Create ASP.Net MVC Web application for IPL with Master Page and minimum 4 Pages

2. Create the Master Page (Layout)

1. **Add a Layout (Master) Page**:
   - In the **Views** folder, under **Shared**, right-click and select **Add > New Item**.
   - Choose **MVC Layout Page** and name it `Layout.cshtml`.
2. **Edit `Layout.cshtml`**:

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>@ViewData["Title"] - College Application</title>
    <link href="~/Content/site.css" rel="stylesheet" />
</head>
<body>
    <header>
        <nav>
            <ul>
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
                <li>@Html.ActionLink("Departments", "Index",
"Departments")</li>
                <li>@Html.ActionLink("Contact", "Contact",
"Home")</li>
            </ul>
        </nav>
    </header>
    <main>
        @RenderBody()
    </main>
    <footer>
        <p>© 2024 College Application</p>
    </footer>
</body>
</html>
```

---

3. Create Controllers and Views

1. **HomeController**:

```
csharp
Copy code
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";
```

```
        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";
        return View();
    }
}
```

2. **DepartmentsController**:

```
csharp
Copy code
public class DepartmentsController : Controller
{
    public ActionResult Index()
    {
        var departments = new List<string> { "Computer Science",
"Mathematics", "Physics", "History" };
        return View(departments);
    }
}
```

---

4. Create Views

1. **Home/Index.cshtml**:

```
html
Copy code
@{
    ViewData["Title"] = "Home";
}
<h2>Welcome to the College Application</h2>
<p>Explore our departments and learn more about us.</p>
```

2. **Home/About.cshtml**:

```
html
Copy code
@{
    ViewData["Title"] = "About";
}
<h2>About Us</h2>
<p>@ViewBag.Message</p>
```

3. **Home/Contact.cshtml**:

```
html
Copy code
@{
    ViewData["Title"] = "Contact";
}
<h2>Contact Us</h2>
<p>@ViewBag.Message</p>
```

4. **Departments/Index.cshtml**:

```
html
Copy code
@{
    ViewData["Title"] = "Departments";
}
<h2>Our Departments</h2>
<ul>
    @foreach (var dept in Model)
    {
        <li>@dept</li>
    }
</ul>
```

---

5. Configure the Application

1. **Set Default Route**: Open `RouteConfig.cs` in the `App_Start` folder:

```
csharp
Copy code
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
    );
}
```

Create a Task model with properties like Id, Title, Description, and IsCompleted. Implement CRUD operations for adding, viewing, updating, and deleting tasks. Use an in-memory data store (like a list) to store tasks. Implement pagination for the GET endpoint to list tasks with a limit (e.g., GET /tasks?page=1&limit=5).

Folder and File Structure

Here's the folder structure we'll use:

```
markdown
Copy code
/TaskManagement
    /Controllers
    /Models
    /Services
    /Data
    /Program.cs
    /Startup.cs (if applicable)
```

3. Step-by-Step Implementation

Step 1: Create the Task Model

1. **Right-click the `TaskManagement` project** > **Add** > **New Folder**.
   - Name the folder **Models**.
2. **Inside the `Models` folder**, right-click > **Add** > **Class**.
   - Name the class `Task.cs`.
3. **Define the Task Model**:

```
csharp
Copy code
namespace TaskManagement.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 2: Create an In-Memory Data Store

1. **Right-click the `TaskManagement` project** > **Add** > **New Folder**.
   - Name the folder **Data**.
2. **Inside the `Data` folder**, right-click > **Add** > **Class**.
   - Name the class `TaskDataStore.cs`.
3. **Implement the Data Store**:

```
csharp
Copy code
using TaskManagement.Models;
using System.Collections.Generic;

namespace TaskManagement.Data
{
    public static class TaskDataStore
    {
        public static List<Task> Tasks { get; set; } = new
List<Task>();

        static TaskDataStore()
        {
            // Seed with initial data
            Tasks.Add(new Task { Id = 1, Title = "Task 1",
Description = "Description 1", IsCompleted = false });
            Tasks.Add(new Task { Id = 2, Title = "Task 2",
Description = "Description 2", IsCompleted = true });
        }
    }
}
```

Step 3: Create the Task Controller

1. **Right-click the `TaskManagement` project** > **Add** > **New Folder**.

- Name the folder **Controllers**.

2. **Inside the `Controllers` folder**, right-click > **Add** > **Controller**.
   - Choose **API Controller - Empty**.
   - Name it `TaskController`.
3. **Implement CRUD and Pagination**:

```
csharp
Copy code
using Microsoft.AspNetCore.Mvc;
using TaskManagement.Data;
using TaskManagement.Models;
using System.Linq;

namespace TaskManagement.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class TaskController : ControllerBase
    {
        // GET /api/task?page=1&limit=5
        [HttpGet]
        public IActionResult GetTasks([FromQuery] int page = 1,
[FromQuery] int limit = 5)
        {
            if (page <= 0 || limit <= 0)
            {
                return BadRequest("Page and limit must be greater
than 0.");
            }

            var tasks = TaskDataStore.Tasks
                .Skip((page - 1) * limit)
                .Take(limit)
                .ToList();

            return Ok(tasks);
        }

        // GET /api/task/{id}
        [HttpGet("{id}")]
        public IActionResult GetTask(int id)
        {
            var task = TaskDataStore.Tasks.FirstOrDefault(t => t.Id
== id);
            if (task == null) return NotFound("Task not found.");
            return Ok(task);
        }

        // POST /api/task
        [HttpPost]
        public IActionResult CreateTask([FromBody] Task newTask)
        {
            if (newTask == null) return BadRequest("Invalid task.");

            newTask.Id = TaskDataStore.Tasks.Count + 1;
            TaskDataStore.Tasks.Add(newTask);

            return CreatedAtAction(nameof(GetTask), new { id =
newTask.Id }, newTask);
        }
```

```
        // PUT /api/task/{id}
        [HttpPut("{id}")]
        public IActionResult UpdateTask(int id, [FromBody] Task
updatedTask)
        {
            var task = TaskDataStore.Tasks.FirstOrDefault(t => t.Id
== id);
            if (task == null) return NotFound("Task not found.");

            task.Title = updatedTask.Title;
            task.Description = updatedTask.Description;
            task.IsCompleted = updatedTask.IsCompleted;

            return NoContent();
        }

        // DELETE /api/task/{id}
        [HttpDelete("{id}")]
        public IActionResult DeleteTask(int id)
        {
            var task = TaskDataStore.Tasks.FirstOrDefault(t => t.Id
== id);
            if (task == null) return NotFound("Task not found.");

            TaskDataStore.Tasks.Remove(task);
            return NoContent();
        }
    }
}
```

---

4. Configure the Application

1. **Run the Application**:
   - In **Program.cs** (for .NET 6+):

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

     builder.Services.AddControllers();

     var app = builder.Build();

     app.UseHttpsRedirection();
     app.MapControllers();

     app.Run();
     ```

2. **Test with Swagger**:
   - Navigate to `/swagger` to test the API or use **Postman** to test:
     - `GET /api/task?page=1&limit=5`
     - `POST /api/task` with a JSON body:

       ```
       json
       Copy code
       {
       ```

```
    "title": "New Task",
    "description": "Task Description",
    "isCompleted": false
}
```

---

5. Additional Enhancements

- **Add Validation**: Use `DataAnnotations` to validate input.

  ```
  csharp
  Copy code
  [Required]
  public string Title { get; set; }
  ```