Introduction

In the previous assignment, you implemented some of the common interaction patterns on standalone entities. In this assignment, you will continue working with standalone entities as you build your knowledge of LINQ. Afterward, you will work with associated data for display-only. It is recommended you read the entire assignment before you get started.

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 50% deduction. All assignments must be successfully completed and submitted to pass the course.

Objective(s)

You will create several actions that will work with **Track** objects. You will gain experience 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” and “get one” use cases for the **Track** and **Invoice** objects.
- Create the Controller and Manager classes that will work together for data service operations.
- Create Views that will interact with the user.

Getting started

Using the “**Web App Project Template S2022 V1**” project template provided by your professor, create a new web app and name it as follows: “**S2022A2**” + your initials. For example, your professor would call the web app “S2022A2*NKR*”. 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, error-free, 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 “Assignment Help” channel.
- Email your professor and include error message details and screenshots when possible.

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.

Customize the app’s appearance

You will customize the appearance of 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, including the link in the page header.

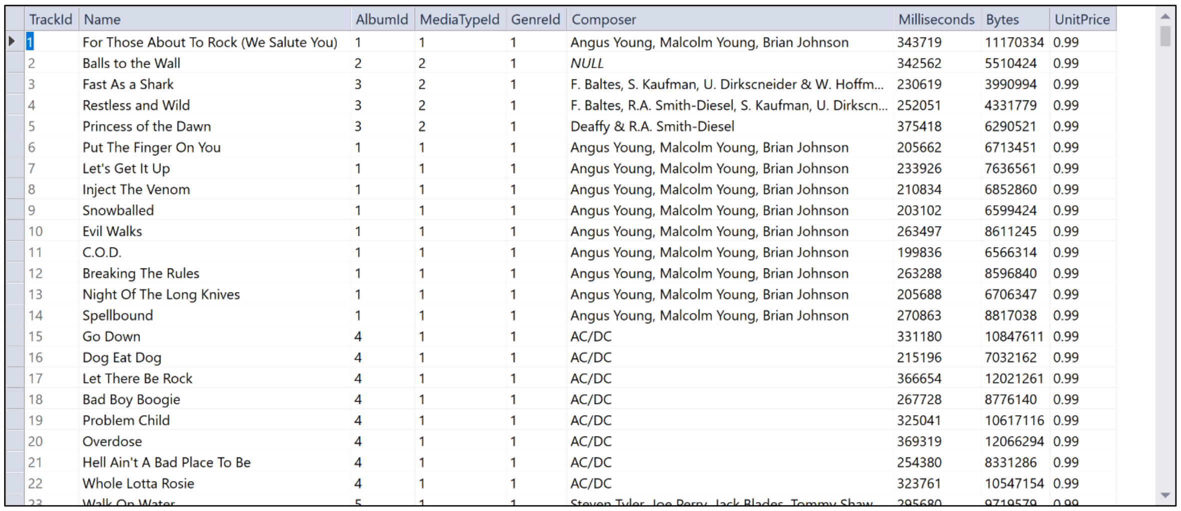
Please feel free to delete or comment out the About and Contact pages. Please also remove the link from the header.

You may also delete or comment out the three columns on the home index page. DO NOT remove the <h1> and “lead” paragraph. You must fill this in with the appropriate information.

Track “get all” use cases

Create the Track base view model

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. Dirkscneider & 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 Shaw	295680	9719579	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.

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. As you would expect, each method will return an **IEnumerable<TrackBaseViewModel>**.

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

TrackGetAllRockMetal – Filter where **GenreId** is 1 or 3 and sort in ascending order by **Name** then by **Composer**.

TrackGetAllTylerVallance – Filter where the **Composer** contains the string “Steven Tyler” **and** contains the string “Jim Vallance”. Sort the tracks in ascending order by **Composer** then by track **Name**.

TrackGetAllTop50Longest – Sort in descending order by **Milliseconds**; use the Take() method to limit the results to 50 items only.

TrackGetAllTop50Smallest – Sort in ascending order by **Bytes**; limit 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 "All Tracks" "get all" method 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 word "get" in each method name.

