**ASP.NET Core 3.1 MVC Code 1st Development with EF in Visual Studio**

This tutorial introduces the reader to developing an ASP.NET Core 3.1 application that uses the *Code First* approach with SQL Server. Before you proceed with this tutorial, make sure that the following pre-requisites are met:

* You are using the Windows 10 Operating System
* You have Visual Studio 2019 installed on your computer
* .NET Core 3.1 is installed on your computer
* You have installed the *dotnet-ef* tool. If you have not, then you need to run this command:

dotnet tool install --global dotnet-ef --version 3.1.0-\*

Code first development is the approach where you do not need to create the database tables. This is done for you from the code model of classes in your application. This approach does not require you to know any SQL.

The objective is to model NBA (National Basketball Association) teams and players as shown below:

**Player**

PlayerId  
FirstName  
LastName  
Position

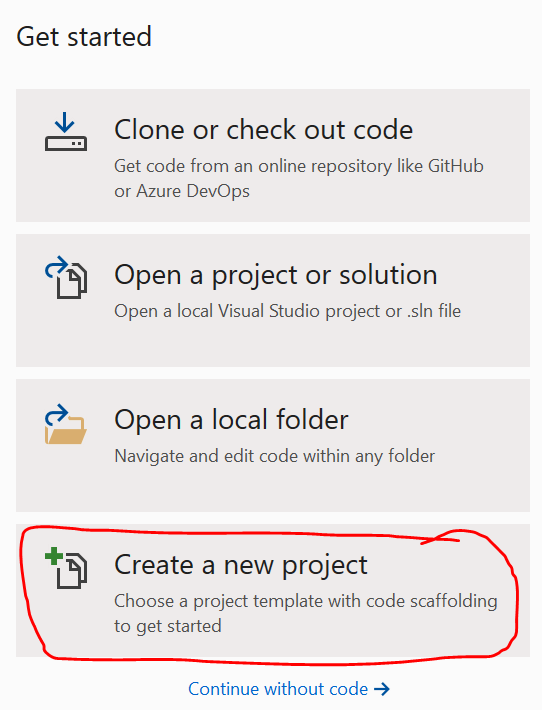
**Team**

TeamName  
City  
Province  
Country

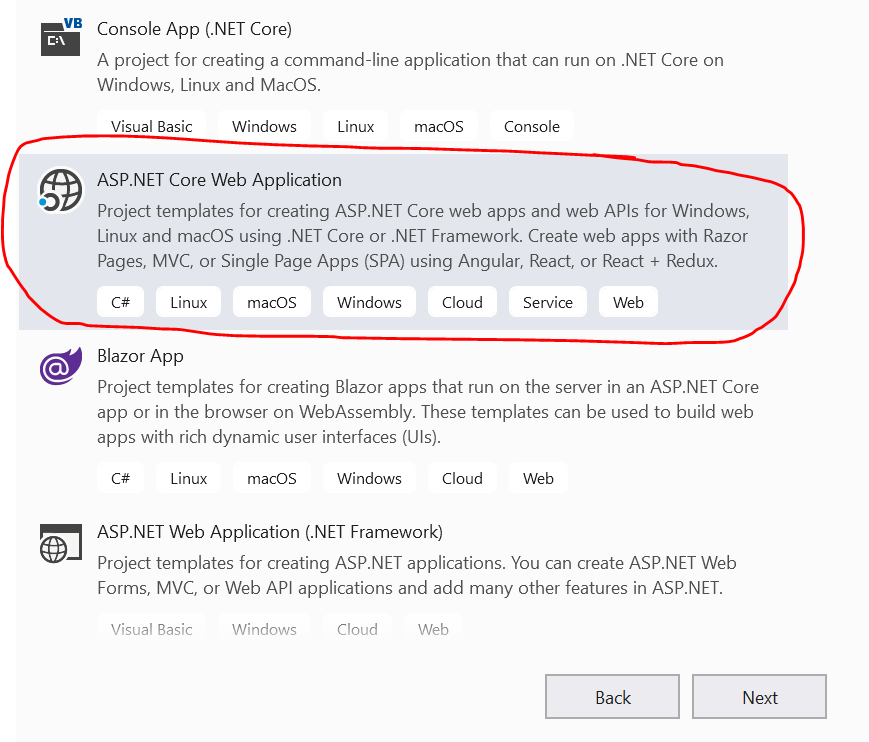
**The Visual Studio Project**

Start your Visual Studio 2019

Click on “Create a new project”



