

Introduction

In the previous assignment, you implemented some of the common interaction patterns on standalone entities. In this assignment, you will build on your knowledge of LINQ by implementing various “display-only” use cases with Track and Invoice objects.

This assignment is worth **9%** of your final course grade. If you wish to submit the assignment before the due date, you can do that. If submitted after the due date, **you will receive a 10% deduction for every 24-hour period the assignment is late**. An assignment handed in **more than five days** late will receive a **mark of zero**.

Objective(s)

You will create several actions that will work with **Track** objects. You will gain experience working with associated data and sorting and filtering the tracks using LINQ.

You will work with and display associated data for **Invoice** objects. Your app will enable users to view a list of invoices and display details about each invoice.

Specifications overview and work plan

Here is a brief list of specifications that you must implement:

- Follow best practices
- Implement the recommended system design guidance
- Customize the appearance of your web app
- Create view models that will implement the “get all” use case for **Track** and **Invoice** objects and the “get one” use case for **Invoice** objects.
- Create the Controller and Manager classes that will work together for data service operations.
- Create Views that will interact with the user. Try to recreate the views shown in the assignment. Remember to remove broken links and clean-up headings.

Getting started

Using the “**Web App Project Template V1**” project template provided by your professor, create a new web app and name it as follows: **your initials + “2247A2”**. For example, your professor would call the web app “**KY2247A2**”. The project template is available on Blackboard.

Build and run the web app immediately after creating the solution to ensure that you are starting with a working, errorfree, base. As you write code, you should build frequently. If your project has a compilation error and you are unsure how to fix it:

- Search the internet for a solution. Sites like Stack Overflow contain a plethora of solutions to many of the common problems you may experience.
- Post on the MS Teams “Course Help” channel.
- Email your professor and include error message details and screenshots when possible.
- Keep in mind the academic integrity policies when getting help.

View your web app in a browser by pressing F5 or using the menus (“Debug” > “Start”).

Update the project code libraries (NuGet packages)

The web app includes several external libraries. It is a good habit to update these libraries to their latest stable versions. You must update the NuGet packages before you begin working on this assignment. You can refer to the instructions in Assignment 1 for more information on this topic. Do not update AutoMapper.

Customize the app’s appearance

You will customize the appearance of all your web apps and assignments. **Never submit an assignment that has the generic auto-generated text content.** Please make the time to customize the web app’s appearance.

For this assignment, you can defer this customization work until later. Come back to it at any time and complete it before you submit your work.

Follow the guidance from Assignment 1 to customize the app’s appearance. Please feel free to delete or comment out the About and Contact pages. If you delete these pages, remember to remove the links from the header.

Track “get all” use cases

Create the TrackBaseViewModel and TrackWithDetailViewModel

Create an appropriate view model for displaying **Track** objects. The **Track** data looks something like this:

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
2	Balls to the Wall	2	2	1	NULL	342562	5510424	0.99
3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirksneider & W. Hoffm...	230619	3990994	0.99
4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscn...	252051	4331779	0.99
5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99
6	Put The Finger On You	1	1	1	Angus Young, Malcolm Young, Brian Johnson	205662	6713451	0.99
7	Let's Get It Up	1	1	1	Angus Young, Malcolm Young, Brian Johnson	233926	7636561	0.99
8	Inject The Venom	1	1	1	Angus Young, Malcolm Young, Brian Johnson	210834	6852860	0.99
9	Snowballed	1	1	1	Angus Young, Malcolm Young, Brian Johnson	203102	6599424	0.99
10	Evil Walks	1	1	1	Angus Young, Malcolm Young, Brian Johnson	263497	8611245	0.99
11	C.O.D.	1	1	1	Angus Young, Malcolm Young, Brian Johnson	199836	6566314	0.99
12	Breaking The Rules	1	1	1	Angus Young, Malcolm Young, Brian Johnson	263288	8596840	0.99
13	Night Of The Long Knives	1	1	1	Angus Young, Malcolm Young, Brian Johnson	205688	6706347	0.99
14	Spellbound	1	1	1	Angus Young, Malcolm Young, Brian Johnson	270863	8817038	0.99
15	Go Down	4	1	1	AC/DC	331180	10847611	0.99
16	Dog Eat Dog	4	1	1	AC/DC	215196	7032162	0.99
17	Let There Be Rock	4	1	1	AC/DC	366654	12021261	0.99
18	Bad Boy Boogie	4	1	1	AC/DC	267728	8776140	0.99
19	Problem Child	4	1	1	AC/DC	325041	10617116	0.99
20	Overdose	4	1	1	AC/DC	369319	12066294	0.99
21	Hell Ain't A Bad Place To Be	4	1	1	AC/DC	254380	8331286	0.99
22	Whole Lotta Rosie	4	1	1	AC/DC	323761	10547154	0.99
23	Walk On Water	5	1	1	Steven Tyler, Joe Perry, Jack Blades, Tommy Chan...	265680	6710570	0.99

