Solution Design Document

Developer Code Project for NTARA

Version: 1.00

Updated: 05/21/2025

Author: Dean Taylor

1. Overview

I have been given a task to build a code challenge project. The basic instructions are as follows:

You have been provided a comma-separated value (.csv) file. This data set should remain unedited. Your mission, if you choose to accept it, is as follows:

- Create a single web page in asp.net C# with a textbox and a button that, when clicked, searches a given dataset and returns the entire record of results on screen
- Include a drop-down that defaults to all columns but can be selected to search only one column
- Provide error checking where appropriate
- Please use a technology framework like Angular, React, or Vue to help showcase how you'll interact with the frontend.
- You will be evaluated on functionality not UI design.

2. Requirement Details

- Please use a Database (SQLite or In-Memory) is sufficient
- Please use Entity Framework to serve the data
- This is to showcase your code, organization, and standards please showcase them to the team
- Your project should work on any developer's machine when turned in as long as they have Visual Studio and run the project as a web application. If it doesn't work you won't pass. It is that important.
- Provide a write-up about your project. What was your approach? Provide things that you might change about the project if given the opportunity. Observations you made to improve the page. Length of time spent coding the application, not documentation.
- Focus on how a user would use the application
- If you've been asked to do something you haven't mastered yet. It is perfectly fine to explain that in your documentation. We love to learn and grow at our company. So, if you are learning something new is not a negative. Just let us know in the write-up.

- Delivery of the project on or before the above date you provide.
- A document explaining your approach to the project so the team can see your thoughts and methodology.
- Estimated amount of time spent on the project (for coding only, NOT documenting).
- Project in a .git-based repository of your choice.

3. Summary

This is a from-scratch development project and all requirements need to be met and/or exceeded with extra bells and whistles.

The exercise is meant to show level of experience in full stack development.

End User Story: As a sports announcer, I would like to search for football teams and see their statistical records during a game day announcement.

4. Assumptions and Prerequisites

- This project will be built, designed and run locally in Visual Studio 2022 Professional or equivalent.
- A github account will be used for git repository.

5. Solution Design Framework(s)

- ReactJS and ASP.Net Core.

6. Disclaimer

As I have most recently been developing in Salesforce and inRiver, I have not had any real experience using either the Entity Framework or doing Full Stack development with ReactJS.

I have worked on React projects before, but it has been several years.

So, a lot of this is new to me.

7. Solution

When looking at the Overview and Requirements of this project, I can make the assumption that it would be best if I create a new Visual Studio 2022 Professional using ReactJS and ASP.Net Core Template.

Before jumping into that project, I would like to look at the raw data provided in the CSV.

This is to just eyeball the data to see what it looks like and to quickly identify both the size of the data (columns and rows) and the cleanliness of the data.

Additionally, this is to use a Database First approach to the project as a whole.

A. Database First Approach

After reviewing the raw data, I decided to create a separate Data Loader project within the solution.

This project will parse the CSV file locally (within the project) and then map the data into a SQLite database for use in the Full Stack project.

To parse the data I created a Model for Football Stats in the Data Loader project. This Model additionally has a Class Map as well for use with CSVHelper to handle mapping the column headers, etc.

B. Data Modeling / Data Typing

*** NOTE ***

Normally, this is where I would discuss with a client about the data coming into a system.

It is important that we use the correct Data Types.

In this particular case, I made assumptions based on my review of the data what was integer, decimal (real), string, etc.

The Model reflects this, and I made it so that invalid data not be imported into a given field if it exists.

Usually, when reading and parsing the incoming data with CSVHelper, I would separate good records from bad records and then converse with the client about how they would like to deal with that data going forward.

However, for this demo project, I have read in all records and filtered out any data that was invalid and replaced it with null values.

I did NOT alter the CSV in any manner, but the resulting SQLite DB will data typed and NOT contain 1-1 data as the CSV.

Again, this is not the normal procedure for dealing with data.

C. Running the Data Loader

The Data Loader is a simple Console Application that displays a menu to the user for creating a database.

It includes packages for SQLite and CSVHelper.

The program runs and creates a SQLite DB and a Table for holding the records parsed by CSVHelper.

The resulting statsDatabase.db is created locally containing all the data gleaned from the csv file.

*** NOTE ***

It did not appear to be a requirement of the final application to include this step as part of the server.

This is why it was kept as a separate Data Creation step resulting in the database we want the server to use.

D. Creating the Client/Server

In Visual Studio 2022, I created a new ReactJS and ASP.Net Core Template project which builds the client and server projects with some base files in it (weather forecast).

I added the previously created database into a Data folder in the Server project.

I used Scaffolding to auto-create the base data access layer and models for my database.

Ex: Scaffold-DbContext "Data Source=Data/statsDatabase.db" Microsoft.EntityFrameworkCore.Sqlite -OutputDir Models -Tables tbl_FootballStats

Then, I updated the client jsx to call the server and populate stats data on the page.

Next, I just configured the project for a multi startup with server first (no browser), then the client to open in Chrome.

