

## Introduction

You have been requested to build an app to serve the needs of companies in the television business. You will produce an app that will enable a production company to manage their actors, TV series, and episodes.

As you write this app, keep in mind that the people who work at the production company have various responsibilities, roles, and job duties. You will focus on four job titles, to complete the required tasks of this assignment:

**Executive** – Manages the talent acquisition and management process.

**Coordinator** – Oversees or coordinates the work on a TV series.

**Clerk** – Supports the coordinator and executive, during the production of TV episode.

**Admin** – The software developer will configure the database and publish to Microsoft Azure.

This assignment is worth 10% of your final course grade.

## Specifications overview and work plan

Here is a brief work plan sequence:

1. Create the project using V2 of the web app project template.
2. Customize the app's appearance.
3. Create the design model classes.
4. Create view models and mappers that cover the use cases.
5. Configure the security settings for the app.
6. Add methods to the Manager class that:
  - a. load initial data
  - b. handle the use cases
7. Implement the use cases by adding controllers, actions, and views.
8. Publish your web app to Azure.

## Getting started

Using the “**Web App Project Template V2**” project template provided by your professor, create a new web app and name it as follows: **your initials + “2237A5”**. For example, your professor would call the web app “*NKR2237A5*”. 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 a discussion board (on the course page in Blackboard).
- 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”).

## 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.

After creating the web app, customize the home page. Add a button called “**Assignment 5 on Azure**” and set the button link to the URL of your assignment on Azure.

## Create the design model classes

Add new design model classes as individual code files in the **Data** folder. You will need to add the following entities:

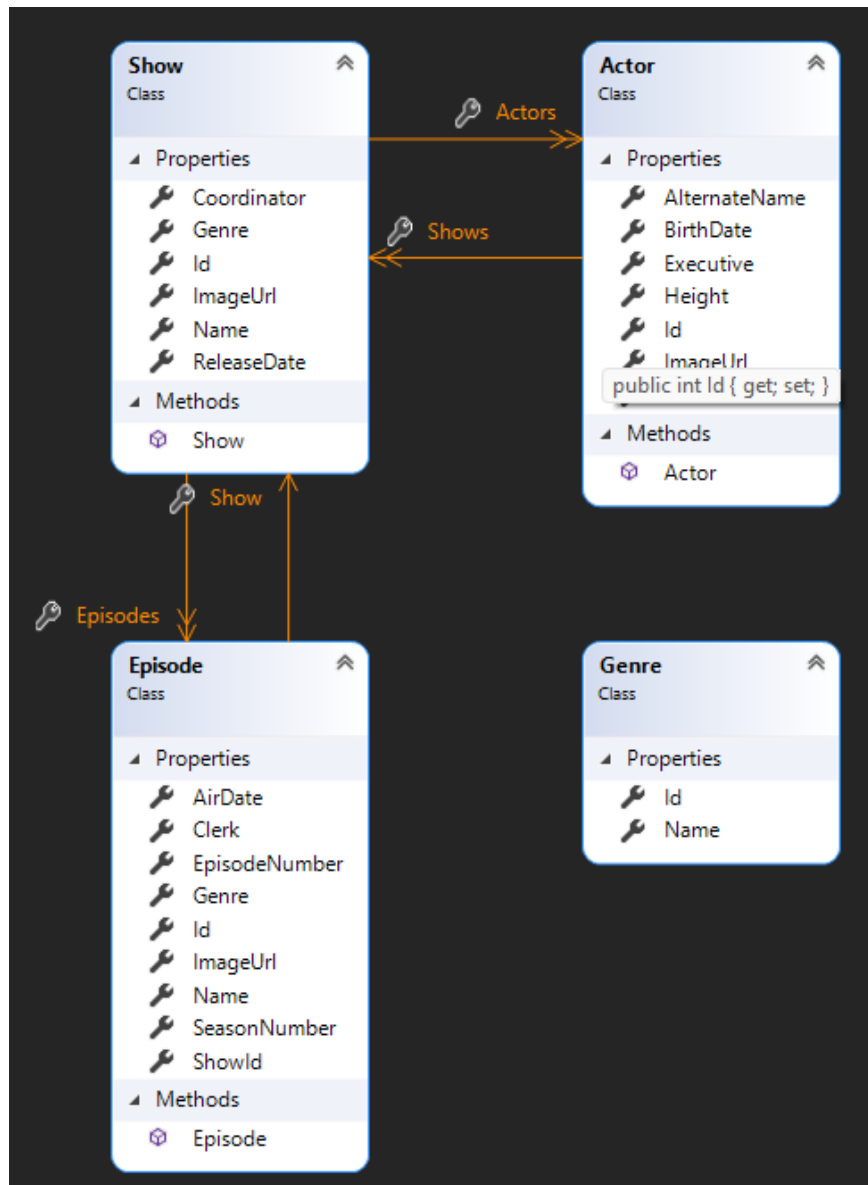
- Actor
- Show
- Episode
- Genre

When writing the design model classes, follow the guidance (rules and conventions) from the lecture notes presented during week 8:

- Name the unique identifier property Id and use int data type.
- Valid data annotations are pretty much limited to [Required] and [StringLength(n)].
- Do NOT configure scalar properties (example: int, double) with the [Required] attribute.
- When defining an association between two classes, navigation properties MUST be configured in both classes.
- Required “to-one” navigation properties must include the [Required] attribute.
- “To-many” navigation properties must be of type ICollection<T>, initialized as HashSet<T>.
- Initialize DateTime properties and default data in the no-argument constructor.

Following the above rules and conventions will ensure that the database is initialized and configured correctly. They will improve the quality of your work and simplify other coding tasks as you build the app.

Here is an image of the completed data model, from the “class diagram” feature in Visual Studio.



## Genre

The **Name** property is the name of the genre *[Required, Max Length: 50]*. For example, “Comedy”.

## Actor

The **Name** property is the actor’s full name *[Required, Max Length: 150]*. For example, “Dwayne Johnson”.

The **AlternateName** property is used for a person who uses a stage name *[Max Length: 150]*. For example, “The Rock”.

The **BirthDate** property may or may not be known.

The **Height** property may or may not be known. It stores the height of the person in meters and may contain decimals.

The **ImageUrl** property will hold a URL to a photo of the actor *[Required, Max Length: 250]*. You can find a URL for an actor almost anywhere, including Google or Bing image search, Wikipedia, or IMDB.

The **Executive** property holds the username (e.g. executive@example.com) of the authenticated user who completed the process of adding the new **Actor** object *[Required, Max Length: 250]*.

### Warning:

Do NOT make associations with the security classes, from your own design model classes. Instead, use the security APIs. For example, assume that you need to keep track of a username in an entity (for whatever reason). Here is how to approach this task:

- Get the username (i.e. email address) from the current execution context (**User** property in the controller, or **HttpContext.Current.User** property in the manager object).
- Or use the security API to fetch/lookup the desired user and get its username.
- Store that value as a non-associated string property, in your entity.

The username (email address) is GUARANTEED to be unique, so it is safe to use it.

## Show

The **Name** property is the full name of the TV series *[Required, Max Length: 150]*. For example, “WWE Smackdown!”.

The **Genre** property holds an unassociated genre name *[Required, Max Length: 50]*. In the user interface, when creating a new **Actor** object, the available genres will be shown in an item-selection element.

The **ReleaseDate** property denotes when the show will (or has) released *[Required]*. It can be a date in the future. Default it to the current date.

The **ImageUrl** property will hold a URL to a photo of the TV series cover art *[Required, Max Length: 250]*.

The **Coordinator** property holds the username (e.g. coordinator@example.com) of the authenticated user who completed the process of adding the new **Show** object *[Required, Max Length: 250]*.

## Episode

The **Name** property is the full name of a single episode *[Required, Max Length: 150]*. For example, “Final SmackDown! Of the Year 2000”.

The **SeasonNumber** property is an integer starting at 1.

The **EpisodeNumber** property is an integer starting at 1. The episode number restarts at one for each season.

The **Genre** property holds a genre string/value *[Required]*. In the user interface, when creating a new **Episode** object, the available genres will be shown in an item-selection element.

The **AirDate** property denotes when the show will (or has) aired *[Required]*. It can be a date in the future. Default it to the current date.

The **ImageUrl** property will hold a URL to a photo of the episode cover art *[Required, Max Length: 250]*.

The **Clerk** property holds the username (e.g. clerk@example.com) of the authenticated user who completed the process of adding the new **Episode** object *[Required, Max Length: 250]*.

## Associations

An **Actor** has a to-many association with **Show** and therefore an actor/actress can perform in many shows.

