

## **FINAL ASSIGNMENT COMPLETE SUBMISSION**

### **GITHUB PAGES LINK**

<https://adavis-cs.github.io/CapstoneProject/>

### **Professional Self-Assessment**

I have greatly enjoyed my career as a student in the SNHU Computer Science program. There was a lot to learn about theory, technology, and the application of programming principles in software development. Although my time in the program is coming to an end, and I will be happily graduating with my degree in Computer Science, I am aware that the professional field of technology is an ever-evolving space, and I will have to be proactive about learning new developments in the industry to keep my skills and coding practices sharp.

There was a lot that I have learned about the underlying Computer Science fundamentals in this program. Specifically, the role that Data Structures and Algorithms play in software development. I believe that a lot of the fundamentals are being abstracted away with new technologies that make a developer's life easy, however it is important to be aware of the fact that all of these new technologies are relying on the same fundamentals that got us to where we are here today with respect to all things Computer Science. So, having had the experience of learning about the fundamentals, I have attained a great appreciation of their importance and the role they play in the grand scheme of things.

With regards to the capstone course, I am appreciative of the fact that I was able to bring all my knowledge together and demonstrate it in the form of updating an existing application that I have worked on in the past. It was a great recap of the many things I have learned thus far, and their application to an actual scenario, which I will be further updating in the future.

I cannot wait to showcase my skills to my future employers and start my journey towards a career as a technology professional.

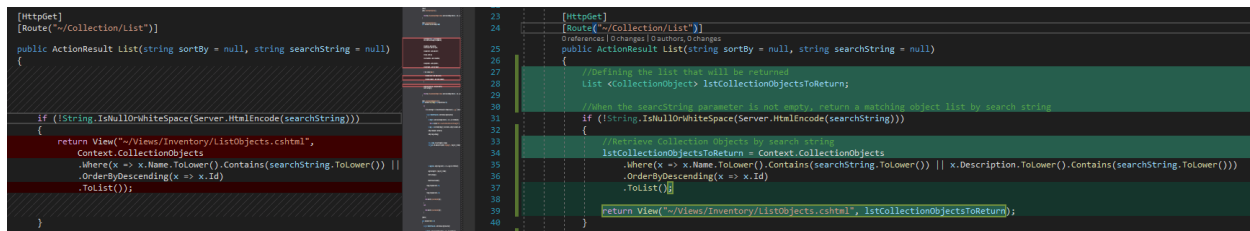
### **Milestone Two - Enhancement One: Software Design/Engineering**

The artifact I am including for this assignment is a code file that serves as the code-behind for the Controller Methods of the ASP.Net MVC web application project that is written in C#. This file contains methods for the various controller actions that are responsible for building the Model objects and returning them along with the Views to the user as a response to their HTTP request. This controller is responsible for the vast majority of the pages that the user will be interacting with, it was created as each individual View of the web application was created and required an associated controller method.

Alan Davis  
CS 499 Capstone  
8/14/21

This artifact was included because it showcases the majority of the logic that is used by the application, and it is a good example that showcases my ability to utilize various programming techniques to manipulate the behavior and output of the application based on user input. I have made several changes to the artifact that I have identified as beneficial to enhancing the code's readability and maintainability, exception handling, as well as security of the application.

Throughout the code, I added comments that enable both myself and future developers understand the functionality a code segment is attempting to achieve. I'll include snippets from a diff view in my IDE to demonstrate the changes I've made below:



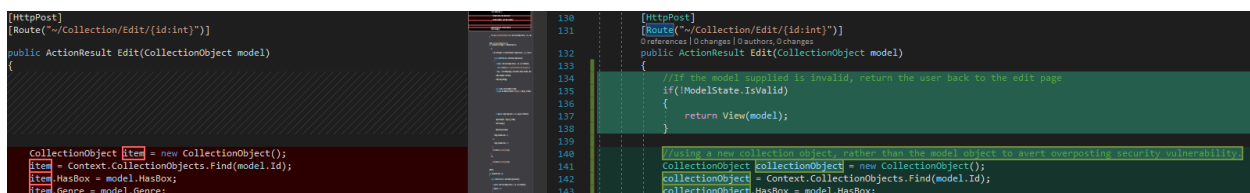
```
[HttpGet]
[Route("~/Collection/List")]
public ActionResult List(string sortBy = null, string searchString = null)
{
    if (!string.IsNullOrEmpty(Server.HtmlEncode(searchString)))
    {
        return View("~/Views/Inventory/ListObjects.cshtml",
            Context.CollectionObjects
                .Where(x => x.Name.ToLower().Contains(searchString.ToLower()) ||
                    .OrderByDescending(x => x.Id)
                    .ToList());
    }
}
```

```
[HttpGet]
[Route("~/Collection/List")]
[Authorize(Roles = "Admin, Manager")]
public ActionResult List(string sortBy = null, string searchString = null)
{
    //Defining the list that will be returned
    List <CollectionObject> listCollectionObjectsToReturn;
    //When the search string is null, we use the default object list by search string
    if (!string.IsNullOrEmpty(Server.HtmlEncode(searchString)))
    {
        //Retrieve Collection Objects by Name
        listCollectionObjectsToReturn = Context.CollectionObjects
            .Where(x => x.Name.ToLower().Contains(searchString.ToLower()) || x.Description.ToLower().Contains(searchString.ToLower()))
            .OrderByDescending(x => x.Id)
            .ToList();
    }
    return View("~/Views/Inventory/ListObjects.cshtml", listCollectionObjectsToReturn);
}
```

In addition to adding comments that help the reader trace the code, I have also decoupled the generation of the model object from the View method that is being called by the controller method. This allows for better readability, because the reader will know what type of object is being generated for the View model.

Further, I made changes to my switch statement. Where before I used to return the View that had a LINQ expression generating the model object by querying the database through Entity Framework. Now, my switch statement no longer returns the view after each case, but assigns the value of the defined model object in the beginning of the controller method, and the object is then separately passed to the View method that is called at the return statement of the controller. This mitigates user error and code duplication, by reducing the number of times that a returned View method has to be defined after each 'case' of a switch statement.

Another change that I have introduced to the artifact were more checks to determine whether the supplied model object is valid, that is, if it conforms to the model data definition, before the code continues any further.



```
[HttpPost]
[Route("~/Collection/Edit/{id:int}")]
public ActionResult Edit(CollectionObject model)
{
    //If the model supplied is invalid, return the user back to the edit page
    if (!ModelState.IsValid)
    {
        return View(model);
    }
}
```

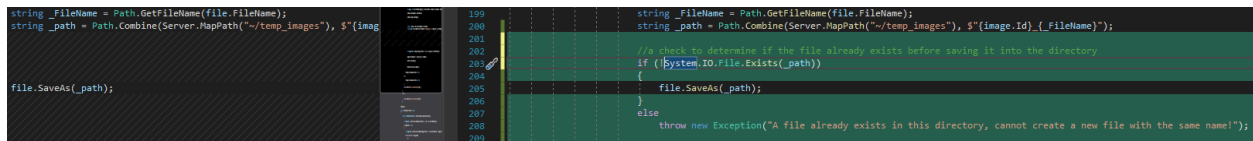
```
[HttpPost]
[Route("~/Collection/Edit/{id:int}")]
[Authorize(Roles = "Admin, Manager")]
public ActionResult Edit(CollectionObject model)
{
    //Using a new CollectionObject rather than the model object to avert overposting security vulnerability
    CollectionObject collectionObject = new CollectionObject();
    collectionObject = Context.CollectionObjects.Find(model.Id);
    collectionObject.HasBox = model.HasBox;
    collectionObject.HasGenre = model.HasGenre;
}
```

In the above snippet, you can see that I added logic to check whether the Model State of the supplied model object is valid. If it is not, then the user is returned to the original view that is making the request. Further, as I discussed in my code review, I now strive to follow better naming conventions, and therefore renamed the one of the objects from 'item' to 'collectionObject', which follows camel-case formatting, and enhances readability. Comments

Alan Davis  
CS 499 Capstone  
8/14/21

were also added here to make my intent clear, specifically as to why a separate object was declared of the same type as the model that was already passed into the method. In this case, it was to circumvent a vulnerability in this web-application known as ‘overposting’.

Yet another change I made to the artifact involves additional checks around file operations. Specifically, whether the file already exists within a directory, before my application attempts to create another file with the same name inside a folder – thereby causing an exception.

