**Conestoga College - ACS & IT**

**Programming Microsoft Web Technologies – PROG2230**

**Final Exam (Section-5)**

**Total marks: 30 Worth: 30%**

### Introduction

This final is a hands-on exam. You are given some starting code (found in the Final folder in eConestoga) and you must follow the steps outlined in this document to complete the code for final submission. The instructions here are sometimes complemented by comments in the starting code. Good luck, and feel free to submit multiple versions but, as always, only your last (i.e. most recent) solution will be graded.

### Please note

I remind you to make sure you do your own work on this final and resist any urge to copy code from any other source - e.g. your classmates. Everyone’s solution will be run through [Moss](https://theory.stanford.edu/~aiken/moss/) to check for academic integrity violations. ***There is zero-tolerance for such violations*** and any encountered with be dealt with in accordance with [Conestoga’s policy](https://lib.conestogac.on.ca/academic-integrity/penalties).

### Some advice

It is always best to make small changes and test it - i.e. work in small *edit-run-test* cycles. The starting code is a runnable app so do everything you can to keep the app runnable as you progress through the steps below. It is ***far better*** to have something runnable at the end that might not be complete than something you thought was complete but doesn’t run! Other than that, the best advice is to slow down and read carefully - the steps below outline everything you need to do to complete this exam.

### What/how to submit?

Zip up your entire solution into ***one zip file*** and submit that file to the eConestoga dropbox for the final. You can submit multiple solutions but only the latest (i.e. most recent) one will be looked at and evaluated.

### How will it be graded?

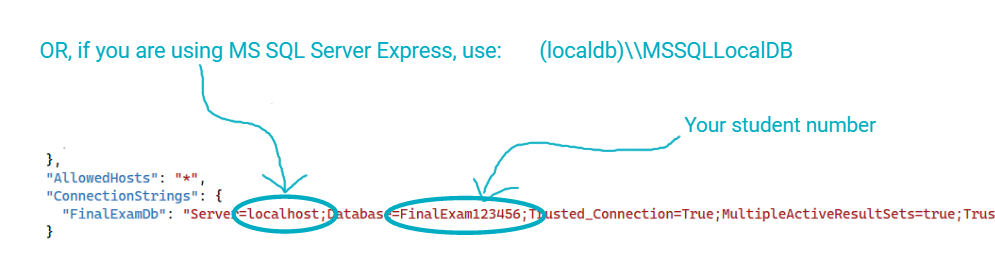
Accompanying the exam will be an ***Excel marking sheet*** that details how your grade will be calculated so you will want to make sure that you are doing everything as it’s laid out there. The Professor’s expectations when marking your solution are that they can:

* Download and extract your latest solution as a single zip file
* Open the solution in VS 2022
* Open the package manager console (PMC) and run “*Update-Database*”
* And run the solution and it should open your web app without errors in the browser

### What does the start-up project give you?

For this exam there is a code in the final-starting-code zip file. It set up some nice infrastructure for you to help get you started. More specifically, with the starting code you get the following:

* It includes a connection string in the *appsettings.json* file - BUT you must alter it so that the ***Database name includes your student number***, namely:



* It has 3 projects:
  + A class library with the entities for the app
  + An xUnit project intended to be used to test the entities class library with a project reference to the entities library project
  + An ASP.NET Core MVC Web App that also already has a project reference to the entities library project
* The web app project already includes a reference to the NuGet package for EF Core
* jQuery UI is in place so that the date picker component gets used for the user to input dates
* There is some seeded data already.
* The web app can be run and implements the following basic functionality:
  + A home page that loads with 1 view component in use and a placeholder for another
  + View Team entities in an HTML table
  + Each row in this table has the following Action links:
    - Edit: which takes the user to functioning page to edit a Team entity
    - Delete: which takes the user to functioning page to delete a Team entity
    - Manage: which takes the user to a Details page that basically consists of only placeholders because you will be implementing that page
* As mentioned, it is a runnable app and has some functionality so, after an update database, you should be able to run and use it.

### What you need to do

The final is based on an app that is very similar in structure to the database-driven web apps that have been covered in the course, especially in terms of what you encountered in the assignments and in-class/practice assignments. It allows users to view, add, edit, and delete Team entities and then manage some related child entities on a Details page.

The screenshots below will be very helpful but to complete this exam you need to complete the following high-level steps:

### Part 0 - Setup:

1. Download and unzip the “***final-starting-code.zip***” file included in the Final folder, and then open the VS solution in it to use as your starting point.
2. In the “*Home/Index*” view change “*Jasveen Kaur Taneja (Student #: 123456)*” to your full name and student number in brackets.
3. If you want to run the app as is then run update-database.