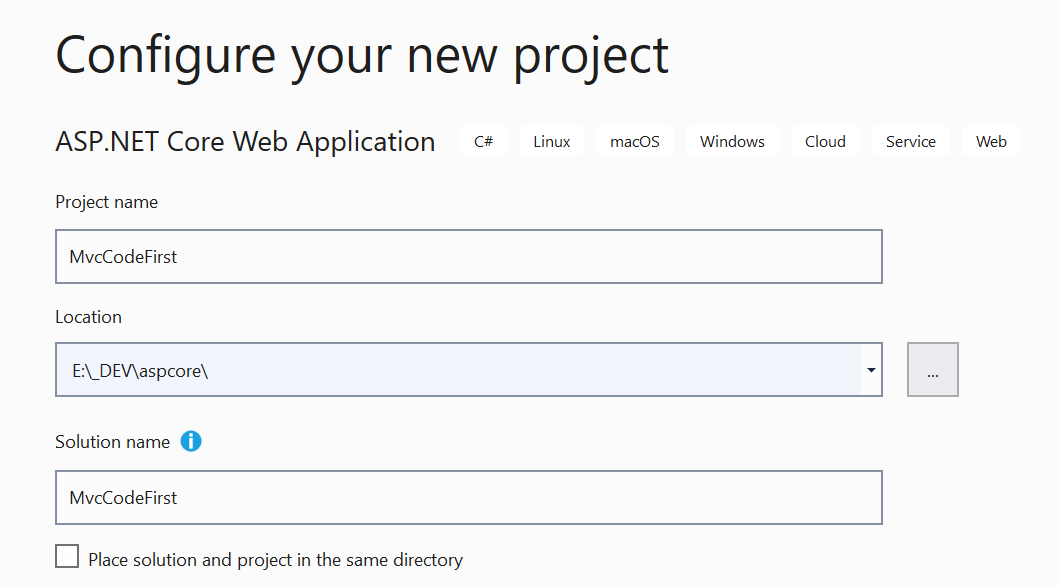
Choose “ASP.NET Core Web Application”



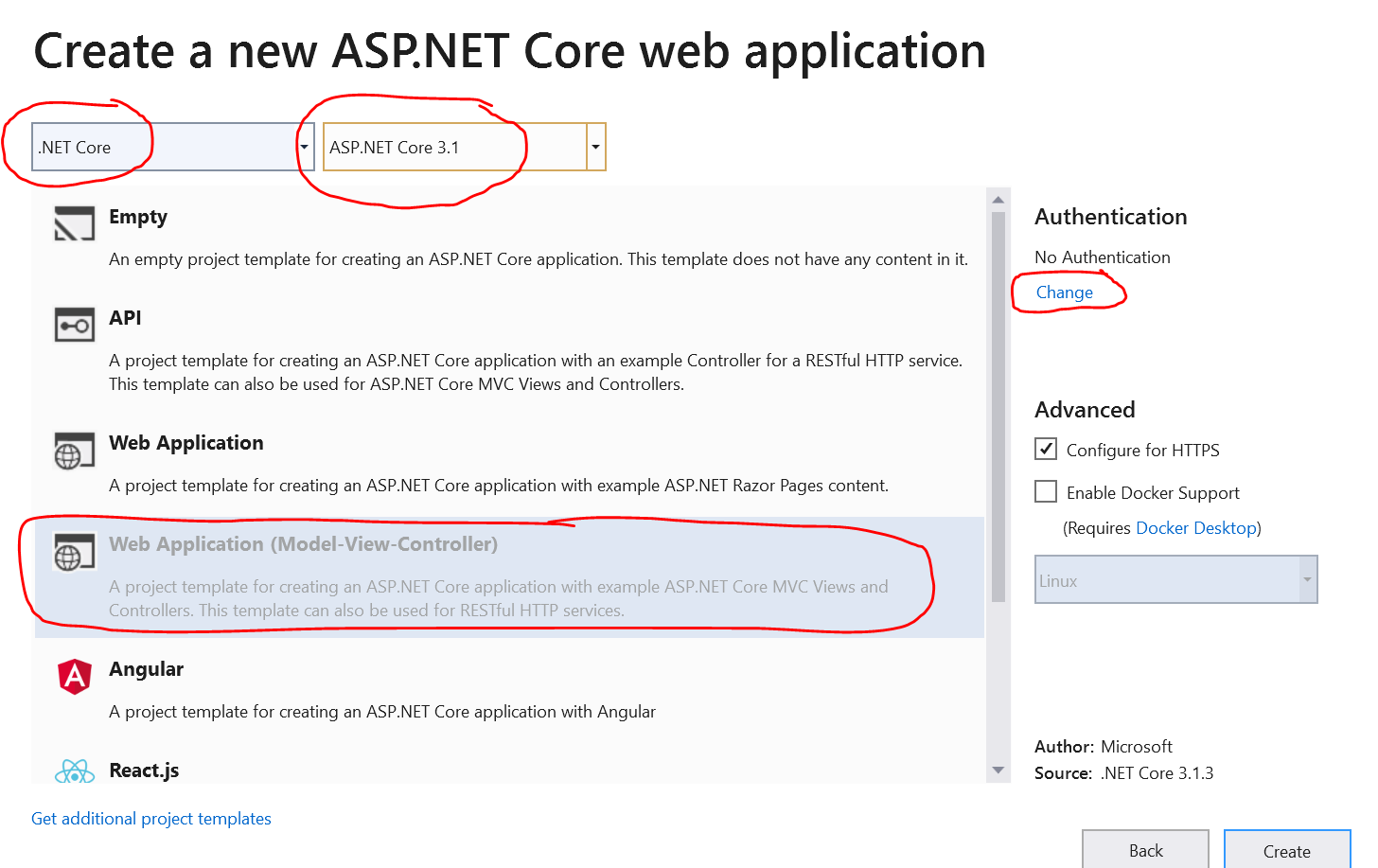
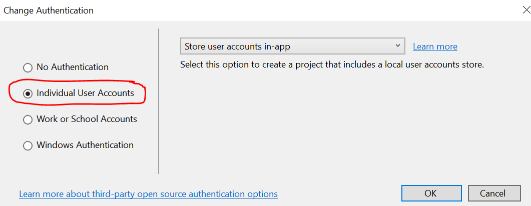
*File* >> *New* >> *Project...*