As you copy properties from the **Track** design model class to your view model class, you will notice that there is a property decorated with the **Column** attribute. The **Column** attribute cannot be used in a view model class so do not include it. You do not need the **AlbumId**, **GenreId** or **MediaTypeId** properties in your view model class.

Later, you will need to show the album title and genre name along with the track info. You will use the **AutoMapper** flattening feature for this. Create a new view model to inherit from the base view model and introduce the appropriate properties.

All view models require suitable data annotations. Study each property and determine which data annotations should be added and which should be excluded. Please refer to the lecture notes for help. Do not forget to include a **Display** attribute on the necessary properties to make the scaffolded views look nicer.

Add methods to the Manager class to handle the use cases

The class notes and code examples have all the information you will need to implement this part of the work plan.

For the “get all” use cases on the **Track** entity, several methods will be needed, and each will use LINQ query expressions to filter and sort the results. Each method will return an **IEnumerable<TrackWithDetailViewModel>**.

TrackGetAll

Typical “get all” method sorted in ascending order by track **Name**.

TrackGetBluesJazz

Filter where **GenreId** is 2 or 6 and sort in ascending order by genre **Name** then by track **Name**.

TrackGetCantrellStaley

Filter where the **Composer** contains the string “Jerry Cantrell” and contains the string “Layne Staley”. Sort the tracks in ascending order by **Composer** then by track **Name**.

TrackGetTop50Longest

Display the 50 longest tracks based on the **Milliseconds** property. Sort the tracks in ascending order by track **Name**.

TrackGetTop50Smallest

Display the 50 smallest tracks based on the **Bytes** property. Sort the tracks in ascending order by track **Name**.

Note: For the last two methods, you can use the Take() method to limit the results to 50 items only.

Add mappings to the Manager class to handle the use cases

Define the **AutoMapper** mappings that each use case will need. At this point in time, you should have enough experience to know which maps are required.

Create a new Controller

Now create a new controller called **TracksController** using the scaffolding feature as previously seen. The **TracksController** class will have a method named **Index()**. It will call the **TracksGetAll()** method in the **Manager** and return each track in the typical manner.

Think carefully about the other method names because they will become part of the URL. Do not include the “TrackGet” part in each action name.

Also, keep in mind that:

- Each action method will call a method in the **Manager** object.
- Each action method will return the same **Index** view and pass it the fetched collection. This means you only need to scaffold a single view for all the actions in the controller.
- You should remove any unused actions from the controller since actions such as **Create**, **Details**, and **Edit** will just cause unnecessary bloat.

Creating the Index.cshtml view

All track lists will be hosted in a single view, the **Index** view. Create the view using the scaffolding feature. The actual tracks that are displayed can be controlled by calling different actions in the controller.

You will add some links – as **ActionLink** HTML Helpers – near the top of the **Index** view. Each link will call the appropriate action/method in the **TracksController**. Make sure that you include a fifth link, which will call the **Index** action to show all tracks.

You will also need to use the name of the method (action) to show an appropriate heading. Add a code block to the top of the HTML.