A **Show** has a to-many association with **Actor** and therefore a show can have many actors/actresses casted.

A **Show** has a to-many association with **Episode** and therefore a show can have many episodes.

An **Episode** has a to-one association with **Show** and therefore an episode will only exist for a single show.

## DB Sets

Remember to modify the **IdentityModels.cs** file and add **DBSet<Entity>** properties for each design model class.

## Add methods to the Manager class to load initial data

Create methods that load initial data for each entity. You will need initial data for all entities – **RoleClaim**, **Genre**, **Actor**, **Show**, and **Episode**. Ensure you enter real data, not silly data (like “aaaa” or “1234”).

- LoadRoles()
- LoadGenres()
- LoadActors()
- LoadShows()
- LoadEpisodes()

When you are coding the **Load()** methods for the **Actor**, **Show**, and **Episode** entities, you will need the name of the currently authenticated user for setting the value of the **Executive**, **Coordinator** or **Clerk** properties.

*The data you load for each entity should be unique to each student. Do not use the data provided in the screenshots supplied in this document.*

## Initial data – roles

The **Register()** method in the **AccountController** will use role claims. You will need to add the roles to the database first. Include the following role claims:

- Admin
- Executive
- Coordinator
- Clerk

## Initial data – genres

Open IMDB (or a streaming app like Netflix). Identify ten genres and add them to the data store.

*Note: When you are adding shows and episodes, you must enter a genre. To make your coding task easier, you can use simple strings. Make sure that you use the exact same string as an existing genre (case sensitive).*

## Initial data – actors

Identify three of your favourite actors in show business. If you need information, Wikipedia is often a pretty good source for data. Add these three actors to the data store.

## Initial data – shows

Select one of your *just-added* actors from above and add two **Show** objects to the data store. First, fetch the **Actor** object from the data store. For example:

```
var theRock = ds.actors.SingleOrDefault(a => a.Name == "Dwayne Johnson");
```

Next, add each show to the data store. Notice that a **Show** has an **Actors** property, which is a collection of **Actor** objects. Logically, when adding a new **Show**, the collection is empty so that you do not have to worry about overwriting it and losing data. As a result, you can simply create a new **Actor** collection containing a single **Actor** object only, our selected actor. For example:

```
ds.shows.Add(new Show {  
    Actors = new Actor[] { theRock },  
    Name = "WWE Smackdown!",  
    // etc.  
});
```

## Initial data – episodes

For each show (above), identify three episodes. Add these six episodes to the data store.

You will need a reference to the **Show** object as you add each **Episode**. Plan to fetch the **Show** objects from the data store before you add episodes.

## Load Data Controller

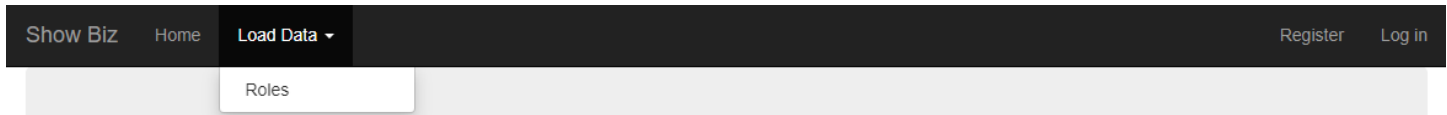
A special-purpose controller can be used to call the manager methods above. Add actions to the load data controller to load the initial data. Name the actions: **Roles()**, **Genres()**, **Actors()**, **Shows()**, and **Episodes()**.

Except for the **Roles()** action, protect the controller actions so that only users with the “Admin” role can load data. If you protect the **Roles()** action you will not be able to load Roles without signing up first, causing a paradox. Remember, you can use the **[AllowAnonymous]** attribute on this action.