I ran the app and reviewed that the stat data was being displayed on the page.

E. Creating GitHub repository (SEGUE)

I segued here since I had base functionality working and spun up a new repo in github.

I connected my Visual Studio and pushed the base to a master branch.

Using a gitflow, I switched my working branch to a branch off develop and got back to work making commits to that working branch.

F. Begin work on React front end and installed some react components.

I npm installed a react data table to the client project for styling, sorting and pagination of the data.

I got that working and then styled for missing / erroneous data.

*** NOTE ***

I am not sure if its normal, but even though it was configured to run the server project first, it seems as though the client will ultimately still launch before the server is ready.

To get around this, I created another GET route in the server that just returns OK.

Then, on the front end, I keep pinging that endpoint until I get OK before attempting to handle the Get for the data.

To make this look more seamless, I npm installed a skeleton for doing a ‘stencilling’ for the table.

Also, I npm installed a custom dropdown for handling the requirement for searching within columns.

G. Created folder structure, context and routing within React front end

I created folders for assets, components, context and pages.

I created a StatsContext that handles all the server calls and returned context values to a StatsContextProvider

I updated the Main jsx to wrap the app with both the browser router and the provider.

H. Created components and pages within React front end

I created react components for Navbar, SearchBar and Footer.

I created pages for Home, About, Team, and Documents

I updated the routes in App jsx to handle routes to these pages.

I. Created Search functionality in the Server Data Access Layer

Handles both search term and column searching.

Updated Context to use Search

J. Wired everything up

Imported components and created handlers for all actions in search bar and navbar.

Commented as much as possible on everything to ensure other developers follow the actions.

K. Making things look pretty and responsive (styling. Lots of styling)

While the data table was already handled properly, everything else needed to be styled.

I created a nice color scheme for the site, and then added logos and icons for buttons, etc.

*** NOTE ***

It's pretty hard to have a site with a somewhat large table to work properly down to a mobile phone screen.

The site is responsive and usable to a point, but I opted here to not spend much more time making it work for mobile as I do not believe that its intended use – i.e., from the requirement user story, this is likely used for announcements during a game, like commentary. In which case, they would have a computer or a notepad in front of them.

L. Bonus Material

I wrote some code to rip out all the team logos from ESPN and have loaded those assets into the project just to give it a little more look-n-feel.

Clicking on the Team in the data table will open that row of statistics in a Team page.

Hey, it just looks nice.

About/Document pages. Again, just something to make it feel more like a real website.

8. Unit Tests

*** NOTE – THIS ONLY PARTIALLY COMPLETE ***

For the Server side, I have created a separate project called TestFootballStatsLiveServer which implements xUnit. I have created some basic Unit tests for the Controller.

For the Client side, I have installed ViTest and a few basic component tests.

9. Personal Testing

To avoid embarrassment of the project not running, I have used a separate computer and user to pull the project from github and test running it locally.

Additionally, I have Published the project As-Is to the azure cloud and witnessed that it successfully runs client and server and that the site performs as expected.

The published site is available here:

https://footballstatsliveserver20250520012711-fbh3andmc0hqadg3.eastus-01.azurewebsites.net/

10. Issues during development

Apart from time spent doing some basic research, the single largest hurdle I had was actually getting the project to deploy to Azure.

While this seems like it should have been a straightforward thing to do (which I have done before), it simply would not work as expected.

It turns out the front end was just not getting deployed with the backend.

The solution was very difficult to figure out, but eventually, I found When you create a new React project in Visual Studio 2022, you have to change the client .esproj file Microsoft.VisualStudio.JavaScript.SDK version to 1.0.2125207 to have the wwwroot folder being published.

11. Observations

- Did not like how I exactly implemented the Data. As mentioned above, there would be a talk with the client how to manage bad/missing data.
- If data set were larger, I would need to see if a better searching solution needs to be implemented.
- Would have handled sports icons differently. As is, there are all rather large and could potentially slow down the table data from being displayed because of their size. But, this was just bonus material so I did it as quickly as possible instead.
- My website logo sucks. I did not spend much time on that. It does not match the look and feel of the site.
- The favicon isn't bad.
- I've already mentioned the responsiveness, but it's actually ok for the purpose of this demo. I am not a web designer, but I think it'll do.
- I normally would not leave Unit tests until last, but I did in this case to just get the project done. I would have liked to build that out more.

12. Changes to the Project

- The demo project was frankly a bit more than I expected, but I understand its purpose and intent.
- I probably wouldn't change much to it as it does cover most things you'd expect to glean from a developer doing full stack projects.
- The data itself is stale. I would probably include more data stretching a few years.
- Probably it would be good to have a data grooming piece. As mentioned, there wasn't anything about data typing or communication with a 'client' about bad data. Frankly, in the essence of PIM development this is a critical piece.

13. Final Thoughts

- It was kind of a fun project that someone can throw themselves into.
- It's easy to spend more time on things unrelated to the project functionality such as styling, responsiveness and publishing than the project itself. Even if these things are not 'graded' on, it feels like something that has to be done for personal completeness.