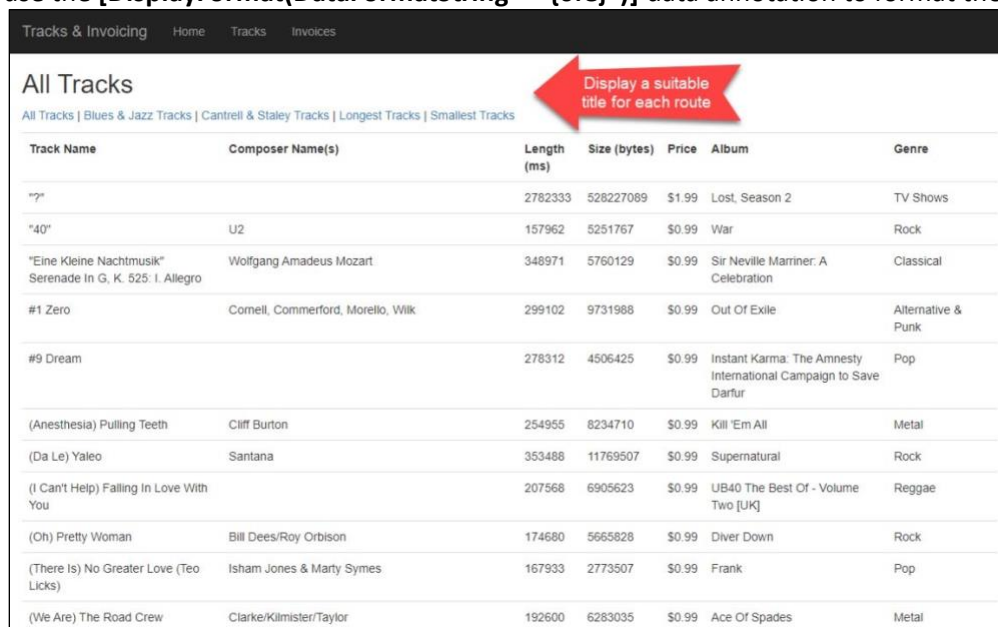
```
@{
    string actionName =
    ViewContext.RouteData.GetRequiredString("action");
    string title
    = "All Tracks";
    switch(actionName)
    {
    case "Top50Smallest":
        title = "Smallest
        Tracks";
        break;
    }

    ViewBag.Title = title;
}
```

Edit the <h2> header element near the top so that it looks like this:

```
<h2>@title</h2>
```

The page will look like the following screenshot. Notice the column names and that the price has been formatted correctly. You can use the `[DisplayFormat(DataFormatString = "{0:C}")]` data annotation to format the price.



Track Name	Composer Name(s)	Length (ms)	Size (bytes)	Price	Album	Genre
"7"		2782333	528227089	\$1.99	Lost, Season 2	TV Shows
"40"	U2	157962	5251767	\$0.99	War	Rock
"Eine Kleine Nachtmusik"	Wolfgang Amadeus Mozart	348971	5760129	\$0.99	Sir Neville Marriner: A Celebration	Classical
#1 Zero	Cornell, Commerford, Morello, Wilk	299102	9731988	\$0.99	Out Of Exile	Alternative & Punk
#9 Dream		278312	4506425	\$0.99	Instant Karma: The Amnesty International Campaign to Save Darfur	Pop
(Anesthesia) Pulling Teeth	Cliff Burton	254955	8234710	\$0.99	Kill 'Em All	Metal
(Da Le) Yaleo	Santana	353488	11769507	\$0.99	Supernatural	Rock
(I Can't Help) Falling In Love With You		207568	6905623	\$0.99	UB40 The Best Of - Volume Two [UK]	Reggae
(Oh) Pretty Woman	Bill Dees/Roy Orbison	174680	5665828	\$0.99	Diver Down	Rock
(There Is) No Greater Love (Teo Licks)	Isham Jones & Marty Symes	167933	2773507	\$0.99	Frank	Pop
(We Are) The Road Crew	Clarke/Kilmister/Taylor	192600	6283035	\$0.99	Ace Of Spades	Metal

Invoice “get all” and “get one” use cases

Create view models

We will be working with the **Invoice** entity and some of its associated entities. The following use cases will need view models:

- Invoice – “get all”
- Invoice – “get one”

Go ahead and write the base view model class. Remember to add the [Key] data annotation to your view model classes.

