**ASP.NET Core Capstone**

**Review and build a multi-page ASP.NET Core application that displays a sports league. Teams and players are stored in a SQL database and retrieved with Entity Framework and LINQ. The data appears in tables with loops, filtering, and sorting.**

**Objective**

Now that you have mastered the basics of building ASP.NET Core applications, it is time to venture out on your own and test those skills. Your goal is to build a complete web app that displays a sports league, complete with teams, players, and sorting and filtering options.

We’ll provide you with a starter template with sample data in a SQL database. Some of the backend code is added so that Entity Framework and other middleware is ready to inject into the pages you create.

**Completed Project**

Before you start building, you can [view a sample of the completed project here](https://content.codecademy.com/courses/asp-dot-net/capstone/League-Solution.zip). Once you download the project and run it in a browser, try these features:

On the Home page:

* The “Home” and “League” links on the top nav navigate to this page
* Displays the league logo and name

On the Teams page:

* The “Team” nav link and the URL route **/teams** navigate to this page
* Displays a list of teams in row and column format
* Teams belong to a division which belongs to a conference
* Teams are ordered by the win/loss record
* A dropdown list on the heading allows selection of a single favorite
* The entire row of the favorite team has a gold background
* Each team row is a link and displays a cyan background on hover
* Clicking on any team row navigates to the Team Detail page

On the Team Detail page:

* Has a route URL of **/team/{id}**
* Displays the team logo, a picture of the team stadium, the seating chart for anyone interested in purchasing tickets, and list of players in row and column format
* Each player row is a link and displays a cyan background on hover
* Clicking on any player row navigates to the Player Detail page

On the Players page:

* The “Player” nav link and the URL route **/players** navigate to this page
* Displays a list of players in row and column format
* The header has dropdowns for filtering by team and position
* The header has a search box to filter by name (case sensitive)
* The header has a dropdown for sorting by 3 columns
* The Players row appears in bold if their depth is 1 (means they are a starter)
* The Player row has a gold background if they are on your favorite team
* Each player row is a link and displays a cyan background on hover
* Clicking on any player row navigates to the Player Detail page

On the Player Detail page:

* Has a route URL of **/player/{id}**
* Displays the team logo and player details

Your challenge is to create a similar set of pages to retrieve and display the provided data. Your app should have all of the functionality listed above.

**Starter Project**

You can [download the starter template at this link](https://content.codecademy.com/courses/asp-dot-net/capstone/League-Starter.zip).

It uses a similar approach to the 6 data lessons you completed earlier. If you’d like to refresh those concepts, you can re-read those lessons and/or use the recap below:

The dotnet command line utility was used to create a project from a template built into the .NET Core SDK:

dotnet new webapp -n League

This statement does the following:

* invoke the **dotnet** utility
* create a **new** resource
* use the built-in **webapp** template
* call the folder and the C# namespace **League**

Some images are pre-loaded in the **wwwroot/image** folder. Notice that the file names are coded by team ids. The **wwwroot/css/site.css** has some extra CSS styles applied to help you display favorites and hover over rows in grids.

There are 3 additional framework libraries required for Entity Framework support. The project already has these loaded:

* dotnet add package Microsoft.EntityFrameworkCore.Tools
* dotnet add package Microsoft.EntityFrameworkCore.Design
* dotnet add package Microsoft.EntityFrameworkCore.SQLite

There are 2 folders already created to support the Entity Framework configuration. The first folder is the **Models** folder, which has the data models that define SQL table structure and C# data types. Each is a C# class with public members:

* **League.cs**
* **Conference.cs**
* **Division.cs**
* **Team.cs**
* **Player.cs**

Take a look at these .cs files so you know which data column names you can work with.

The second folder is the **Data** folder, which contains 2 files.

* **LeagueContext.cs** is the Entity Framework *context*. A context is a C# class that acts as an intermediary between the SQL database and your C# code. You *inject* this context into each of your pages that need to retrieve data. The context class references the EF library in **using** statements. The **DbSet** method is used to wire a C# model class to a SQL table.
* **DbInitializer.cs** injects the EF context and uses **Add()** and **SaveChanges()** to write sample data records into the tables. This class is only called when the sample database does not exist. It then seeds the database. It gets an instance of the EF context and queries to see if any records exist with **Any()**.

The name of the database is established in the **appsettings.json** file. The database connection string is named the same as the EF context by convention.

The **Startup.cs** file stores a C# class that configures your application. There is a service section that establishes the EF context with **AddDbContext()** so that it can be injected into other pages. There is an additional service configured called **AddSession()**. This sets some options so that cookies can be used to pass your favorite team selection from one page to another. In the **Configure()** method, there is an extra statement to **UseSession()**. This demonstrates the common pattern for adding middleware services to your projects. In this case, the middleware is provided by the .NET Core framework. Note that you could also write your own services.

The final piece of the starter project is **Program.cs**. The **Main()** method is the starting point for any .NET Core application. The **Main()** method was modified to call **CreateDbIfNotExists()**. It calls the **DbInitializer()** class mentioned above. This checks to see if the database exists and has data in it.

Once you have the starter project loaded, try deleting the **League.db** database file. Then run the project so you can watch the database rebuild itself.

**General Hints**

You have all you need to begin! If you want additional hints, this section will guide you through more detail.