Select *ASP.NET Core Web Application*

Give your project a name (like *MvcCodeFirst*)



On the next dialog, after you click on *OK*, choose *ASP.NET Core 3.1 and Web Application (Model-View-Controller)*. Click on the*Change* button and select *Individual User Accounts*.



Click on *Create*.  
  
Let us add two classes (*Team*& *Player*) that represent the entities that were mentioned beforehand. Create two class files in the *Models*folder: *Team.cs* & *Player.cs*. Replace the class code with the following:

**Team class**

public class Team {

[Key]

[MaxLength(30)]

public string TeamName { get; set; }

public string City { get; set; }

public string Province { get; set; }

public string Country { get; set; }

public List<Player> Players { get; set; }

}

Make sure you resolve the namespaces for the "*Key*" and "*MaxLength*" classes.

**Player class**

public class Player {

public int PlayerId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Position { get; set; }

public string TeamName { get; set; }

public Team Team { get; set; }

}

Add the following to the *Data/ApplicationDbContext.cs* before or after the constructor.

public DbSet<Team> Teams { get; set; }

public DbSet<Player> Players { get; set; }

Make sure you resolve the inherited *DbContext, Team, and Player*classes.  
  
Build your application to ensure that you do not have any compiler errors.

Open *appsettings.json* for editing and change the database name so that it is simply *NBA*and not a long non-sense name. The appropriate connection string setting in *appsettings.json* will look like this:

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=NBA;Trusted\_Connection=True;MultipleActiveResultSets=true"

Developers prefer having sample data when building data driven applications. Therefore we will create some sample data to ensure that our application behaves as expected. Create a class file named *SampleData*in the *Data*directory and add to it the following code:

public class SampleData {

public static void Initialize(IApplicationBuilder app) {

using (var serviceScope = app.ApplicationServices.GetService<IServiceScopeFactory>().CreateScope()) {

var context = serviceScope.ServiceProvider.GetService<ApplicationDbContext>();

context.Database.EnsureCreated();

// Look for any teams.

if (context.Teams.Any()) {

return; // DB has already been seeded

}

var teams = GetTeams().ToArray();

context.Teams.AddRange(teams);

context.SaveChanges();

var players = GetPlayers(context).ToArray();

context.Players.AddRange(players);

context.SaveChanges();

}

}

public static List<Team> GetTeams() {

List<Team> teams = new List<Team>() {

new Team() {

TeamName="Lakers",

City="Los Angeles",

Province="California",

Country="USA",

},

new Team() {

TeamName="Golden State Warriors",

City="Oakland",

Province="California",

Country="USA",

},

new Team() {

TeamName="Rockets",

City="Houston",

Province="Texas",

Country="USA",

},

new Team() {

TeamName="Thunder",

City="Oklahoma City",

Province="Oklahoma",

Country="USA",

},

new Team() {

TeamName="Pelicans",

City="New Orleans",

Province="Louisiana",

Country="USA",

},

new Team() {

TeamName="Raptors",

City="Toronto",

Province="Ontario",

Country="Canada",

},

new Team() {

TeamName="Celtics",

City="Boston",

Province="Massachusetts",

Country="USA",

},

};

return teams;

}

public static List<Player> GetPlayers(ApplicationDbContext context) {

List<Player> players = new List<Player>() {

new Player {

FirstName = "LeBron",

LastName = "James",

TeamName = context.Teams.Find("Lakers").TeamName,

Position = "Shooting Guard"

},

new Player {

FirstName = "Kevin",

LastName = "Durant",

TeamName = context.Teams.Find("Golden State Warriors").TeamName,

Position = "Power Forward"

},

new Player {

FirstName = "Stephen",

LastName = "Curry",

TeamName = context.Teams.Find("Golden State Warriors").TeamName,

Position = "Point Guard"

},

new Player {

FirstName = "James",

LastName = "Harden",

TeamName = context.Teams.Find("Rockets").TeamName,

Position = "Shooting Guard"

},

};

return players;

}

}

Resolve all outstanding namespaces.  
  