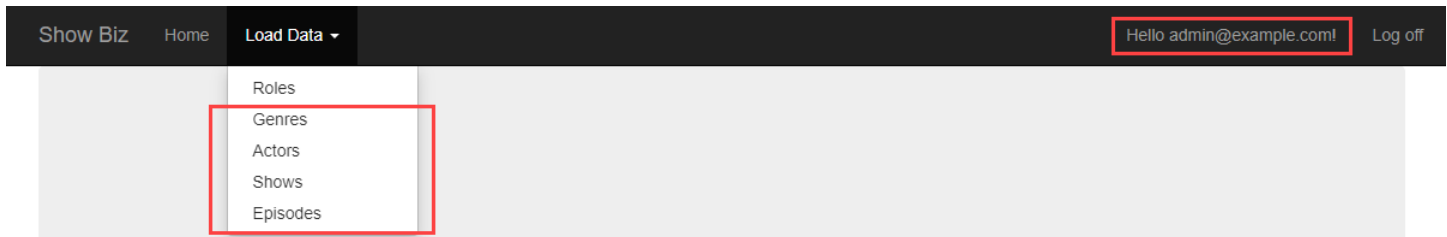
*Do not delete the controllers or load data functionality. Your professor will need to mark this part of the project.*

## Load Data Navigation

Modify the navigation bar to display load data links. All users will see the “Roles” option, but the remaining options are only visible to administrators. Use the **RequestUser.IsAdmin** property to help you.



When an admin is logged in, display something like:





## Configure the security settings for the app

Register some new user accounts. The safest internet top-level domain name to use is “example.com” because it is not processed by the internet’s DNS infrastructure. Please use the password **Password123!** for all accounts so that your teacher can run and test the app with each account.

Role claims can be hierarchical in nature or not, it depends upon the situation. For this app, create the following user accounts and set the roles accordingly:

- Admin (**admin@example.com**) should be configured with the **Admin** role claim.
- Executive (**exec@example.com**) should be configured with the **Executive** and **Coordinator** role claims.
- Coordinator (**coord@example.com**) should be configured with the **Coordinator** and **Clerk** role claims.
- Clerk (**clerk@example.com**) should be configured with the **Clerk** role claim.

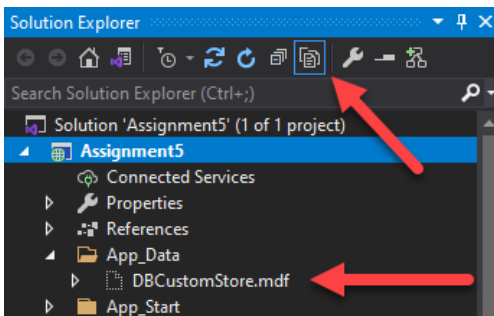
*After creating the accounts, you will need to run each load data action in order.*

## Confirm your data is loaded

If you have correctly-defined your design model classes, created the accounts, and loaded the data, the database will exist and contain at least three actors, two shows, and six episodes.

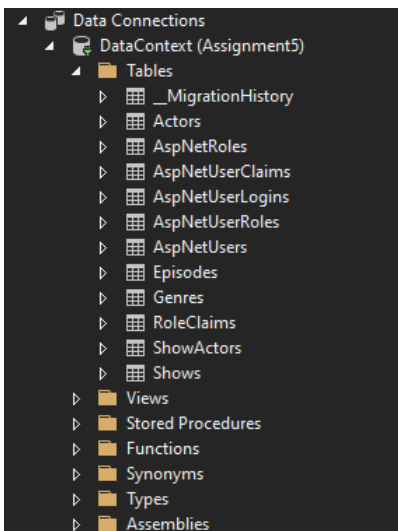
If you are careful, you can open the database, and check your progress. How?

First, in Solution Explorer, select/highlight the project item (shown in bold below). On the toolbar, choose **Show All Files**. Open (reveal) the contents of the **App\_Data** folder, it will show a file named **DBCustomeStore.mdf**, (*the file name may differ from the screenshot*). This is the file that is used to hold the contents of the database.



Double-click it, and it will open on the left-side **Server Explorer** panel. Look for the item named **DefaultConnection** or **DataContext**.

Open (reveal) the contents of the **Tables** folder.



You will notice several things:

- There are many tables that begin with **AspNet...** – those hold the ASP.NET Identity security related data
- You can see tables for your design model entities – **Actors, Shows, Episodes, Genres**.
- To support many-to-many associations, there are “bridge” tables, for example **ShowActors**.

