Secure Software Development Diploma in CSF AY 2022/23 - Semester 3	Week 7
Razor Pages Security 4 - Audit Trail , Session and Error Management	

This practical will explore the following:

- **Part 1 - Creating an Audit Trail Log**
- **Part 2 - Session Management – Protecting Application Identity Cookie**
- **Part 3 - Error Management – Custom Error Display and Handling**
- **Part 4 - Publishing Razor Application to Azure Cloud Portal (Optional)**

Part 1 – Creating an Audit Trail Log

Wikipedia: "An audit trail (also called audit log) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event".

Audit logging is a critical step for adding security to your applications. Often times, audit logs are used to trace an attacker's steps, provide evidence in legal proceedings, and used to detect and prevent attacks as they are occurring. Some regulatory compliance laws, such as HIPAA, also require security-specific audit logs to be kept.

Events to be audited and logged

The first step is deciding which events require logging. Here are some general actions to consider logging in your applications:

- Administrative actions, such as adding, editing, and deleting user accounts / permissions.
- Authentication attempts
- Authorized and unauthorized access attempts
- Reading or writing to sensitive information, such as encryption keys, authentication tokens, API keys, etc.
- Validation failures that could indicate an attack against the application

Details to Log:

Each audit log entry must provide enough information to identify the user and their action in the application. The following details provide some details to consider:

- Timestamp identifying when was an operation/action performed?
- Who performed the action? This could be achieved with unique user identifier, session token hash value, and source IP address.
- What operation/action was performed? Include the URL, action / method, and action result.
- Which record(s) were impacted?
- What values were changed?

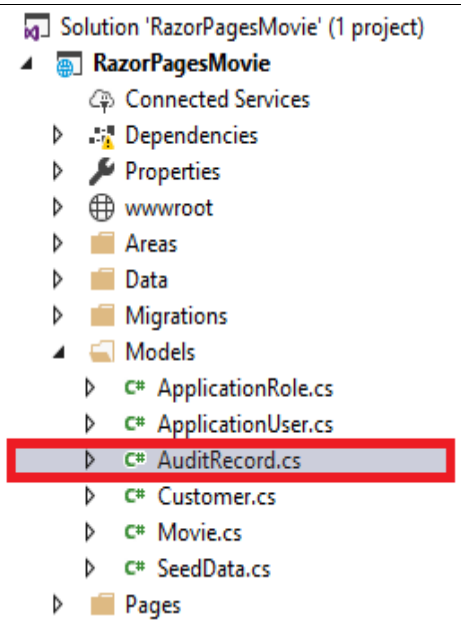
For this part of the practical, we will be creating a simple audit log to track the following events:

- **When a movie record is added**
- **When a movie record is deleted**
- **When a failed login attempt has occurred**

This is a continuation from the earlier Razor Pages Security 3 lab.

1. Create a **AuditRecord class** in **Models** folder in the existing Movie Razor application. Add the following code.

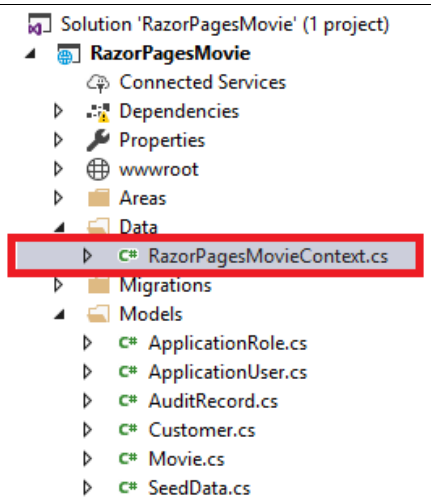
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace RazorPagesMovie.Models
{
    public class AuditRecord
    {
        [Key]
        public int Audit_ID { get; set; }

        [Display(Name = "Audit Action")]
        public string AuditActionType { get; set; }
        // Could be  Login Success /Failure/ Logout, Create, Delete, View, Update

        [Display(Name = "Performed By")]
        public string Username { get; set; }
        //Logged in user performing the action

        [Display(Name = "Date/Time Stamp")]
        [DataType(DataType.DateTime)]
        public DateTime DateTimeStamp { get; set; }
        //Time when the event occurred

        [Display(Name = "Movie Record ID ")]
        public int KeyMovieFieldID { get; set; }
        //Store the ID of movie record that is affected
    }

}
```



- The above are some audit items to be logged. In addition, other useful items may include Client IP address, field that is being edited and objects before change and after change.

2. Modify the existing **RazorPagesMovieContext.cs** and add in the AuditRecords as shown (as highlighted).

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace RazorPagesMovie.Models
{
    public class RazorPagesMovieContext : IdentityDbContext<ApplicationUser, ApplicationRole,
string>
    {
        public RazorPagesMovieContext(DbContextOptions<RazorPagesMovieContext> options)
            : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
            // Customize the ASP.NET Identity model and override the defaults if needed.
            // For example, you can rename the ASP.NET Identity table names and more.
            // Add your customizations after calling base.OnModelCreating(builder);
        }


        public DbSet<RazorPagesMovie.Models.Movie> Movie { get; set; }
        public DbSet<RazorPagesMovie.Models.Customer> Customers { get; set; }
        public DbSet<RazorPagesMovie.Models.AuditRecord> AuditRecords { get; set; }
    }
}
```



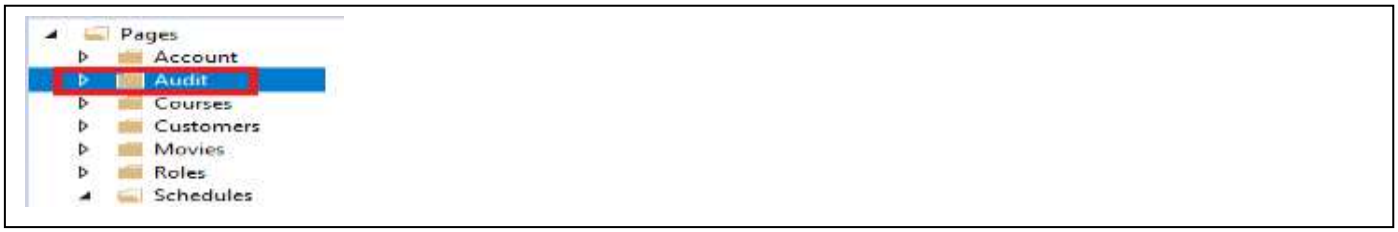
3. Create AuditRecords table in database with the following commands in **Package Manager Console**:

```
PM> Add-Migration AddAudit
PM> Update-database
```

This will create the audit table in the existing database. [If there is some errors, you may want to delete the existing database together with the Migration folder in the project and run the two commands again to rebuild the database.]

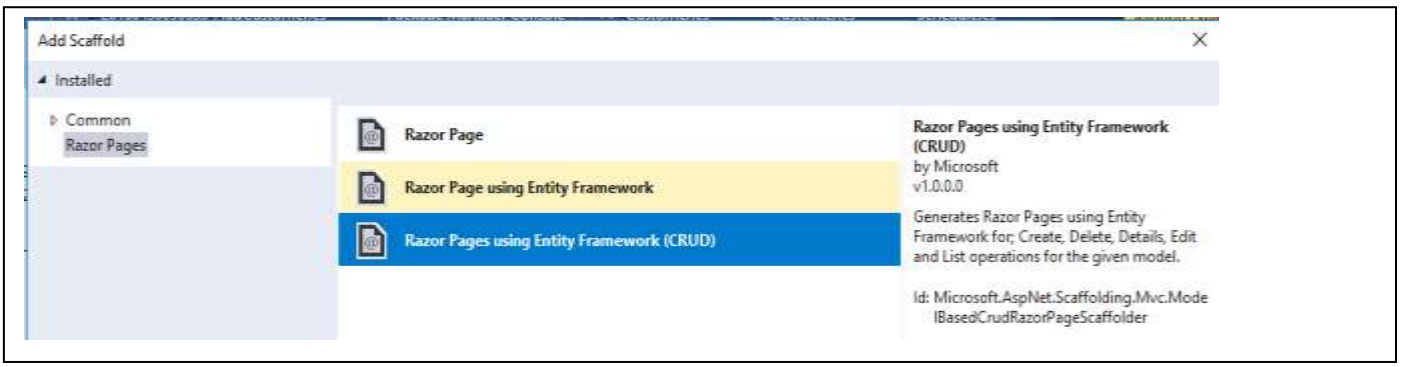
Verify using **SQL Server Object Explorer** that AuditRecords table is created (as shown below).

4. Create a folder called **Audit** in **Pages** Folder. Right-click Pages folder=>**Add**=> **New Folder**.

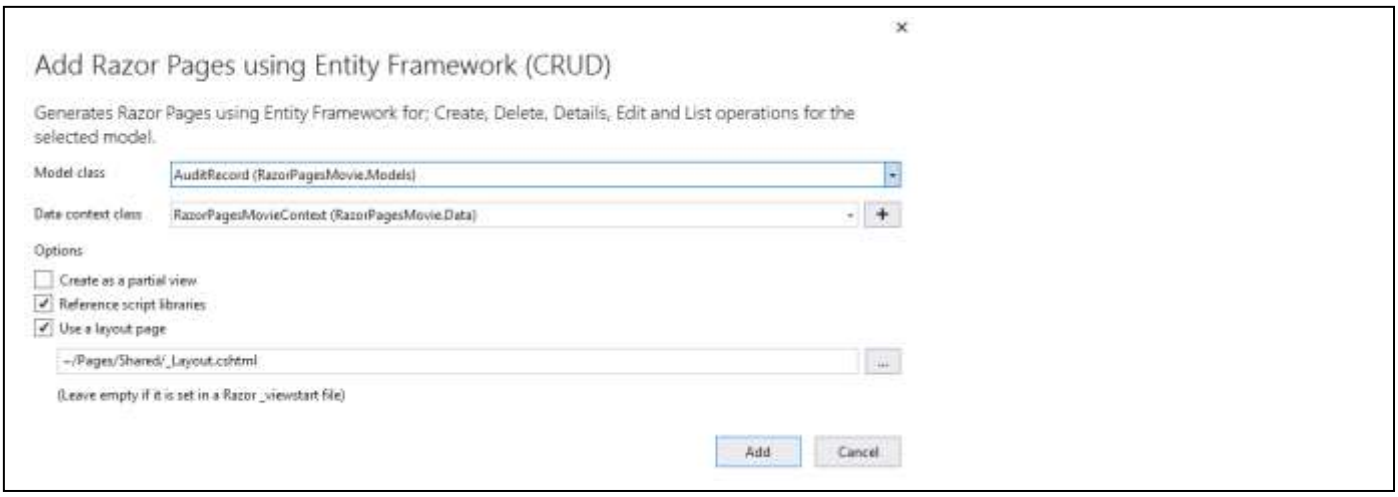


5. Next use scaffolding tool to create CRUD (Create, Edit , Update, Delete and Index) pages for AuditRecord model.

   Right click on **Audit** folder, select **Add** => **Razor Page** => Select **Razor Pages using Entity Framework (CRUD)**.

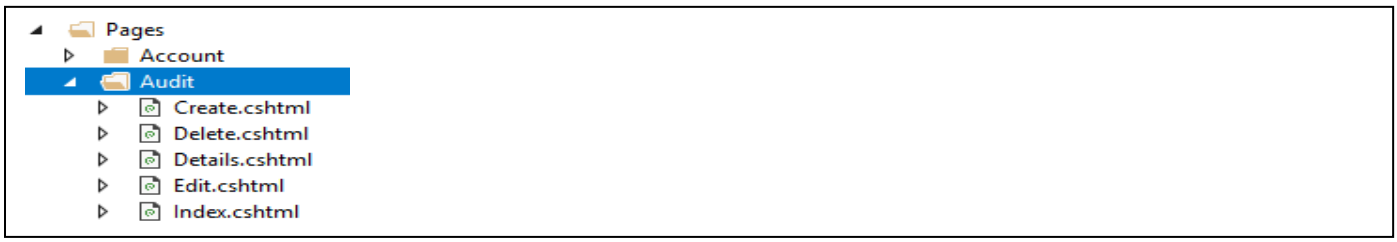


6. For model class: select **AuditRecord**. For Data context class, choose **RazorPagesMovieContext.**



Click **Add**.

7. Check that ***Create.cshtml, Delete.cshtml, Details.cshtml, Edit.cshtml & Index.cshtml*** are created in Audit folder.



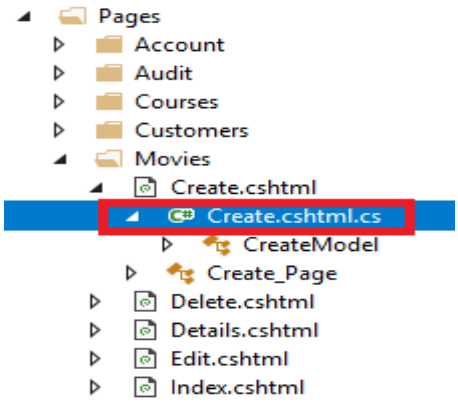
8. Run the application - **Ctrl F5** (http://localhost:XXXXX/Audit/Create) to create a new audit record. Check that it is running correctly.

Next we will create an audit record **when a movie record is added.**

9. Modify the existing **Create.cshtml.cs** file in **Pages/Movies** folder.

   Add/Edit in the following highlighted code. **Comment out** the authorization attribute using //.

