Step 2: Create the Task Model

1. **Right-click** on the `Models` folder in the **Solution Explorer** > **Add** > **Class**.
2. Name it **Task.cs**.

Task.cs:

```
csharp
Copy code
namespace TaskApp.Models
{
    public class Task
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public bool IsCompleted { get; set; }
    }
}
```

Step 3: Create a Task Service (In-Memory Data Store)

To simulate storing tasks, we will create an in-memory data store. This will allow us to manage tasks without needing a database.

1. **Right-click** on the **Services** folder (if it doesn't exist, create it) > **Add** > **Class**.
2. Name it **TaskService.cs**.

TaskService.cs:

```
csharp
Copy code
using System.Collections.Generic;
using System.Linq;
using TaskApp.Models;

namespace TaskApp.Services
{
    public class TaskService
    {
        private static List<Task> _tasks = new List<Task>();
        private static int _nextId = 1;

        public TaskService()
        {
            // Seed with some data (optional)
            if (!_tasks.Any())
            {
                _tasks.Add(new Task { Id = _nextId++, Title = "Sample
Task", Description = "This is a sample task", IsCompleted = false });
            }
        }
        // Add a task
        public Task AddTask(Task task)
        {
            task.Id = _nextId++;
            _tasks.Add(task);
            return task;
        }

        // Get all tasks with pagination
        public List<Task> GetTasks(int page = 1, int limit = 5)
        {
            return _tasks.Skip((page - 1) * limit).Take(limit).ToList();
        }

        // Get a task by Id
        public Task GetTaskById(int id)
        {
            return _tasks.FirstOrDefault(t => t.Id == id);
        }

        // Update a task
        public bool UpdateTask(int id, Task updatedTask)
        {
            var task = _tasks.FirstOrDefault(t => t.Id == id);
            if (task == null) return false;

            task.Title = updatedTask.Title;
            task.Description = updatedTask.Description;
            task.IsCompleted = updatedTask.IsCompleted;
            return true;
        }

        // Delete a task
        public bool DeleteTask(int id)
        {
            var task = _tasks.FirstOrDefault(t => t.Id == id);
            if (task == null) return false;

            _tasks.Remove(task);
            return true;
        }
    }
}
```

Step 4: Register the Service in Startup

You need to add the `TaskService` to the dependency injection container so that it can be used in controllers.

1. **Open `Startup.cs`** (or `Program.cs` in .NET 6+).
2. In the `ConfigureServices` method, add:

In `Startup.cs` (for .NET Core 3.1 or .NET 5):

```
csharp
Copy code
public void ConfigureServices(IServiceCollection services)
```

```
{
    services.AddControllersWithViews();
    services.AddSingleton<TaskService>();  // Register the TaskService
}
```

In `Program.cs` (for .NET 6+):

```
csharp
Copy code
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddSingleton<TaskService>();  // Register the TaskService

var app = builder.Build();
```

Step 5: Create the Controller

1. **Right-click on the `Controllers` folder** > **Add** > **Controller**.
2. Select **MVC Controller - Empty**.
3. Name the controller **TaskController**.

TaskController.cs:

```
csharp
Copy code
using Microsoft.AspNetCore.Mvc;
using TaskApp.Models;
using TaskApp.Services;

namespace TaskApp.Controllers
{
    public class TaskController : Controller
    {
        private readonly TaskService _taskService;

        public TaskController(TaskService taskService)
        {
            _taskService = taskService;
        }

        // GET: /task
        public IActionResult Index(int page = 1, int limit = 5)
        {
            var tasks = _taskService.GetTasks(page, limit);
            return View(tasks);
        }

        // GET: /task/details/{id}
        public IActionResult Details(int id)
        {
            var task = _taskService.GetTaskById(id);
            if (task == null)
                return NotFound();

            return View(task);
        }
```

```
// GET: /task/create
public IActionResult Create()
{
    return View();
}

// POST: /task/create
[HttpPost]
public IActionResult Create(Task task)
{
    if (ModelState.IsValid)
    {
        _taskService.AddTask(task);
        return RedirectToAction("Index");
    }
    return View(task);
}

// GET: /task/edit/{id}
public IActionResult Edit(int id)
{
    var task = _taskService.GetTaskById(id);
    if (task == null)
        return NotFound();

    return View(task);
}

// POST: /task/edit/{id}
[HttpPost]
public IActionResult Edit(int id, Task task)
{
    if (ModelState.IsValid)
    {
        bool updated = _taskService.UpdateTask(id, task);
        if (!updated) return NotFound();

        return RedirectToAction("Index");
    }
    return View(task);
}

// GET: /task/delete/{id}
public IActionResult Delete(int id)
{
    var task = _taskService.GetTaskById(id);
    if (task == null)
        return NotFound();

    return View(task);
}

// POST: /task/delete/{id}
[HttpPost, ActionName("Delete")]
public IActionResult DeleteConfirmed(int id)
{
    bool deleted = _taskService.DeleteTask(id);
    if (!deleted) return NotFound();

    return RedirectToAction("Index");
```

```
        }
    }
}
```