You can right-click any of these tables, and choose **Show Table Data**, to show the data in a grid in the Visual Studio editor area.

dbo.Actors [Data]							
Max Rows: 1000							
	Id	Name	AlternateName	BirthDate	Height	ImageUrl	Executive
▶	1	Bryan Cranston	NULL	1956-03-07 00:00:00	1.79	https://upload.wikimedia.org/wiki...	exec@example.com
	2	Dwayne Douglas Johnson	Dwayne "The Rock" Johnson	1972-05-02 00:00:00	1.96	https://upload.wikimedia.org/wiki...	exec@example.com
	3	Julia Louis-Dreyfus	NULL	1961-01-13 00:00:00	1.6	https://upload.wikimedia.org/wiki...	exec@example.com
⊕	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**Very important:** When you are done looking, right-click **DefaultConnection** or **DataContext**, and choose **Close connection**. If you forget, some of your coding and testing tasks may fail.

## Data loading problems?

If your data did not load correctly, then you should remove the data, fix the coding problem, and attempt to load the data again.

How do you remove the data? Create another method in the **Manager** class for this purpose. (Later on, you can call this method from a controller action.)

Remove ALL the objects in each entity collection. Here is an example of how you would do this for the **Episode** entity collection; you would repeat this code for the other entity collections. Remember to delete the entities in reverse order.

```
foreach (var e in ds.Episodes)
{
    ds.Entry(e).State = System.Data.Entity.EntityState.Deleted;
}
ds.SaveChanges();
```

Even though there are other ways to remove objects, this is probably the easiest.

## The database is really messed up! Can I delete it and start again?

Maybe your data items are messed up. Maybe the design model classes are wrong. Maybe there are just too many little problems, and they are preventing progress. You are thinking, “I want to delete the database and start fresh”. Here is how:

The data context has a database **Delete()** method. Be careful!

### From MSDN:

Deletes the database on the database server if it exists, otherwise does nothing. Calling this method from outside of an initializer will mark the database as having not been initialized. This means that if an attempt is made to use the database again after it has been deleted, then any initializer set will run again and, usually, will try to create the database again automatically.

It is suggested that you add another method to the **Manager** class (which can then be called from a controller action/method). Its code:

```
public bool RemoveDatabase()
{
    try
    {
        // Delete database
        return ds.Database.Delete();
    }
    catch (Exception)
    {
        return false;
    }
}
```

After the database is deleted, run the data load steps again.

## Implement the required use cases

Implement the following use cases for the **Genre**, **Actor**, **Show** and **Episode** objects. At this point you should be comfortable creating the view models, mappings, **Manager** methods, controller actions and views.

Potential view model names have been supplied below, this will help you organize the structure of your view models. Your structure may be different, that is okay.

- **Genre** – get all (sorted by **Name**)
- **Actor** – get all (sorted by **Name**), get one (with detail), add new
- **Show** – get all (sorted by **Name**), get one (with detail), add new
- **Episode** – get all (sorted by **Show.Name**, **SeasonNumber**, **EpisodeNumber**), get one (with detail), add new

### Considerations for the “get all” and “get one” use cases

You will need a combination of “base” view model classes and “detail” view model classes that include associated data for display purposes. Carefully plan this step. You can use a mix of techniques, including composed properties, or by including associated objects or collections. Do not worry about getting this step perfect right away. You can modify these classes as you progress.

### Considerations for the Show and Episode “add new” use cases

It is suggested that you use the following procedure to assemble the “add new” view model classes:

1. Write the **...AddForm** classes first, by copying the properties from the design model class.
  - a. Add additional data annotations, as appropriate.
  - b. For each **Genre** property, add an accompanying select list property used for the HTML Form item-selection. Remove the **Genre** property (and keep only the list) from the view model as it may conflict with preselection.
2. Write the **...Add** classes next, they do NOT inherit from other classes.
  - a. Add additional data annotations, as appropriate.
3. Refer to the screenshots (which can be found in the sections below) to help you create the view models and design the views.




## Genre – get all (GenreBaseViewModel)

This page is accessible to all users. There is no need to modify **Genre** objects so remove all the links.

Show Biz	Home	Genres	Actors	Shows	Episodes	Load Data ▾	Register	Log in
Genre List								
<b>Name</b>								
Biography								
Comedy								
Crime								
Documentary								
Drama								
History								
Kids								
Music								
Mystery								
Thriller								

## Actor – get all (ActorBaseViewModel : ActorAddViewModel)

This page is accessible to all users however you must hide the “Create New” link for everyone except users with the “Executive” role. Make additional modifications as outlined in the screenshot.