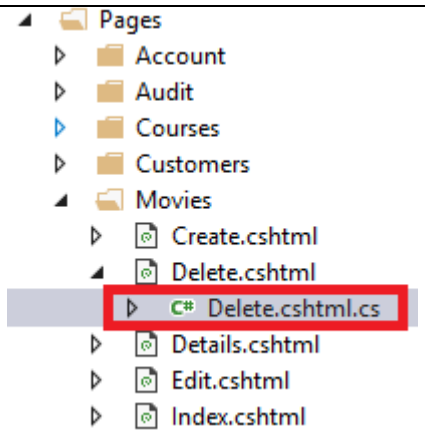
```
using System;
using System.Threading.Tasks;
using System.Web;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Movies
{
    //[Authorize(Roles = "Admin")]
    public class CreateModel : PageModel
    {
        private readonly RazorPagesMovie.Models.MovieContext _context;

        [BindProperty]
        public Movie Movie { get; set; }

        public CreateModel(RazorPagesMovie.Models.MovieContext context)
        {
            _context = context;
        }

        public IActionResult OnGet()
        {
          return Page();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

              _context.Movie.Add(Movie);
             //await _context.SaveChangesAsync();

            // Once a record is added, create an audit record
            if (await _context.SaveChangesAsync()>0)
            {
              // Create an auditrecord object
                var auditrecord = new AuditRecord();
                auditrecord.AuditActionType = "Add Movie Record";
                auditrecord.DateTimeStamp = DateTime.Now;
                auditrecord.KeyMovieFieldID = Movie.ID;
                // Get current logged-in user
                var userID = User.Identity.Name.ToString();
                auditrecord.Username = userID;

                _context.AuditRecords.Add(auditrecord);
                await _context.SaveChangesAsync();
            }
            return RedirectToPage("./Index");
        }
    }
}
```



Next we will create an audit record **when a movie record is deleted.**

10. Modify the existing **Delete.cshtml.cs** file in **Pages/Movies** folder. Add/Edit in the following highlighted code. **Comment out** the authorization attribute using //.

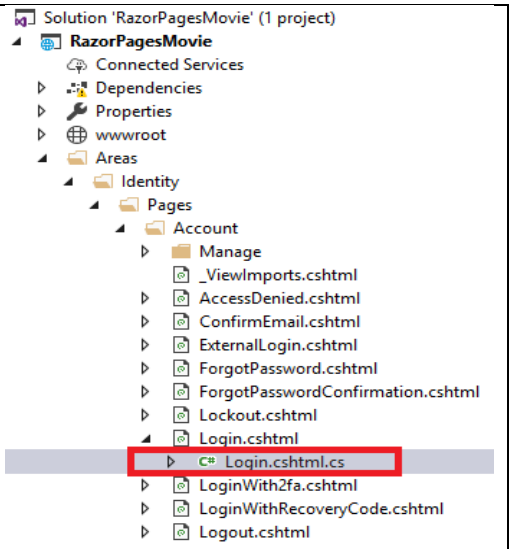
```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Movies
{
    //[Authorize(Roles = "Admin")]
    public class DeleteModel : PageModel
    {
        private readonly RazorPagesMovie.Models.MovieContext _context;

        public DeleteModel(RazorPagesMovie.Models.MovieContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Movie Movie { get; set; }

        public async Task<IActionResult> OnGetAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Movie = await _context.Movie.SingleOrDefaultAsync(m => m.ID == id);

            if (Movie == null)
            {
                return NotFound();
            }
            return Page();
        }

        public async Task<IActionResult> OnPostAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            Movie = await _context.Movie.FindAsync(id);

            if (Movie != null)
            {
                _context.Movie.Remove(Movie);
                //  await _context.SaveChangesAsync();

                // Once a record is deleted, create an audit record
                if (await _context.SaveChangesAsync() > 0)
                {
                    var auditrecord = new AuditRecord();
                    auditrecord.AuditActionType = "Delete Movie Record";
                    auditrecord.DateTimeStamp = DateTime.Now;
                    auditrecord.KeyMovieFieldID = Movie.ID;
                    var userID = User.Identity.Name.ToString();
                    auditrecord.Username = userID;
                    _context.AuditRecords.Add(auditrecord);
                    await _context.SaveChangesAsync();
                }
            }

            return RedirectToPage("./Index");
        }
    }
}
```



Next we will create an audit record **when a failed login attempt has occurred.**

11. Modify the existing **Login.cshtml.cs** file in **Areas/Identity/Pages/Account** folder. Add/Edit in the following highlighted code.

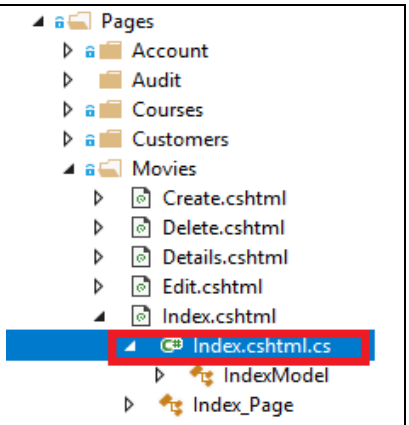
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Account
{
    public class LoginModel : PageModel
    {
        //private readonly UserManager<ApplicationUser> _userManager;
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly ILogger<LoginModel> _logger;
        private readonly RazorPagesMovie.Data.RazorPagesMovieContext _context;

        public LoginModel(SignInManager<ApplicationUser> signInManager,
ILogger<LoginModel> logger, RazorPagesMovie.Data.RazorPagesMovieContext
context)

        {
            // _signInManager = signInManager;
            _logger = logger;
            _context = context;
        }

        [BindProperty]
        public InputModel Input { get; set; }

        public IList<AuthenticationScheme> ExternalLogins { get; set; }

        public string ReturnUrl { get; set; }

        [TempData]
        public string ErrorMessage { get; set; }

        public class InputModel
        {
            [Required]
            [EmailAddress]
            public string Email { get; set; }

            [Required]
            [DataType(DataType.Password)]
            public string Password { get; set; }

            [Display(Name = "Remember me?")]
            public bool RememberMe { get; set; }
        }

        public async Task OnGetAsync(string returnUrl = null)
        {
            if (!string.IsNullOrEmpty(ErrorMessage))
            {
                ModelState.AddModelError(string.Empty, ErrorMessage);
            }

            // Clear the existing external cookie to ensure a clean login process
            await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

            ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

            ReturnUrl = returnUrl;
        }
```



```
        public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            ReturnUrl = returnUrl;

            if (ModelState.IsValid)
            {
                var result = await _signInManager.PasswordSignInAsync(Input.Email,
Input.Password, Input.RememberMe, lockoutOnFailure: true);

                if (result.Succeeded)
                {
                    _logger.LogInformation("User logged in.");
                    return LocalRedirect(Url.GetLocalUrl(returnUrl));
                }
                else
                {
                    // Login failed attempt - create an audit record
                        var auditrecord = new AuditRecord();
                        auditrecord.AuditActionType = "Failed Login";
                        auditrecord.DateTimeStamp = DateTime.Now;
                        auditrecord.KeyMovieFieldID = 999;
                        // 999 – dummy record

                        auditrecord.Username = Input.Email;
                        // save the email used for the failed login
                        _context.AuditRecords.Add(auditrecord);
                        await _context.SaveChangesAsync();
                }

                if (result.RequiresTwoFactor)
                {
                    return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl,
RememberMe =
                    Input.RememberMe });
                }
                if (result.IsLockedOut)
                {
                    _logger.LogWarning("User account locked out.");
                    return RedirectToPage("./Lockout");
                }
                else
                {
                    ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                    return Page();
                }
            }
            return Page();
        }
    }
}
```

12. Modify the existing **Index.cshtml.cs** file in Movies folder. **Search** for the highlighted code and **comment out** the authorization attribute using //. This is to ensure that when the audit trail is being tested, the authorization will not affect the testing.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Authorization;

namespace RazorPagesMovie.Pages.Movies
{
    //[Authorize(Roles = "Admin, Users")]
    public class IndexModel : PageModel
    {
        private readonly RazorPagesMovie.Models.MovieContext _context;

        public IndexModel(RazorPagesMovie.Models.MovieContext context)
        {
            _context = context;
        }
.........................
```



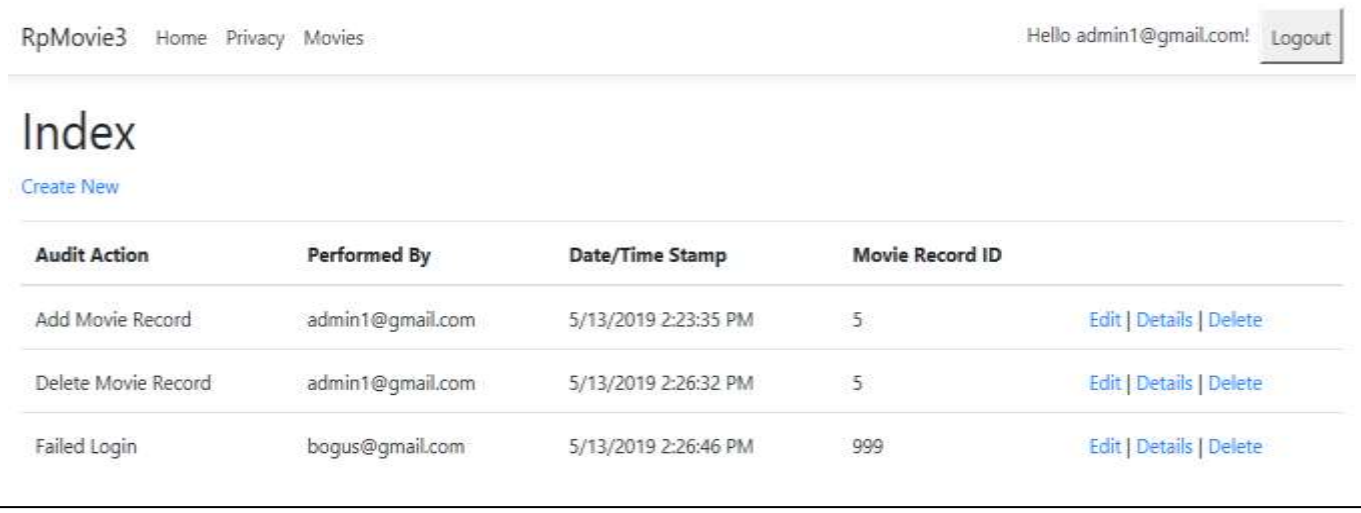
13. Run the application – **Press Ctrl F5.**

Do the following:

- Login as a user with admin role (i.e. admin1@gmail.com) to create a new movie record.
- Delete a movie record.
- Logout and attempt to login with a **non-existent user account** (i.e. bogus@gmail.com).

Login again to check that the audit records are created by going to https://localhost:XXXXX /Audit

Sample Screenshot



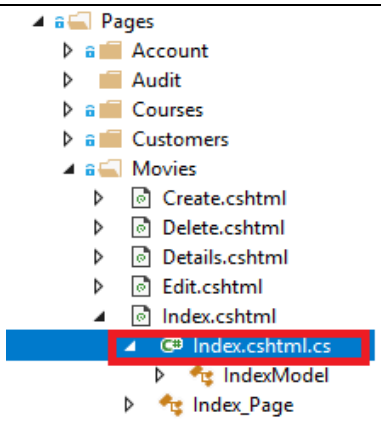
Audit Action	Performed By	Date/Time Stamp	Movie Record ID	
Add Movie Record	admin1@gmail.com	5/13/2019 2:23:35 PM	5	Edit | Details | Delete
Delete Movie Record	admin1@gmail.com	5/13/2019 2:26:32 PM	5	Edit | Details | Delete
Failed Login	bogus@gmail.com	5/13/2019 2:26:46 PM	999	Edit | Details | Delete

If the audit trail feature is working correctly, there should be 3 records showing 3 events being logged.

Note that for a complete audit solution, other important events such as update, login etc. should be audited and data before change and after change should also be logged.

14. Modify the existing **Index.cshtml.cs** file in Movies folder.
Uncomment out the authorization attribute by removing the using //.
We need to use this for the next part of the lab.

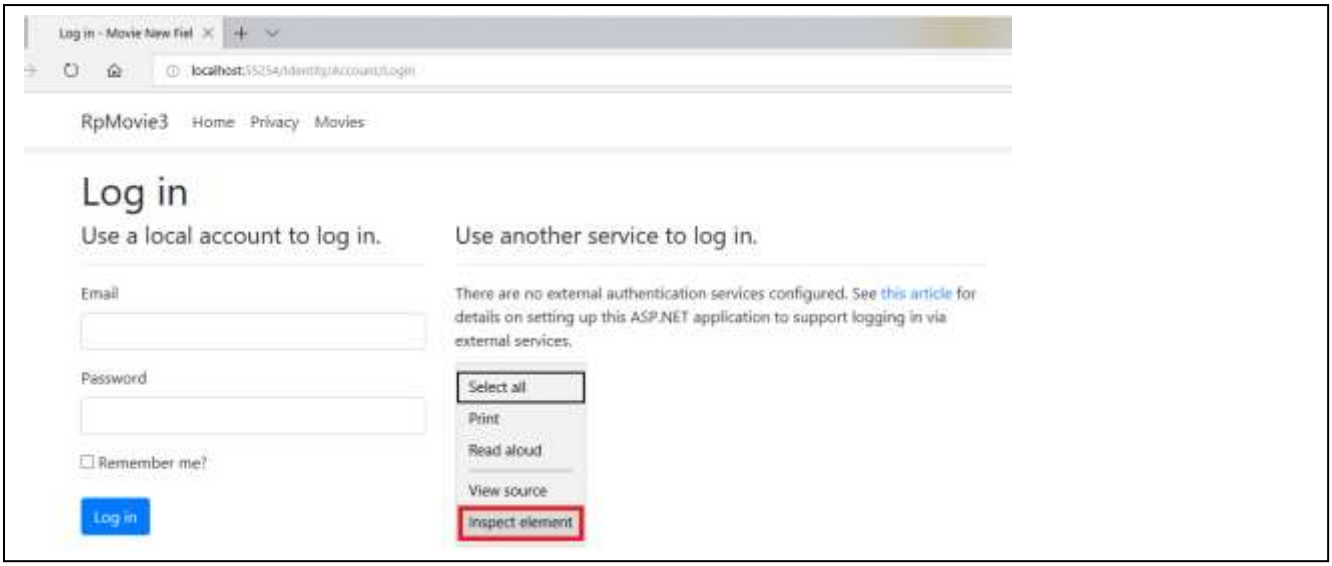
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Authorization;

namespace RazorPagesMovie.Pages.Movies
{
    [Authorize(Roles = "Admin, Users")]
    public class IndexModel : PageModel
    {
        private readonly RazorPagesMovie.Models.MovieContext _context;