Each action method will call a method in the **Manager** object. Each action method will return the same **Index** view and pass on the fetched collection.

Please remember to remove any unused actions from the controller. 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, for example:

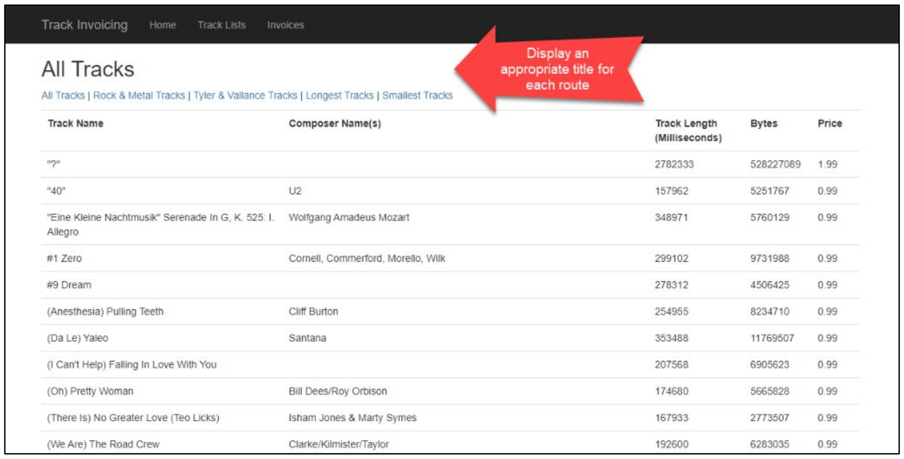
```
@{
     string actionName = ViewContext.RouteData.GetRequiredString("action");
     string title;

     switch(actionName) {
          case "Index":
               title = "All Tracks";
               break;
     }
}
```

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:



All Tracks

All Tracks | Rock & Metal Tracks | Tyler & Vallance Tracks | Longest Tracks | Smallest Tracks

Track Name	Composer Name(s)	Track Length (Milliseconds)	Bytes	Price
"?"		2782333	528227089	1.99
"40"	U2	157962	5251767	0.99
"Eine Kleine Nachtmusik" Serenade In G, K. 525: I. Allegro	Wolfgang Amadeus Mozart	348971	5760129	0.99
#1 Zero	Cornell, Commerford, Morello, Wilk	299102	9731988	0.99
#9 Dream		278312	4506425	0.99
(Anesthesia) Pulling Teeth	Cliff Burton	254955	8234710	0.99
(Da Le) Yaleo	Santana	353488	11769507	0.99
(I Can't Help) Falling In Love With You		207568	6905623	0.99
(Oh) Pretty Woman	Bill Dees/Roy Orbison	174680	5665828	0.99
(There Is) No Greater Love (Teo Licks)	Isham Jones & Marty Symes	167933	2773507	0.99
(We Are) The Road Crew	Clarke/Kilmister/Taylor	192600	6283035	0.99

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

Create view models and mappers that cover the use cases

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 those view model classes now. Do not worry about the associated entities yet, you should focus on the standalone “get all” and “get one” use cases first. You should be comfortable with this task as you, but if not, please review the previous milestones.

Remember to add the [Key] data annotation to your view model classes.

Once you have created your view models, define the maps for each use case.

