ASP.NET Practical Documentation

Practical 5: Implementing Security in ASP.NET Application

Objective:
To implement user authentication and authorization in an ASP.NET application.

Steps:

1. Open Visual Studio 2019 → Create a new project → Select ASP.NET Web Application (.NET Framework).

2. Choose MVC template and check the option Authentication → Individual User Accounts.

3. Visual Studio scaffolds the default login and register pages.

4. Run the application → Register a new user. The details are stored in the local database (AspNetUsers table).

5. Add a new Controller DashboardController with an Index view.

6. In DashboardController, apply [Authorize] attribute:

```
[Authorize]
public class DashboardController : Controller {
    public ActionResult Index() {
        ViewBag.Message = "Welcome " + User.Identity.Name;
        return View();
    }
}
Views/Dashboard/Index.cshtml
@{
    ViewBag.Title = "Dashboard";
}
<h2>@ViewBag.Message</h2>
<p>This is a protected page.</p>
open Views/Shared/_Layout.cshtml and add inside the <ul class="nav navbar-nav">:
<li>@Html.ActionLink("Dashboard", "Index", "Dashboard")</li>
```

7. **Change logout redirect to Dashboard or Login**
Open AccountController.cs → LogOff() action.
By default, it looks like this:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LogOff()
{
   AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);
   return RedirectToAction("Index", "Home");
}
```

Change it to redirect to Login directly:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LogOff()
{
   AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);
   return RedirectToAction("Login", "Account");
}
```

7. Run the project:

If logged in → Dashboard page opens.

If not logged in → Redirected to Login.

Practical 8: Creating RESTful Services in ASP.NET Web API

Objective:
To create RESTful services using ASP.NET Web API.

Steps:

1. Open Visual Studio 2019 → Create a new project → Select ASP.NET Web Application (.NET Framework).

2. Choose Web API template.

3. Create a model class Product.cs:

```
public class Product {
    public int ProductId { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public int Quantity { get; set; }
}
```

4. Create a ProductsController.cs under Controllers:

```
public class ProductsController : ApiController {
    static List<Product> products = new List<Product>() {
        new Product { ProductId=1, Name="Laptop", Price=50000, Quantity=5 },
        new Product { ProductId=2, Name="Mouse", Price=500, Quantity=10 }
    };

    public IEnumerable<Product> Get() {
        return products;
    }

    public Product Get(int id) {
        return products.FirstOrDefault(p => p.ProductId == id);
    }

    public void Post(Product product) {
        products.Add(product);
    }

    public void Put(int id, Product product) {
        var existing = products.FirstOrDefault(p => p.ProductId == id);
        if(existing != null) {
            existing.Name = product.Name;
            existing.Price = product.Price;
            existing.Quantity = product.Quantity;
        }
    }

    public void Delete(int id) {
        var product = products.FirstOrDefault(p => p.ProductId == id);
        if(product != null) products.Remove(product);
    }
}
```

5. Run the application → Test APIs using Postman.

Responses are in JSON/XML by default.

Practical 10: Creating Applications using Razor Views and Helpers

Objective:
To create an ASP.NET MVC application using Razor views and HTML Helpers.

Steps:

1. Open Visual Studio 2019 → Create a new project → Select ASP.NET Web Application (.NET Framework) → MVC template.

2. Create a model class Student.cs:

```
public class Student {
    public int Id { get; set; }
    public string Name { get; set; }
    public string Course { get; set; }
    public string Email { get; set; }
}
```

3. Add a StudentController.cs:

```
public class StudentController : Controller {
    static List<Student> students = new List<Student>();

    public ActionResult Index() {
        return View(students);
    }
    public ActionResult Create() {
        return View();
    }

    [HttpPost]
    public ActionResult Create(Student s) {
        students.Add(s);
        return RedirectToAction("Index");
    }
}
```

4. Add Razor Views:

Create.cshtml (Form using HTML Helpers):

```
@model YourNamespace.Models.Student
@using (Html.BeginForm()) {
    <div>
        @Html.LabelFor(m => m.Name)
        @Html.TextBoxFor(m => m.Name)
    </div>
    <div>
        @Html.LabelFor(m => m.Course)
        @Html.TextBoxFor(m => m.Course)
    </div>
    <div>
        @Html.LabelFor(m => m.Email)
        @Html.TextBoxFor(m => m.Email)
    </div>
    <input type="submit" value="Save" />
}
```

Index.cshtml (Table of students):

```
@model IEnumerable<yournamespace.Models.Student>

<table>
    <tr><th>Name</th><th>Course</th><th>Email</th></tr>
    @foreach (var s in Model)
    {
        <tr>
            <td>@s.Name</td>
            <td>@s.Course</td>
            <td>@s.Email</td>
        </tr>
    }
</table>
<p>@Html.ActionLink("Add Student", "Create")</p>
```

5. Run → Add students via form → Displayed in table.