        public IndexModel(RazorPagesMovie.Models.MovieContext context)
        {
            _context = context;
        }
……................
```



Part 2 - Session Management – Protecting Application Identity Cookie

1. Press **Ctrl F5** to run the Movie Razor application using **Microsoft Edge.**

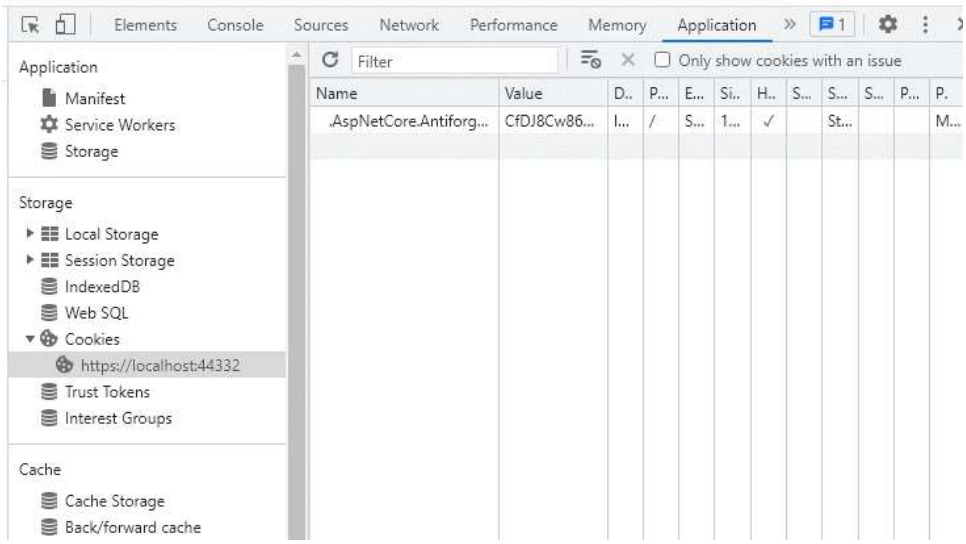
If you are already logged-in, log out first. At the **login page**, right click on the page and click on **inspect element.** (If you are unable to see inspect element, press F12).



2. Another debugging page will load. Click on **Application** tab => **Cookies** =>**https://localhost:xxxxx**.(as shown).

There is only one cookie - this cookie is an antiforgery token used to prevent cross site request forgery attacks. **Authentication cookie used by the identity framework does not exist at this time.** Do not close this page yet.

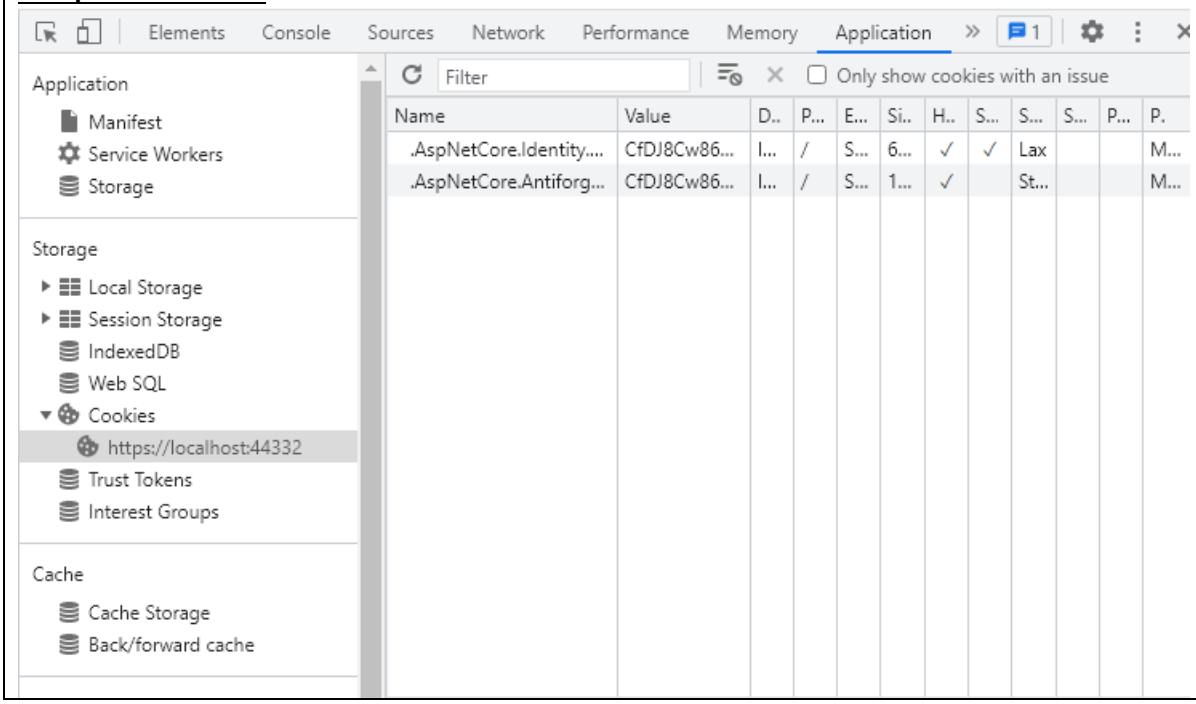
Sample Screenshot



3. Go back to the login page/portion. **Login** with a valid user account (i.e. admin1@gmail.com).

   Click on **Application** tab => **Cookies** =>**https://localhost:xxxxx**.(as shown).

Sample Screenshot



The authentication cookie named .AspNetCore.Identity.Application with its (randomly generated) value is shown.

User authentication in Identity framework involves the use of cookies. This cookie is set and sent by the server on the client browser once a user is properly authenticated.

For subsequent requests, the client will send the cookie along in its own header and the server will use it to recognize the authenticated client and provide continuity of the now-authenticated user's session. That makes the cookie equivalent to a password during the time the session is valid. The server will not continue to send back the cookies, it will only send them if there is a change.

It is important to ensure that hackers do not steal the application authentication cookie. SSL can offer some protection of the authentication cookie.

Session Attack – Stealing Session Authentication Cookie

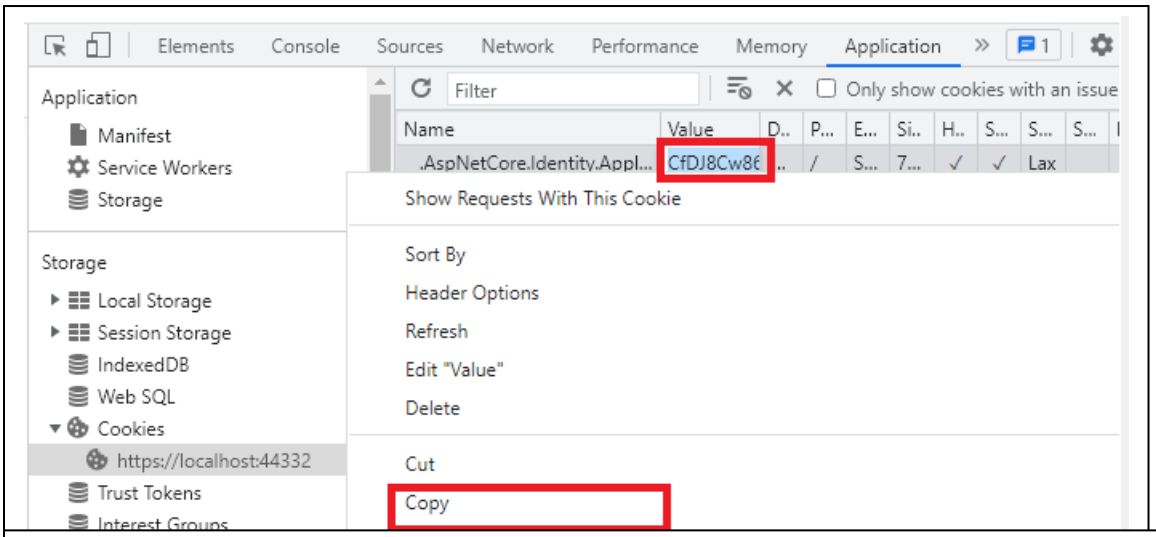
Let simulate a session attack and see how stealing an authentication cookie can compromise the security of an application. For this exercise, we need to 2 web browsers: Microsoft Edge and Chrome and a user account.

The steps are as follows:

4. Press **Ctrl F5** to run the Movie Razor application (http://localhost:XXXXX/Movies) using **Microsoft Edge.** Login with a valid user account (i.e. admin1@gmail.com).

   After login, at the index page right click and select **inspect element** to get debugger page. Click on **Application** Tab=> **Cookies => https://localhost:xxxxx**.

   Right click on **AspNetCore.Identity.Application** cookie and select **Copy selected items**.