Show Biz	Home	Genres	Actors	Shows	Episodes	Load Data ▾	Hello exec@example.com!	Log off
Actor List								
<a href="#">Create New</a> <span style="color: red;">← Only visible for "Executive" role</span>								
Name	Alternate Name	Birth Date	Height (m)	Image	Executive			
Bryan Cranston		1956-03-07	1.79		exec@example.com	<a href="#">Details</a>		
Dwayne Douglas Johnson	Dwayne "The Rock" Johnson	1972-05-02	1.96		exec@example.com	<a href="#">Details</a>		
Julia Louis-Dreyfus		1961-01-13	1.60		exec@example.com	<a href="#">Details</a>		

Set the date format. Use [DisplayFormat] attribute

Show two decimals. Use [DisplayFormat] attribute

Show as an <img> with max-width and max-height of 100px.



## Show – get all (ShowBaseViewModel)

This page is accessible to all users. Make additional modifications as outlined in the screenshot.

Show BizHomeGenresActorsShowsEpisodesLoad Data ▾

Hello exec@example.com!Log off

Show List

Name	Genre	Release Date	Image	Coordinator	
Better Call Saul	Crime	2015-02-08		coord@example.com	<a href="#">Details</a>
Breaking Bad	Drama	2008-01-20		coord@example.com	<a href="#">Details</a>

Set the date format. Use [DisplayFormat] attribute

Show as an <img> with max-width and max-height of 100px.

## Episode – get all (EpisodeWithShowNameViewModel : EpisodeBaseViewModel)

This page is accessible to all users. You will need to display the **Name** (in italic font) of the associated **Show**. You can use an **AutoMapper** composed property to help. Make additional modifications as outlined in the screenshot.

Show Biz

Home

Genres

Actors

Shows

Episodes

Load Data ▾

Hello exec@example.com!

Log off

Episode List

Name

Season

Episode

Genre

Date Aired

Image

Clerk


Uno

1

1

Drama

2015-02-08



clerk@example.com

[Details](#)

Better Call Saul


Mijo

1

2

Thriller

2015-02-09



clerk@example.com

[Details](#)

Better Call Saul


Nacho

1

3

Crime

2015-02-16



clerk@example.com

[Details](#)

Better Call Saul


Pilot

1

1

Drama

2008-01-20



clerk@example.com

[Details](#)

Breaking Bad


Cat's in the Bag...

1

2

Thriller

2008-01-27



clerk@example.com

[Details](#)

Breaking Bad


...And the Bag's in the River

1

3

Crime

2008-02-10



clerk@example.com

[Details](#)

Breaking Bad

Set the date format.  
Use [DisplayFormat]  
attribute

Show as an <img>  
with max-width and  
max-height of 100px.

## Actor – add new (ActorAddViewModel)

This page can only be accessed by users that are part of the “Executive” role. Remember to protect the “add new” controller actions using an **Authorize** attribute.

This is a standard “add new” use case that behaves like tasks you have implemented in the past.

Do not worry about associated data when adding an **Actor**. Treat it as the upper most object in the **Actor-Show-Episode** interrelationship. An **Actor** object must always get created before you create a **Show** object.

In the Manager class, remember to set the **Executive** property of the **Actor** object with the username of the current security principal (i.e. authenticated user). Otherwise, the **SaveChanges()** method will fail, because an empty **Executive** property will not pass validation.

After successfully adding the **Actor** object, redirect to the **Details** view for that object.

Show Biz Home Genres Actors Shows Episodes Load Data ▾ Hello exec@example.com! Log off

### Create Actor

Name

Alternate Name

Birth Date

Height (m)

Image

Create

[Back to List](#)

## Actor – get one (ActorWithShowInfoViewModel : ActorBaseViewModel)

This page is accessible to all users however you must hide the “Add New Show” link for everyone except users with the “Coordinator” role. Make additional modifications as outlined in the screenshot.

To show the actor names, you can set up a “to-many” navigation property in your view model with many **ShowBaseViewModel** objects. **AutoMapper** will handle this use case.

In the view, use a **@foreach** loop to display the name of each actor. There is a “.Count()” LINQ function that will help you determine the number of episodes.

The screenshot shows a web application interface for an actor's profile. The page title is "Bryan Cranston". Below the title, there is a table of personal information: Name (Bryan Cranston), Alternate Name, Birth Date (1956-03-07), Height (m) (1.79), and Image (a photo of Bryan Cranston). Below the image, it says "Executive" and "exec@example.com". At the bottom, there is a section "Appeared In" with a "2 shows" badge, listing "Breaking Bad" and "Better Call Saul". There are two links: "Add New Show" and "Back to List".