Add the following code to seed data to the end of the *Configure()* method in *Startup.cs*:

SampleData.Initialize(app);

## Migrations

What is a migration?

A migration when the model in your application is used to automatically produce commands that create the database & tables in SQL server.

It is time to run some migration script in order to inspect the model and create database changes to accommodate the *Team* and *Player* entities. This can be done either at the terminal window or in the package manager console.

Let us first delete the migration that already exist. Delete the *Migrations* folder under *Data*.

Drop into a terminal window in the project folder that contains the *.csproj* file.  
  
Type the following dotnet commands in a terminal window inside the project folder that contains the *\*.csproj* file:

dotnet-ef migrations add InitialCreate -o Data/Migrations

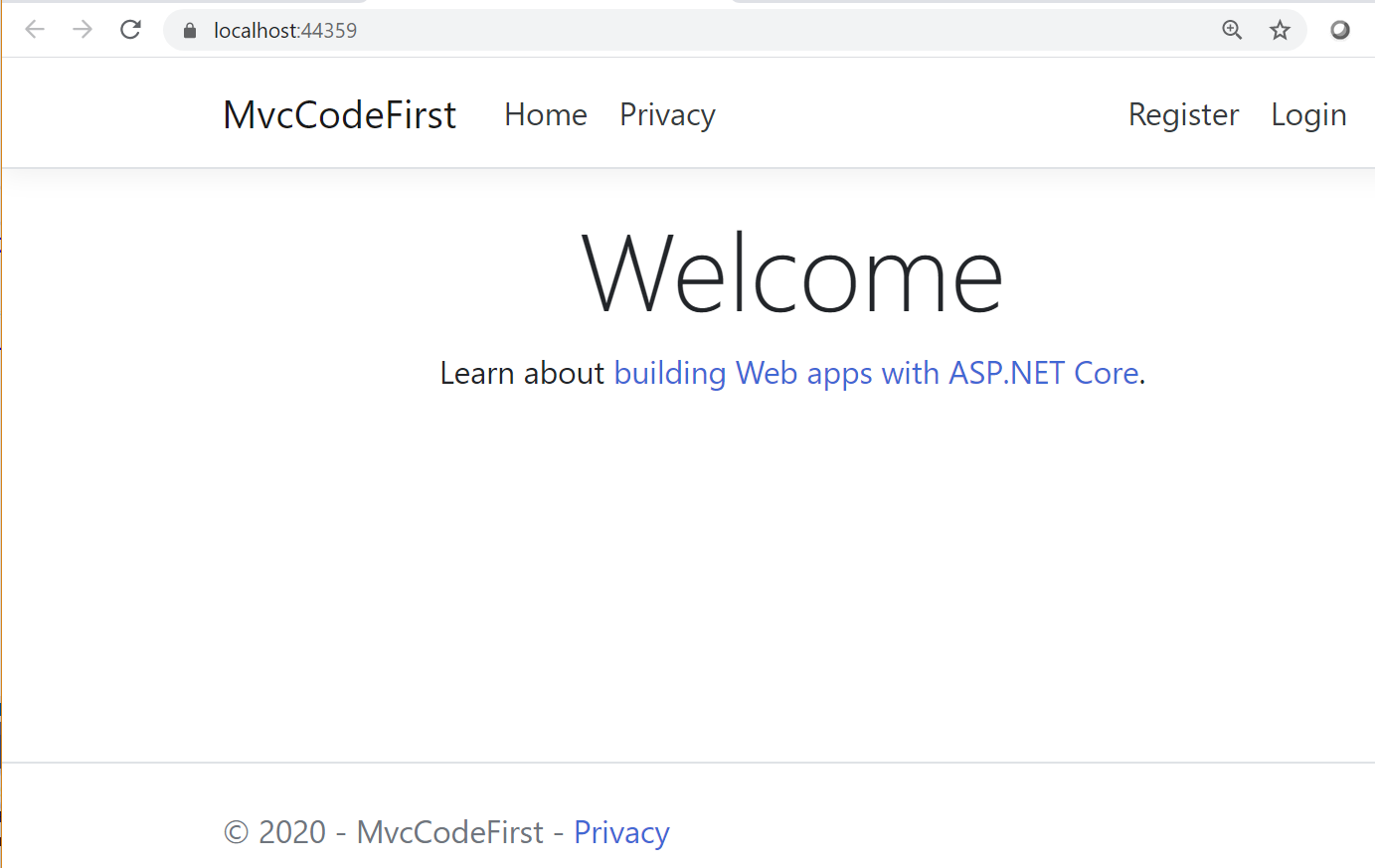
At this point, you will find a .cs file created in Data/Migrations folder that contains all the instructions for creating tables for Teams & Players in the database.

