Shefqet <Ceti> Zyko, P.L. (Eng.), P.T. (Eng.)

Data Analyst | Full-Stack Software Developer | Quality Assurance

Calgary, Alberta

403.293.0458 | Email: szceti@outlook.com | www.linkedin.com/in/CetisZ | https://github.com/CetisZ

PROFESSIONAL SUMMARY

- Passionate Technology Enthusiastic and life-long student working towards professional certifications in Cloud AWS, JavaScript Algorithms, Data Analysis & Visualization, and ML - AI
- Expertise utilizing web development technologies such as JavaScript, NodeJS, C#, ASP.NET, ….
- APEGA + ASET member, Professional Licensee (Eng..), Professional Technologist (Eng.)
- 17 years of data collection - management, quality assurance, and automated testing of various applications in oil, natural gas, manufacturing, and municipal water sectors to solve problems.
- Creative Thinking, Communication, Time Management, Decision Making, Organization and Docs.

Ceti's SportsPro Technical Support Project

Creation of SportsPro Technical Support website, using ASP.NET Core MVC language, which will track technical support service calls (incidents) in a database that also stores information about company's customers, software products, and technicians. It will also use C#, HTML, CSS, Bootstrap, and more.

Description:

Website: SportsPro Technical Support website consist of webpages that support two types of users.
First, administrators - can manage the products, technicians, customers, incidents, and product registrations that are in the data base.
Second, technicians – can update incidents that have been assigned to them.

Database: It will store data that is needed to track technical support incidents. The Incidents table contains one row for each technical support incident.

Each row in the Incidents table is related to:
- one row in the Customers table, which contains data about the company's customers
- one row in the Products table, which contains data about the company's products, and
- one row in the Technicians table, which contains data about the company's technical support staff.

In addition, each row in the Customers table is related to one row in the Countries table, which stores a list of available countries.

The Product Registrations table, on the other hand, is a linking table that creates a many-to-many relationship between the Customers and Products tables.
As a result, each row relates one customer to one product. This allows one customer to register many products, and it allows one product to be registered by many customers.

Deliverables:

1a. *Manage Products* - Admin user can view, edit and delete existing products as well as add new products.

1b. *Manage Technicians* - Admin user can view, edit and delete existing technicians as well as add new technicians.

1c. *Manage Customers* - Admin user can view, edit and delete existing customers as well as add new customers.

1d. *Manage Incidents* - Admin user can view, edit and delete existing incidents as well as add new incidents.

1e. *The URLs for webpages* – Added an About page that is displayed by the About action of the Home controller. And use attribute routing to shorten some of the URLs.

1f. *Razor views* – example: the Products link is active for Product controller. Same for the rest of other controllers.

1g. *Use of TempData to display messages*: In the Product controller the TempData is used to store a success message after each successful add, edit, or delete operation is performed. Then message is displayed on the Product Manager page. Same logic is used for other controllers.

1h. *Use of View Model:* Used view model to pass data to the list Manager page and its Incident page.

1i. *Update Incidents:* Technicians can view all of their assigned and open incidents. And can edit an incident's description and also specify a close date for the open incident.