Annotations with red arrows point to specific elements:

- Red arrow pointing to "Bryan Cranston": Display the name of the actor as the heading
- Red arrow pointing to the image: Show as an <img> with max-width and max-height of 150px.
- Red arrow pointing to the "2 shows" badge: List the Name of each show the actor appeared in
- Red arrow pointing to the "Add New Show" link: Only visible for "Coordinator" role
- Red arrow pointing to the "Back to List" link: Display as a Bootstrap badge

## Show – add new (ShowAddFormViewModel and ShowAddViewModel)

This page can only be accessed by users that are part of the “Coordinator” role.

This “add new” use case will probably be the most challenging to handle. When adding a new **Show**, you can assume that you are currently working with – and have a reference to – an existing known **Actor** object.

When adding a new **Show**, you’ll need to setup an associated **Actor**. As a starting point, assume that it will work the same as it does in the “Associations Add/Edit” code example, where a vehicle could be added while working with a manufacturer. Here, you will be adding a **Show** for a known **Actor** however you will also need to allow the user to select other actors, if any.



In the `FormViewModel`, you must:

- Add a **string** property to display to your `FormViewModel` containing the name of the known **Actor**.
- Add a **multi-select** list property to display all actors. Remember to pre-select the known actor as you create the multi-select list object. The multi-select list object constructor takes a collection of selected values, so you will have to package the known actor identifier inside a new collection.
- Add a select list object to hold the list of genres.

In the `AddViewModel`, you can expect to receive a collection of integer values (**Actor** identifiers), each of which must be validated and configured (within the **Manager** class).

For this use case, you will add two **AddShow()** controller actions to the **Actor** controller instead of the **Show** controller. These actions must be protected using the **Authorize** attribute. You must also apply attribute routing, so the URLs make sense (for example: “Actors/{id}/AddShow”).

*Detour:* The project template does not include the command that activates attribute routing. You must add it. How? Open the **App\_Start > RouteConfig.cs** source code file. In the **RegisterRoutes()** method, add this statement:  
**routes.MapMvcAttributeRoutes();**

In the **Manager** class, remember to set the **Coordinator** property of the **Show** object with the username of the current security principal (i.e. authenticated user). Otherwise, the **SaveChanges()** method will fail, because an empty **Coordinator** property will not pass validation.

After successfully adding the **Show** object, redirect to the **Details** view for that object.

The screenshot shows a web application interface for adding a show. The page title is "Add Show for Bryan Cranston". The form includes the following fields and annotations:

- Name:** A text input field with a red arrow pointing to it and the annotation "Set 'autofocus'".
- Release Date:** A date input field showing "2023-03-14" with a calendar icon. A red arrow points to it with the annotation "Default to current date".
- Image:** A text input field.
- Genre:** A dropdown menu showing "Biography". A red arrow points to it with the annotation "Preselect the first item".
- Actors:** A section containing three checkboxes:
  - ☒ Bryan Cranston (A red arrow points to it with the annotation "Preselect the current artist. Ensure the checkboxes are bootstrapped and that clicking the label toggles the checkbox.")
  - ☐ Dwayne Douglas Johnson
  - ☐ Julia Louis-Dreyfus

At the bottom of the form is a "Create" button. A red arrow points to the heading "Add Show for Bryan Cranston" with the annotation "Display the name of the actor in the heading". A red arrow points to the "Actors" section with the annotation "This is a bootstrap 'well'".

The top navigation bar includes links: Show Biz, Home, Genres, Actors, Shows, Episodes, Load Data. The user is logged in as "Hello coord@example.com!" and can click "Log off".

A "Back to List" link is located at the bottom left of the form.

## Show – get one (ShowWithInfoViewModel : ShowBaseViewModel)

This page is accessible to all users however you must hide the “Add New Episode” link for everyone except users with the “Clerk” role. Make additional modifications as outlined in the screenshot.

A **Show** object is in the “middle” of two associations (**Actor** and **Episode**), so you will need to include data from both. You can use “to-many” navigation properties in your view models.

In the view, use a **@foreach** loop to display the name of each episode. There is a “.Count()” LINQ function that will help you determine the number of episodes.