dotnet-ef database update

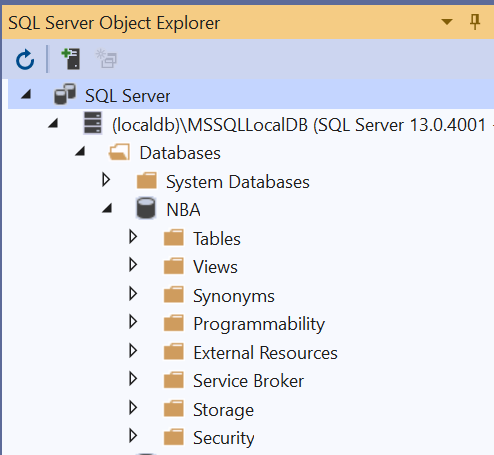
You will find that a database has been created for you containing Teams and Players data.

**Test Application**

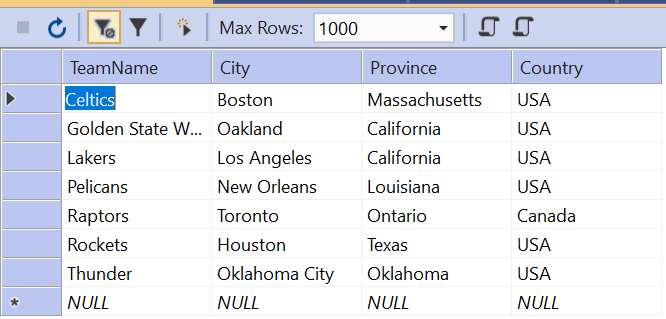
We are now ready to run the application. Hit *Ctrl+F5* on your keyboard. The application will run and will look like this:



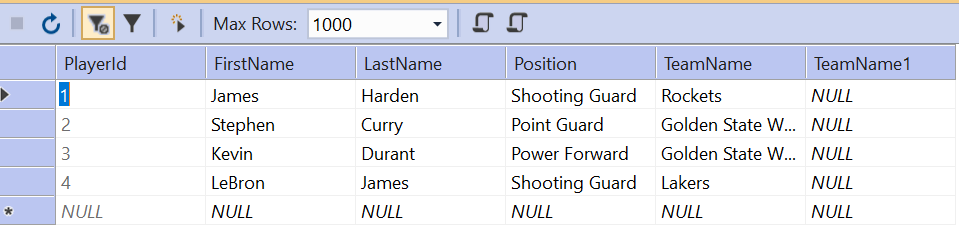
Of course, we do not see any data yet. To view the sample data in Visual Studio, click on *View*>> *SQL Server Object Explorer*. This opens up a pane in Visual Studio. Expand nodes database server >> *Databases*>> *NHL*>> *Tables*.



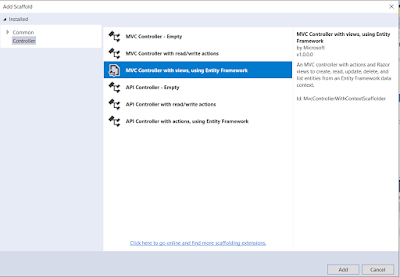
Right-click on *dbo.Teams* then select *View Data*. You should see *Teams*sample data in the database.



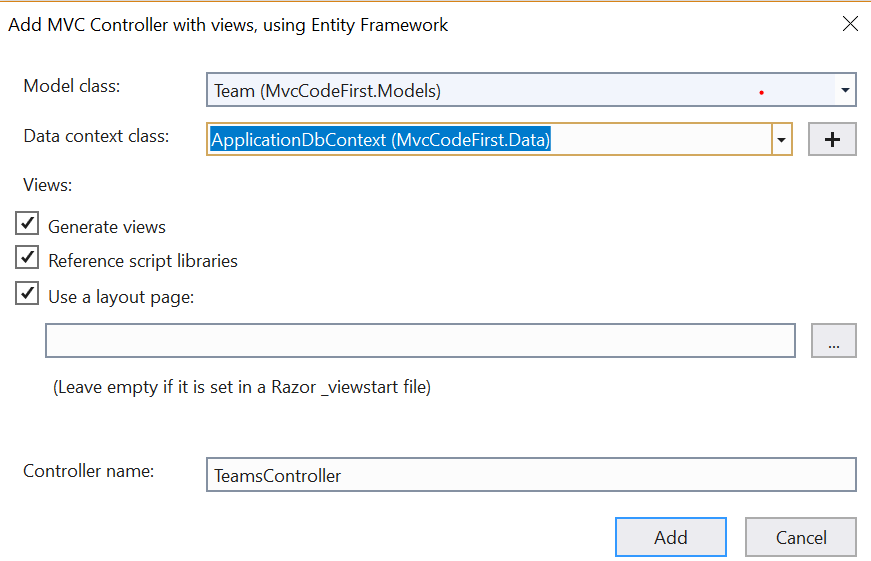
Likewise, view *Players*sample data.