A screenshot of a code editor with a dark theme. The code is in C# and shows a method for saving a file. It includes a comment: `// a check to determine if the file already exists before saving it into the directory`. The code uses `System.IO.File.Exists(_path)` to check if the file exists. If it does, an exception is thrown: `throw new Exception("A file already exists in this directory, cannot create a new file with the same name!");`. The code is surrounded by a light blue border, and a yellow highlight is visible on the left side of the editor window.

```
string fileName = Path.GetFileName(file.FileName);
string _path = Path.Combine(Server.MapPath("~/temp_images"), $"{image.Id}_{fileName}");

file.SaveAs(_path);

199
200
201
202
203 // a check to determine if the file already exists before saving it into the directory
204 if (!System.IO.File.Exists(_path))
205 {
206     file.SaveAs(_path);
207 }
208 else
209     throw new Exception("A file already exists in this directory, cannot create a new file with the same name!");
```

Here you can see that I added an if clause, which checks whether a file with the specified path already exists. If it does not, then the file is created within the system. Otherwise, an exception is thrown and is caught and processed by the ‘catch’ block of the code segment further down in the method. I also added comments to make this apparent.

I believe that I have satisfied the course objectives defined in the earlier module. Part of writing good software is writing clear and concise code, which I think I am getting more proficient at, with better usage of comments and other programming practices that help both myself, and other developers who may look at this application down the line, quickly and seamlessly trace the code and understand my intent when I was originally writing this logic.

While making changes to this artifact, I really enjoyed reflecting on my coding practices before my experience in the relevant Computer Science courses, and how my coding style and perspective has changed since then. What this assignment is, essentially, is a refactoring exercise. And one challenge I have encountered is the importance of being judicious of my time and effort invested when reviewing code. Because, since I have picked up many new coding techniques over time, as I was reviewing my code, many times I was tempted to overhaul the method and re-write it differently. However, a refactor does not mean a complete re-write of the application, performing this exercise made me appreciate the challenge of identifying potentially problematic code and having to trace through blocks of ambiguous and undocumented code, because I had forgotten what the code was trying to accomplish. Subsequently adding clear comments where applicable will serve me and possibly other developers well into the future.

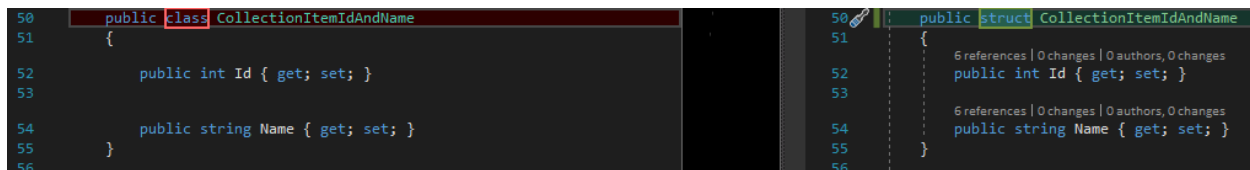
### **Milestone Three – Algorithms and Data Structures**

Module three afforded me an opportunity to revisit my implementation of Algorithms and Data Structures within the Web Application solution that I am using throughout this capstone course. The application is an ASP.Net Web Application that follows the MVC design pattern, coded in C# and utilizing Entity Framework as the ORM. I selected this application as my artifact, because in my initial creation of the application, I have not taken into consideration the importance of using correct types. So, although the performance implications from my changes in this module may be negligible for its current use case, if this application were to scale from

Alan Davis  
CS 499 Capstone  
8/14/21

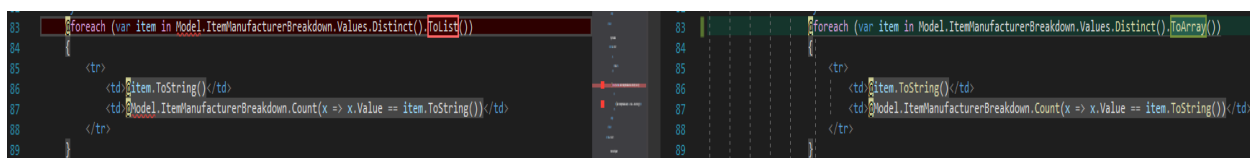
processing a couple thousand objects to many millions, the performance implications would become more apparent. This is because lighter data types are less costly than their heavier counterparts i.e. using structs instead of classes, when applicable. Likewise, using correct algorithm to search through an array of objects is also important, because making the correct assumptions in advance, will yield faster search times from less processing.

