

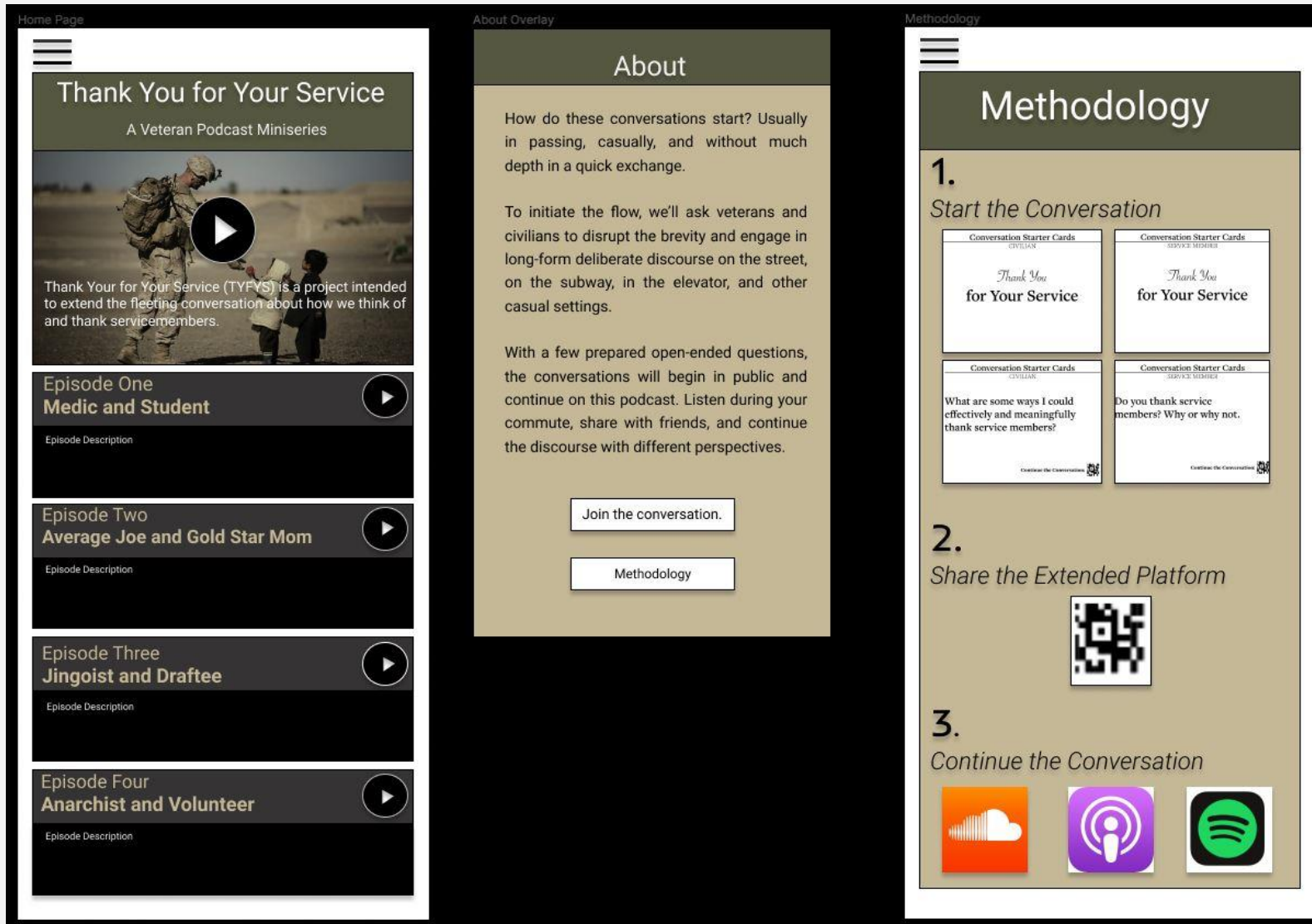
# PODCAST WEBSITE

K. David Pearce

Student Project

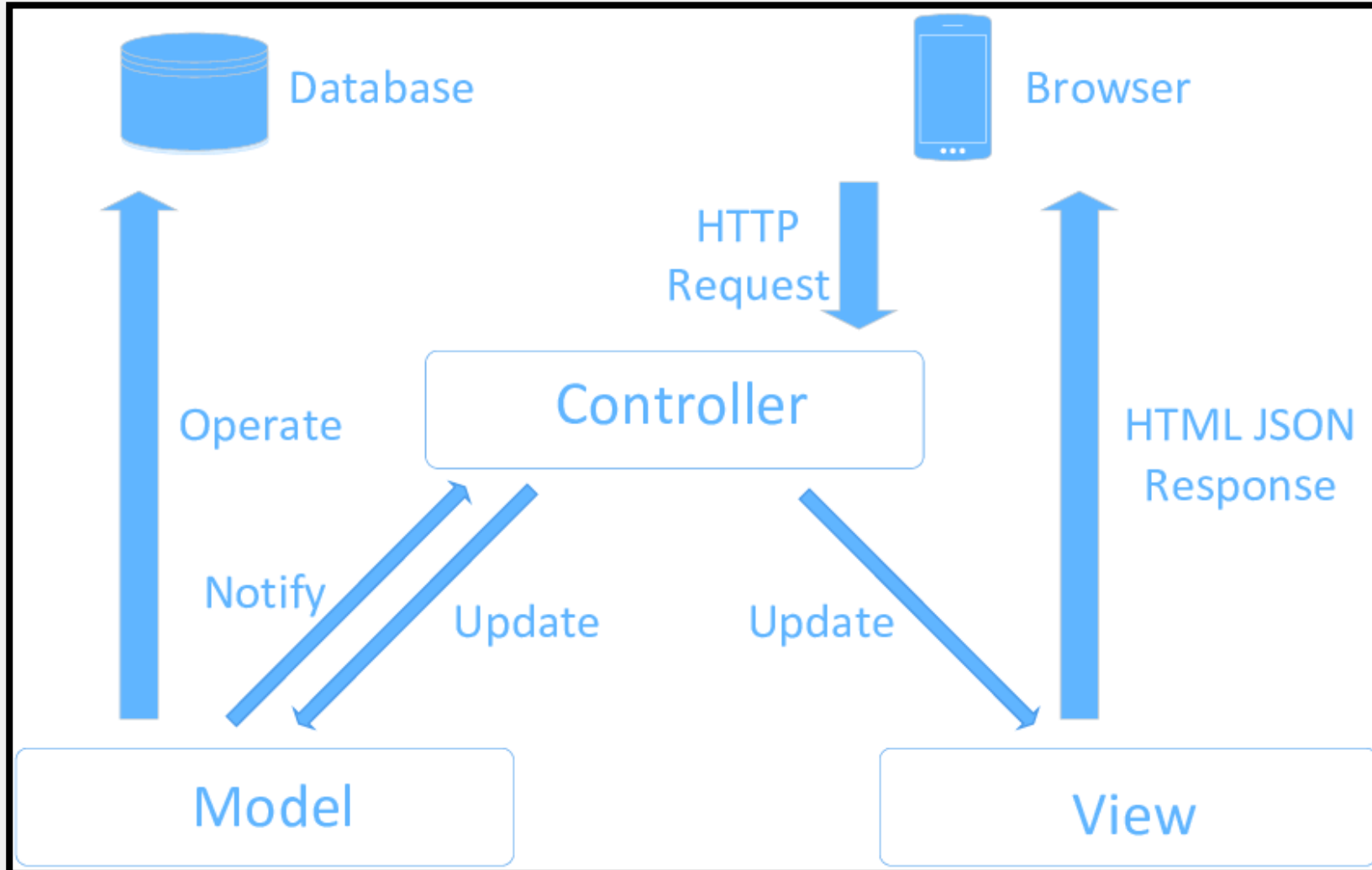
MSSA San Antonio Cohort 5

# PURPOSE



- My wife wants to create a podcast titled “Thank You For Your Service” that extends the fleeting conversation between veterans and civilians about how we think of and thank service members
- She designed a prototype of the website (see left image) using a website called Figma
- The website will host her podcasts and will also have an online store where fans can purchase things like shirts and coffee mugs