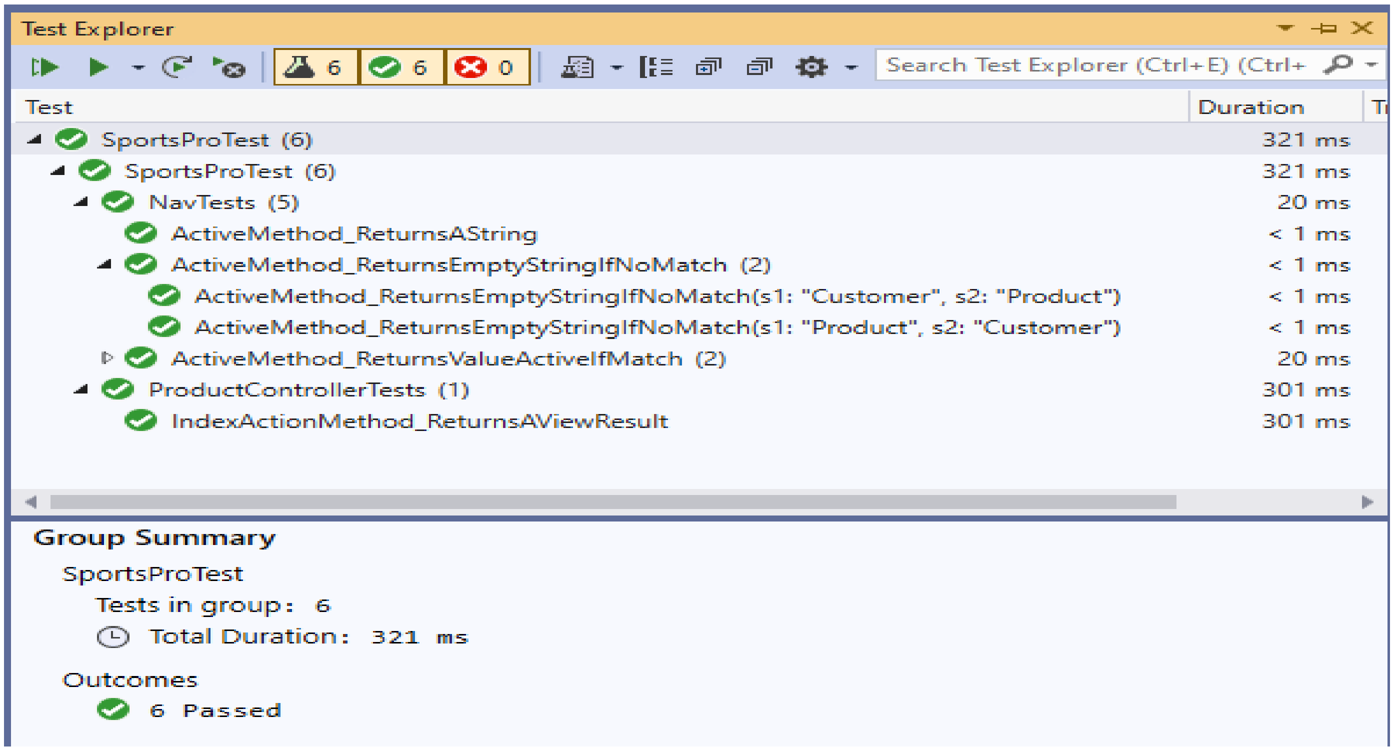
1j. *Adding filtering to a webpage:* Added filtering to Incident Manager webpage, the Admin user is allowed to filter incidents by only displaying unassigned incidents or only displaying open incidents.

1k. *Validation:* Added data validation for the Add/Edit Customer page.
Note: The same email address cannot be accepted twice.

1l. *Manage registrations:* Added webpages that let Admin user to view the Product Registration for a customer and add a new product registration as well.

1m. *Encapsulate the data layer:* Website uses the repository pattern and unit of work pattern to encapsulate the data layer.
Incident controller uses the "use unit of work class". Product controller uses "repository class".

