm @(Completed ? "btn-secondary" : "btn-success")"

@onclick="MarkComplete"

disabled="@(Completed)">

@(Completed ? "✅ Completed" : "✔ Mark Complete")

</button>

@code {

[Parameter] public string TaskName { get; set; } = string.Empty;

[Parameter] public bool Completed { get; set; }

[Parameter] public EventCallback<string> OnTaskCompleted { get; set; }

private Task MarkComplete() => OnTaskCompleted.InvokeAsync(TaskName);

}