Paste the items on a **word document**. **Take note of your own application cookie value** which is highlighted as shown below. Different users would have different cookie values. Will use the value later.

Sample Screenshot

name,value,domain,path,expires,httpOnly,secure

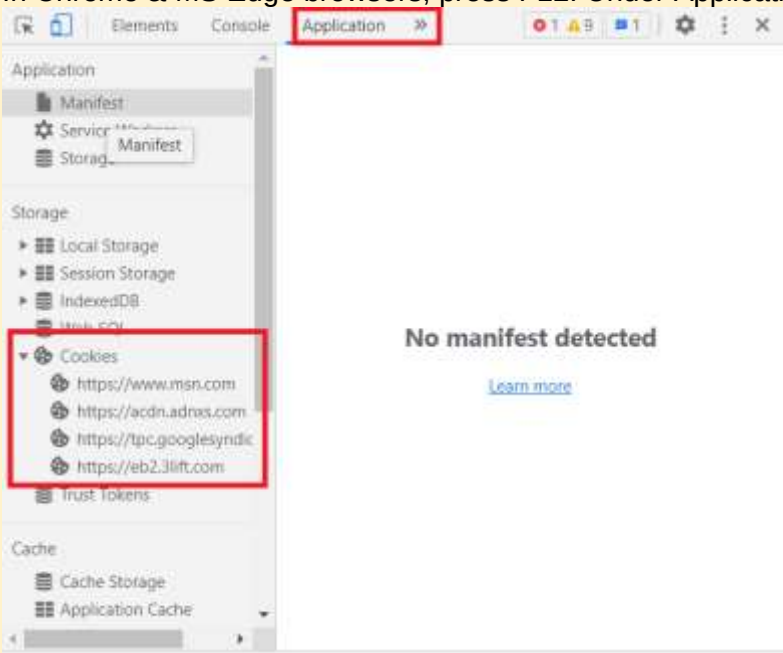
.AspNetCore.Identity.Application,CfDJ8D4_TfAenWRIl6F2d6Lhp8mYHCItABeiO5pvt_he4H8dhZEqC0jRIP9S0okqS7AoolovTYQJ-5TacsX55ZB4fP6O1VpE5ZnRoNzF1oeRyajdSXhHe2sG2ifMhn75C-V6MfYEZPXcM8WWnuQDxxsCvdwQa-VV-CLY3ZKusEDyvP2gBoBG-Imckv0ztHsrJ0y4O7dHwoUeIBwf64CL9-4f1kw2HxyE1zDnPVBInwGALDnSaxsLlStkFI4Rk8FrSNU0NQqsRBKjJN6vK2iiVdOT6Acq2OxexGC3Fq8Jq_gupbWsNGGK11uNs_SDSlmubZ-ZzvLTwUukIZwfU250uZqS8jJyzzNKtZS0-IEP6kXfGvnRymOMpikD1HGAQorVrvMkI_OF-tSAX5Wmx_HfqIDxgb66tt1F0gGrThOQ1ENpYM2xqUaqB9ZGKHnGfXB_tYZaBtPLanOtGdTlxXYNzBm4WgQF9aU9sEZqyLLgAybAu4tierXZEM-eq60HutbApM3L99TDFCW1Q84s_Qky4WIpNumd3afYUgejO7gS08QKCX_5ug5EgnuHAF0eVBv3Cs1aEtWatUr9tv2tK81TVENcOLTEcr1Pxonb3GgpF81qRSlRri-vayGxW-M18AdgUXVvlzPodxl8VUbIXHfZQujYP0Kb19ovV3EFnbWHRjbmxPK3PhUN0d8G0W0lSExNj4yCtQ,localhost,/,Session,true,false

Here we are assuming that a hacker is able to steal this application cookie and use it to launch an attack using another browser.

Note

Please note that you can use your own browser's cookie inspector. It is not necessary to install a cookie inspector add-on or plugin as per shown in step 5.

In Chrome & MS Edge browsers, press F12. Under Application, there is Cookie information.

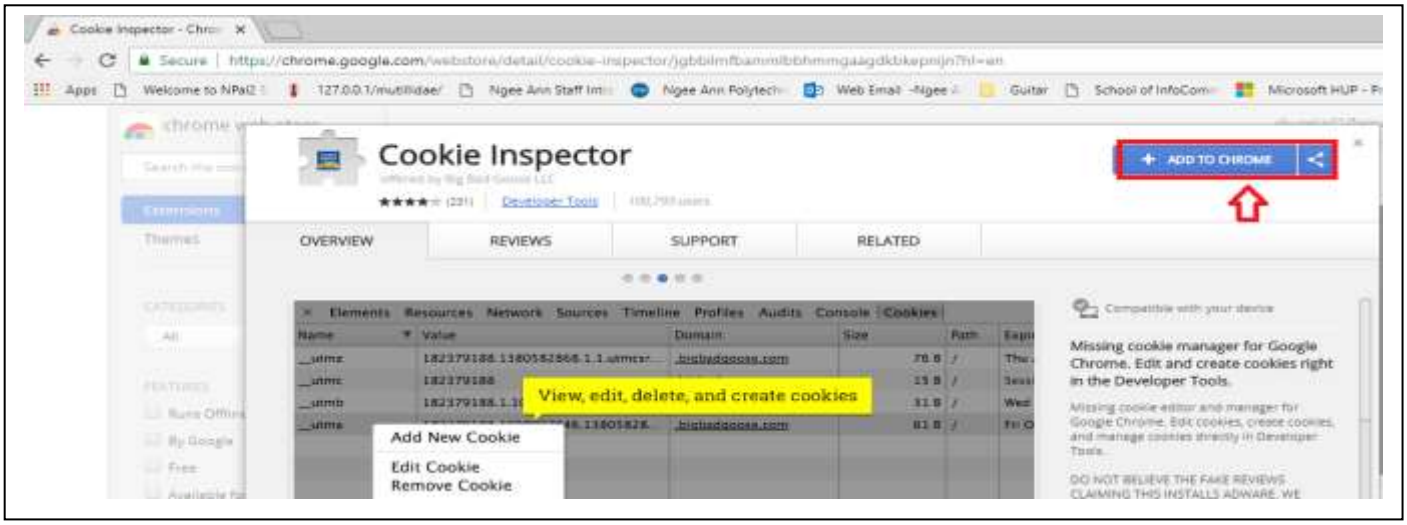


Please note that Step 5 is optional.

(Optional Step 5 if you want to use cookie inspector to insert cookie value in chrome)

5. Do not close the Edge Browser.
   Open Chrome Browser. Go to the following URL:
   https://chrome.google.com/webstore/detail/cookie-inspector/jgbbilmfbammlbbhmmgaagdkbkepnijn?hl=en

   When prompted, click **ADD TO CHROME** to install Cookie Inspector – an application to add and edit cookie in chrome browser.

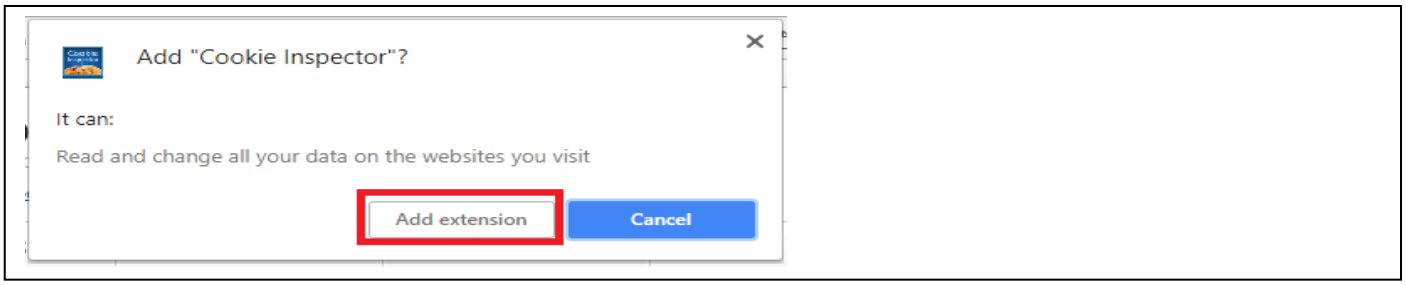


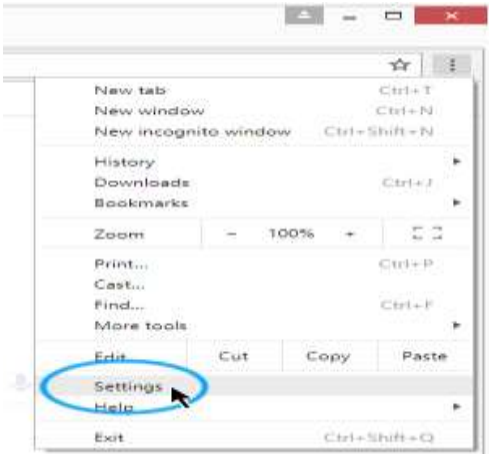
When prompted, click **Add extension.** Wait for it to be installed.

Next enable Cookies in Chrome. The steps in **enabling Cookies in Google Chrome** is as follows:

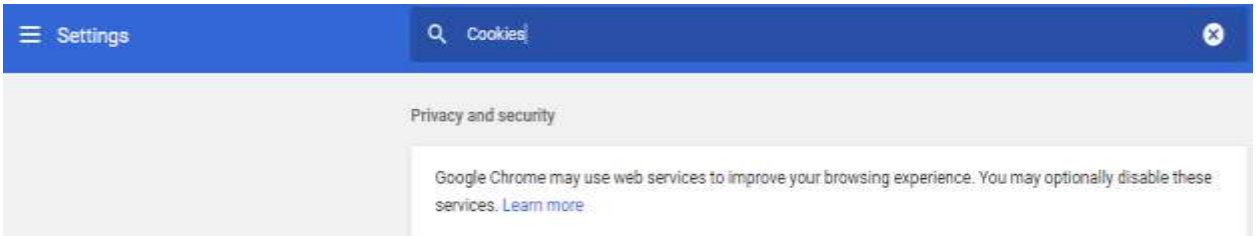
(a) Click the **"Customize and Control"** button.

(b) Select the **"Settings"** menu item.

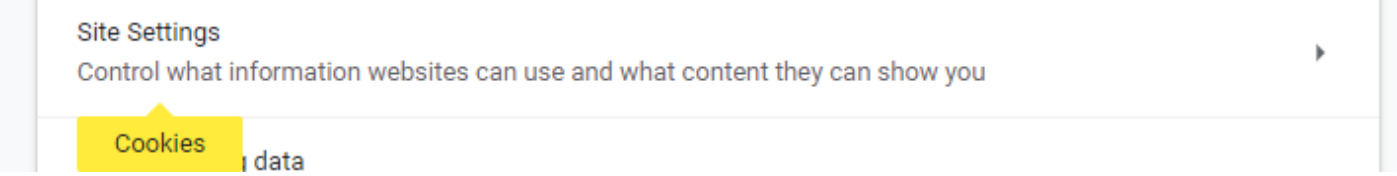




(c) Search for **Cookies** settings. Enter Cookies to search.

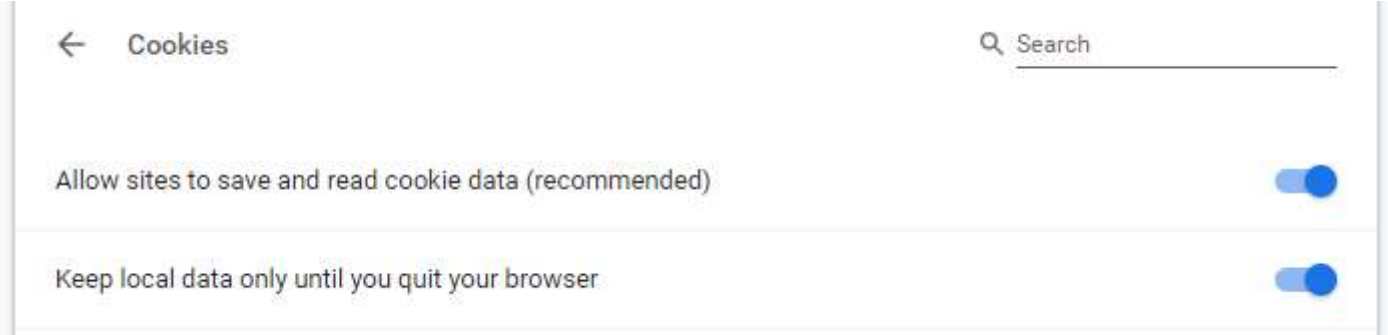


(d) Scroll down to "**Site Settings**" and click it.

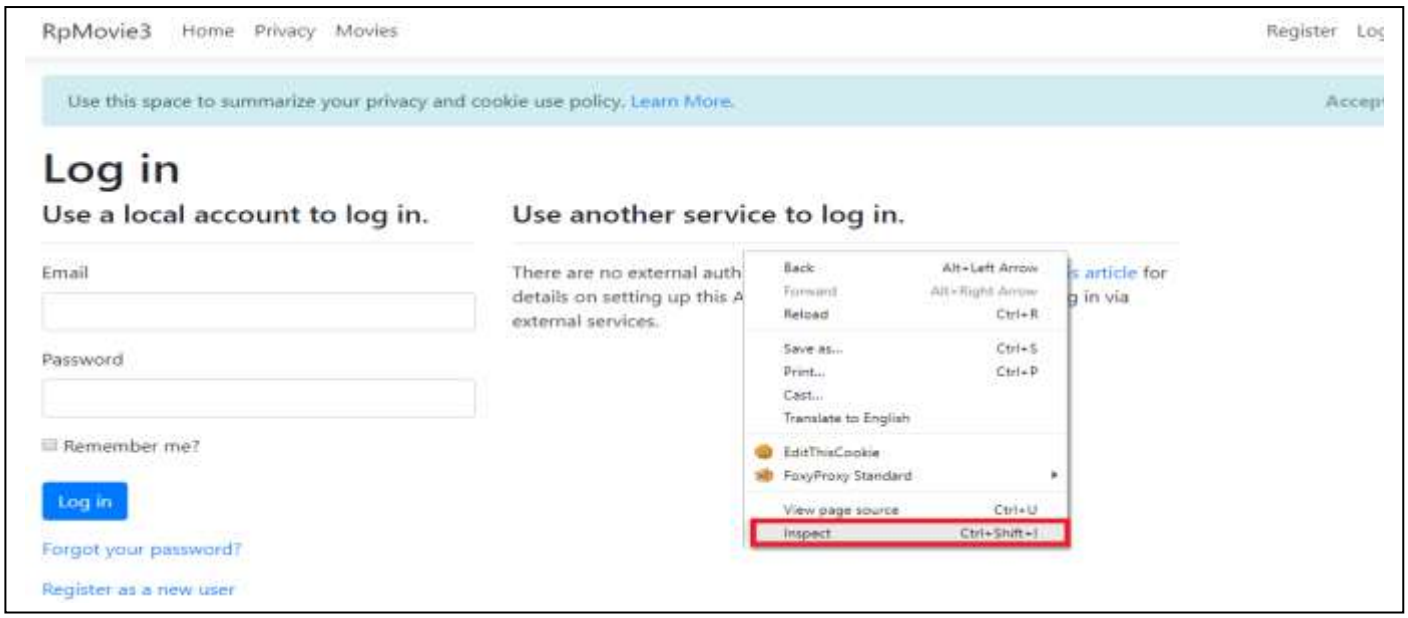


(e) Click on "**Cookies**" items.

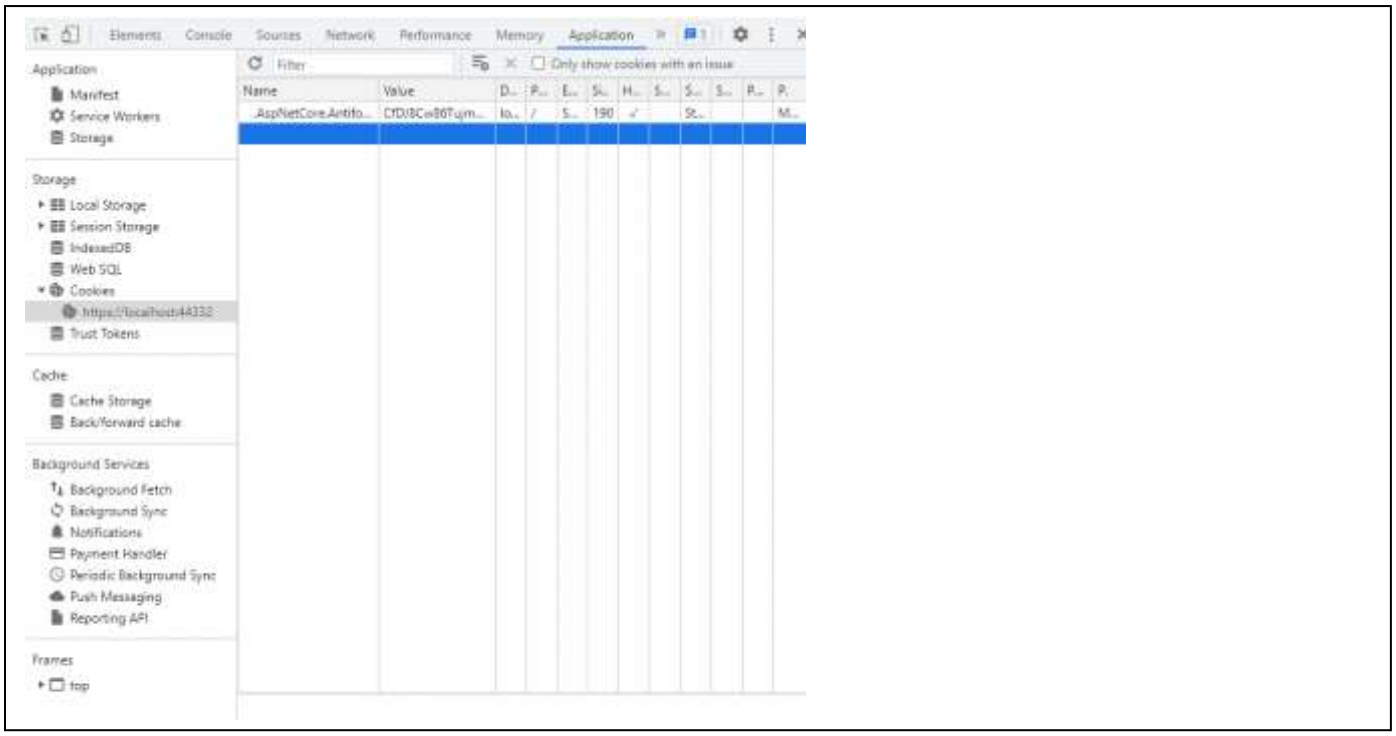
(f) Turn on **"Allow sites to save and read cookie data (recommended)" and "Keep local data only until you quit your browser"**



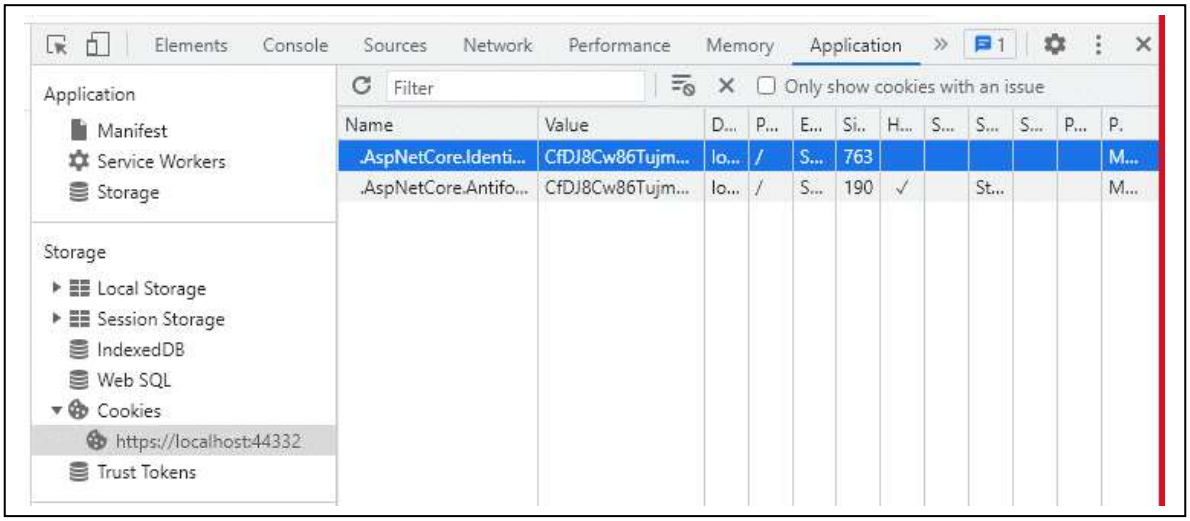