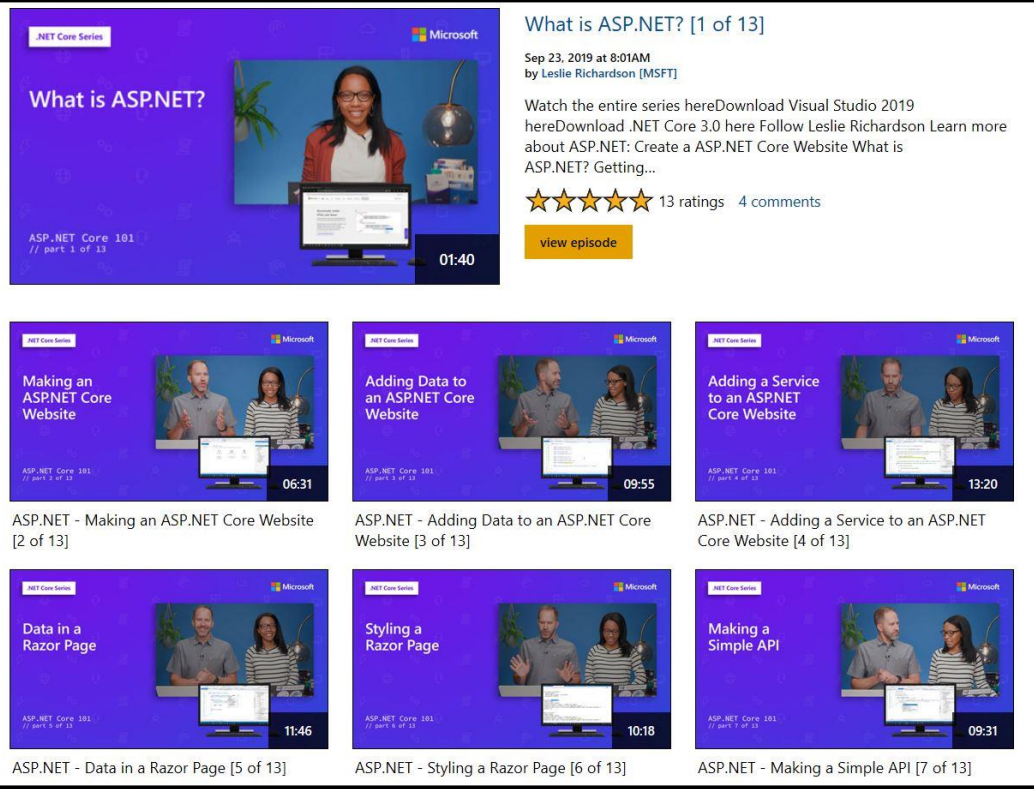
# FUNCTIONAL REQUIREMENTS



- The user will click a button on the website to listen to a podcast episode
- The browser will send an HTTP request to the Controller
- The Controller will access the Model which retrieves the correct episode from the database
- The controller will then access the View which sends the HTML response back to the browser

# LITERATURE TO ASSIST ME (PART I)

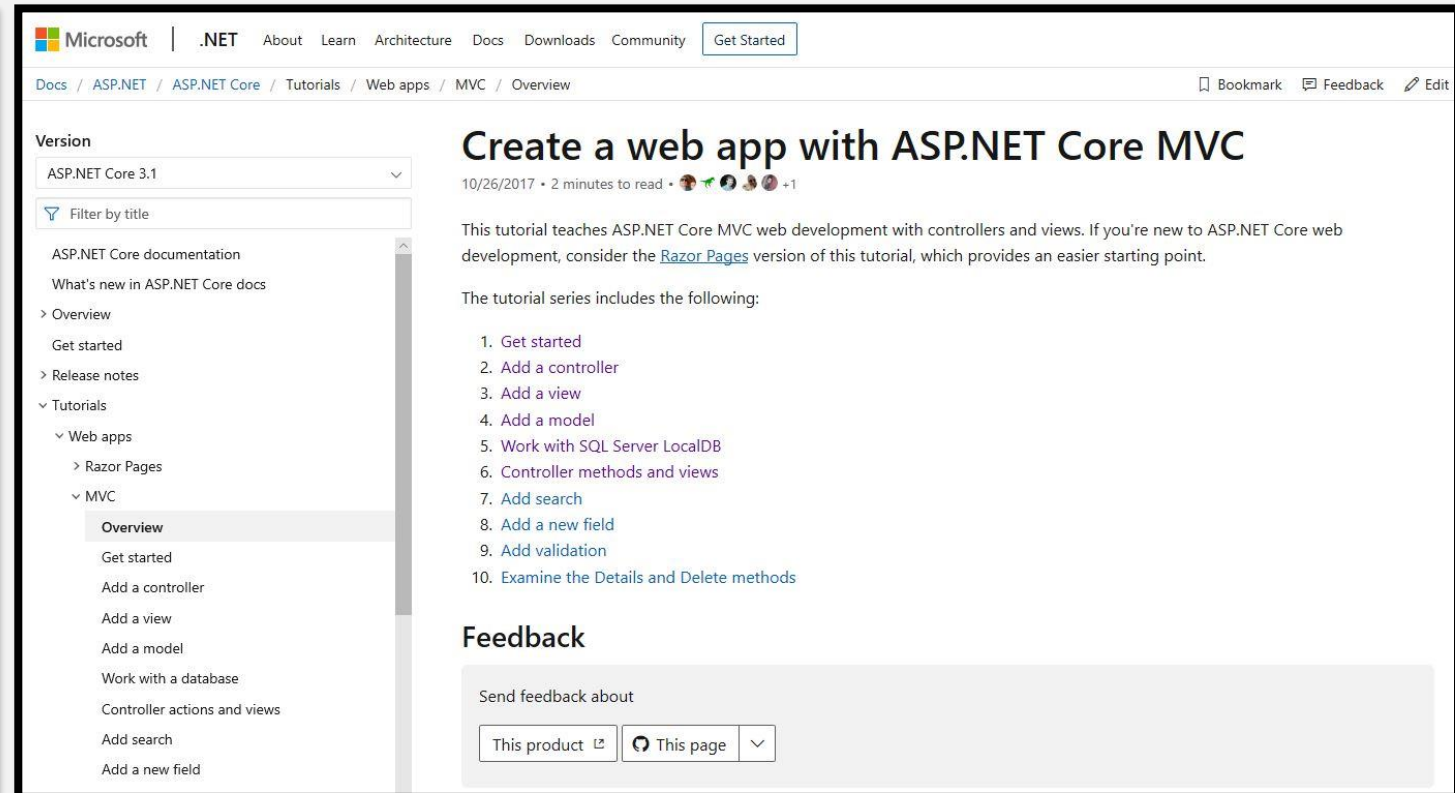
- I will use two websites to assist me in creating the website:
- First is the ASP.NET tutorial website (left)
- Second is the ASP.NET example (right)