Step 6: Create the Views

1. Index View (List all tasks with pagination)

- Right-click the `Views/Task` folder > **Add** > **View**.
- Name it **Index.cshtml**.

Index.cshtml:

```
html
Copy code
@model List<TaskApp.Models.Task>

@{
    ViewBag.Title = "Tasks";
}

<h2>Tasks</h2>
<table class="table">
    <thead>
        <tr>
            <th>Title</th>
            <th>Description</th>
            <th>Status</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var task in Model)
        {
            <tr>
                <td>@task.Title</td>
                <td>@task.Description</td>
                <td>@(task.IsCompleted ? "Completed" : "Pending")</td>
                <td>
                    @Html.ActionLink("Edit", "Edit", new { id = task.Id })
|
                    @Html.ActionLink("Delete", "Delete", new { id = task.Id
})
                </td>
            </tr>
        }
    </tbody>
</table>

<div>
    @Html.ActionLink("Create New Task", "Create")
</div>
```

2. Create View (Form to create a new task)

- Right-click the `Views/Task` folder > **Add** > **View**.
- Name it **Create.cshtml**.

Create.cshtml:

```
html
Copy code
@model TaskApp.Models.Task

@{
    ViewBag.Title = "Create Task";
}

<h2>Create Task</h2>

@using (Html.BeginForm())
{
    <div class="form-group">
        @Html.LabelFor(m => m.Title)
        @Html.TextBoxFor(m => m.Title, new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Description)
        @Html.TextBoxFor(m => m.Description, new { @class = "form-control"
})
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.IsCompleted)
        @Html.CheckBoxFor(m => m.IsCompleted)
    </div>
    <button type="submit" class="btn btn-primary">Create</button>
}
```

3. Edit View (Form to edit a task)

- Right-click the `Views/Task` folder > **Add** > **View**.
- Name it **Edit.cshtml**.

Edit.cshtml:

```
html
Copy code
@model TaskApp.Models.Task

@{
    ViewBag.Title = "Edit Task";
}

<h2>Edit Task</h2>

@using (Html.BeginForm())
{
    <div class="form-group">
        @Html.LabelFor(m => m.Title)
        @Html.TextBoxFor(m => m.Title, new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Description)
        @Html.TextBoxFor(m => m.Description, new { @class = "form-control"
})
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.IsCompleted)
        @Html.CheckBoxFor(m => m.IsCompleted)
    </div>
    <button type="submit" class="btn btn-primary">Save Changes</button>
}
```

4. Delete View (Confirm delete task)

- Right-click the `Views/Task` folder > **Add** > **View**.
- Name it **Delete.cshtml**.

Delete.cshtml:

```
html
Copy code
@model TaskApp.Models.Task

@{
    ViewBag.Title = "Delete Task";
}

<h2>Delete Task</h2>

<div>
    <h3>Are you sure you want to delete this task?</h3>
    <p>@Model.Title</p>
    <form method="post">
        <button type="submit" class="btn btn-danger">Delete</button>
        @Html.ActionLink("Cancel", "Index", null, new { @class = "btn btn-
secondary" })
    </form>
</div>
```

Create ASP.Net MVC Web application for Festivals in India with Master Page and minimum 4 Pages

Step 2: Create the Festival Model

1. **Right-click on the `Models` folder** > **Add** > **Class**.
2. Name the class `Festival.cs`.