6. Using Google **Chrome**, go to http://localhost:XXXXX/Movies (use the same URL as when you were using Microsoft Edge Browser). Right click on the Chrome page and click on **Inspect** as shown.



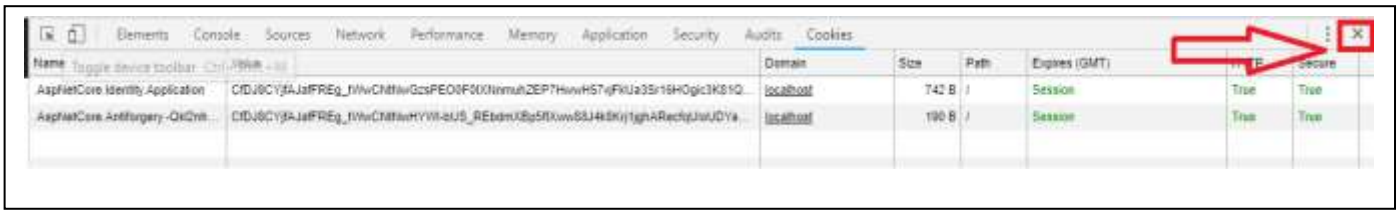
The following page will appear. Click on **Application→Cookies→https://localhost:XXXX**. (Cookies Tab will only appear if Cookie Inspector is installed).



7. On the right side, double-click on respective column entry to enter the details of the new cookie to add:
   Name: **.AspNetCore.Identity.Application**
   Value : (this is value of the application cookie that you copied earlier. Be careful to select correctly only the VALUE from the word document)
   (To paste the values, use Ctrl-v keyboard shortcut)

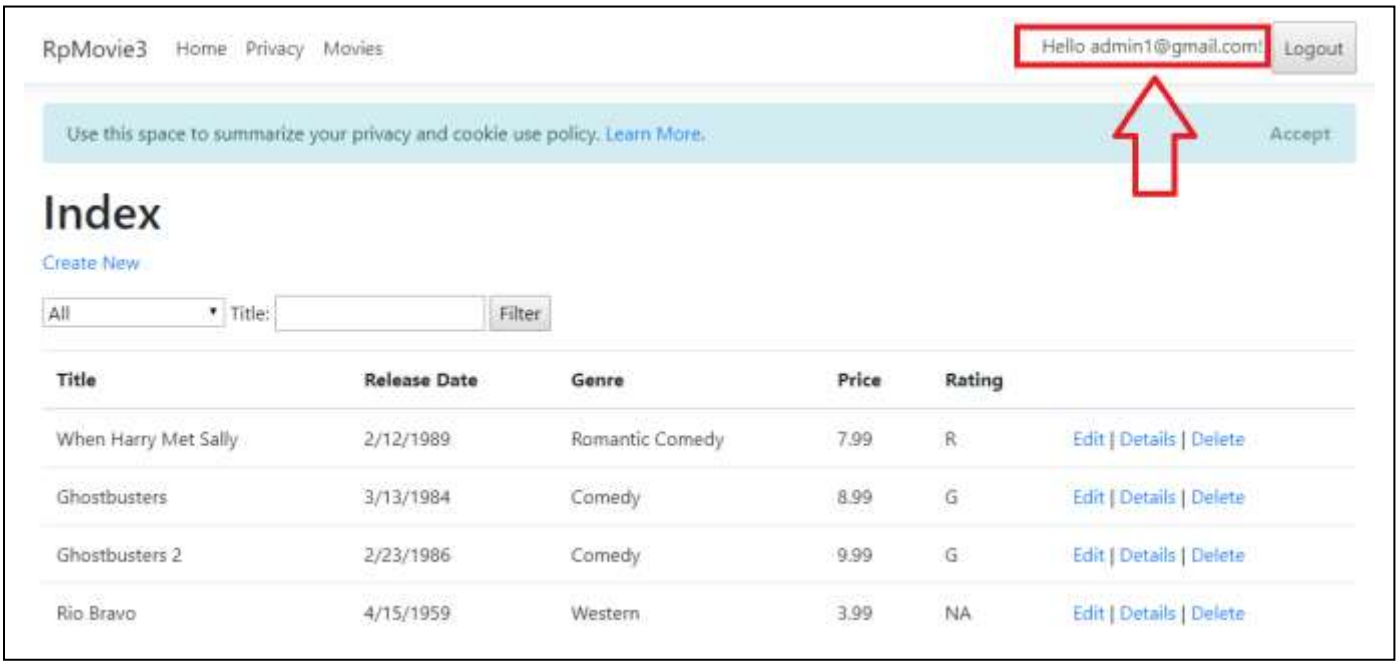


8. Once the cookie values are entered, close the debugger by clicking **x** on the right top corner.



9. On the Chrome browser, type in the **same URL** as before for the Movie application as in step 6 (i.e. http://localhost:XXXXX/Movies) and press **Enter.** It will redirect you to the index page of the Movie application **(without actually entering credentials in the login page)**.

   Notice that on the right top corner, you should be able to see the user account that is associated with the authentication cookie (i.e. admin4@gmail.com).

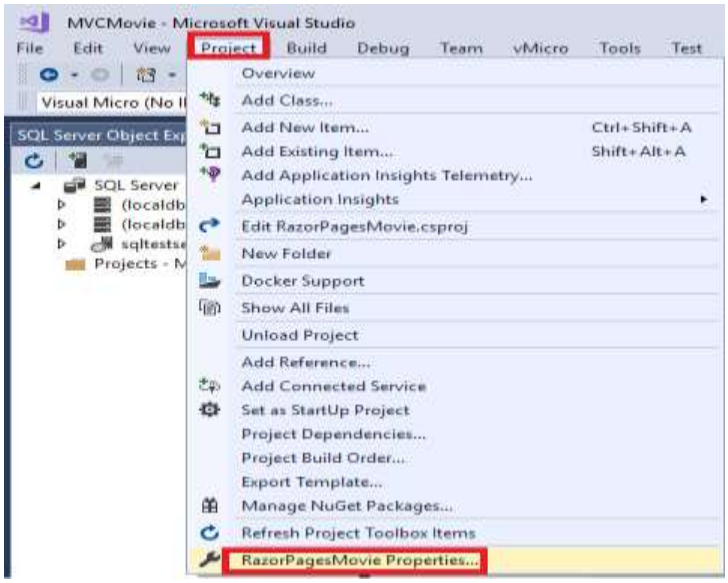


This is how a hacker can steal an authentication cookie, place it on his browser and use it to login and access the application. If you remove or edit the cookie, you will be log out of the application.

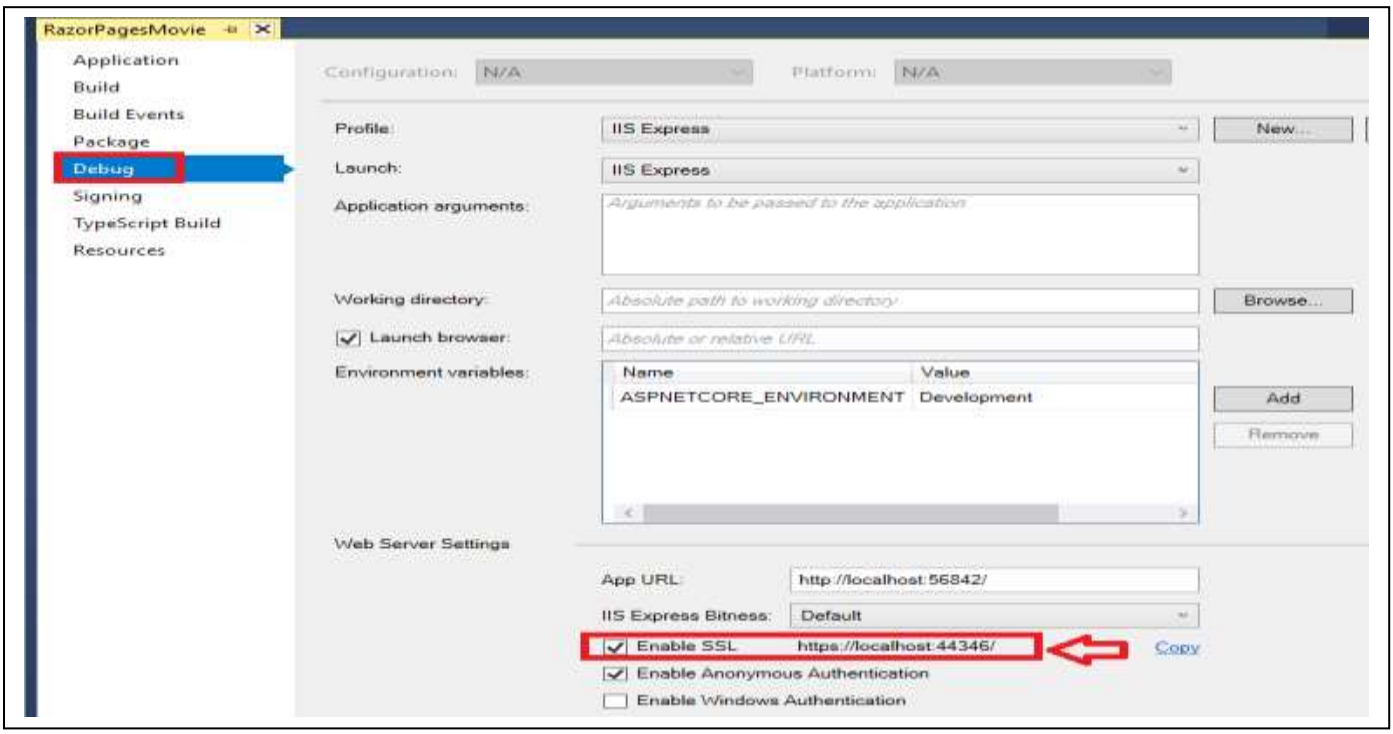
Protecting the Application Identity Cookie

One way to protect application cookie is to use SSL (Secure Socket Layer - HTTPS). Using SSL would ensure that the HTTP packet headers which contain cookies are encrypted when it is being transmitted over the network. This would protect the application from man-in-the-middle attacks in which the end user cookie value could be stolen by a man-in-the-middle using traffic sniffing tools (i.e. Wireshark).

10. To enable SSL functionality on Razor Movie application, click on the **Project Menu** and then **RazorPagesMovie Properties.**



11. On the left side, click on **Debug**. **Enable SSL option. (This is enabled by default)**
You may need to scroll down. Note the URL link for SSL is shown (i.e. https:/localhost:44346/)**.**



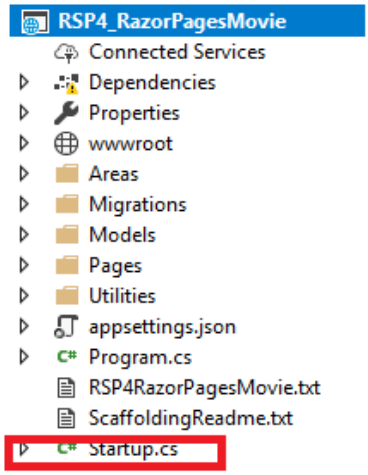
Press **Ctrl -F5** to run the application.

The application would be running with SSL with an URL as https://localhost:XXXXX.

Another way to mitigate the security risks is to configure the application's cookie settings.

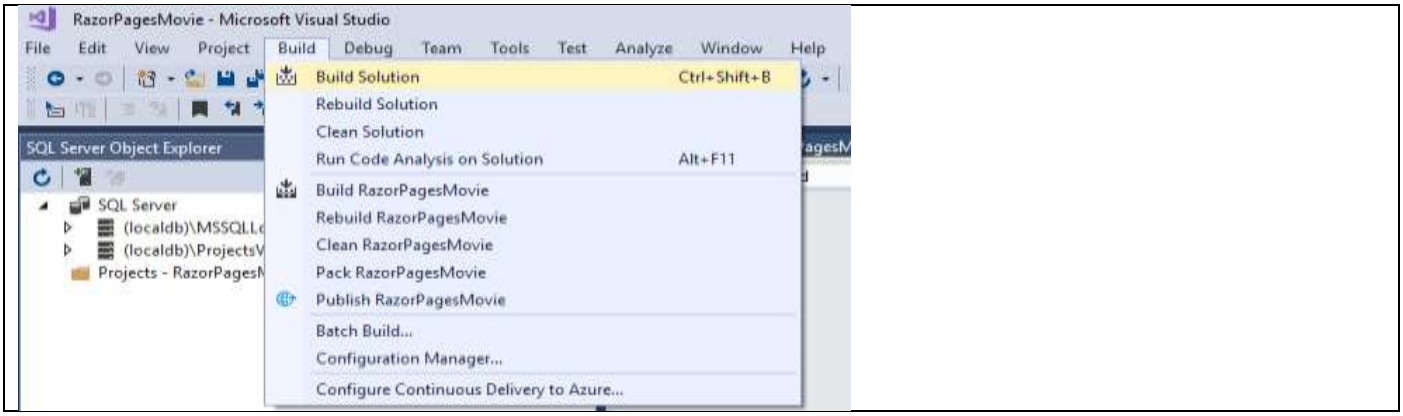
12. Add in the following highlighted code in the **ConfigureServices** method of the existing **Startup.cs**.

