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).

## Step 2: Define the Folder Structure

Your folder structure will look like this:

```
bash
Copy code
/TaskManagementAPI
    /Controllers
    /Models
    /Program.cs
    /appsettings.json
```

## Step 3: Create the Task Model

1. **Right-click the `Models` folder** > **Add** > **Class**.
    - Name it `Task.cs`.

**Task.cs**:

```
csharp
Copy code
namespace TaskManagementAPI.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 4: Create the Task Controller

1. **Right-click the `Controllers` folder** > **Add** > **Controller**.
    - Choose **API Controller - Empty**.
    - Name it `TasksController`.

**TasksController.cs**:

```
csharp
Copy code
using Microsoft.AspNetCore.Mvc;
using TaskManagementAPI.Models;

namespace TaskManagementAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TasksController : ControllerBase
    {
        // In-memory data store
        private static List<Task> tasks = new List<Task>
        {
            new Task { Id = 1, Title = "Task 1", Description = "Description
for Task 1", IsCompleted = false },
            new Task { Id = 2, Title = "Task 2", Description = "Description
for Task 2", IsCompleted = true },
            new Task { Id = 3, Title = "Task 3", Description = "Description
for Task 3", IsCompleted = false },
            new Task { Id = 4, Title = "Task 4", Description = "Description
for Task 4", IsCompleted = true },
            new Task { Id = 5, Title = "Task 5", Description = "Description
for Task 5", IsCompleted = false },
            new Task { Id = 6, Title = "Task 6", Description = "Description
for Task 6", IsCompleted = true }
        };

        // GET api/tasks?page=1&limit=5
        [HttpGet]
        public ActionResult<IEnumerable<Task>> GetTasks(int page = 1, int
limit = 5)
        {
            var pagedTasks = tasks.Skip((page - 1) *
limit).Take(limit).ToList();
            return Ok(pagedTasks);
        }

        // GET api/tasks/5
        [HttpGet("{id}")]
        public ActionResult<Task> GetTask(int id)
        {
            var task = tasks.FirstOrDefault(t => t.Id == id);
            if (task == null)
            {
                return NotFound();
            }
            return Ok(task);
        }

        // POST api/tasks
        [HttpPost]
        public ActionResult<Task> CreateTask([FromBody] Task newTask)
        {
            if (newTask == null)
            {
                return BadRequest();
            }

            newTask.Id = tasks.Max(t => t.Id) + 1;  // Assign new unique ID
            tasks.Add(newTask);
            return CreatedAtAction(nameof(GetTask), new { id = newTask.Id
}, newTask);
        }

        // PUT api/tasks/5
        [HttpPut("{id}")]
        public IActionResult UpdateTask(int id, [FromBody] Task
updatedTask)
```

```
        {
            var existingTask = tasks.FirstOrDefault(t => t.Id == id);
            if (existingTask == null)
            {
                return NotFound();
            }

            existingTask.Title = updatedTask.Title;
            existingTask.Description = updatedTask.Description;
            existingTask.IsCompleted = updatedTask.IsCompleted;

            return NoContent();
        }

        // DELETE api/tasks/5
        [HttpDelete("{id}")]
        public IActionResult DeleteTask(int id)
        {
            var task = tasks.FirstOrDefault(t => t.Id == id);
            if (task == null)
            {
                return NotFound();
            }

            tasks.Remove(task);
            return NoContent();
        }
    }
}
```

## Explanation of the Controller Methods:

- **GET /tasks**: Fetches a paginated list of tasks. The `page` and `limit` query parameters allow for pagination.
- **GET /tasks/{id}**: Fetches a specific task by its ID.
- **POST /tasks**: Adds a new task to the in-memory list. The task is created with a new ID.
- **PUT /tasks/{id}**: Updates an existing task based on its ID.
- **DELETE /tasks/{id}**: Deletes the task based on its ID.

## Step 5: Configure the Program.cs

1. **Open Program.cs** and make sure the following code is present for setting up the API routes:

### Program.cs:

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

// Add services to the container.
builder.Services.AddControllers();

var app = builder.Build();

// Configure the HTTP request pipeline.
app.UseHttpsRedirection();
app.MapControllers();
app.Run();
```

## Step 6: Run the Application

1. **Build the project** by pressing **Ctrl + Shift + B**.
2. **Run the project** by pressing **F5** or **Ctrl + F5**.

Your **Task Management API** will be running, and you can use tools like **Postman** or **Swagger** (if added) to test the API endpoints.

## Step 7: Testing the API

1. **GET /tasks?page=1&limit=5**: Fetches the first 5 tasks.
2. **GET /tasks/{id}**: Fetches a task by its ID (e.g., `/tasks/1`).
3. **POST /tasks**: Adds a new task by sending a POST request with a JSON body like:

```
json
Copy code
{
    "Title": "New Task",
    "Description": "Description for new task",
    "IsCompleted": false
}
```

4. **PUT /tasks/{id}**: Updates an existing task by sending a PUT request with a JSON body like:

```
json
Copy code
{
    "Title": "Updated Task",
    "Description": "Updated task description",
    "IsCompleted": true
}
```

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

## Step 2: Set up the Folder Structure

The default folder structure for an MVC application includes:

- **Controllers**: Contains the controller classes that handle requests.
- **Models**: Contains classes for the data structure.
- **Views**: Contains the .cshtml files that render HTML.
- **Content**: For CSS, images, and static resources.
- **Scripts**: For JavaScript libraries.

Your application will look like this:

```
vbnet
Copy code
/IPLWebApp
    /Controllers
    /Models
    /Views
        /Home
        /Shared
    /Content
    /Scripts
    /App_Start
    /Global.asax
    /Web.config
```

## Step 3: Create the IPL Model

1. **Right-click on the `Models` folder** > **Add** > **Class**.
    - Name it **Player.cs** (this model represents a cricket player).