Add data from Customer and Employee entities

Do we need “customer” view model classes? No, you can use composed property names along with the AutoMapper’s “flattening” feature. In a class, a composed property name is a string concatenation of the property names from the navigation property in the current class to the property in the destination class. On the way to the destination, you can pass through other navigation properties.

Create an **InvoiceWithDetailViewModel** class that will inherit from **InvoiceBaseViewModel**. Add some composed property names from the **Customer** and **Employee** design model classes.

For example, the **Invoice** class has a **Customer** navigation property. If we want to include the **Customer** class **FirstName** property, the composed property name will be **CustomerFirstName**.

In addition, the **Customer** class has an **Employee** navigation property. If we want the **Employee** class **FirstName** property, the composed property name will be **CustomerEmployeeFirstName**.

For now, we need:

- The customer’s first name, last name, state, and country
- The employee’s first name and last name (of the customer’s sales rep).

Add methods and mappings to the Manager class

In the Manager class, add the methods that support the use cases. Don't forget to configure **AutoMapper** mappings.

- **InvoiceGetAll()** – Return a collection of **InvoiceBaseViewModel** classes sorted by **InvoiceDate** in descending order.
- **InvoiceGetByIdWithDetail()** – Return an **InvoiceWithDetailViewModel** for the **Invoice** with the specified **ID**.

The **InvoiceGetByIdWithDetail()** method must use the **Include()** method for the **Customer** and **Employee** properties.

How can you get employee information? The problem is that **Invoice** does NOT have a direct association with **Employee**. Instead, the path is **Invoice > Customer > Employee**.

Can we easily get employee information? Yes, as you may recall reading the MSDN documentation for the [Include\(\)](#) method, the “path” parameter is a “dot-separated list of related objects to return in the query results”. So, perhaps something like “Customer.Employee” would work.

Add a controller for invoices

Create an **InvoicesController** for the **Invoice** entity. Go ahead and code the **Index** and **Detail** actions. Remove the unused actions from the controller.

Create the “get all” view

The “get all” use case will show the default view when working with **Invoice** objects. Your view should look like this. Notice the following:

- Each invoice will have a “Details” link at the right side. Other links have been removed since they are unused.
- The column names and order.
- The invoice total has been formatted as a currency using the **[DisplayFormat]** attribute (like the track price from earlier).
- The invoice date has been formatted as a date “MMM d, yyyy” using the **[DisplayFormat]** attribute.

Tracks & Invoicing							
Home Tracks Invoices							
Invoices							
Customer	Billing Address	City	State	Country	Postal/Zip	Date	Total
58	12,Community Centre	Delhi		India	110017	Dec 22, 2013	\$1.99
44	Porthankatu 9	Helsinki		Finland	00530	Dec 14, 2013	\$13.86
35	Rua dos Campeões Europeus de Viena, 4350	Porto		Portugal		Dec 9, 2013	\$8.91
29	796 Dundas Street West	Toronto	ON	Canada	M6J 1V1	Dec 6, 2013	\$5.94
25	319 N. Frances Street	Madison	WI	USA	53703	Dec 5, 2013	\$3.96
21	801 W 4th Street	Reno	NV	USA	89503	Dec 4, 2013	\$1.98
23	69 Salem Street	Boston	MA	USA	2113	Dec 4, 2013	\$1.98

Create the “get one” view

The “get one” use case will show the default view for an invoice object. Here is an example screen capture.