Annotations from the screenshot:

- Display the name of the show as the heading
- Show as an `<img>` with max-width and max-height of 150px.
- List the Name of each actor that appears in this show
- List the Name of each episode
- Only visible for "Clerk" role
- Display as a Bootstrap badge

## Episode – add new (EpisodeAddFormViewModel and EpisodeAddViewModel)

This page can only be accessed by users that are part of the “Clerk” role. Make additional modifications as outlined in the screenshot.

When adding a new **Episode**, you will need to configure an associated **Show**. In other words, you will be adding an **Episode** to a known **Show**. Do not allow the user to select the show, instead, add properties to the AddFormViewModel and AddViewModel classes that enable you to display information about the associated **Show**, and configure its identifier as a hidden HTML Form element. Expect to get a single integer value (**Show** identifier), which must be validated and configured (in the **Manager** class).

In your AddFormViewModel you will also need to add the **Show** name as a **string** property and include a select list object to hold the list of genres.

For this use case, you will add two **AddEpisode()** controller actions to the **Show** controller instead of the **Episode** controller. These actions must be protected using the **Authorize** attribute. You must also apply attribute routing, so the URLs make sense (for example: "Shows/{id}/AddEpisode").

In the **Manager** class, remember to set the **Clerk** property of the **Show** object with the username of the current security principal (i.e. authenticated user). Otherwise, the **SaveChanges()** method will fail, because an empty **Clerk** property will not pass validation.

After successfully adding the **Episode** object, redirect to the **Details** view for that object.

The screenshot shows a web form titled "Add Episode to Breaking Bad". The form includes the following fields and annotations:

- Name:** A text input field with a red arrow pointing to it and the annotation "Set 'autofocus'".
- Season:** A text input field containing the value "1".
- Episode:** A text input field containing the value "1".
- Date Aired:** A date input field containing "2023-03-14" and a calendar icon. A red arrow points to it with the annotation "Default to current date".
- Image:** An empty text input field.
- Genre:** A dropdown menu with "Biography" selected. A red arrow points to it with the annotation "Preselect the first item".
- Create:** A button at the bottom of the form.
- Back to List:** A link at the bottom left.
- Annotations:**
  - A red arrow points to the heading "Add Episode to Breaking Bad" with the annotation "Display the name of the show in the heading".

## Episode – get one (EpisodeWithShowNameViewModel)

This page is accessible to all users. Make additional modifications as outlined in the screenshot.

Notice you will need to display the **Show** name in addition to the **Episode** properties.

The screenshot shows a web page titled "Cat's in the Bag...". The page displays the following information and annotations:

- Heading:** "Cat's in the Bag..." with a red arrow pointing to it and the annotation "Display the name of the episode as the heading".
- Episode Details:**
  - Show Name:** Breaking Bad
  - Name:** Cat's in the Bag...
  - Season:** 1
  - Episode:** 2
  - Genre:** Thriller
  - Date Aired:** 2008-01-27
  - Image:** A small image of a person in a yellow jacket. A red arrow points to it with the annotation "Show as an <img> with max-width and max-height of 150px."
- Clerk:** clerk@example.com
- Back to List:** A link at the bottom left.

## Publish to Azure

Follow the guidance in the previous assignment for help to deploy/publish your web app to Azure. Remember, you will need to create a new SQL Database and change the “tier” to “basic”, so you don’t waste your Azure credits.

Use the following names for your Azure resources (replacing **nromanidis** with your Seneca ID)

- Resource group: **WEB524**
- Database server: **nromanidis-ds-web524** (*already created last assignment*)
- Database: **nromanidis-db-web524-a5**
- Web app: **nromanidis-wa-web524-a5.azurewebsites.net**

Once published to Azure, make sure you load the roles, actors, shows, and episodes to the database.

Remember to test that the registration and the authentication processes work on Azure correctly.

Your professor will test on Azure. If there are malfunctions due to incorrectly publishing to Azure or failing to load data, you may lose up-to 25% of your assignment mark.

## Testing your work

While designing and coding your web app, use the Visual Studio debugger to test your algorithms, and inspect the data that you are working with.

In a browser, test your work by doing tasks that fulfill the use cases in the specifications.

**Reminder: The “Learn more >>” button on the project home page must be customized to link to your assignment on Azure. Your professor will use this link to test and mark your assignment.**

## 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 **NKR2237A6** 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 **NKR2237A6** 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 on the left-side navigator. Follow the link for this lab.
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.