### **Player.cs**:

```
csharp
Copy code
namespace IPLWebApp.Models
{
    public class Player
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Team { get; set; }
        public string Role { get; set; }
        public int Runs { get; set; }
        public int Wickets { get; set; }
    }
}
```

2. Add a second model for the **Team**.

### **Team.cs**:

```
csharp
Copy code
namespace IPLWebApp.Models
{
    public class Team
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string City { get; set; }
        public string Captain { get; set; }
        public int TotalWins { get; set; }
    }
}
```

## Step 4: Create the IPL Controller

1. **Right-click on the `Controllers` folder** > **Add** > **Controller**.

- Choose **MVC Controller - Empty**.
- Name it **HomeController**.

**HomeController.cs**:

```
csharp
Copy code
using IPLWebApp.Models;
using System.Collections.Generic;
using System.Web.Mvc;

namespace IPLWebApp.Controllers
{
    public class HomeController : Controller
    {
        // Sample data to be displayed
        private static List<Player> players = new List<Player>
        {
            new Player { Id = 1, Name = "Virat Kohli", Team = "RCB", Role =
"Batsman", Runs = 6000, Wickets = 0 },
            new Player { Id = 2, Name = "Rohit Sharma", Team = "MI", Role =
"Batsman", Runs = 5000, Wickets = 0 },
            new Player { Id = 3, Name = "Jasprit Bumrah", Team = "MI", Role
= "Bowler", Runs = 100, Wickets = 120 }
        };

        private static List<Team> teams = new List<Team>
        {
            new Team { Id = 1, Name = "RCB", City = "Bangalore", Captain =
"Virat Kohli", TotalWins = 0 },
            new Team { Id = 2, Name = "MI", City = "Mumbai", Captain =
"Rohit Sharma", TotalWins = 5 }
        };

        // GET: Home
        public ActionResult Index()
        {
            return View(players);
        }

        // GET: Home/Teams
        public ActionResult Teams()
        {
            return View(teams);
        }

        // GET: Home/Player/{id}
        public ActionResult Player(int id)
        {
            var player = players.Find(p => p.Id == id);
            return View(player);
        }

        // GET: Home/Team/{id}
        public ActionResult Team(int id)
        {
            var team = teams.Find(t => t.Id == id);
            return View(team);
        }
    }
}
```

## Step 5: Create Views

1. **Index View**: Display a list of all players.
   - **Right-click the `Views/Home` folder > Add > View.**
   - Name it **Index.cshtml**.

### Index.cshtml:

```
html
Copy code
@model List<IPLWebApp.Models.Player>

@{
    ViewBag.Title = "Home";
}

<h2>Players</h2>
<table>
    <tr>
        <th>Name</th>
        <th>Team</th>
        <th>Role</th>
        <th>Runs</th>
        <th>Wickets</th>
        <th>Details</th>
    </tr>
    @foreach (var player in Model)
    {
        <tr>
            <td>@player.Name</td>
            <td>@player.Team</td>
            <td>@player.Role</td>
            <td>@player.Runs</td>
            <td>@player.Wickets</td>
            <td>@Html.ActionLink("Details", "Player", new { id = player.Id
})</td>
        </tr>
    }
</table>
```

2. **Teams View**: Display a list of all teams.
   - **Right-click the `Views/Home` folder > Add > View.**
   - Name it **Teams.cshtml**.

### Teams.cshtml:

```
html
Copy code
@model List<IPLWebApp.Models.Team>

@{
    ViewBag.Title = "Teams";
}

<h2>Teams</h2>
```

```
<table>
    <tr>
        <th>Name</th>
        <th>City</th>
        <th>Captain</th>
        <th>Total Wins</th>
        <th>Details</th>
    </tr>
    @foreach (var team in Model)
    {
        <tr>
            <td>@team.Name</td>
            <td>@team.City</td>
            <td>@team.Captain</td>
            <td>@team.TotalWins</td>
            <td>@Html.ActionLink("Details", "Team", new { id = team.Id
})</td>
        </tr>
    }
</table>
```

3. **Player Details View**: Display the details of a player.
    - **Right-click the `Views/Home` folder** > **Add** > **View**.
    - Name it **Player.cshtml**.

**Player.cshtml**:

```
html
Copy code
@model IPLWebApp.Models.Player

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

<h2>@Model.Name</h2>
<p>Team: @Model.Team</p>
<p>Role: @Model.Role</p>
<p>Runs: @Model.Runs</p>
<p>Wickets: @Model.Wickets</p>
```

4. **Team Details View**: Display the details of a team.
    - **Right-click the `Views/Home` folder** > **Add** > **View**.
    - Name it **Team.cshtml**.

**Team.cshtml**:

```
html
Copy code
@model IPLWebApp.Models.Team

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

<h2>@Model.Name</h2>
<p>City: @Model.City</p>
<p>Captain: @Model.Captain</p>
```

```
<p>Total Wins: @Model.TotalWins</p>
```

## Step 6: Create the Layout (Master Page)

1. **Right-click the `Views/Shared` folder > Add > View.**
   - Name it `_Layout.cshtml`.

**_Layout.cshtml**:

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>@ViewBag.Title - IPLWebApp</title>
</head>
<body>
    <header>
        <nav>
            <ul>
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("Teams", "Teams", "Home")</li>
            </ul>
        </nav>
    </header>

    <div>
        @RenderBody()
    </div>
</body>
</html>
```

## Step 7: Configure the `Program.cs` and `RouteConfig`

Ensure you have proper routing setup. For MVC, the default configuration in `Program.cs` is enough:

**Program.cs**:

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

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```