Festival.cs:

```
csharp
Copy code
namespace FestivalApp.Models
{
    public class Festival
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Date { get; set; }
        public string Description { get; set; }
    }
}
```

Step 3: Create a Service to Manage Festivals

We'll create an in-memory service to manage festival data for the sake of this example. You can later extend it to use a database if necessary.

1. **Right-click on the `Services` folder** (create the folder if it doesn't exist) > **Add** > **Class**.
2. Name the class `FestivalService.cs`.

FestivalService.cs:

```
csharp
Copy code
using System.Collections.Generic;
using FestivalApp.Models;
using System.Linq;

namespace FestivalApp.Services
{
    public class FestivalService
    {
        private static List<Festival> _festivals = new List<Festival>();
        private static int _nextId = 1;

        public FestivalService()
        {
            // Seed with some data (optional)
            if (!_festivals.Any())
            {
                _festivals.Add(new Festival { Id = _nextId++, Name =
"Diwali", Date = "November", Description = "Festival of lights." });
                _festivals.Add(new Festival { Id = _nextId++, Name =
"Holi", Date = "March", Description = "Festival of colors." });
            }
        }

        // Get all festivals
        public List<Festival> GetFestivals()
        {
            return _festivals;
        }

        // Get a festival by ID
        public Festival GetFestivalById(int id)
        {
            return _festivals.FirstOrDefault(f => f.Id == id);
        }

        // Add a new festival
        public Festival AddFestival(Festival festival)
        {
            festival.Id = _nextId++;
            _festivals.Add(festival);
            return festival;
        }

        // Update a festival
        public bool UpdateFestival(int id, Festival festival)
        {
            var existingFestival = _festivals.FirstOrDefault(f => f.Id ==
id);
            if (existingFestival == null) return false;

            existingFestival.Name = festival.Name;
            existingFestival.Date = festival.Date;
            existingFestival.Description = festival.Description;
            return true;
        }

        // Delete a festival
        public bool DeleteFestival(int id)
        {
            var festival = _festivals.FirstOrDefault(f => f.Id == id);
            if (festival == null) return false;

            _festivals.Remove(festival);
            return true;
        }
    }
}
```

Step 4: Register the Service in `Startup.cs`

You need to register the **FestivalService** class in the DI container.

1. **Open `Startup.cs`** (or `Program.cs` for .NET 6+).
2. In the `ConfigureServices` method, add:

```
csharp
Copy code
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddSingleton<FestivalService>();  // Register the
FestivalService
}
```

Step 5: Create the Controller

1. **Right-click on the `Controllers` folder** > **Add** > **Controller**.
2. Select **MVC Controller - Empty**.
3. Name the controller **FestivalController**.

FestivalController.cs:

```
csharp
Copy code
using Microsoft.AspNetCore.Mvc;
using FestivalApp.Models;
using FestivalApp.Services;

namespace FestivalApp.Controllers
{
    public class FestivalController : Controller
    {
        private readonly FestivalService _festivalService;

        public FestivalController(FestivalService festivalService)
        {
            _festivalService = festivalService;
        }

        // GET: /Festival
        public IActionResult Index()
        {
            var festivals = _festivalService.GetFestivals();
            return View(festivals);
        }

        // GET: /Festival/Details/{id}
        public IActionResult Details(int id)
        {
            var festival = _festivalService.GetFestivalById(id);
            if (festival == null)
                return NotFound();

            return View(festival);
        }

        // GET: /Festival/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: /Festival/Create
        [HttpPost]
        public IActionResult Create(Festival festival)
        {
            if (ModelState.IsValid)
            {
                _festivalService.AddFestival(festival);
                return RedirectToAction("Index");
            }
            return View(festival);
        }

        // GET: /Festival/Edit/{id}
        public IActionResult Edit(int id)
        {
            var festival = _festivalService.GetFestivalById(id);
            if (festival == null)
                return NotFound();

            return View(festival);
        }

        // POST: /Festival/Edit/{id}
        [HttpPost]
        public IActionResult Edit(int id, Festival festival)
        {
            if (ModelState.IsValid)
            {
                var updated = _festivalService.UpdateFestival(id,
festival);
                if (!updated) return NotFound();
                return RedirectToAction("Index");
            }
            return View(festival);
        }

        // GET: /Festival/Delete/{id}
        public IActionResult Delete(int id)
        {
            var festival = _festivalService.GetFestivalById(id);
            if (festival == null)
                return NotFound();

            return View(festival);
        }

        // POST: /Festival/Delete/{id}
        [HttpPost, ActionName("Delete")]
        public IActionResult DeleteConfirmed(int id)
        {
            bool deleted = _festivalService.DeleteFestival(id);
            if (!deleted) return NotFound();

            return RedirectToAction("Index");
        }
    }
}
```

Step 6: Create the Views

You will need to create views for the **Index**, **Details**, **Create**, **Edit**, and **Delete** actions.

