# Introduction

## Problem

For this lab you will be moving the existing lab to the web as an ASP.NET MVC application. Only the UI will be changed in this lab. All the existing code should continue to work unchanged.

## Solution

You will be removing the existing Winforms project and adding a new ASP.NET MVC project. No other changes to the existing code should be needed.

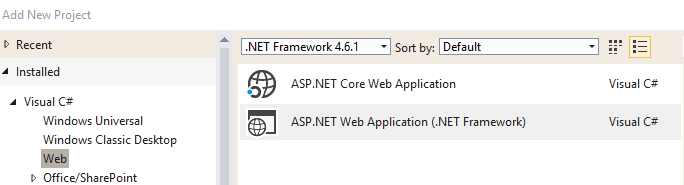
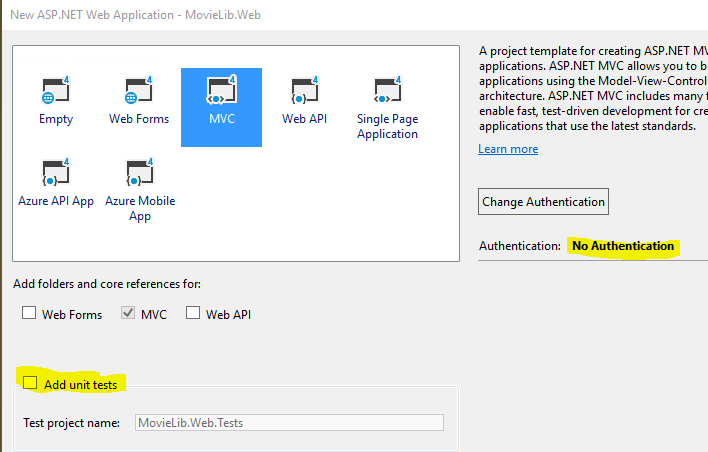
Note: All screenshots are for demonstration purposes. You can make adjustments to meet your needs.

# Setting Up the Solution

As with previous labs, copy the entire solution (folder and all) into a new folder. This will allow you to reuse all your existing code but still keep it separate from previous lab assignments.

Remove the Winforms UI project from the solution and folder. It is no longer needed.

# Creating the Web Project

1. Add a new ASP.Net Web Application project to the solution called MovieLib.Web.  
     
   
2. In the wizard select the MVC template, verify the Authentication option is No Authentication and do not generate unit tests.  
     
   
3. Set the project as the startup project.
4. Add references to the core project assemblies.
   1. MovieLib
   2. MovieLib.Data.Sql
5. Remove references to the following assemblies.
   1. Microsoft.AI.\*
   2. Microsoft.ApplicationInsights
   3. System.Web.ApplicationServices
   4. System.Web.DynamicData
   5. System.Web.Entity
   6. System.Web.Extensions
   7. System.Web.Services
   8. System.Web.WebPages.Deployment
6. Delete the following items from the project.
   1. App\_Data folder
   2. ApplicationInsights.config
7. Open the web.config file.
   1. Copy the connectionStrings section from the existing Winforms project
   2. Remove the ApplicationInsightsWebTracking module from the following sections
      1. System.web\httpModules
      2. System.webServer\modules
8. Build and run the project to ensure it loads.

# Cleaning Up Home

1. Edit the Index view for Home.
   1. Replace the contents of the DIV element with the class jumbotron.
      1. <h2>Welcome to MovieLib Web</h2>
   2. Replace the DIV element with the class row.
      1. <div>Your name</div>
      2. <div>ITSE 1430</div>
2. Edit the \_Layout view under Views\Shared.
   1. Change the title from My ASP.NET Application to MovieLib in the header.
   2. Remove the About and Contact links in the navbar.
   3. Change the copyright notice in the footer to contain your name.
   4. Change the link with text Application Name to MovieLib.
3. Edit the HomeController.
   1. Remove the actions for everything but Index.

# Defining the Models

To isolate the UI from the business layer you need to add a model for each type you want to expose. In the web project do the following.

1. Add a new class called MovieViewModel to the Models folder.
2. Add properties for each of the properties already defined on the Movie business object.

## Implement IValidableObject

Implement the IValidatableObject interface. Instead of putting the logic in the Validate method, use attributes so the client can enforce the rules. For the Length property use a custom error

To simplify the controller logic, create conversion methods to convert between Movie and MovieViewModel types. Use extension methods.