```
……................

    public void ConfigureServices(IServiceCollection services)
    {
      ……
      ……

      services.ConfigureApplicationCookie(options =>
      {
       // options.Cookie.Name = "YourCookieName";
       //  options.Cookie.Domain=
       // options.LoginPath = "/Account/Login";
       // options.LogoutPath = "/Account/Logout";
       // options.AccessDeniedPath = "/Account/AccessDenied";

          options.Cookie.HttpOnly = true;
          options.ExpireTimeSpan = TimeSpan.FromSeconds(30);
          options.SlidingExpiration = true;

      });
      ……..
```



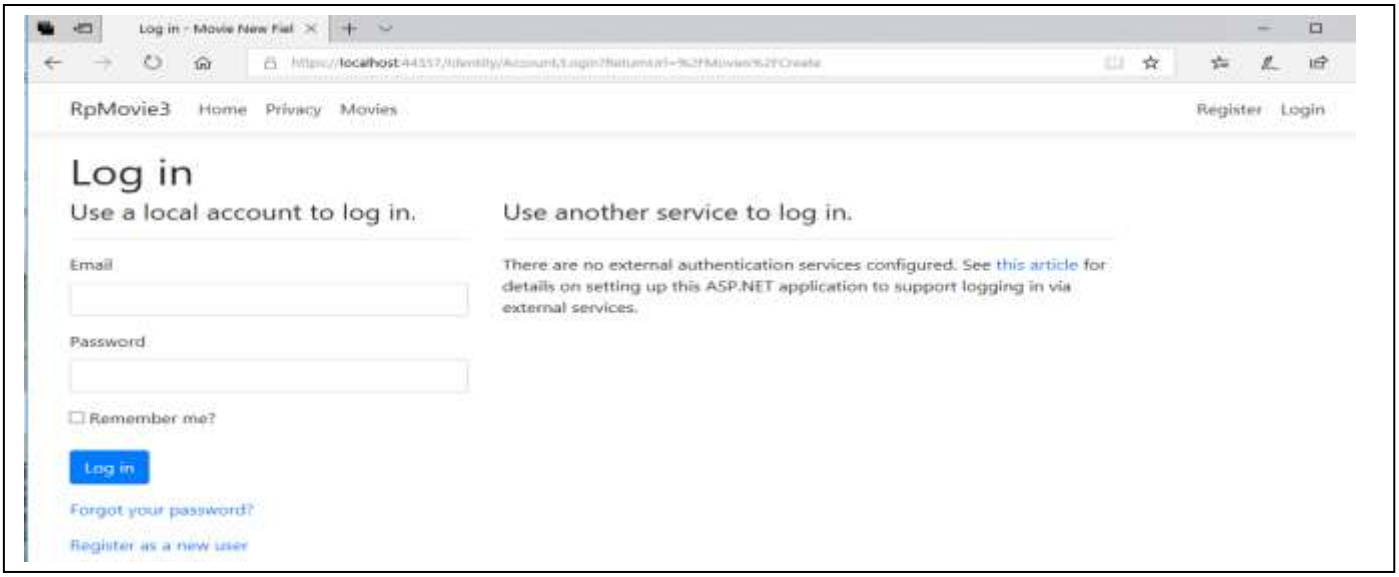
Cookie Configuration Settings Explanation	
Options.Cookie.HttpOnly =true;	• When set to true, it does not allow cookies to be read with client side script like JavaScript. This would mitigate potential XSS vulnerabilities -attempts to read cookie and send it back to attacker. • When a cookie is tagged with the HttpOnly flag, it tells the browser that this cookie should only be accessed by server. Any attempt to access it from client script (using JavaScript <script> alert(document.cookie) ;</script>) is strictly forbidden. Most browsers (IE, Firefox) support HTTP Only flag.
Options.ExpireTimeSpan = TimeSpan.FromSeconds(30);	• This set how long the issued cookie is valid for. Default is 14 days. For this exercise we set to 30 seconds for testing purposes. • If SlidingExpiration is false, user will have to sign back once the 30 seconds is up. If SlidingExpiration is true, then the cookie would be re-issued on any request half way through the ExpiresTimeSpan. • For example, if the user logged in and then made a second request 31 seconds later, the cookie would be re-issued for another 30 seconds. If the user logged in and then made a second request 31 seconds later then user would be prompted to log in. • This setting can keep the validity of authentication cookie's window short and reduces the risk from some types of session hijacking threats.
Options.SlidingExpiration = true;	• When set to true, it resets the expiration time for a valid authentication cookie if a request is made and more than half of the timeout interval has elapsed.
Options.Cookie.Name ="YourCookieName";	• Set the name of the authentication being created. If not named, the default is .AspNetCore.Identity.Application.
Options.LoginPath ="/Account/Login";	• Set the path where an anonymous user will be redirected to sign in with the credentials. Default value is /Account/Login.
Options.LogoutPath ="/Account/Logout";	• When a user is logged, they are redirected to this path. Default value is /Account/Login

14. Click on Build Menu followed by **Build Solution**. Ensure there is no errors when building the solution.



Press **Ctrl F5** to run Movie application using Microsoft Edge.
Login with a valid user account (i.e. admin1@gmail.com). Go to URL https://localhost:XXXXX/Movies.
This will direct you to the index page of Movie application (as shown below).
Do not do anything for more than 30 seconds. After that, try to click on **Create New** link.

You would not be able to create a new record. The application will **prompt you to login again**.

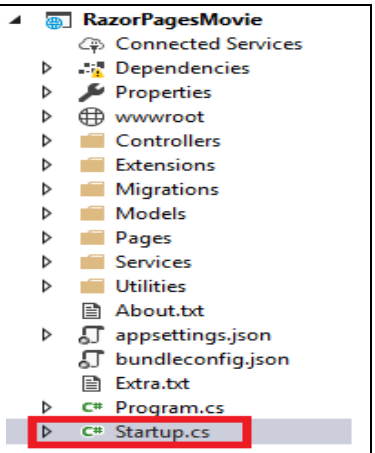


Reason as to why the Movie Create page was not loaded?

- This is because the authentication cookie was only set to be valid for 30 seconds. After 30 seconds the cookie is no longer valid and as such, the application will redirect you to the login page to authenticate.
- In this way, it is implementing a so-called session timeout which is a security control for any application. It specifies the length that an application will allow a user to remain logged in or allow the session to be inactive before forcing the user to re-authenticate.

15. Change the ExpireTimeSpan to a higher value (500 seconds) in the **ConfigureServices** method of the existing **Startup.cs**. This is done so that there may not be errors later on when testing the application.

```
……................

    public void ConfigureServices(IServiceCollection services)
    {
      ……

       services.ConfigureApplicationCookie(options =>
       {
        // options.Cookie.Name = "YourCookieName";
        //  options.Cookie.Domain=
        // options.LoginPath = "/Account/Login";
        // options.LogoutPath = "/Account/Logout";
        // options.AccessDeniedPath = "/Account/AccessDenied";

          options.Cookie.HttpOnly = true;
          options.ExpireTimeSpan = TimeSpan.FromSeconds(500);
          options.SlidingExpiration = true;
       });
      ……..
```



Part 3: Error Management – Custom Error Display and Handling

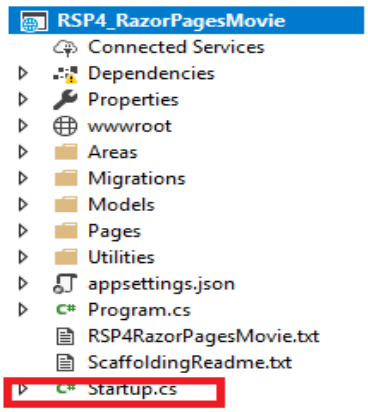
Improper error handling can lead to security problems for a web site. The most common problem is information leakage – when error messages such as stack traces, database dumps and error codes are displayed. These information provide clues on potential flaws in the web site and such messages are also disturbing to normal users.

Another common security problem caused by improper error handling is the fail-open security check. All security mechanisms should deny access until specifically granted, not grant access until denied, which is a common reason why fail open errors occur. Errors also can cause the system to crash or consume significant resources, effectively denying or reducing service to legitimate users.

Let look at show improper error handling can lead to information disclosure

1. Take a look at the error handling code in **Configure** method of the **Startup.cs file** in the Movie Razor application**.**

```
……..
 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
     public void Configure(IApplicationBuilder app, IHostingEnvironment env)
     {

    if (env.IsDevelopment())
      {
        app.UseDeveloperExceptionPage();
        }
        else
        {
       app.UseExceptionHandler("/Error");
       }

           app.UseStaticFiles();
           app.UseAuthentication();
           app.UseMvc();
     }
……..
```

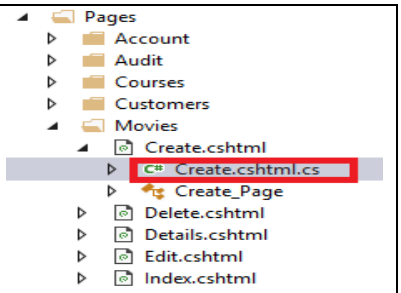


Explanation

- Since we are in a development environment using Visual Studio, the exception handling that is being used is **developerExceptionPage()**. This will give you a stack trace and other useful information. This is suitable if it is used in a development environment but is a risk a production environment.

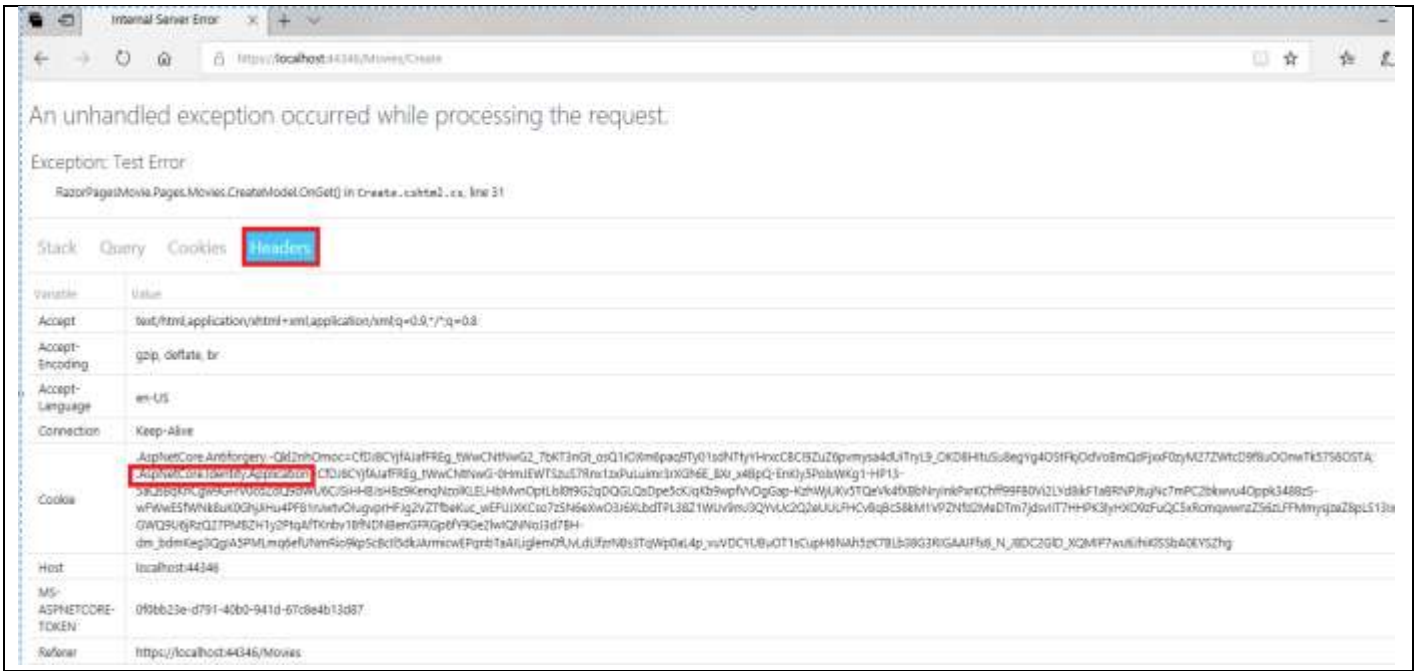
2. Edit **OnGet** method of **Create.cshtml.cs** file of the Movie application.
   Add in the following highlighted code. Here we are simulating an exception when the Create page of Movie app is loaded up (onGet method).

```
……..
public IActionResult OnGet()
     {

        //Movie = new Movie
        //{
        //    Title = "The Good, the bad, and the ugly",
       //    Genre = "Western",
       //    Price = 1.19M,
       //    ReleaseDate = DateTime.Now

       // };
        throw new Exception("Test Error");
       // return Page();
     }
……..
```



Note: Comment out some of the codes as shown.

3. Press **Ctrl F5** to run the Movie application (https://localhost:XXXXX/Movies).
   Login with a valid user account (i.e. admin1@gmail.com).
   After logging in, click on the **Create New** of the index page, the following error page is shown.

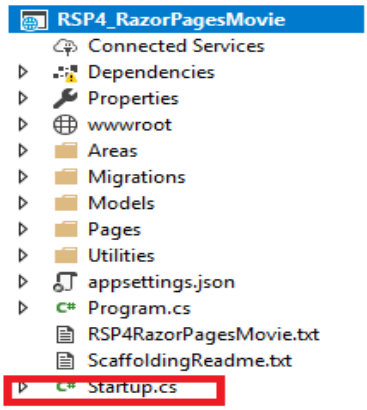


- The page includes several tabs with information about the exception and the request. First tab includes a stack trace. **Click on Headers or Cookies**. The cookie used for authentication can be seen and a hacker can use it to hijack a session.
- This error page for development environment shows all relevant details (Stack, Query, Cookies and Headers) that developers require to resolve the issue. However, this page is **not suitable for production/end-user**, it gives too much information which could be used for malicious reasons.

Let configure for custom error display and handling

4. Edit the **Configure method** of the **Startup.cs** file in the Movie Razor application**. Comment out** certain code as shown.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {

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

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseAuthentication();
        app.UseCookiePolicy();

        app.UseMvc();
    }
```



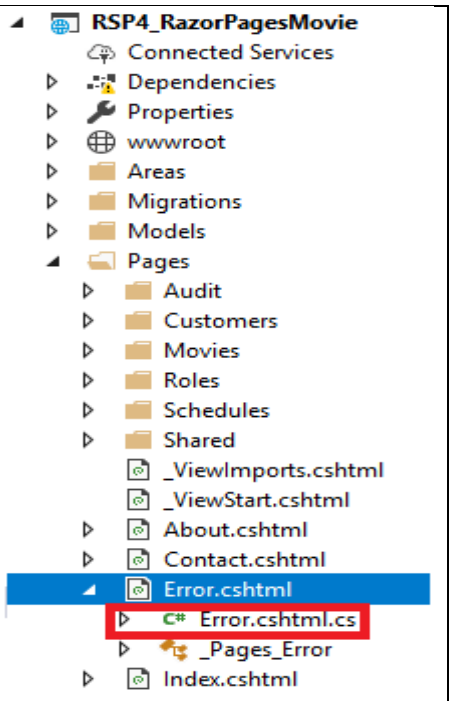
Explanation

- UseExceptionHandler() method uses a **specified error handler** to deal with the exception. It might log the exceptions and show a friendly message to the end user. It gives flexibility on how to build an error response.