1. Index View (List of festivals)

- **Right-click on `Views/Festival` > Add > View**.
- Name it **Index.cshtml**.

Index.cshtml:

```
html
Copy code
@model IEnumerable<FestivalApp.Models.Festival>

@{
    ViewBag.Title = "Festivals";
}

<h2>Festivals</h2>

<table class="table">
    <thead>
        <tr>
            <th>Name</th>
            <th>Date</th>
            <th>Description</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var festival in Model)
        {
            <tr>
                <td>@festival.Name</td>
                <td>@festival.Date</td>
                <td>@festival.Description</td>
                <td>
                    @Html.ActionLink("Details", "Details", new { id =
festival.Id }) |
                    @Html.ActionLink("Edit", "Edit", new { id = festival.Id
}) |
                    @Html.ActionLink("Delete", "Delete", new { id =
festival.Id })
                </td>
            </tr>
        }
    </tbody>
</table>

<div>
    @Html.ActionLink("Create New Festival", "Create")
</div>
```

2. Details View (Festival details)

- **Right-click on `Views/Festival`** > **Add** > **View**.
- Name it **Details.cshtml**.

Details.cshtml:

```
html
Copy code
@model FestivalApp.Models.Festival

@{
    ViewBag.Title = "Festival Details";
}

<h2>Festival Details</h2>

<div>
    <h4>@Model.Name</h4>
    <p><strong>Date:</strong> @Model.Date</p>
    <p><strong>Description:</strong> @Model.Description</p>
</div>

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

3. Create View (Form to create a festival)

- **Right-click on `Views/Festival`** > **Add** > **View**.
- Name it **Create.cshtml**.

Create.cshtml:

```
html
Copy code
```

```
@model FestivalApp.Models.Festival

@{
    ViewBag.Title = "Create Festival";
}

<h2>Create Festival</h2>

@using (Html.BeginForm())
{
    <div class="form-group">
        @Html.LabelFor(m => m.Name)
        @Html.TextBoxFor(m => m.Name, new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Date)
        @Html.TextBoxFor(m => m.Date, new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Description)
        @Html.TextAreaFor(m => m.Description, new { @class = "form-control"
})
    </div>
    <button type="submit" class="btn btn-primary">Create</button>
}
```

4. Edit View (Edit festival details)

- **Right-click on `Views/Festival` > Add > View.**
- Name it **Edit.cshtml**.

Edit.cshtml:

```
html
Copy code
@model FestivalApp.Models.Festival

@{
    ViewBag.Title = "Edit Festival";
}

<h2>Edit Festival</h2>

@using (Html.BeginForm())
{
    <div class="form-group">
        @Html.LabelFor(m => m.Name)
        @Html.TextBoxFor(m => m.Name, new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Date)
        @Html.TextBoxFor(m => m.Date, new { @class = "form-control" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Description)
        @Html.TextAreaFor(m => m.Description, new { @class = "form-control"
})
    </div>
    <button type="submit" class="btn btn-primary">Save Changes</button>
}
```

5. Delete View (Confirm deletion)

- **Right-click on `Views/Festival`** > **Add** > **View**.
- Name it **Delete.cshtml**.

Delete.cshtml:

```
html
Copy code
@model FestivalApp.Models.Festival

@{
    ViewBag.Title = "Delete Festival";
}

<h2>Delete Festival</h2>

<div>
    <h3>Are you sure you want to delete this festival?</h3>
    <p>@Model.Name</p>
    <form method="post">
        <button type="submit" class="btn btn-danger">Delete</button>
        @Html.ActionLink("Cancel", "Index", null, new { @class = "btn btn-
secondary" })
    </form>
</div>
```