Add methods to the Manager class that handle the use cases

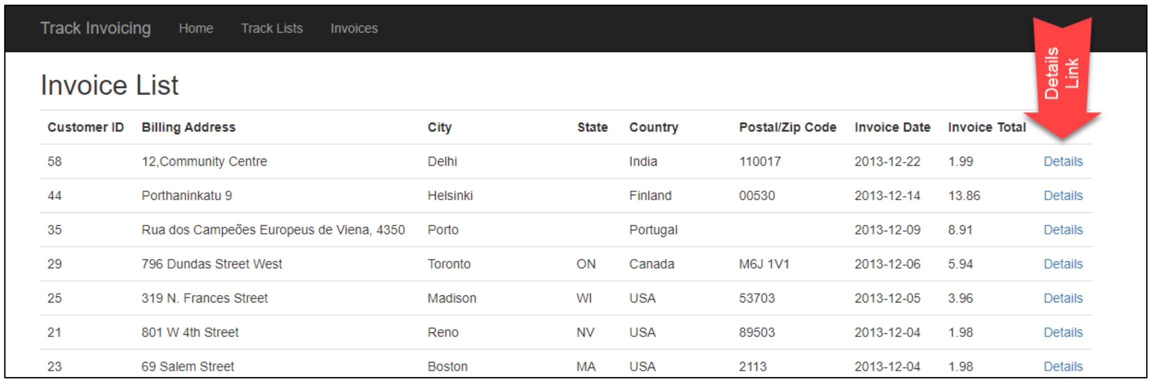
Again, focus on the “get all” and “get one” use cases for now. Do not worry about associated entities yet. In the Manager class, add the methods that support these use cases. For the “get all” use case, you should use a LINQ query expression to sort the results by **InvoiceDate** in descending order.

Add a controller that will work with the Manager object

Create a controller for the **Invoice** entity. Focus on the "get all" and "get one" use cases. Do not worry about associated entities yet.

Create the "get all" view

The "get all" use case will show the default view when working with Invoice objects. Each invoice will have a "Details" link at the right side. Here is an example screen capture.

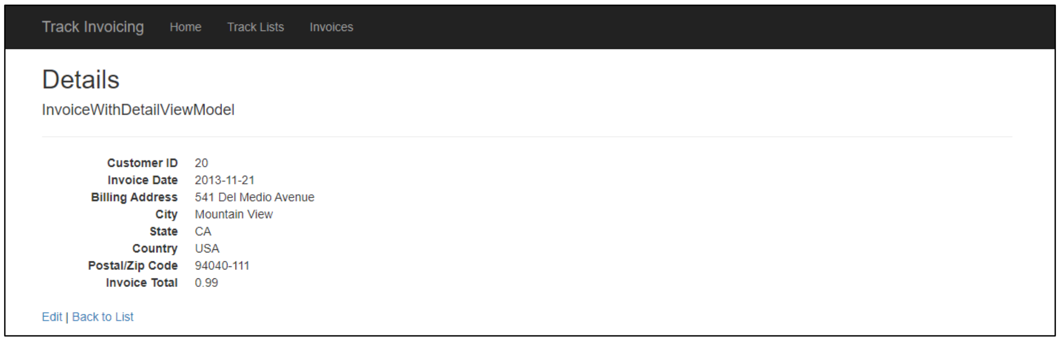


Invoice List

Customer ID	Billing Address	City	State	Country	Postal/Zip Code	Invoice Date	Invoice Total	
58	12,Community Centre	Delhi		India	110017	2013-12-22	1.99	Details
44	Porthaninkatu 9	Helsinki		Finland	00530	2013-12-14	13.86	Details
35	Rua dos Campeões Europeus de Viena, 4350	Porto		Portugal		2013-12-09	8.91	Details
29	796 Dundas Street West	Toronto	ON	Canada	M6J 1V1	2013-12-06	5.94	Details
25	319 N. Frances Street	Madison	WI	USA	53703	2013-12-05	3.96	Details
21	801 W 4th Street	Reno	NV	USA	89503	2013-12-04	1.98	Details
23	69 Salem Street	Boston	MA	USA	2113	2013-12-04	1.98	Details

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.



Details

InvoiceWithDetailViewModel

Customer ID	20
Invoice Date	2013-11-21
Billing Address	541 Del Medio Avenue
City	Mountain View
State	CA
Country	USA
Postal/Zip Code	94040-111
Invoice Total	0.99

Edit | Back to List

Incrementally add associated data

Now, it is time to add associated data. We will add some data from these entities incrementally:

- Customer
- Employee (via Customer)
- InvoiceLine
- Track (via InvoiceLine)
- Album (via Track and InvoiceLine)
- MediaType (via Track and InvoiceLine)
- Artist (via Album and Track and InvoiceLine)

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.

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

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.

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

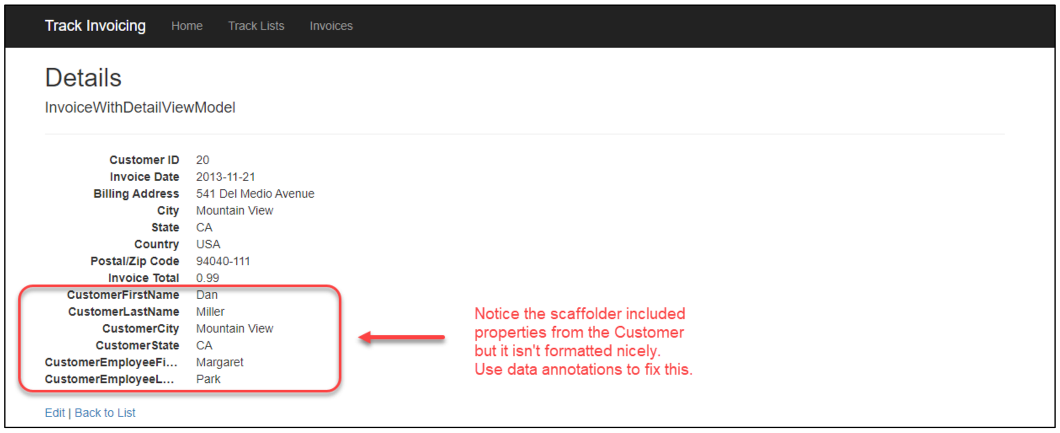
Add a mapper, from **Invoice** to **InvoiceWithDetailViewModel**.

In the Manager class, add another "get one" **Invoice** method – call the new method **InvoiceGetByIdWithDetail()**. The purpose of this method will be to fetch an **Invoice** with some associated data. It 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.

At this point in time, you may have something that looks like this: *(image on next page)*



Details

InvoiceWithDetailViewModel

Customer ID	20
Invoice Date	2013-11-21
Billing Address	541 Del Medio Avenue
City	Mountain View
State	CA
Country	USA
Postal/Zip Code	94040-111
Invoice Total	0.99
CustomerFirstName	Dan
CustomerLastName	Miller
CustomerCity	Mountain View
CustomerState	CA
CustomerEmployeeFi...	Margaret
CustomerEmployeeL...	Park

Notice the scaffolder included properties from the Customer but it isn't formatted nicely. Use data annotations to fix this.

Make the view look better. Hand-edit the view:

- Format the numbers and dates so they look nicer.
- Group the invoice-specific info together (and display the invoice identifier).
- Group the customer info together.
- Format the numbers and dates.

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. For example, you can format dates and numbers, and you can concatenate strings in a more convenient manner.

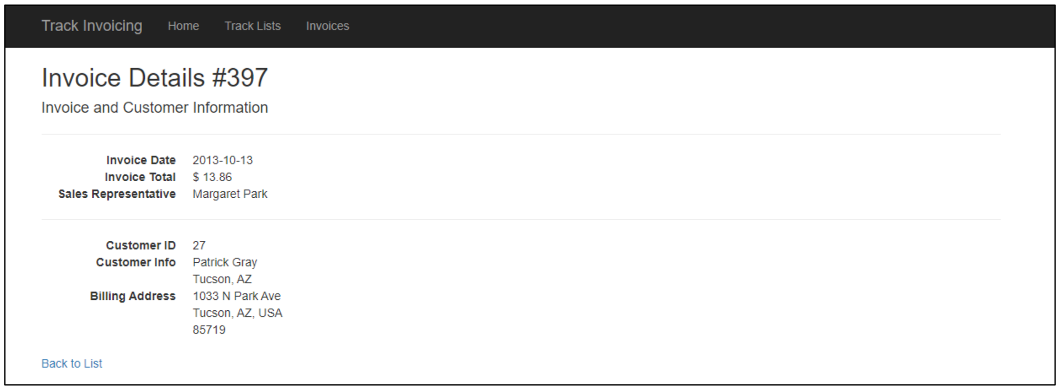
Note: There is a **DisplayFormat** data annotation. This can be quite useful for date and number formatting too.

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. Remember, a view code file can hold HTML and code expressions.

Gather the customer info together

Get rid of those atomic customer-related elements and create new concatenated strings (in the code block at the top of the view). You should end up with something like this.



Invoice Details #397

Invoice and Customer Information

Invoice Date	2013-10-13
Invoice Total	$ 13.86
Sales Representative	Margaret Park
Customer ID	27
Customer Info	Patrick Gray Tucson, AZ
Billing Address	1033 N Park Ave Tucson, AZ, USA 85719

Back to List

Add data from InvoiceLine

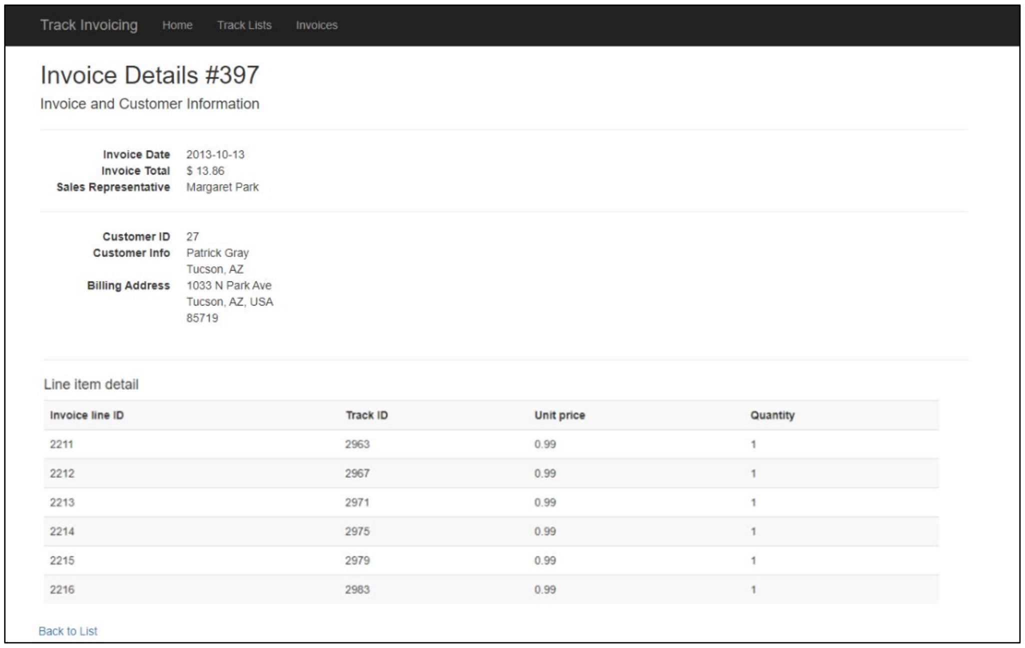
Add invoice line-item detail to the page. First, we will need a "base" view model class for **InvoiceLine**. We also need a mapper.

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. Include the **InvoiceLines** property in the statement. The controller method is good as is, with no changes.

Finally, in the view code, 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. You will end up with something similar to the following image:

(image on next page)



Invoice Details #397

Invoice and Customer Information

Invoice Date	2013-10-13
Invoice Total	$ 13.86
Sales Representative	Margaret Park
Customer ID	27
Customer Info	Patrick Gray Tucson, AZ
Billing Address	1033 N Park Ave Tucson, AZ, USA 85719

Line item detail

Invoice line ID	Track ID	Unit price	Quantity
2211	2963	0.99	1
2212	2967	0.99	1
2213	2971	0.99	1
2214	2975	0.99	1
2215	2979	0.99	1
2216	2983	0.99	1

Back to List

Add another column to the table to hold the line item total.

Note: The data in the screenshots may be differ from your assignment.

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.

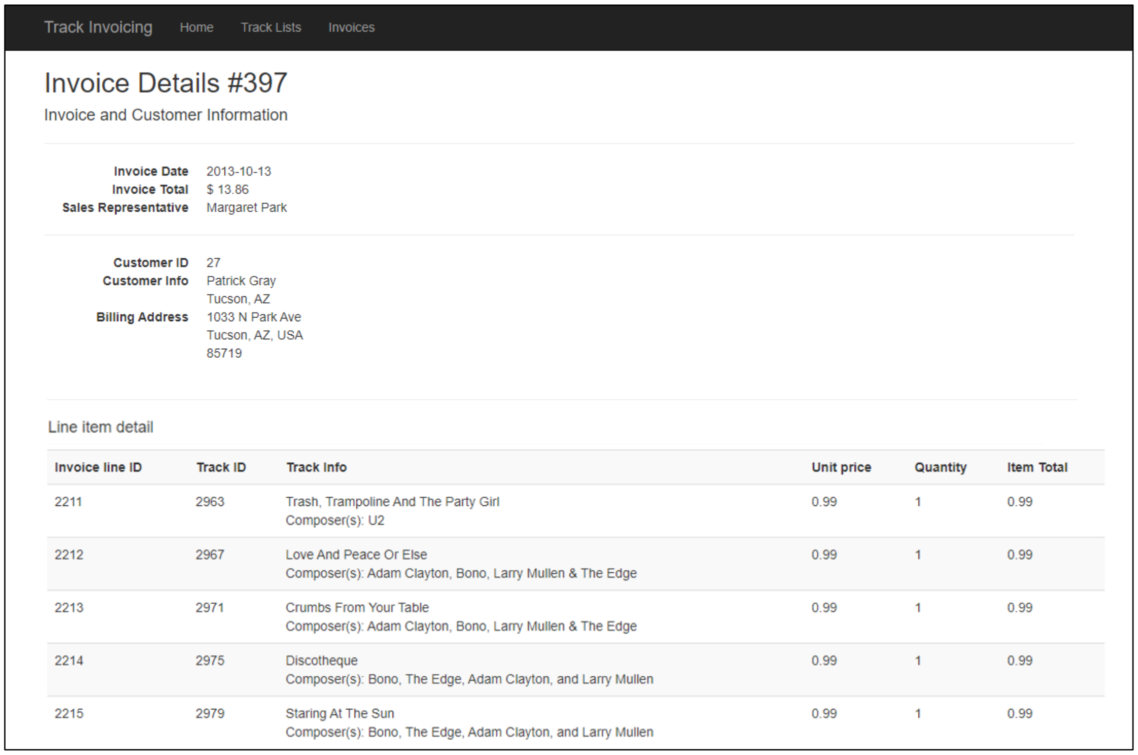
Create an **InvoiceLineWithDetailViewModel** class that inherits from **InvoiceLineBaseViewModel**. Add some composed property names from the **Track** design model class. At minimum, we want the track the **Name** property. Add a couple of others such as the **Composer** property.

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:

(image on next page)



Invoice Details #397

Invoice and Customer Information

Invoice Date	2013-10-13
Invoice Total	$ 13.86
Sales Representative	Margaret Park
Customer ID	27
Customer Info	Patrick Gray Tucson, AZ
Billing Address	1033 N Park Ave Tucson, AZ, USA 85719

Line item detail

Invoice line ID	Track ID	Track Info	Unit price	Quantity	Item Total
2211	2963	Trash, Trampoline And The Party Girl Composer(s): U2	0.99	1	0.99
2212	2967	Love And Peace Or Else Composer(s): Adam Clayton, Bono, Larry Mullen & The Edge	0.99	1	0.99
2213	2971	Crumbs From Your Table Composer(s): Adam Clayton, Bono, Larry Mullen & The Edge	0.99	1	0.99
2214	2975	Discotheque Composer(s): Bono, The Edge, Adam Clayton, and Larry Mullen	0.99	1	0.99
2215	2979	Staring At The Sun Composer(s): Bono, The Edge, Adam Clayton, and Larry Mullen	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.