- When an uncaught exception happens, it will send the request to the route listening on the path you specified (in this case to **Error.cshtml.cs** of Pages folder). Here we call UseExceptionHandler() method and specify /Error action as a string parameter. **So whenever there is any unhandled exception control will be taken to Error.cshtml.cs.**

5. Edit **Error.cshtml.cs** file in Pages folder as shown. Any exceptions will be routed to this OnGet method. Edit as shown.

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Diagnostics;

namespace RazorPagesMovie.Pages
{
    public class ErrorModel : PageModel
    {
        public string RequestId { get; set; }

        public int iStatusCode { get; set; }
        public string Message { get; set; }
        public string StackTrace { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

        public void OnGet()
        {
            RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;

            // Get the details of the exception that occurred
            var exception = HttpContext.Features.Get<IExceptionHandlerFeature>();

            iStatusCode = HttpContext.Response.StatusCode;
            Message = exception.Error.Message;
            StackTrace = exception.Error.StackTrace;

        }
    }
}
```

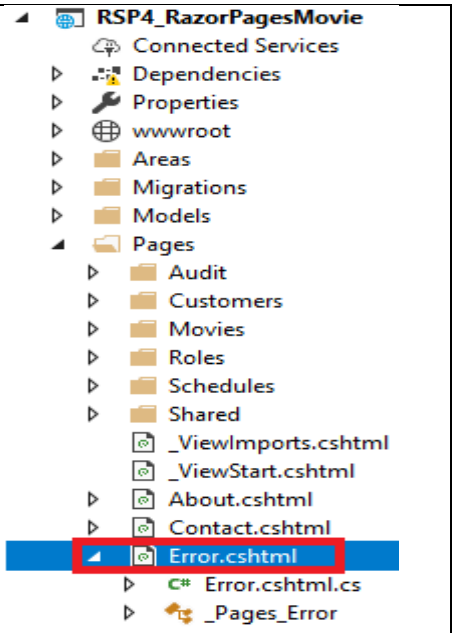


Explanation

- In the OnGet method, the exception details are extracted using Features.Get() method on HttpContext. The HTTP status code is obtained using the Response.StatusCode property. The status code, exception message and exception stack trace are stored in variables for display.

6. Edit **Error.cshtml file** in Pages folder. This is the custom error page to be shown to users. Edit as shown.

```
@page
@model ErrorModel
@{
    ViewData["Title"] = "Error";
}

<h1 class="text-danger">Custom Error Page</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>

@if (Model.ShowRequestId)
{
    <p>
        <strong>Request ID:</strong> <code>@Model.RequestId</code>
    </p>
}

<h3>Custom Error Message</h3>
<p>
    Status  Code : @Model.iStatusCode
</p>
<p>
    Error Message : @Model.Message
</p>
<p>
    Stack Trace : @Model.StackTrace
</p>
```



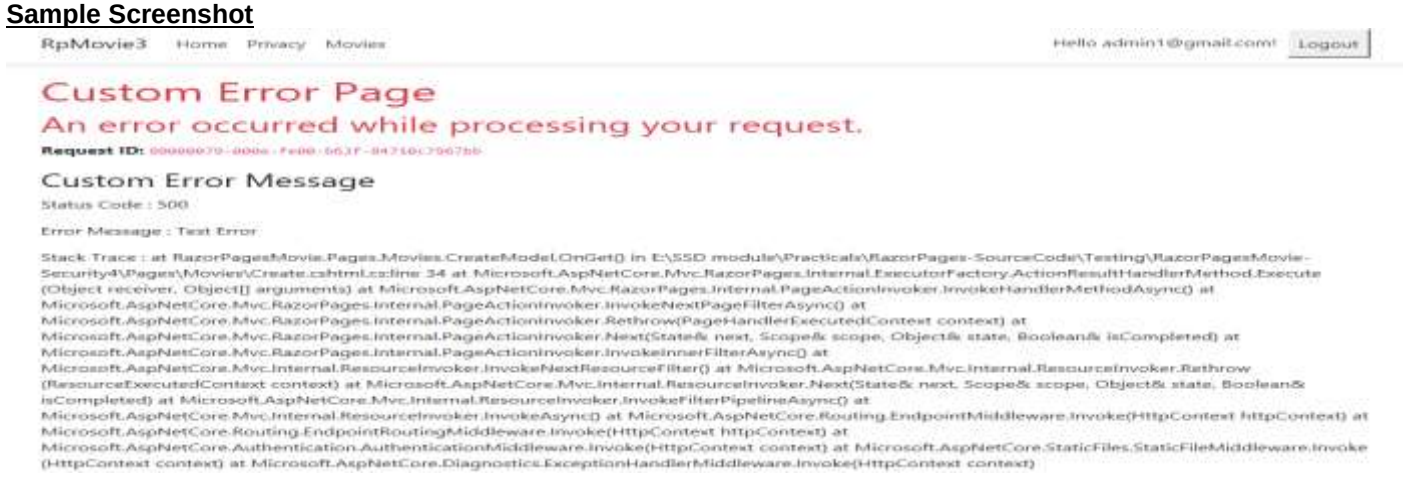
Explanation

- Here, we show every debugging details. For now, we just leave it for testing purpose. But for production use, we should omit out the Error Message and StackTrace.

7. Press **Ctrl F5** to run the Movie application (https://localhost:XXXXX/Movies).
   Login as a user (i.e. admin1@gmail.com).

   After logging in, click on the **Create New** of the index page, the following customized error page is shown. Of course, for production use, the stack and other details should be omitted.

Sample Screenshot



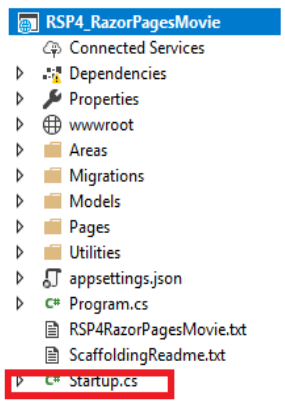
Show Error Pages for HTTP status codes

The two techniques discussed so far deal with the unhandled exceptions arising from your code. However, that is not the only source of errors. Many a times errors are generated due to internal server errors, non-existent pages, web server authorization issues and so on. These errors are reflected by the HTTP status codes such as 500, 404 and 401. A list of HTTP Status Codes is found on this link (https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

Let configure how to display custom error pages for HTTP status codes.

8. Edit the **Configure method** of the **Startup.cs file** in the Movie Razor application**.** Edit as shown.

```
.........
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {

      //   if (env.IsDevelopment())
      //   {
      //       app.UseDeveloperExceptionPage();
      //   }
      //   else
      //   {
      app.UseStatusCodePages("text/html", "<h1>Status code page</h1> <h2>Status Code: {0}</h2>");
      app.UseExceptionHandler("/Error");
      // app.UseHsts();
      //  }

      app.UseHttpsRedirection();
      app.UseStaticFiles();
      app.UseAuthentication();
      app.UseCookiePolicy();
       app.UseMvc();
    }
.......
```

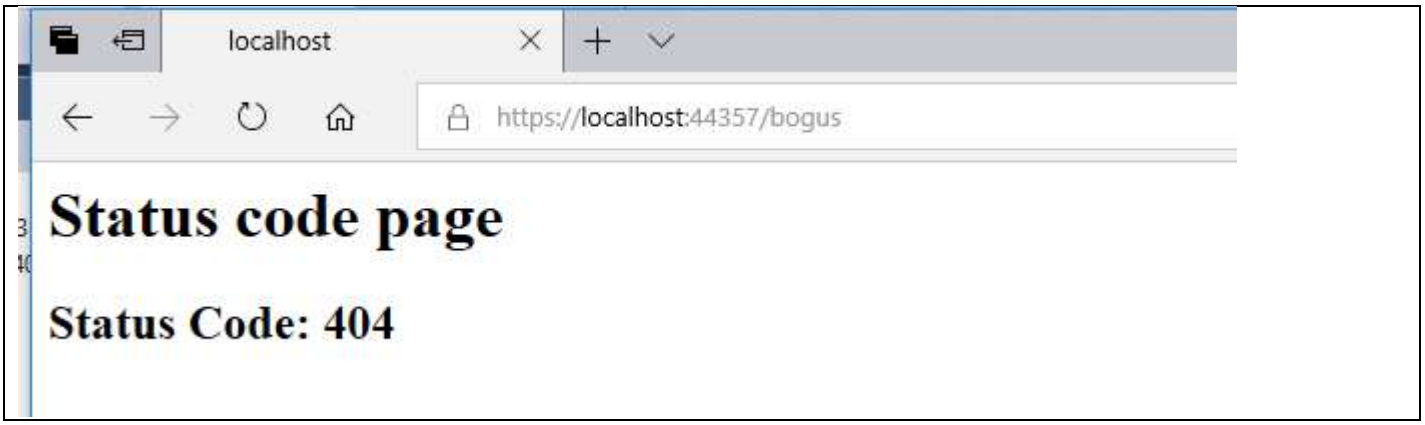


Explanation

- UseStatusCodePages() method uses HTML markup to render the status code. Notice the use of {0} to output the status code at a specific place within the markup.
- Another way is to use **app.UseStatusCodePagesWithRedirects("/CustomErrorPages/{0}.html")** method**.** This code redirects to separate HTML pages stored as 404.html, 500.html and so on. This way you can display different custom error pages for different status codes.

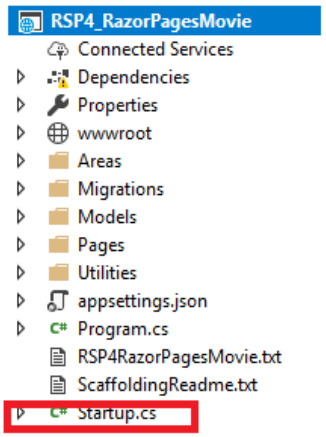
9. Press **Ctrl F5** to run the Movie application (https://localhost:XXXXX/Movies).
   Login as a user (i.e. admin1@gmail.com).
   After logging in, go an non-existent web page (i.e. **https://localhost:XXXXX/bogus**).

   The custom error page showing the HTTP status code is shown . Status Code 404 basically means that the URL is not recognized and web server cannot find the requested resource.



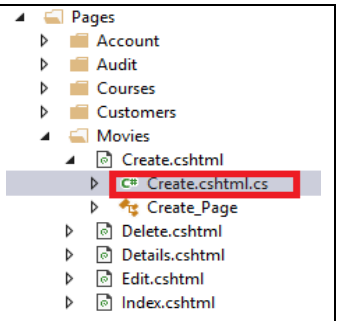
10. Change the code in **Configure method** of the **Startup.cs file** back to original form as shown below**.**

```
.........
 public void Configure(IApplicationBuilder app, IHostingEnvironment env)
   {
       if (env.IsDevelopment())
       {
         app.UseDeveloperExceptionPage();
       }
       else
       {
       app.UseStatusCodePages("text/html","<h1>Status code page</h1> <h2>Status Code: {0}</h2>");
       app.UseExceptionHandler("/Error");
      }

      app.UseHttpsRedirection();
      app.UseStaticFiles();
      app.UseAuthentication();
      app.UseCookiePolicy();
      app.UseMvc();
   }

........
```



11. Comment out the throw exception code in the **OnGet** method of **Create.cshtml.cs** file of the Movie application. .

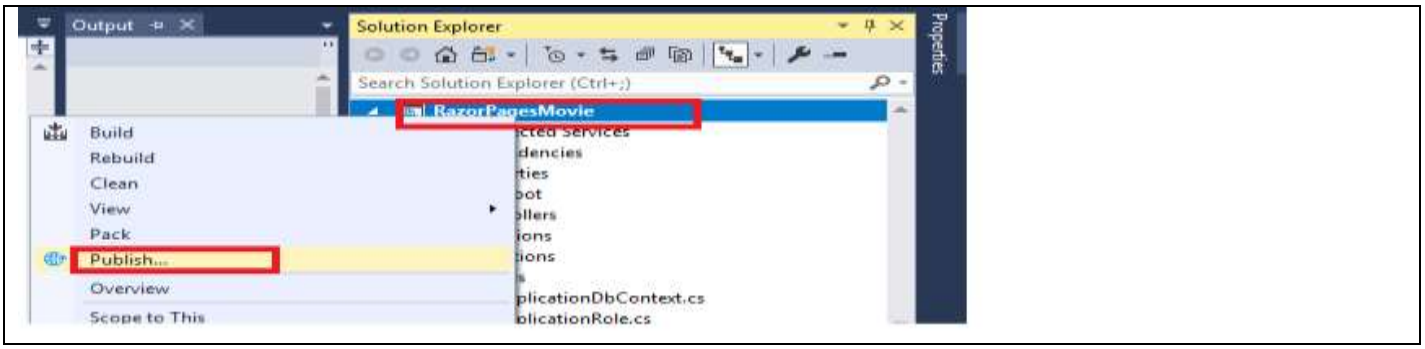
```
.............

 public IActionResult OnGet()
    {
       Movie = new Movie
       {
          Title = "The Good, the bad, and the ugly",
          Genre = "Western",
          Price = 1.19M,
          ReleaseDate = DateTime.Now