In this narrative I will be focusing on the changes I have made to a specific type in my application, namely the 'CollectionItemIdAndName' class type. I noticed that within the application, I have a class type defined that only serves to house two properties within it. Defining the type as a class, may have been a mistake, since this type is only used to store values such as an integer reference to the 'ID' of a collectible item (i.e. action figure) that the object relates to. This class type is then stored within a dictionary object within the application, which is then used as a model object in the 'Report' view that displays statistical information to the user about their inventory. Since, this report view is generated by making calculations and aggregations on objects contained within the dictionary, accessing properties within a class object inside the dictionary would be additional processing that would have to occur to follow the references from the heap where the class type stores its references, onto the stack, where object's values are actually stored. Thus, I have changed this object to a struct. In doing so, I am able to make the application more performant by accessing the values within the object's properties directly on the stack without first having to reference the heap. A screenshot of the diff of my changes is represented below. Again, while this might be a trivial change for an application that only processes several thousand records, it would yield noticeable performance dividends if it was dealing with a much larger volume of objects.



The screenshot displays a side-by-side comparison of code changes. On the left, the original code defines a 'public class CollectionItemIdAndName' with two properties: 'public int Id { get; set; }' and 'public string Name { get; set; }'. On the right, the modified code defines a 'public struct CollectionItemIdAndName' with the same two properties. The diff highlights the change from 'class' to 'struct' and the removal of the 'References' and 'Changes' metadata associated with the class.

Another modification that I have made in this artifact that relates to making it more performant by choosing the correct data structure, is I have opted to use an Array type in place of a Generic List type that is provided in the .Net framework, where appropriate. I noticed that there is a View within the application that iterates over a collection of objects that is implemented via a List type. The List type available in .Net affords the programmer select features to work with the objects contained within the list. However, in this instance, I see that the code only iterates through the objects in the List, without taking advantage of any additional features the list has to offer. So, in the interest of refactoring to optimize, I have opted to change the List type to a lighter 'Array' type in the code, over which the loop will iterate. In effect, this should execute the loop faster through an Array type rather than a List type.



The screenshot shows a code diff for a foreach loop. The left side shows the original code: 'foreach (var item in Model.ItemManufacturerBreakdown.Values.Distinct().ToList())'. The right side shows the modified code: 'foreach (var item in Model.ItemManufacturerBreakdown.Values.Distinct().ToArray())'. The diff highlights the change from 'ToList()' to 'ToArray()'.

Alan Davis  
CS 499 Capstone  
8/14/21

I am very thankful that I can now be cognizant of the trade-offs between my design choices, and my selection of data types for a given application. This is one of the outcomes that is defined within the Computer Science program that I have definitely realized. I can not think of any apparent gaps within the coursework and its applicability, however one thing I have noticed, is that the coursework focuses a lot on the fundamentals of computer science, which is definitely important, and less on different technologies such as the .Net framework. Different frameworks may obscure a lot of the underlying plumbing to programmers, such as using different Data Structures within its native type system. So, it makes it slightly challenging to see how the different frameworks are implementing the data structures within their types, and which one is the best to select for a particular use case. I am happy I was able to experience and grow as a programmer through my career as a student in this program.

### **Milestone Five – Databases**

In this module, I will discuss the architecture of my application, as it relates to its use of storing and accessing data in a database. I will then discuss the modifications I have made to the application, and the benefits my updates will yield to the performance and overall security of my software. The artifact that I have selected for this capstone is a Web Application that I have built using Microsoft's ASP.Net Framework, following the MVC model as the design pattern, and using C# as the programming language. I have selected this artifact, because I believe that web applications are an amalgam of many aspects of computer science and are a perfect solution to demonstrate how everything I have learned in my Computer Science program comes together.

The basic purpose behind my web application is to allow the user to store and retrieve information about the user's collectible toy inventory to simplify what used to be a very manual process for the user for whom I created it. Therefore, since this is a data-driven web application, a database back-end is necessary. For this, I have opted to implement an ORM solution, specifically Entity Framework, that points to a database powered by Microsoft's SQL Server, which are both also Microsoft technologies. I chose Entity Framework, because it plays well with other technologies in the Microsoft software development ecosystem, and because it allows me to take a 'Code-First' approach when architecting the application. Since I was the only person writing the application, being able to take a code-first approach saved me a lot of time, and I was able to focus more on writing code, and less on direct database management.