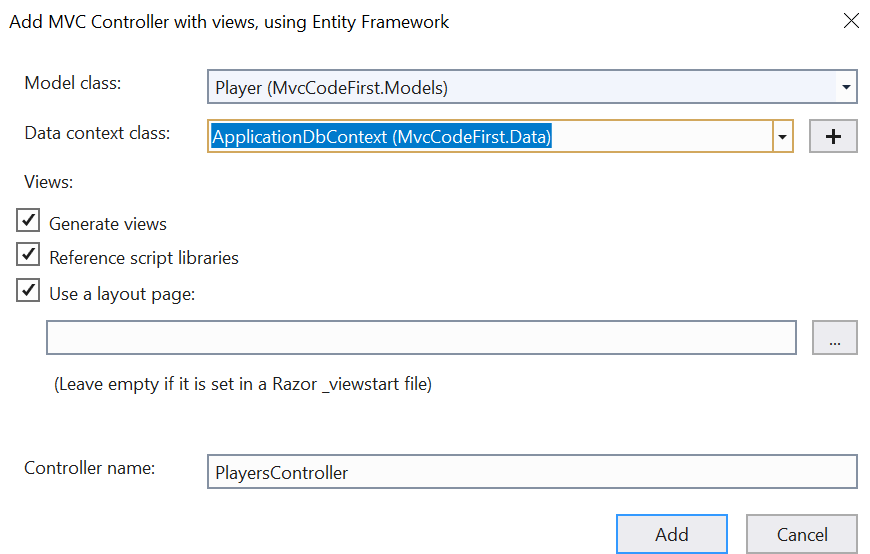
Notice that the *TeamName* column in the *Players*entity is a foreign key into the *Teams*entity.  
  
Let us now scaffold the MVC controllers for both of these entities. Back in *Solution Explorer*, right-click on *Controllers*then select *Add*>> *Controller*. Select "*MVC Controller with views, using Entity Framework*".

[](https://3.bp.blogspot.com/-j0Bn8q4JLHA/WzKbt1l9YKI/AAAAAAAACFA/Ic117lJzdagR_UyCVqoQObn7EVvpLvwXgCLcBGAs/s1600/Capture7.PNG)

Choose *Team*for *Model class* and *NhlContext*for *Data context class*.

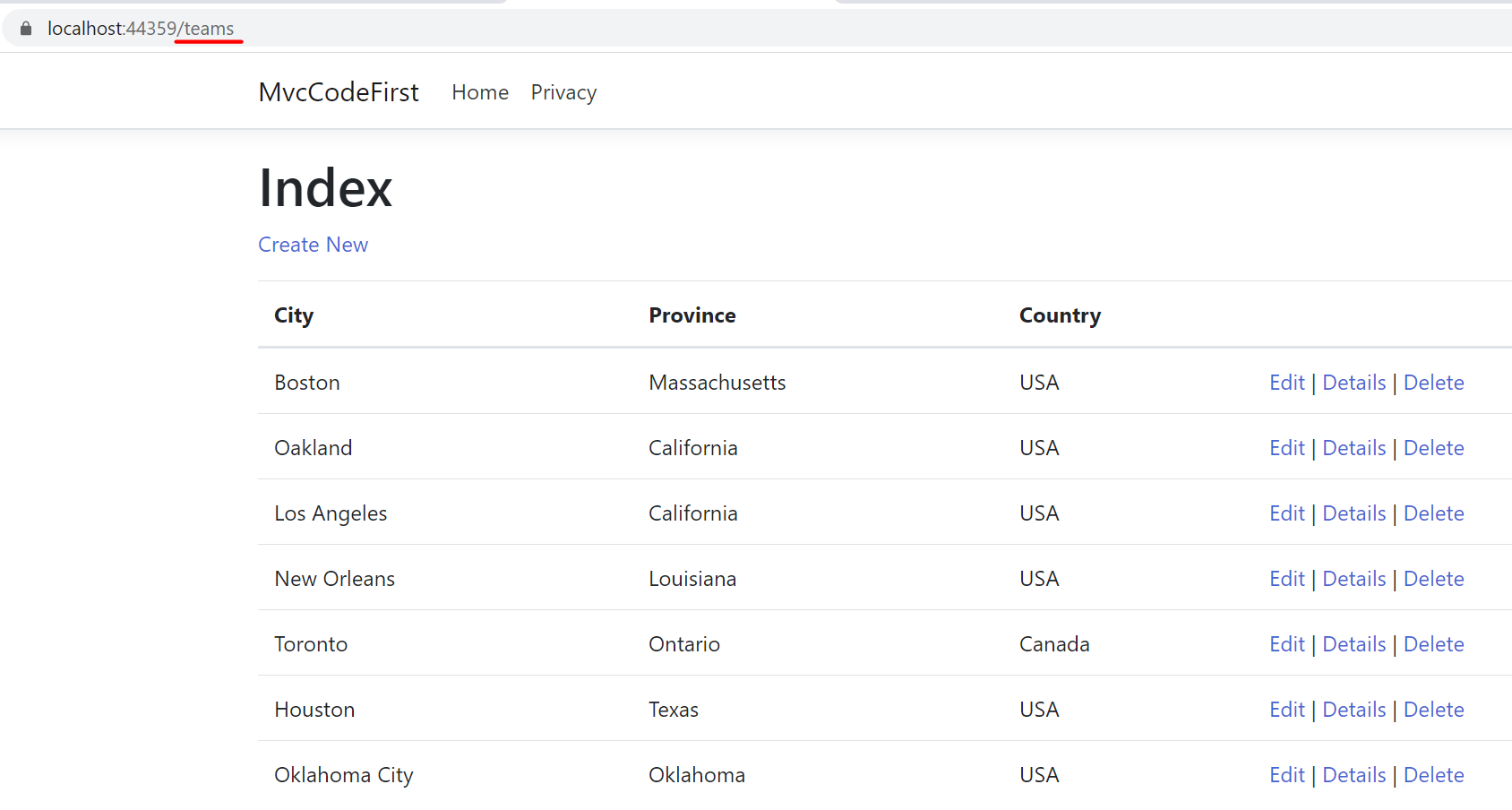


When you click on the *Add*button, the controller for *Teams*is scaffold-ed (created) for you. This includes the action methods for displaying, creating, editing and deleting data.  
  
Just like you created a controller for the *Team*table, do the same for the *Player*table.

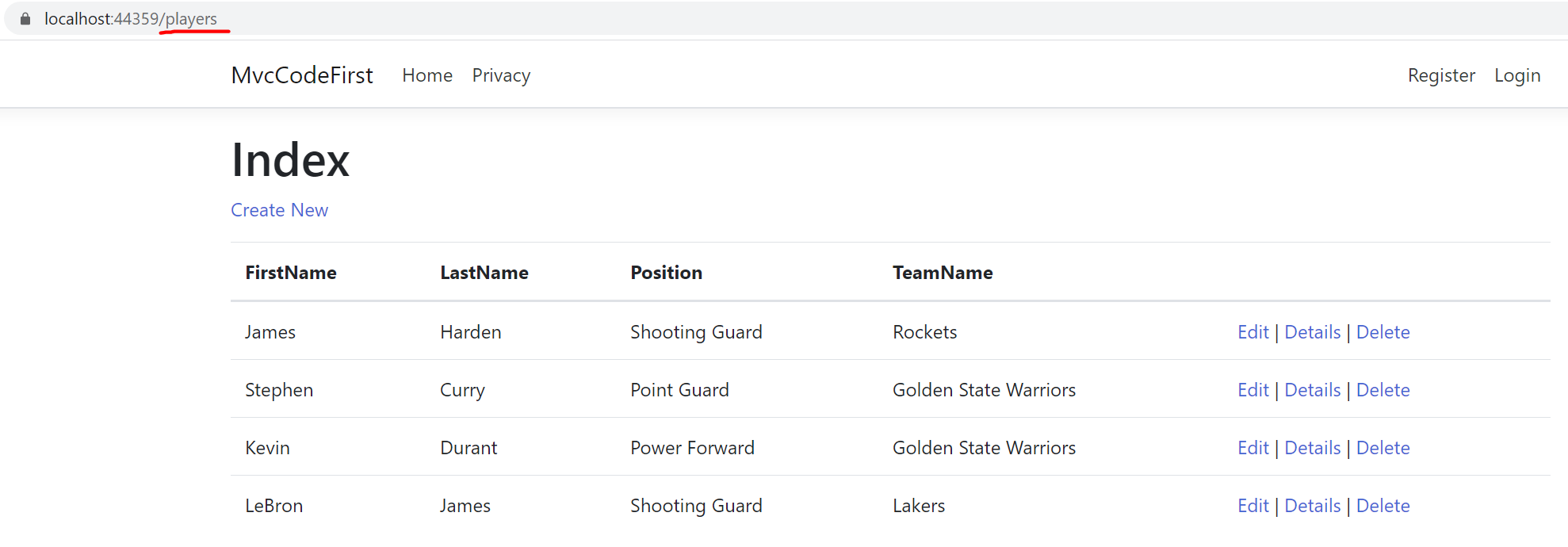


To view the output of the controllers created, you can run the application by hitting *Ctrl + F5* then add either */teams* or */players* in the address-line of your browser to view the *Teams*or *Players*controllers respectively.