       };
       //throw new Exception("Test Error");
       return Page();
    }
.............
```



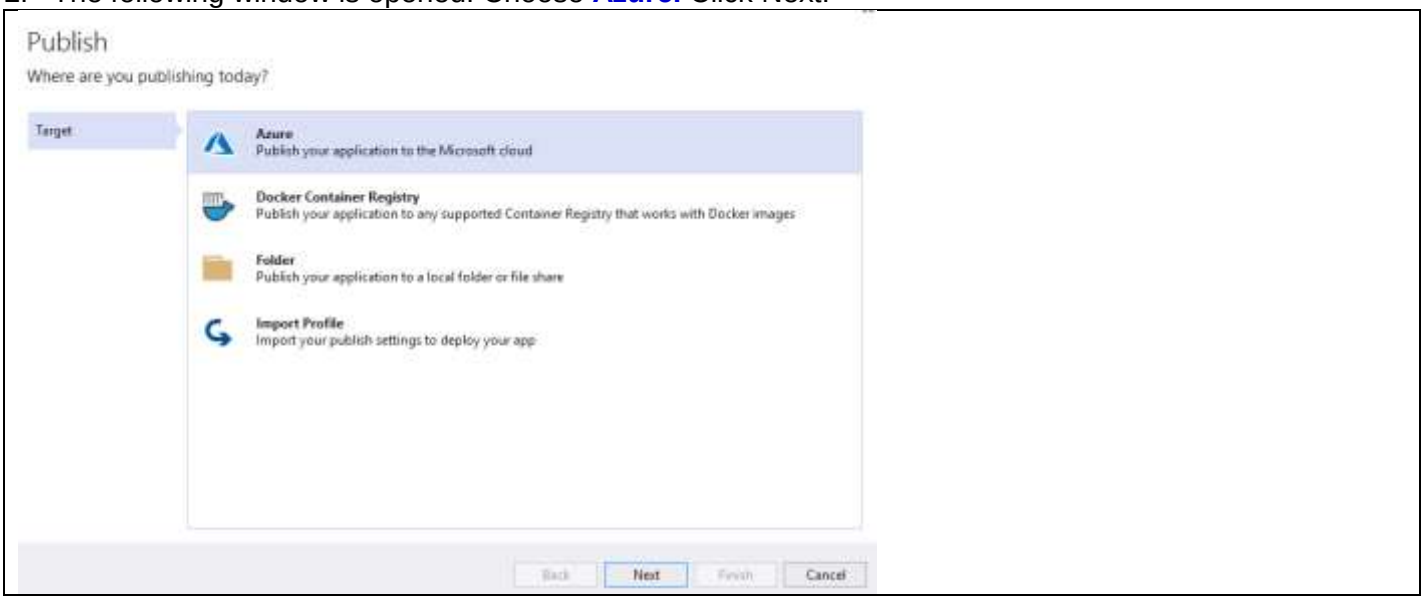
Part 4 – Publishing Razor Application to Azure Cloud Portal (Optional Activity)

Next we will publish the web application to the Azure portal on the Internet. You may need to register in Azure portal to get a free Microsoft Azure account.

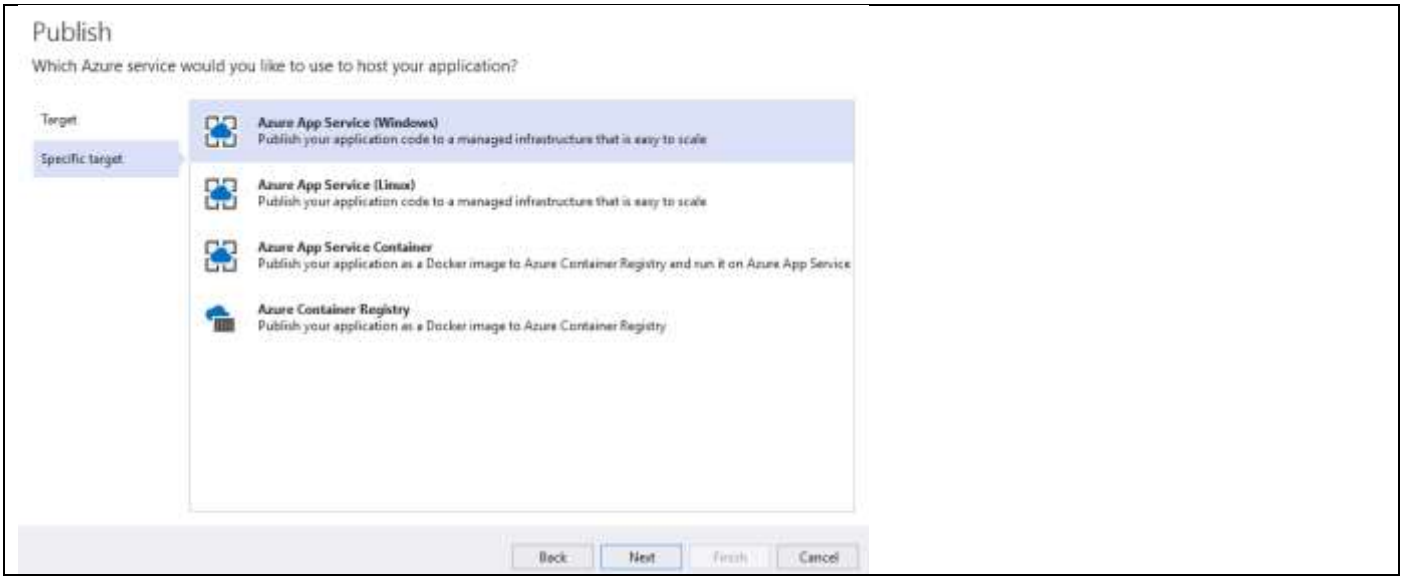
1. Right click on the RazorPagesMovie project from Solution Explorer and click to **"Publish"** option.



2. The following window is opened. Choose **Azure.** Click Next.



3. Select **Azure App Service (Windows).** Click Next.



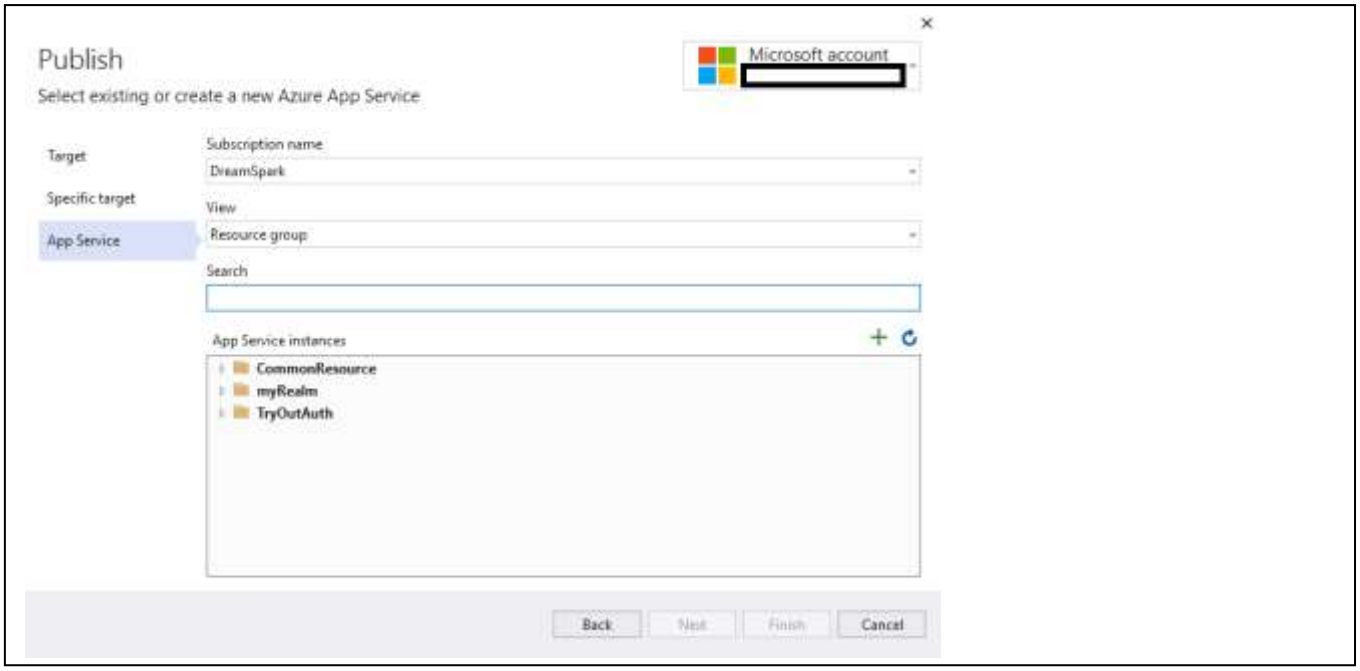
4. If you do not have an existing account, click on **Create your Free Azure Account.**

If you already have an existing account, choose **Sign in.**

5. If you click on create a free azure account, the web browser will navigate to the azure portal. Follow the instructions to create a new free account . Details such as email address, phone number and other details are required.

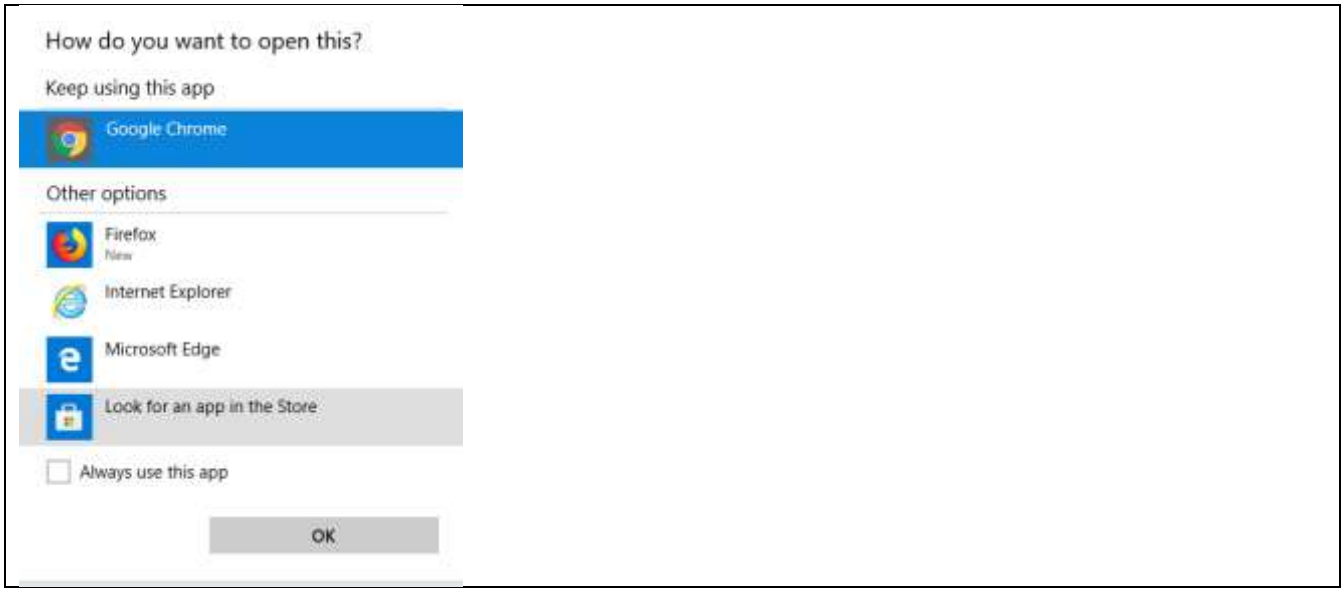
Once a new account has been created, you can now go to the next step 6.

6. Click on Sign In and use your created account to sign in. Thereafter, select your App Service instances where you want to publish to.

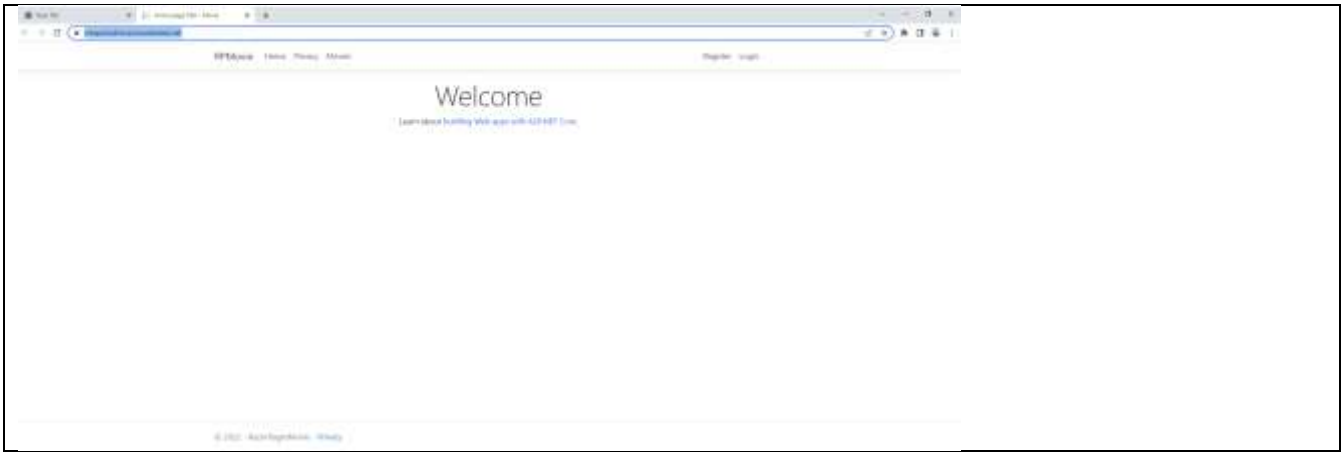


7. The following dialog appears. Click **Publish.**

8. Visual Studio will build and publish the project to the selected app service instance in Azure. Thereafter, the following dialog appears. Choose your selected browser of choice to view**. Click OK.**



9. The browser opens the published site in Azure web server. Note that the URL is no longer localhost.



Note:

- Besides deploying Microsoft NET applications to Azure portal, Azure also supports PHP, Java , Python, Node.js and HTML applications. Windows and Linux virtual machines can also be provisioned to host your applications.

Rerference: You can refer to youtube video below to view how to deploy asp.net core web application to Azure App Service using Visual Studio 2019.
https://www.youtube.com/watch?v=MOG4yp7Zmrc

---------- End ----------