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:

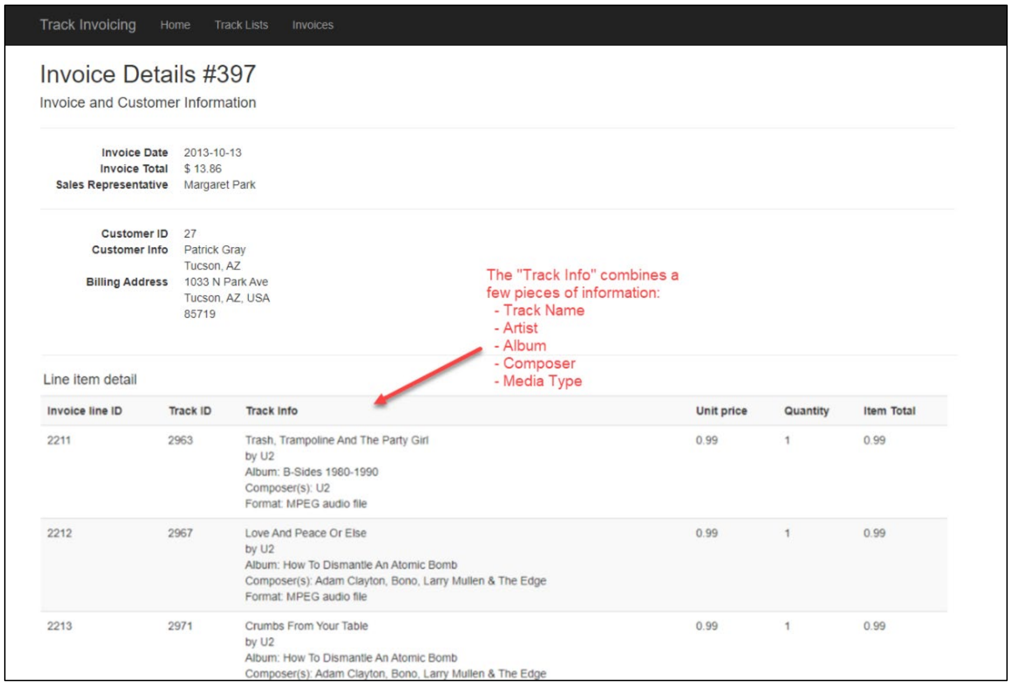
InvoiceLine > **Track** > **Album** > **Artist**

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:



Invoice Details #397

Invoice and Customer Information

Invoice Date	2013-10-13
Invoice Total	$ 13.86
Sales Representative	Margaret Park

Customer ID	27
Customer Info	Patrick Gray Tucson, AZ
Billing Address	1033 N Park Ave Tucson, AZ, USA 85719

The "Track Info" combines a few pieces of information:
- Track Name
- Artist
- Album
- Composer
- Media Type

Line item detail

Invoice line ID	Track ID	Track Info	Unit price	Quantity	Item Total
2211	2963	Trash, Trampoline And The Party Girl by U2 Album: B-Sides 1980-1990 Composer(s): U2 Format: MPEG audio file	0.99	1	0.99
2212	2967	Love And Peace Or Else by U2 Album: How To Dismantle An Atomic Bomb Composer(s): Adam Clayton, Bono, Larry Mullen & The Edge Format: MPEG audio file	0.99	1	0.99
2213	2971	Crumbs From Your Table by U2 Album: How To Dismantle An Atomic Bomb Composer(s): Adam Clayton, Bono, Larry Mullen & The Edge	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

You must comply with Seneca College's Academic Integrity Policy. Although you may interact and collaborate with others, 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.

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.

1. Locate the folder that holds your solution files. You can jump to the folder using the Solution Explorer. Right-click the "Solution" item and choose "Open Folder in File Explorer".

2. Go up one level and you will see your solution folder (similar to **S2022A2NKR** 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 **S2022A2NKR** 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://my.senecacollege.ca/.

7. Open the "Web Programming Using ASP.NET" course area and click the "Assignments" link on the left-side navigator. Follow the link for this lab.

8. Submit/upload your zip file. The page will accept three submissions so you may re-upload the project if you need to make changes. Just make sure you make all your changes before the due date! Only the last submission will be marked.