**The Teams controller**



**The Players controller**



There is one thing we need to fix in the *Teams i*ndex view. Since team name is a primary key, it does not display in the table. The team-name is important to us so we need to modify *Views/Teams/Index.cshtml*. Open the file in Visual Studio and add the following HTML code as the first column title in the table:

<th>

@Html.DisplayNameFor(model => model.TeamName)

</th>

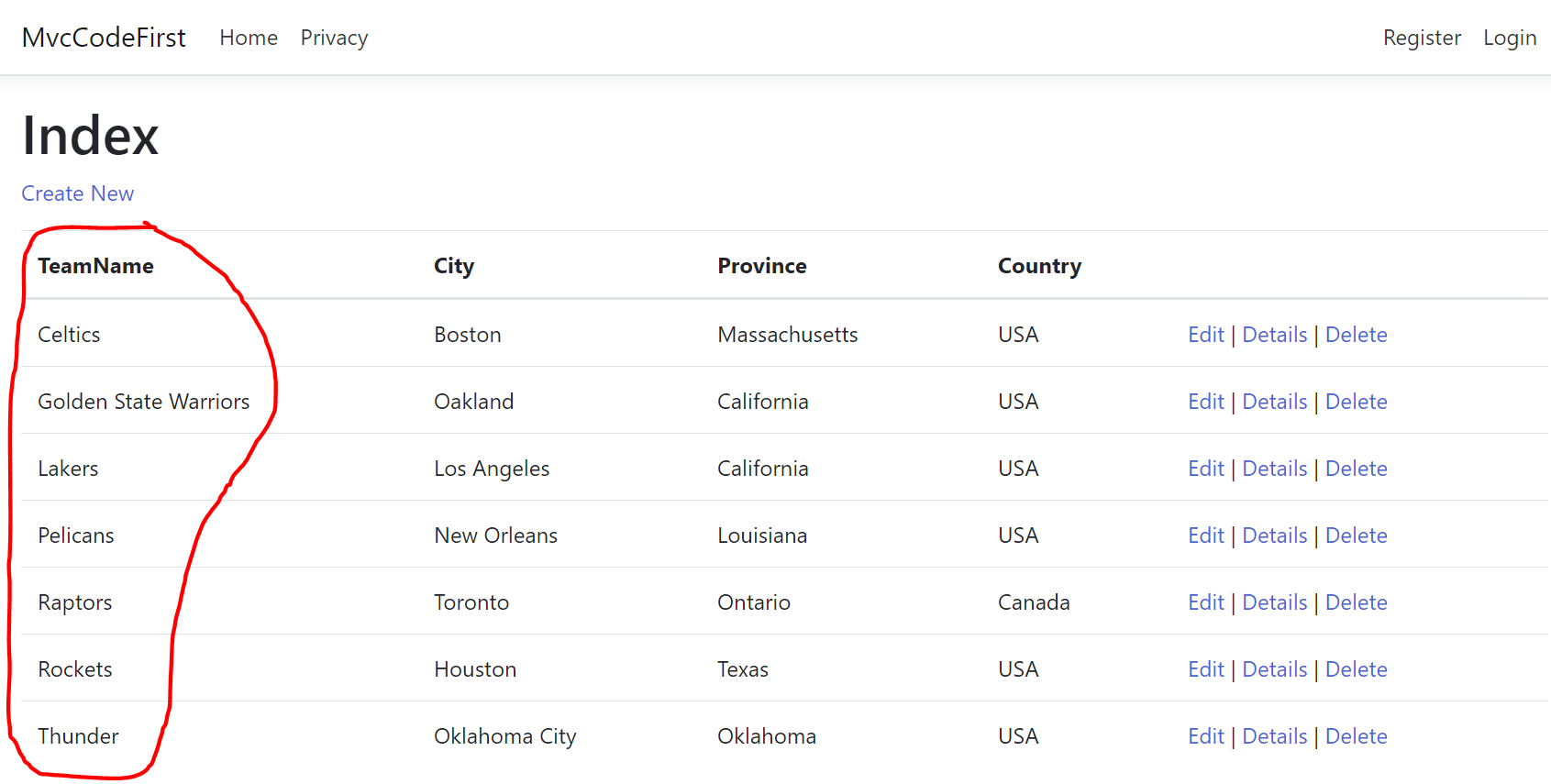
Also, add the following column data to the table with this HTML code:

<td>

@Html.DisplayFor(modelItem => item.TeamName)

</td>

When you run the application with */teams* added to the address line, you should see team names.



Let us add menu items on the home page of our application for *Teams*& *Players*so we do not have to always access these controllers by typing into browser the address line. To do this, edit *Views/Shared/\_Layout.cshtml*. Add the following HTML code right before the closing *</ul>* tag:

<li class="nav-item">

<a class="nav-link text-dark" asp-area="" asp-controller="Teams" asp-action="Index">Teams</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-area="" asp-controller="Players" asp-action="Index">Players</a>

</li>

Now, when you run the application you will see two new menu buttons for Team & Player.