First, to build something similar to the sample app, you will need the following files:

* **Pages/Teams** folder
* **Pages/Teams/Index.cshtml** Razor page
* **Pages/Teams/Index.cshtml.cs** PageModel class
* **Pages/Teams/Team.cshtml** Razor page
* **Pages/Teams/Team.cshtml.cs** PageModel class
* **Pages/Players** folder
* **Pages/Players/Index.cshtml** Razor page
* **Pages/Players/Index.cshtml.cs** PageModel class
* **Pages/Players/Player.cshtml** Razor page
* **Pages/Players/Player.cshtml.cs** PageModel class

Every one of the above pages accesses the database so you need to inject the EF context into each. There should be a **private** **readonly** member that stores the instance of the context.

You are only reading data in each **PageModel** so create an **OnGetAsync()** method. It uses the **readonly** context member to access the database tables via the **DbSet()** defined in the context. There are many variations of this and here are a few examples:

* Retrieve a single record

Player = await \_context.Players.FindAsync(Id);

* Retrieve all Divisions for a given Conference

return Divisions.Where(d => d.ConferenceId.Equals(ConferenceId)).OrderBy(d => d.Name).ToList();

The **Teams/Index** page allows selection of a favorite so it needs to access the **HttpContext** object to set and get cookie strings. The **Players/Index** page also needs to read the cookie.

**Page-Specific Hints**

This section provides hints specific to each page of the application. For additional guidance, reference the course lessons, see “Additional Reading” below, or search the web for clues.

For the Teams page:

* Get a list of all Conferences
* Get all Divisions by Conference and sort them by name
* Get all Teams by Division and sort them by Win and Loss
* Add a **SelectList** for the favorite, populate it from all Team IDs
* Check for an existing session cookie
* Check for a new favorite on each page load and save it to the session cookie
* Create 3 **foreach** loops in the HTML
* Create markup for the header and detail rows
* Modify the class name for the **<a>** tag surrounding each team to show the favorite status

For the Team Detail page:

* Add an **@page** directive for the desired id
* Retrieve a single record based on that id
* Display the logo, stadium, and seating chart located in the **wwwroot/image** folder (note that the images are named the same as the team id)
* Include all the related players
* Add 1 **foreach** loop for the included team players
* Use the same layout you make in the **Players/Index** page

For the Players page:

* Get a list of all Players
* Add a **SelectList** for the teams, populate it from all Team IDs
* Add a **SelectList** for the positions, populate it with distinct positions
* Add a text **<input>** in a **<form>** with a **<button>** for search
* Add a **SelectList** for the sort order, populate it with column names
* Create 1 **foreach** loop in the HTML
* Create markup for the header and detail rows
* Check for an existing session cookie and set the local **Favorite** member
* Modify the class name for the **<a>** tag surrounding each player to show the favorite and starter (**Depth == 1**) status

For the Player Detail page:

* Add an **@page** directive for the desired id
* Retrieve a single record based on that id

**Optional: Adding Your Own League**

This set of tables and pages should work well for many sport leagues:

* MLB
* NBA
* NHL
* FIFA
* Local youth or club sports

To change the project for your own league:

* Find an API to get player rosters or build them from a spreadsheet. Here is a good resource: [RapidAPI](https://rapidapi.com/api-sports/api/api-football?endpoint=apiendpoint_1ca9aabd-dcc7-46a7-9415-491e30378e08" \t "_blank)
* Create a new **Data/DbInitializer.cs** file and substitute your data
* Make a new file in the **Data** folder with a new class name. If you make a new file with a different name, just reference that in **Program.cs**
* Collect images and put them under the **wwwroot/image** folder
* Delete the **League.db** database file; it will rebuild when you run

**Additional Reading**

* This article explains the **dotnet** command line utility: [.NET Core CLI overview](https://docs.microsoft.com/en-us/dotnet/core/tools/)
* Why would you pick .NET Core instead of the older .NET Framework: [.NET Core vs. .NET Framework for server apps](https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server)
* This explains the injected middleware used for cookie management in a session: [Session and state management in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-3.1)
* This is a detailed tutorial on entity framework applied to ASP.NET Core: [Razor Pages with Entity Framework Core in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-3.1&tabs=visual-studio-code)
* Master CSS flexbox for responsive grids: [A Complete Guide to Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)

**Next Steps**

There are many opportunities for a developer that has mastered the basics of C# and ASP.NET Core. Here are some areas you might want to explore:

* Write entire web sites in C# using web assembly: [Introduction to ASP.NET Core Blazor](https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.1)
* Create RESTful services: [Create web APIs with ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-3.1)
* Build real time apps with push capability: [Introduction to ASP.NET Core SignalR](https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-3.1)
* Learn remote procedure calls: [Introduction to gRPC on .NET Core](https://docs.microsoft.com/en-us/aspnet/core/grpc/?view=aspnetcore-3.1)
* Host ASP.NET Core web apps in Azure: [Static Web Apps](https://azure.microsoft.com/en-us/services/app-service/static/)
* Take online training from Microsoft hosted in Azure: [LEARN](https://docs.microsoft.com/en-us/learn/)
* Watch training videos from Microsoft on C# and ASP.NET topics: [LEARN TV](https://docs.microsoft.com/en-us/learn/tv/)
* Download and test the latest .NET Core SDK: [Download .NET](https://dotnet.microsoft.com/download)
* Download and use the most popular editor for C# and .NET: [Visual Studio Code](https://code.visualstudio.com/)