One modification that I will be making in this exercise, is I will define constraints for the data types within the database through my object models that are used by the ORM to create my database tables and columns. An important lesson I have learned through my courses here in this program, is that it is important to be cognizant about data types and their length, as it can have significant performance implications on the application. For instance, currently, my models that the ORM uses create the data tables, only take into consideration the object's native type in C#, and consequently for cases such as 'string' types, create a data type in the database that is typed as 'nvarchar' with the length of 'MAX'. The issue with using the 'MAX' length setting in this case may obstruct the use of indexers on the table, as well as potentially impact performance, if the table contains a lot of data. See screenshots below of the object model and respective table the ORM creates in the database.

Alan Davis  
CS 499 Capstone  
8/14/21

```
#region Buyer Information
[Display(Name = "Buyer's Name")]
2 references | 0 changes | 0 authors, 0 changes
public string BuyerName { get; set; }

[Display(Name = "Buyer's Address")]
[DataType(DataType.MultilineText)]
2 references | 0 changes | 0 authors, 0 changes
public string BuyerAddress { get; set; }

[Display(Name = "Date Shipped")]
[DataType(DataType.Date)]
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
5 references | 0 changes | 0 authors, 0 changes
public DateTime? DateShipped { get; set; }

[Display(Name = "Buyer Comments")]
[DataType(DataType.MultilineText)]
2 references | 0 changes | 0 authors, 0 changes
public string BuyerComments { get; set; }
#endregion
```

Name	Data Type	Allow Nulls	Default
BuyerName	nvarchar(MAX)	<input checked="" type="checkbox"/>	
BuyerAddress	nvarchar(MAX)	<input checked="" type="checkbox"/>	
DateShipped	datetime	<input checked="" type="checkbox"/>	
BuyerComments	nvarchar(MAX)	<input checked="" type="checkbox"/>	

To remedy this issue, I have explicitly added attributes to the properties of the object class that generates the table, to limit the character length of each column. This would be critical in cases where there are millions of records in the database, and the user often performs searches against text in the table. Please see changes demonstrated in the screenshot of the file diff below

```
81      #region Buyer Information
82
83      [Display(Name = "Buyer's Name")]
84      public string BuyerName { get; set; }
85
86      [Display(Name = "Buyer's Address")]
87      [DataType(DataType.MultilineText)]
88      public string BuyerAddress { get; set; }
89
90      [Display(Name = "Date Shipped")]
91      [DataType(DataType.Date)]
92      [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
93      public DateTime? DateShipped { get; set; }
94
95      [Display(Name = "Buyer Comments")]
96      [DataType(DataType.MultilineText)]
97      public string BuyerComments { get; set; }
98
99      #endregion
100
101
102
```

```
81      #region Buyer Information
82
83      [Display(Name = "Buyer's Name")]
84      [StringLength(30)]
85      public string BuyerName { get; set; }
86
87      [Display(Name = "Buyer's Address")]
88      [DataType(DataType.MultilineText)]
89      [StringLength(250)]
90      public string BuyerAddress { get; set; }
91
92      [Display(Name = "Date Shipped")]
93      [DataType(DataType.Date)]
94      [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
95      public DateTime? DateShipped { get; set; }
96
97      [Display(Name = "Buyer Comments")]
98      [DataType(DataType.MultilineText)]
99      [StringLength(3000)]
100      public string BuyerComments { get; set; }
101
102      #endregion
103
```

The updated table definition is demonstrated in the screenshot below:

Name	Data Type	Allow Nulls	Default
BuyerName	nvarchar(30)	<input checked="" type="checkbox"/>	
BuyerAddress	nvarchar(250)	<input checked="" type="checkbox"/>	
DateShipped	datetime	<input checked="" type="checkbox"/>	
BuyerComments	nvarchar(3000)	<input checked="" type="checkbox"/>	

Alan Davis  
CS 499 Capstone  
8/14/21

Overall, I believe my changes will make the application more scalable and performant if there is ever a need for it to support databases with millions of records, and conserve computing resources on the DB server.

With this artifact, I was able to demonstrate my awareness of good database design principles and adhere to these practices, even through a code-first approach to software development. One challenge I faced is figuring out how to influence the data types in my table through the use of an ORM. Although it was a challenge, I was able to overcome it with some research and persistence. Through my time as a student in the Computer Science program, I was able to develop more resourcefulness when it comes to finding solutions for complex technical problems. I am glad I was able to apply my new skills experientially in this capstone.