Tracks & Invoicing Home Tracks Invoices

Details

InvoiceWithDetailViewModel

Customer	27
Date	Oct 13, 2013
Billing Address	1033 N Park Ave
City	Tucson
State	AZ
Country	USA
Postal/Zip	85719
Total	\$13.86
CustomerFirstName	Patrick
CustomerLastName	Gray
CustomerState	AZ
CustomerCountry	USA
CustomerEmployeeL...	Park
CustomerEmployeeFi...	Margaret

Notice the scaffolder included properties from the Customer and Customer.Employee objects.

[Edit](#) | [Back to List](#)

Hand-edit the view to improve its appearance:

- Group the invoice-specific info together (and display the invoice identifier). It is suggested that you create a new <dl> element to hold the invoice-specific information.
- Group the customer info together. Get rid of those atomic customer-related elements and create new concatenated strings

At the top of every view, there is a Razor code expression block (which starts with `@{...}`). You may add code to this block to declare and prepare strings that can be used in the view.

Tracks & Invoicing Home Tracks Invoices

Details for Invoice #397

Customer Info	27 Patrick Gray AZ, USA
Billing Address	1033 N Park Ave Tucson, AZ USA, 85719
Sales Rep	Margaret Park
Date	Oct 13, 2013
Total	\$13.86

[Back to List](#)

Add data from InvoiceLine

Add invoice line-item detail to the page. First, we will need a “base” view model class for **InvoiceLine**. Do not copy the **InvoiceId** or **TrackId** properties, they are not needed.

Add a new property to the view model. This property will contain a getter only:

```
public decimal LinePrice
{
    get
    {
        return Quantity * UnitPrice;
    }
}
```

Next, modify the **InvoiceWithDetailViewModel** class by adding a collection *navigation property* for the **InvoiceLineBaseViewModel** items. Make sure the name of this navigation property matches the design model class.

Now, modify the manager **InvoiceGetByIdWithDetail()** method and **Include()** the **InvoiceLines**. You may also need to add a new **AutoMapper** mapping. The controller method is good as is.

In the view code, manually add HTML and code expressions to build a table below the existing information. Add the classes “table” and “table-striped” to the <table>, these are defined by the Bootstrap framework.

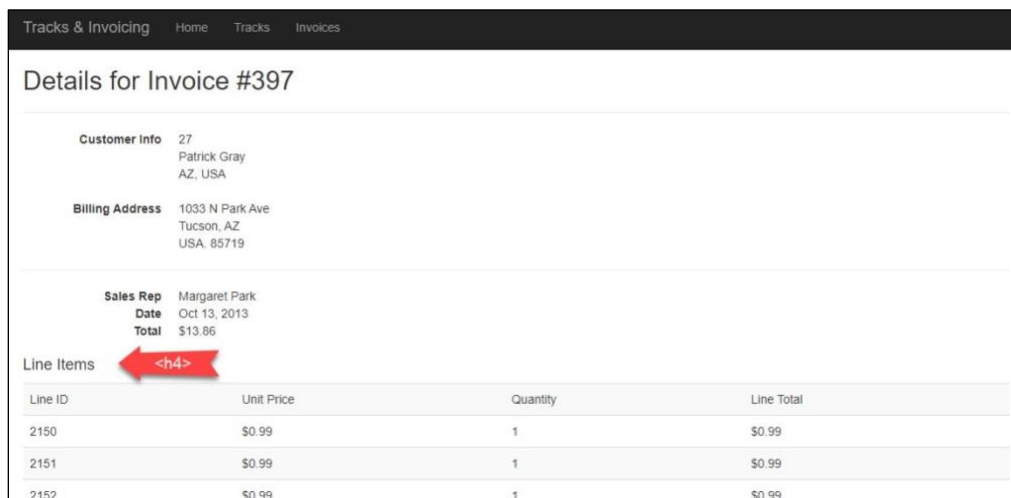
Continue to use HTML helpers in the table. For example, you can render a column title using code like:

```
@Html.DisplayNameFor(model => model.InvoiceLines.First().InvoiceLineId)
```

You can render a table cell using:

```
@Html.DisplayFor(model => line.InvoiceLineId)
```

You will end up with something like this:



Line ID	Unit Price	Quantity	Line Total
2150	\$0.99	1	\$0.99
2151	\$0.99	1	\$0.99
2152	\$0.99	1	\$0.99

Add data from Track (via InvoiceLine)

While the invoice line items are technically correct, they are not very useful. Add information about the track that is readable and understandable. For example, the track name, and other detail.