### Part 1 - Validation:

Add validation for adding/editing Player entities:

1. The fields: LeagueRegistrationNumber, Number (i.e. Jersey number), FirstName, and LastName are all required
2. Values of the LeagueRegistrationNumber property must also adhere to a specified format of: 2 letters, a dash, and 8 digits but case doesn't matter, i.e. "AB-12345678"
3. And Number must be an integer between 1 and 99
4. In addition to these value-based validation requirements, when there are validation errors, ensure that a message appears that indicates the errors in the form, e.g. error messages in red at the top and the field in error highlighted.

### Part 2 - xUnit:

1. Implement only one unit test setup for the project entities (EntitiesUnitTest) and then ensure that the test runs and passes.

### Part 3 - Tag helpers, Partial Views, and View Components:

1. The solution contains a complete solution for a “Last Action Message” TagHelper - all that you need to do for this step is to make use of it in the layout.
2. The existing Edit and Add views for the Team entity contain some common form elements - move those common elements to a partial view and then simply make use of the partial view in those 2 forms
3. The solution contains an existing View Component, namely the “*Recent Past Games*” component and it is used in the “Home/Index” page. Your task here is to implement a second “*Upcoming Home Games*” View Component. It should:
   1. Take two parameters:
      1. Max to display: representing the maximum number of Game entities to display
      2. Number of weeks: the maximum number of weeks into the future to be considered as “upcoming”
   2. And then display (at most) “max to display” future (i.e. within the specified upcoming “number of weeks”) Home Game entities and order them with the most recent at the top. This view should also display the number of weeks in a sub-heading - i.e. see the screenshot below.
   3. Then make use of the new view component in the “Home/Index” page as well, in the specified placeholder

### Part 4 - EF Core, LINQ, and the Details page:

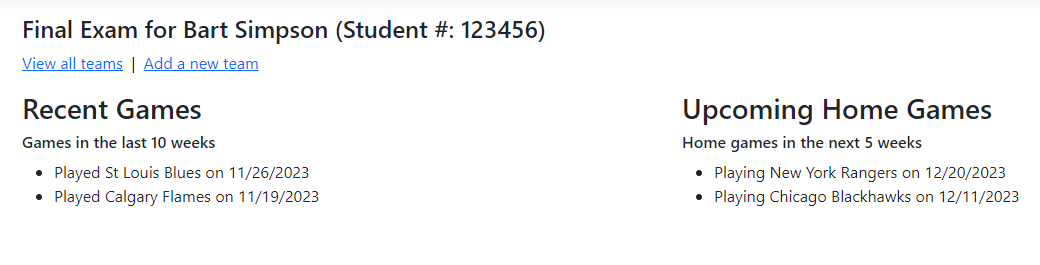
In this part you will implement the Details page that will allow the user to see more details about a specific Team entity and view and add both its related Player and Game entities. To do this, please first see the screenshots below for the Details page, and then complete the following steps:

1. Add a view model that encapsulates the information required for this Details page view:
   1. The active Team entity
   2. A potential New Player object
   3. A potential New Game object
   4. The relevant counts (see screenshot below)
2. Implement the *GetTeamById* action method in order to have it return the required information back to the details view
3. Complete the Details view by replacing the various placeholders with the relevant functionality, namely:
   1. An HTML form to view the current Team entity’s related Player entities
   2. And to the right of that table a form to add a new Player entity
   3. And then below that, another HTML form to view the current Team entity’s related Game entities
   4. And to the right of that 2nd table a form to add a new Game entity
   5. Make the “Edit this team” a functioning link
4. Add new action methods to the Team controller to handle the requests to:
   1. Add new Player entity related to the current Team entity
   2. Add new Game entity related to the current Team entity
   3. And be sure that these action methods:
      1. Return back to the Details view
      2. And result in the display of a last action message if successful
   4. And finally, configure the forms in the Details view to be handled by these action methods

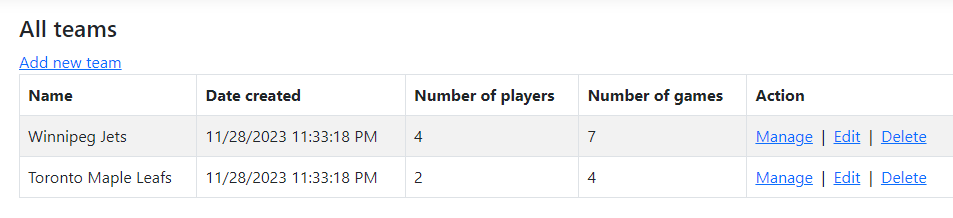
### How it should look at the start A screenshot of a computer Description automatically generated