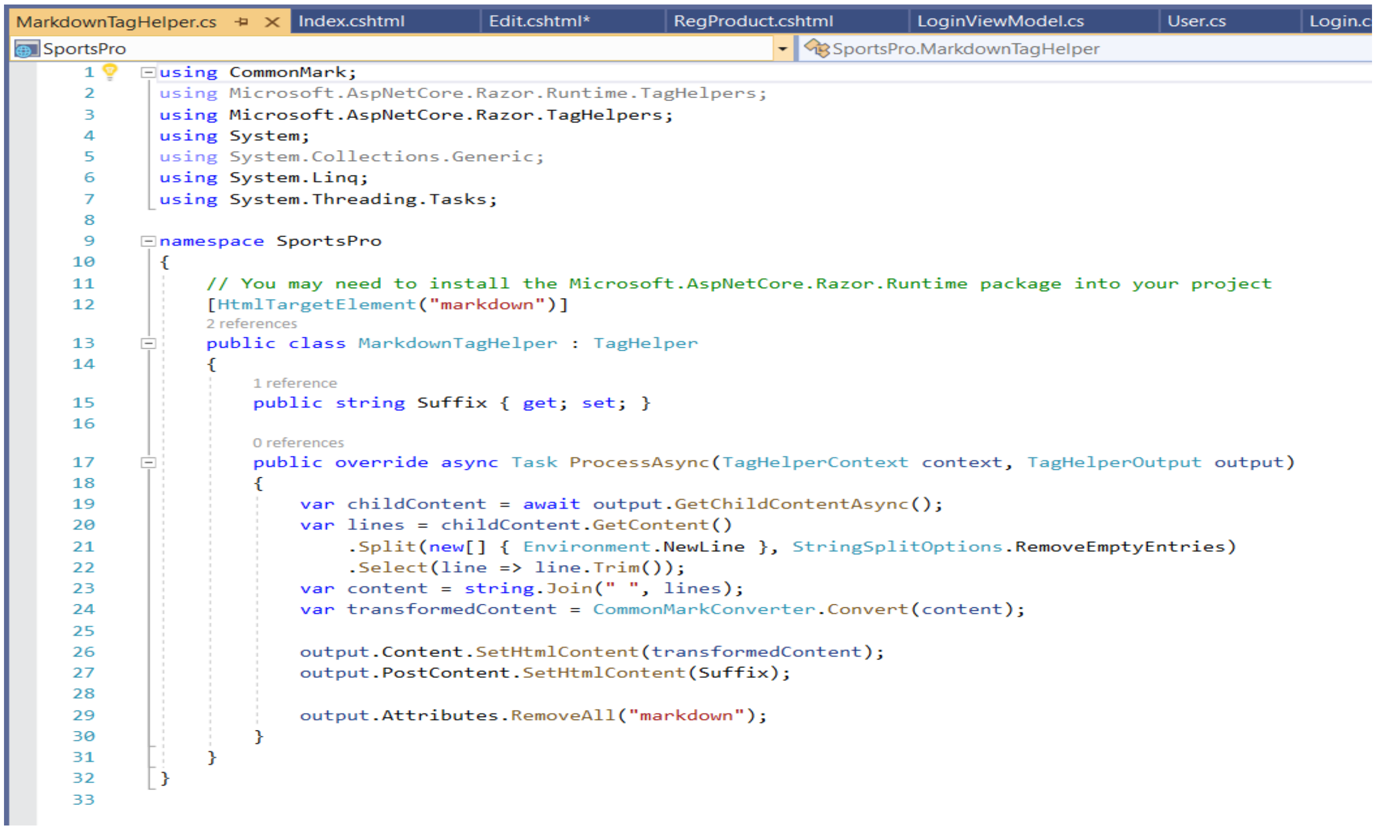
1n. *Automate testing:* To facilitate automation of testing the Product and Customer controllers were updated with the use of dependency injection (DI). See Test Explorer screenshot for illustration.



```
Test                                                                                    Duration
SportsProTest (6)                                                                        321 ms
  SportsProTest (6)                                                                      321 ms
    NavTests (5)                                                                          20 ms
      ActiveMethod_ReturnsAString                                                        < 1 ms
      ActiveMethod_ReturnsEmptyStringIfNoMatch (2)                                       < 1 ms
        ActiveMethod_ReturnsEmptyStringIfNoMatch(s1: "Customer", s2: "Product")          < 1 ms
        ActiveMethod_ReturnsEmptyStringIfNoMatch(s1: "Product", s2: "Customer")          < 1 ms
      ActiveMethod_ReturnsValueActiveIfMatch (2)                                          20 ms
    ProductControllerTests (1)                                                           301 ms
      IndexActionMethod_ReturnsAViewResult                                               301 ms

Group Summary
  SportsProTest
    Tests in group:  6
    Total Duration:  321 ms

  Outcomes
    6 Passed
```

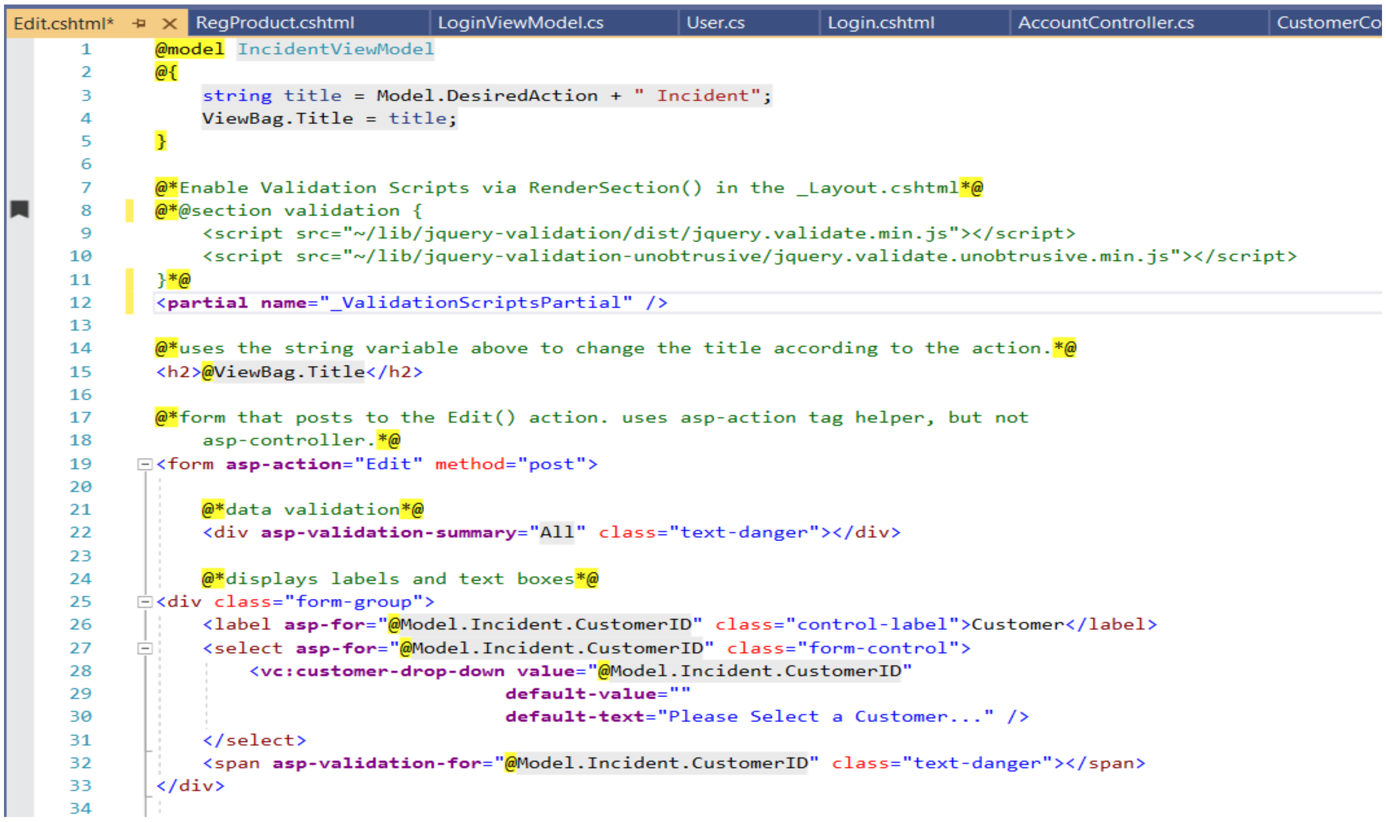
1o. *Use tag helpers, partial views, and view components:* Used tag helpers, partial views, and view components to reduce code duplication in Razor views and layouts, also to simplify controller and view model code.

1o1. Tag helper



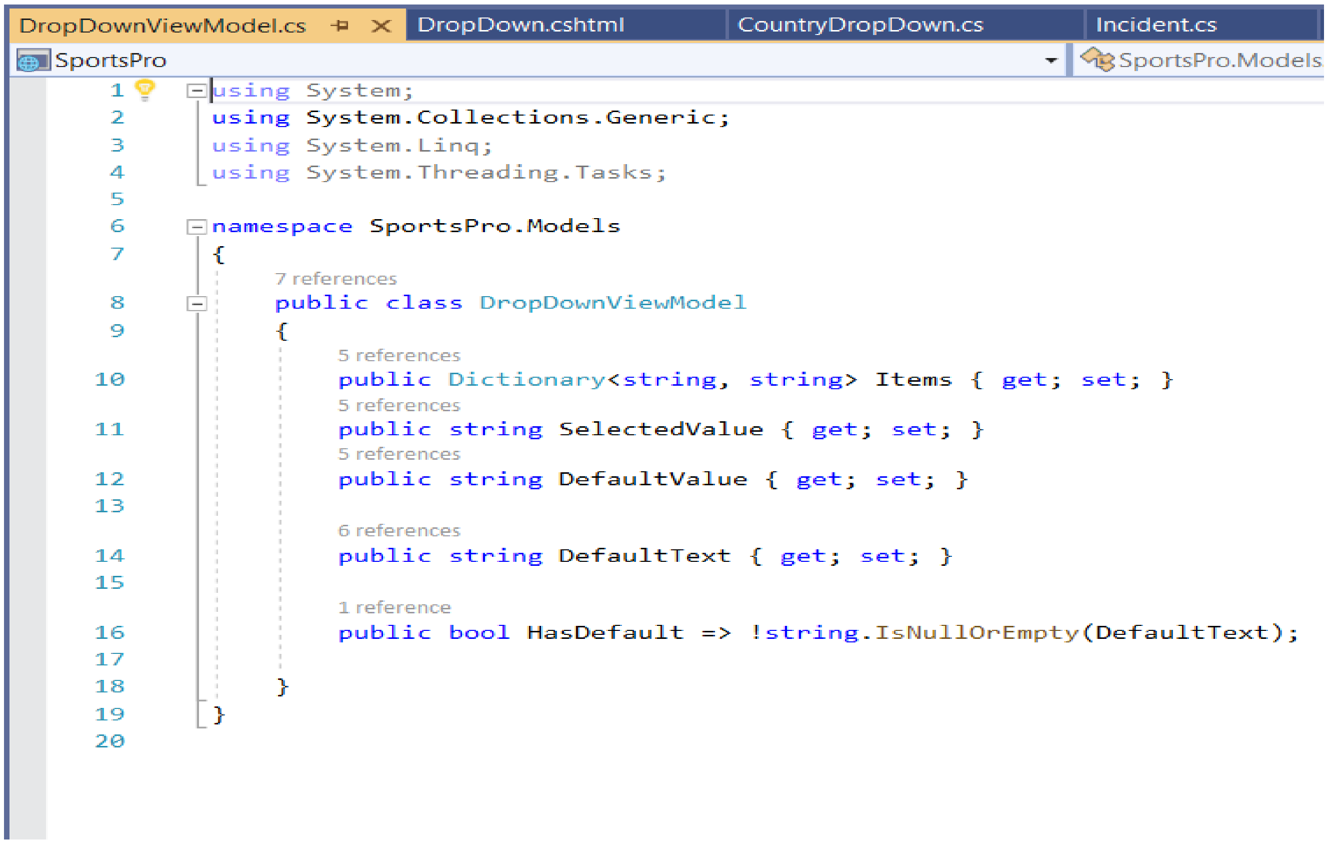
```csharp
using CommonMark;
using Microsoft.AspNetCore.Razor.Runtime.TagHelpers;
using Microsoft.AspNetCore.Razor.TagHelpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SportsPro
{
    // You may need to install the Microsoft.AspNetCore.Razor.Runtime package into your project
    [HtmlTargetElement("markdown")]
    2 references
    public class MarkdownTagHelper : TagHelper
    {
        1 reference
        public string Suffix { get; set; }

        0 references
        public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)
        {
            var childContent = await output.GetChildContentAsync();
            var lines = childContent.GetContent()
                .Split(new[] { Environment.NewLine }, StringSplitOptions.RemoveEmptyEntries)
                .Select(line => line.Trim());
            var content = string.Join(" ", lines);
            var transformedContent = CommonMarkConverter.Convert(content);

            output.Content.SetHtmlContent(transformedContent);
            output.PostContent.SetHtmlContent(Suffix);

            output.Attributes.RemoveAll("markdown");
        }
    }
}
```

1o2. Partial View



```
@model IncidentViewModel
@{
    string title = Model.DesiredAction + " Incident";
    ViewBag.Title = title;
}

@*Enable Validation Scripts via RenderSection() in the _Layout.cshtml*@
@*@section validation {
    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
}*@
<partial name="_ValidationScriptsPartial" />

@*uses the string variable above to change the title according to the action.*@
<h2>@ViewBag.Title</h2>

@*form that posts to the Edit() action. uses asp-action tag helper, but not
    asp-controller.*@
<form asp-action="Edit" method="post">

    @*data validation*@
    <div asp-validation-summary="All" class="text-danger"></div>

    @*displays labels and text boxes*@
<div class="form-group">
    <label asp-for="@Model.Incident.CustomerID" class="control-label">Customer</label>
    <select asp-for="@Model.Incident.CustomerID" class="form-control">
        <vc:customer-drop-down value="@Model.Incident.CustomerID"
                               default-value=""
                               default-text="Please Select a Customer..." />
    </select>
    <span asp-validation-for="@Model.Incident.CustomerID" class="text-danger"></span>
</div>

```

1o3. View Component



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

namespace SportsPro.Models
{
    7 references
    public class DropDownViewModel
    {
        5 references
        public Dictionary<string, string> Items { get; set; }
        5 references
        public string SelectedValue { get; set; }
        5 references
        public string DefaultValue { get; set; }

        6 references
        public string DefaultText { get; set; }

        1 reference
        public bool HasDefault => !string.IsNullOrEmpty(DefaultText);

    }
}

```

1p. *Authenticate and authorize users:* The Admin user functions will only be available to a user who has logged in as an Admin user.
Anonymous users can only view the Home page, the About page, the Register page, and the Login page. If an anonymous user attempts to access any other page, the app displays the Login page.
On startup, the website adds a user with a username of "admin" and a password of "P@ssw0rd" to the Admin role.

1q. *Use images and styling:* Used Bootstrap and consistent styling CSS to show professional website.

1r. *Home page:* This page has a link to Google website.

1s. *Home page:* A free account is used to host this website on Azure cloud services.
https://ceti-sportspro.azurewebsites.net/