# Showing the Movies

## Implement the Action

Add a new, empty MVC controller class called MovieController to manage the movies. Rename the Index action to List. Implement the action.

1. Query the movies from the SQL database (consider using a helper method to get the database).
2. For each movie, convert to a view model.
3. Sort the movies by title.
4. Pass the view models to the view.

## Implement the View

1. Right click inside the action to add the corresponding view.
2. The view should be a List template using MovieViewModel as the model to render the list.
3. Change the Create New to Add and the action to Add.
4. Remove the Details link.
5. Change the view header and title to Movies.

## Expose on Home Page

Add a link to the layout page in the navbar to access the new view. Call the link Movies.

# Adding Movies

## Implement the GET Action

1. Add a new Add action to the Movie controller.
2. Create an empty view model.
3. Pass the view model to the view.

## Implement the POST Action

1. Add a new Add action to the Movie controller that accepts a MovieViewModel.
2. Mark the action as a POST action.
3. Verify the model is valid using ModelState.
4. If valid then add the movie to the database.
5. Catch any exceptions. If any exception occurs add it to ModelState. Reminder that errors will not be shown if you use the overload accepting an Exception object.
6. If validation or any errors occur then return the user to the view with the existing model.
7. If the operation is successful redirect the user back to the list view.

## Implement the View

1. Right click inside the action to add the corresponding view.
2. The view should be an Create template using MovieViewModel as the model.
3. Change the view header and title to Add Movie.
4. Remove the MovieViewModel header element.
5. The description can be long so replace the EditorFor method with TextAreaFor.
6. Fix the back link to use the correct action name.

## Verify the behavior

1. Can navigate to the view from the list view.
2. Entering an empty title or invalid length causes an error to appear.
3. Any server-side errors (i.e. adding a duplicate movie) returns the user to the view with the existing data intact and the error displayed.
4. Successfully adding a movie returns to the list view and the movie appears.

# Editing Movies

## Implement the GET Action

1. Add a new Edit action to the Movie controller. The action should accept an id parameter representing the movie ID.
2. Query the movie from the database.
3. If the movie does not exist return a 404.
4. If the movie does exist then pass the view model to the view.

## Implement the POST Action

1. Add a new Edit action to the Movie controller that accepts a MovieViewModel.
2. Mark the action as a POST action.
3. Verify the model is valid using ModelState.
4. If valid then update the movie in the database.
5. Catch any exceptions. If any exception occurs add it to ModelState.
6. If validation or any errors occur then return the user to the view with the existing model.
7. If the operation is successful redirect the user back to the list view.

## Implement the View

1. Right click inside the action to add the corresponding view.
2. The view should be an Edit template using MovieViewModel as the model.
3. Change the view header and title to the name of the movie being edited.
4. Remove the MovieViewModel header element.
5. The description can be long so replace the EditorFor method with TextAreaFor.
6. Fix the back link to use the correct action name.

## Verify the behavior

1. Can navigate to the view from the list view.
2. The selected movie populates the controls.
3. Entering an empty title or invalid length causes an error to appear.
4. Any server-side errors (i.e. adding a duplicate movie) returns the user to the view with the existing data intact and the error displayed.
5. Successfully updating a movie returns to the list view and the movie appears.

# Deleting Movies

## Implement the GET Action

1. Add a new Delete action to the Movie controller. The action should accept an id parameter representing the movie ID.
2. Query the movie from the database.
3. If the movie does not exist return a 404.
4. If the movie does exist then pass the view model to the view.

## Implement the POST Action

1. Add a new Delete action to the Movie controller that accepts a MovieViewModel.
2. Mark the action as a POST action.
3. Delete the movie from the database.
4. Redirect back to the list view.

## Implement the View

1. Right click inside the action to add the corresponding view.
2. The view should be an Delete template using MovieViewModel as the model.
3. Change the view header and title to the name of the movie being edited.
4. Remove the MovieViewModel header element.
5. Fix the back link to use the correct action name.

## Verify the behavior

1. Can navigate to the view from the list view.
2. The selected movie populates the controls.
3. The movie is deleted from the list.

# Requirements

Your program must meet all the following requirements.

* Compile cleanly with no warnings or errors.
* Meet all the points mentioned in the solution.
* Ensure each file has a file header indicating the course, your name and date.
* Solution, project and code should be uploaded to Github
* Submit lab in MyTCC by providing link to Github URL