A strategy like what we did for **Customer** and **Employee** (above) is recommended. We will take advantage of the AutoMapper “flattening” feature.

Create an **InvoiceLineWithDetailViewModel** class that inherits from **InvoiceLineBaseViewModel**. Add some composed property names from the **Track** design model class. At minimum, we want the track the **Name** property and **Composer** properties.

Add a mapper, from **InvoiceLine** to **InvoiceLineWithDetailViewModel**. Modify the existing manager method so that it includes the path **InvoiceLines > Track**.

Next, go back to the invoice view model classes. The **InvoiceWithDetailViewModel** class has a navigation property that holds the collection of invoice lines. What is its data type? **InvoiceLineBaseViewModel**. We must change the data type to the new view model class that you just created (**InvoiceLineWithDetailViewModel**).

Finally, edit the view code. At this point, you will probably have something that looks like the following:

Tracks & Invoicing				
Home Tracks Invoices				
Details for Invoice #397				
<div> <div>Customer Info</div> <div>27</div> <div>Patrick Gray</div> <div>AZ, USA</div> </div> <div> <div>Billing Address</div> <div>1033 N Park Ave</div> <div>Tucson, AZ</div> <div>USA, 85719</div> </div> <div> <div>Sales Rep</div> <div>Margaret Park</div> <div>Date</div> <div>Oct 13, 2013</div> <div>Total</div> <div>\$13.86</div> </div>				
Line Items				
Line ID	Track	Unit Price	Quantity	Line Total
2150	<div>Name: Black Moon Creeping</div> <div>Composer(s): Chris Robinson/Rich Robinson</div>	\$0.99	1	\$0.99
2151	<div>Name: White Riot</div> <div>Composer(s): Joe Strummer/Mick Jones</div>	\$0.99	1	\$0.99
2152	<div>Name: Train In Vain</div> <div>Composer(s): Joe Strummer/Mick Jones</div>	\$0.99	1	\$0.99

Add data from Album and Artist

Now you will add data from **Album** (via **Track** and **InvoiceLine**) and data from **Artist** (via **Album**, **Track** and **InvoiceLine**). Hopefully, you see a pattern for working with associated data and you will find that adding data from the **Album** and **Artist** objects is surprisingly easy.

Track has a “to-one” association with **Album**. To work with the **Album.Title** property:

1. Add a composed property name to the **InvoiceLineWithDetailViewModel** class.
2. Modify the existing manager method to include the path to the **Album** object.
3. Edit the view code.

Album has a “to-one” association with **Artist**. To work with the **Artist.Name** property:

1. Add a composed property name to the **InvoiceLineWithDetailViewModel** class.
2. Modify the existing manager method to include the path to the **Artist** object.
3. Edit the view code.

Track has a “to-one (or zero)” association with **Genre**. To work with the **Genre.Name** property:

1. Add a composed property name to the **InvoiceLineWithDetailViewModel** class.
2. Modify the existing manager method to include the path to the **Genre** object.
3. Edit the view code.

Add data from MediaType (via Track and InvoiceLine)

Previously when working with **Album** and **Artist** data, we had a straight-line path from the **InvoiceLine** object:

InvoiceLine > Track > Album > Artist

Now we need a piece of data from the **MediaType** object. It has a slightly different path:

InvoiceLine > Track > MediaType

To add data from **MediaType.Name** property, we still follow the same strategy as we did with **Album** and **Artist** but we will need a third **Include()** method to get to the **MediaType** object.

If you did this correctly, you would probably have something that looks like the following screenshot: *(see screenshot on the next page)*

Tracks & Invoicing

Home

Tracks

Invoices

Details for Invoice #397

Customer Info

27

Patrick Gray

AZ, USA

Billing Address

1033 N Park Ave

Tucson, AZ

USA, 85719

Sales Rep

Margaret Park

Date

Oct 13, 2013

Total

\$13.86

Line Items

Line ID	Track	Unit Price	Quantity	Line Total
2150	<div>Name:</div> <div>Black Moon Creeping</div> <div>Album:</div> <div>Live [Disc 2]</div> <div>Artist:</div> <div>The Black Crowes</div> <div>Composer(s):</div> <div>Chris Robinson/Rich Robinson</div> <div>Genre:</div> <div>Blues</div> <div>Media Type:</div> <div>MPEG audio file</div>	\$0.99	1	\$0.99
2151	<div>Name:</div> <div>White Riot</div> <div>Album:</div> <div>The Singles</div> <div>Artist:</div> <div>The Clash</div> <div>Composer(s):</div> <div>Joe Strummer/Mick Jones</div> <div>Genre:</div> <div>Alternative & Punk</div> <div>Media Type:</div> <div>MPEG audio file</div>	\$0.99	1	\$0.99

Testing your work

Test your work by doing tasks that fulfill the use cases in the specifications.

Reminder about academic integrity

Most of the materials posted in this course are protected by copyright. It is a violation of Canada's Copyright Act and [Seneca's Copyright Policy](#) to share, post, and/or upload course material in part or in whole without the permission of the copyright owner. This includes posting materials to third-party file-sharing sites such as assignment-sharing or homework help sites. Course material includes teaching material, assignment questions, tests, and presentations created by faculty, other members of the Seneca community, or other copyright owners.

It is also prohibited to reproduce or post to a third-party commercial website work that is either your own work or the work of someone else, including (but not limited to) assignments, tests, exams, group work projects, etc. This explicit or implied intent to help others may constitute a violation of [Seneca's Academic Integrity Policy](#) and potentially involve such violations as cheating, plagiarism, contract cheating, etc.

These prohibitions remain in effect both during a student's enrollment at the college as well as withdrawal or graduation from Seneca.

This assignment must be worked on individually and you must submit your own work. You are responsible to ensure that your solution, or any part of it, is not duplicated by another student. If you choose to push your source code to a source control repository, such as GIT, ensure that you have made that repository private and have not added collaborators.

A suspected violation will be filed with the Academic Integrity Committee and may result in a grade of zero on this assignment or a failing grade in this course.

Submitting your work

Make sure you submit your assignment before the due date and time. It will take a few minutes to package up your project so make sure you give yourself a bit of time to submit the assignment.

The solution folder contains extra items that will make submission larger. The following steps will help you “clean up” unnecessary files. Please make sure you clean up the project before submitting to prevent issues with Blackboard.

1. Locate the folder that holds your solution files. You can jump to the folder using the Solution Explorer. Rightclick the “Solution” item and choose “Open Folder in File Explorer”.
2. Go up one level and you will see your solution folder (similar to **KY2247A2** but using your initials). Make a copy of your solution and change into the folder where you copied the files. For the remainder of the steps, you should be working in your copied solution!
3. Delete the “packages” folder and all its contents.
4. In the project folder (should be called **KY2247A2** but using your initials) contained within the solution folder, delete the “bin” and “obj” folders.
5. Compress the copied folder into a **zip** file. **Do not use 7z, RAR, or other compression algorithms (otherwise your assignment will not be marked)**. The zip file should not exceed a couple of megabytes in size. If the zip file is larger than a couple of megabytes, do not submit the assignment! Please ensure you have completed all the steps correctly.
6. Login to <https://learn.senecacollege.ca/>.
7. Open the “Web Programming Using ASP.NET” course area and click the “Assignments” link. Follow the link for this assignment.
8. Submit/upload your zip file. The page will accept unlimited submissions so you may re-upload the project if you need to make changes but make sure you make all your changes before the due date. Only the last submission will be marked.
9. **It is highly recommended you download the submitted assignment and test it on your computer.**

== END ==