ASP.NET Core Role-Based Access

Prompt for ASP.NET Core Web Application with User Authentication and Role-Based Access
Objective: Write the code for an ASP.NET Core web application that implements a login system with role-based access control.
Requirements:
Home Page:
The home page should contain:
A textbox for the username.
A textbox for the password.
A login button.
Login Functionality:
When the user enters their credentials and clicks the login button:
Control should transfer to a C# method on the server side.
The method should check if the username and password are both "admin" (for admin privileges) or "guest" (for guest privileges).
If the credentials are "admin/admin", return "admin" to the client.
If the credentials are "guest/guest", return "guest" to the client.
If the credentials are incorrect, return an error message.
Dashboard Page:
Upon successful login, redirect the user to a dashboard page.
The dashboard should include:
A textbox for input.
Three buttons: Add, Edit, and Delete.
Button functionality based on user privileges:
If the user is an admin, all buttons (Add, Edit, Delete) should be enabled.
If the user is a guest, only the Add button should be enabled; Edit and Delete buttons should be disabled.

To implement an ASP.NET Core Web Application with user authentication and role-based access, follow the steps below. The application will check for predefined credentials (admin/guest) and adjust the dashboard functionality based on the user role.

Step 1: Create the ASP.NET Core Web Application

Start by creating a new ASP.NET Core Web Application project using Visual Studio or the .NET CLI.

Step 2: Set Up the Controllers

1. **HomeController**: This controller will handle the login logic.
2. **DashboardController**: This controller will manage the dashboard functionality, based on user roles.

Step 3: Home Page (Login Page)

In the `HomeController`, create an action to handle the login logic and return the appropriate response based on the credentials entered.

HomeController.cs:

```
csharp


  using Microsoft.AspNetCore.Mvc;

  namespace RoleBasedAccess.Controllers
  {
      public class HomeController : Controller
      {
          [HttpGet]
          public IActionResult Index()
          {
              return View();
          }

          [HttpPost]
          public IActionResult Login(string username, string password)
          {
              // Check if the username and password match the predefined values
              if (username == "admin" && password == "admin")
              {
                  // Set the user role to 'admin'
                  TempData["Role"] = "admin";
                  return RedirectToAction("Dashboard", "Dashboard");
              }
              else if (username == "guest" && password == "guest")
              {
                  // Set the user role to 'guest'
                  TempData["Role"] = "guest";
                  return RedirectToAction("Dashboard", "Dashboard");
              }
              else
              {
                  // If credentials are incorrect, show an error message
                  ViewBag.ErrorMessage = "Invalid credentials!";
                  return View("Index");
              }
          }
      }
  }
```

Step 4: Dashboard Page

The dashboard page will check the user's role (stored in `TempData`) and enable or disable buttons accordingly.

DashboardController.cs:

```
csharp


  using Microsoft.AspNetCore.Mvc;

  namespace RoleBasedAccess.Controllers
  {
      public class DashboardController : Controller
      {
          public IActionResult Dashboard()
          {
              // Get the role of the current user (admin or guest)
              string role = TempData["Role"]?.ToString();
```

```
            if (role == null)
            {
                return RedirectToAction("Index", "Home");
            }

            ViewBag.Role = role;
            return View();
        }
    }
}
```

Step 5: Create Views for Login and Dashboard

Views/Home/Index.cshtml (Login Page):

```html
@{
    ViewData["Title"] = "Login";
}

<h2>Login</h2>

<form method="post" action="/Home/Login">
    <div>
        <label for="username">Username</label>
        <input type="text" id="username" name="username" required />
    </div>
    <div>
        <label for="password">Password</label>
        <input type="password" id="password" name="password" required />
    </div>
    <button type="submit">Login</button>
</form>

@if (ViewBag.ErrorMessage != null)
{
    <div style="color: red;">@ViewBag.ErrorMessage</div>
}
```

Views/Dashboard/Dashboard.cshtml (Dashboard Page):

```html
@{
    ViewData["Title"] = "Dashboard";
    string role = ViewBag.Role?.ToString();
}

<h2>Dashboard</h2>

<div>
    <label for="inputText">Input</label>
    <input type="text" id="inputText" />
</div>

<div>
    <button id="btnAdd" @disabled="@(role == "guest" ? "disabled" : "")">Add</button>
    <button id="btnEdit" @disabled="@(role == "guest" ? "disabled" : "")">Edit</button>
    <button id="btnDelete" @disabled="@(role == "guest" ? "disabled" : "")">Delete</button>
</div>

<p>Logged in as: @role</p>
```

Step 6: Configure the Startup (Program.cs)

Make sure to set up routing and add the necessary services in the `Program.cs` file.

```csharp
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace RoleBasedAccess
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Startup.cs (if needed):

```csharp
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace RoleBasedAccess
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                app.UseHsts();
            }

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

            app.UseRouting();

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

Explanation:

1. **Login Page** (`Index.cshtml`):
    - A form allows the user to enter their username and password.
    - Upon submission, it sends a `POST` request to the `Login` method in the `HomeController`, which checks the credentials.
    - If valid credentials are provided, the user is redirected to the `Dashboard` page; otherwise, an error message is displayed.
2. **Dashboard Page** (`Dashboard.cshtml`):
    - Displays a simple form with a textbox and three buttons (`Add`, `Edit`, and `Delete`).
    - The visibility and interactivity of the buttons are determined by the user's role (`admin` or `guest`). If the role is `guest`, only the `Add` button is enabled; the `Edit` and `Delete` buttons are disabled.
    - The `TempData["Role"]` value is used to store and retrieve the user's role during the session.

Step 7: Run the Application

1. Run the application and navigate to the login page.
2. Try logging in with the credentials:
    - `admin/admin` to get full access (all buttons enabled).
    - `guest/guest` to get limited access (only the Add button enabled).
3. The system will redirect you to the dashboard page based on the role after successful login.

This approach uses simple in-memory role-based logic for the demonstration. In production, you would typically use a more secure authentication mechanism, such as ASP.NET Core Identity, JWT, or OAuth, to manage users and roles.