### How it should look in the end

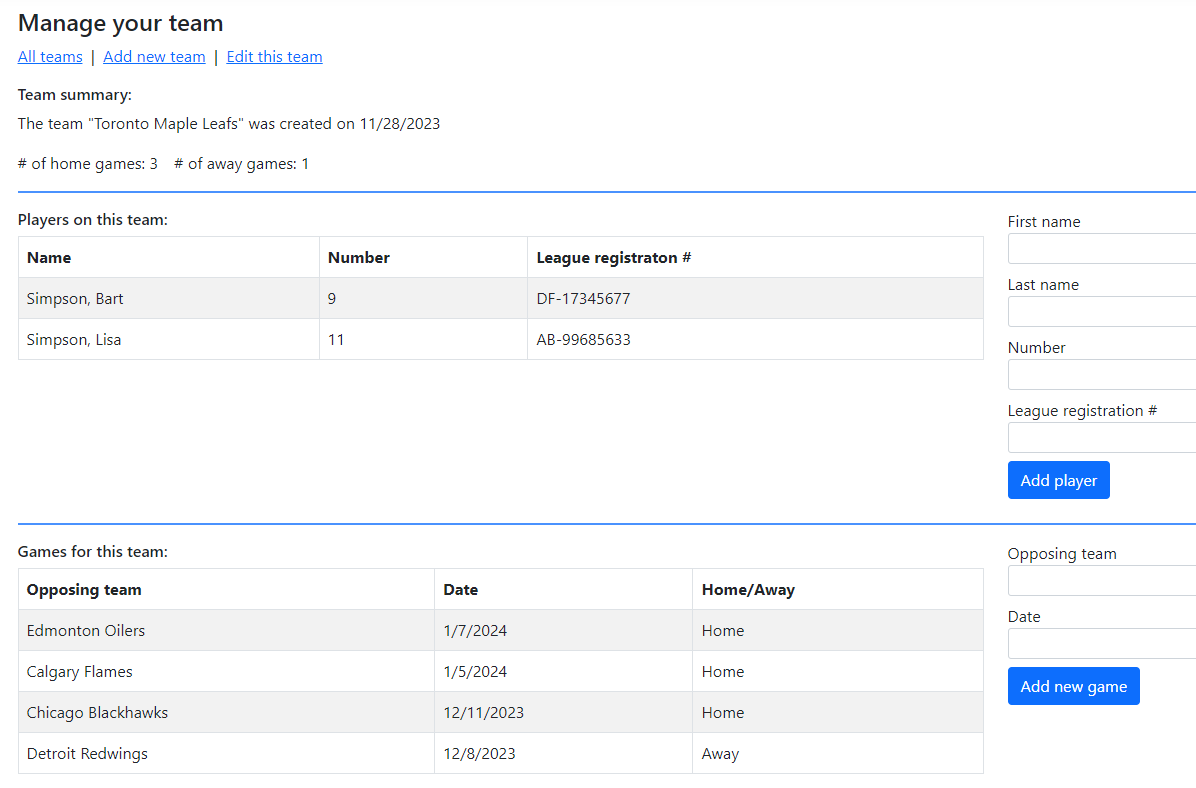
Putting all this together, the following represent some screenshots of a completed version, starting with the home page:



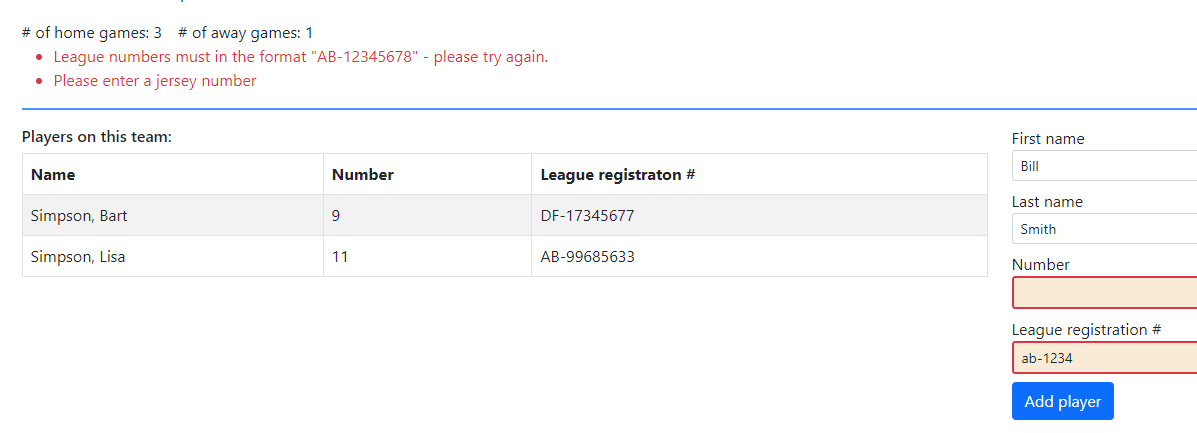
The all Teams view:



The Details view for a given Team:



The Details view after an attempt to add an invalid Player:



The Details view after a successful attempt to add a Player:

