Walkthrough 6 - Error Handling

Setup

This walkthrough will add error handling to the MVC movie tracking application.

1. Open MovieTracker from the end of the previous walkthrough.

Error Handling

1. At the moment, the app doesn't handle errors gracefully.
2. Run the site, navigate to a non-existent URL such as **http://localhost:12345/abc**.
3. The error returned isn't great.

Startup.cs

1. Add middleware for status code pages to the Configure method.
2.
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStatusCodePages();

    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```
3. Save the file, refresh the browser.
4. This is a little better, but can be improved.
5. Add some more middleware that will re-direct the user to the home page.
6.
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStatusCodePages();

    app.Use(async (context, next) =>
    {
        await next();
        if (context.Response.StatusCode == 404)
        {
            context.Request.Path = "/";
            await next();
        }
    });

    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```
7. Save the file, refresh the browser.
8. This is better still, but doesn't inform the user that anything is wrong and could be confusing.
9. Adjust the new middleware to redirect to the error page.
10.
```
...
app.Use(async (context, next) =>
```

```
{
    await next();
    if (context.Response.StatusCode == 404)
    {
        context.Request.Path = "/Home/Error";
        await next();
    }
});
...
```

11. Save the file, refresh the browser.
12. This is better still, but perhaps is too harsh for this common scenario.
13. Adjust the new middleware to redirect to a custom not found page.
14.
```
...
app.Use(async (context, next) =>
{
    await next();
    if (context.Response.StatusCode == 404)
    {
        context.Request.Path = "/Home/~~Error~~CustomNotFound";
        await next();
    }
});
...
```

15. Save the file.

HomeController.cs

1. Add a new method to return a not found page.
2.
```
public IActionResult CustomNotFound()
{
    return View();
}
```

3. Save the file.
4. Right-click in the method and add a Razor View named **CustomNotFound** using the Empty template.

CustomNotFound.cshtml

1. Change the title to **Not Found** and the HTML for the page including a link to the home page.
2.
```
@{
    ViewData["Title"] = "~~Custom~~Not Found";
}

~~<h1>CustomNotFound</h1>~~
<div class="text-center">
    <h1 class="display-1">404</h1>
    <p>The page you are looking for was not found.</p>
    <p><a href="/">Back to Home</a></p>
</div>
```

3. Save the file, refresh the browser.
4. Notice that the _Layout page wasn't specified in CustomNotFound.cshtml, yet it was still found.

_ViewStart.cshtml

1. Open Views / _ViewStart.cshtml to see the layout being specified.

_ViewImports.cshtml

1. Open _ViewImports.cshtml to see the using directives common to all views.

ErrorViewModel.cs

1. Click on the Movies link, then select the details of the first movie.
2. Change the URL to a non-existent movie **http://localhost:12345/Movies/Details/4**, note the error.
3. In the Models folder, open ErrorViewModel.cs.
4. Add a new property for description.
5.
```
public class ErrorViewModel
{
    public string RequestId { get; set; }

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

    public string Description { get; set; }
}
```

6. Save the file.

Error.cshtml

1. In the Views / Shared folder, open Error.cshtml.
2. Add some code for the new Description property and remove the Development Mode text.

3.
```
@model ErrorViewModel
@{
    ViewData["Title"] = "Error";
}

<h1 class="text-danger">Error.</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>
}

<p><strong>Description:</strong> @Model.Description</p>

<h3>Development Mode</h3>
<p>
    Swapping to <strong>Development</strong> environment will display more detailed information about the error that occurred.
</p>
<p>
    <strong>The Development environment shouldn't be enabled for deployed applications.</strong>
    It can result in displaying sensitive information from exceptions to end users.
    For local debugging, enable the <strong>Development</strong> environment by setting the <strong>ASPNETCORE_ENVIRONMENT</strong> enviro
nment variable to <strong>Development</strong>
    and restarting the app.
</p>
```

4. Save the file.

MoviesController.cs

1. Update the error handling code in the Details method.
2.
```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
        return View("Error",
            new ErrorViewModel
            {
                Description = "Movie id invalid."
            });
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return NotFound();
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to find movie with id={id}."
            });
    }
```

3. Save the file, refresh the browser.
4. Update the error handling in the Edit methods.
5.
```
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
        return View("Error",
            new ErrorViewModel
            {
                Description = "Movie id invalid."
            });
    }

    var movie = await _context.Movie.FindAsync(id);
    if (movie == null)
    {
        return NotFound();
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to find movie with id={id}."
            });
    }
    return View(movie);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("Id,Title,DateSeen,Genre,Rating")] Movie movie)
{
    if (id != movie.Id)
    {
        return NotFound();
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Movie ids don't match, id={id} not equal to Movie.id={movie.Id}."
            });
    }

    if (ModelState.IsValid)
```

```
    {
        try
        {
            _context.Update(movie);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MovieExists(movie.Id))
            {
                return NotFound();
                return View("Error",
                    new ErrorViewModel
                    {
                        RequestId = movie.Id.ToString(),
                        Description = $"Unable fo find movie with id={movie.Id}."
                    });
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(movie);
}
```

6. Update the error handling in the Get Delete method.
7.

```
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
        return View("Error",
            new ErrorViewModel
            {
                Description = "Movie id invalid."
            });
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return NotFound();
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to find movie with id={id}."
            });
    }

    return View(movie);
}
```

8. Save the file.
9. Edit a movie and change the URL to a non-existent movie and a non-integer movie id. The errors are handled.
10. Delete a movie and change the URL to a non-existent movie and a non-integer movie id. The errors are handled.

DateSeenValidation.cs

1. Add a movie with a date seen in the future. This doesn't make sense but succeeds.
2. The Range attribute doesn't work with dates and also requires hard-coding the values.
3. In the Models folder, add a new class named **DateSeenValidation.cs**.
4. Make the class inherit the ValidationAttribute class, add the **using System.ComponentModel.DataAnnotations;** directive.
5.

```
public class DateSeenValidation : ValidationAttribute
{
}
```

6. Implement the IsValid method.
7.

```
public class DateSeenValidation : ValidationAttribute
{
    public override bool IsValid(object value)
    {
        var dateSeen = Convert.ToDateTime(value);
        return dateSeen <= DateTime.Now;
    }
}
```

8. Save the file.

Movie.cs

1. Add the newly created annotation to the DateSeen property with an appropriate error message.
2.

```
public class Movie
{
    [Key]
    public int Id { get; set; }

    [Required]
    public string Title { get; set; }

    [DataType(DataType.Date), Display(Name = "Date Seen")]
```

```
    [DateSeenValidation(ErrorMessage = "Date can't be in future.")]
    public DateTime? DateSeen { get; set; }

    public string Genre { get; set; }

    [Range(1, 10)]
    public int? Rating { get; set; }
}
```

3. Save the file.
4. Edit the movie just added and attempt to save with the date seen in the future. It won't be possible. Edit the date seen to the past, save the change. Edit the movie again and remove the date, the validation will also pass.