The screenshot shows the 'What is ASP.NET?' video series page. The main video is 'What is ASP.NET?' (01:40). Below it are six smaller video thumbnails:

- ASP.NET - Making an ASP.NET Core Website [2 of 13] (06:31)
- ASP.NET - Adding Data to an ASP.NET Core Website [3 of 13] (09:55)
- ASP.NET - Adding a Service to an ASP.NET Core Website [4 of 13] (13:20)
- ASP.NET - Data in a Razor Page [5 of 13] (11:46)
- ASP.NET - Styling a Razor Page [6 of 13] (10:18)
- ASP.NET - Making a Simple API [7 of 13] (09:31)

The page also includes a star rating (5 stars) and a 'view episode' button.



The screenshot shows the 'Create a web app with ASP.NET Core MVC' tutorial page. The page includes a navigation menu on the left with links to 'Overview', 'Get started', 'Release notes', 'Tutorials', 'Web apps', and 'MVC'. The main content area features a list of steps for creating a web app with ASP.NET Core MVC:

1. Get started
2. Add a controller
3. Add a view
4. Add a model
5. Work with SQL Server LocalDB
6. Controller methods and views
7. Add search
8. Add a new field
9. Add validation
10. Examine the Details and Delete methods

The page also includes a 'Feedback' section with a 'Send feedback about' button and a dropdown menu to select the feedback target (This product, This page).

# LITERATURE TO ASSIST ME (PART 2)

## CHAPTER 7



### SportsStore: A Real Application

In the previous chapters, I built quick and simple MVC applications. I described the MVC pattern, the essential C# features and the kinds of tools that good MVC developers require. Now it is time to put everything together and build a simple but realistic e-commerce application.

My application, called *SportsStore*, will follow the classic approach taken by online stores everywhere. I will create an online product catalog that customers can browse by category and page, a shopping cart where users can add and remove products, and a checkout where customers can enter their shipping details. I will also create an administration area that includes create, read, update, and delete (CRUD) facilities for managing the catalog; and I will protect it so that only logged-in administrators can make changes.

My goal in this chapter and those that follow is to give you a sense of what real MVC Framework development is like by creating as realistic an example as possible. I want to focus on the MVC Framework, of course, and so I have simplified the integration with external systems, such as the database, and omitted others entirely, such as payment processing.

You might find the going a little slow as I build up the levels of infrastructure I need. Certainly, you *would* get the initial functionality built more quickly with Web Forms, just by dragging and dropping controls bound directly to a database. But the initial investment in an MVC application pays dividends, resulting in maintainable, extensible, well-structured code with excellent support for unit testing.

#### UNIT TESTING

I have made quite a big deal about the ease of unit testing in MVC, and about my belief that unit testing is an important part of the development process. You will see this demonstrated throughout this part of the book because I have included details of unit tests and techniques as they relate to key MVC features.

I know this is not a universal opinion. If you do not want to unit test, that is fine with me. To that end, when I have something to say that is purely about testing, I put it in a sidebar like this one. If you are not interested in unit testing, you can skip right over these sections, and the *SportsStore* application will work just fine. You do not need to do any kind of unit testing to get the technology benefits of ASP.NET MVC, although, of course, support for testing is a key reason for adopting the MVC Framework.

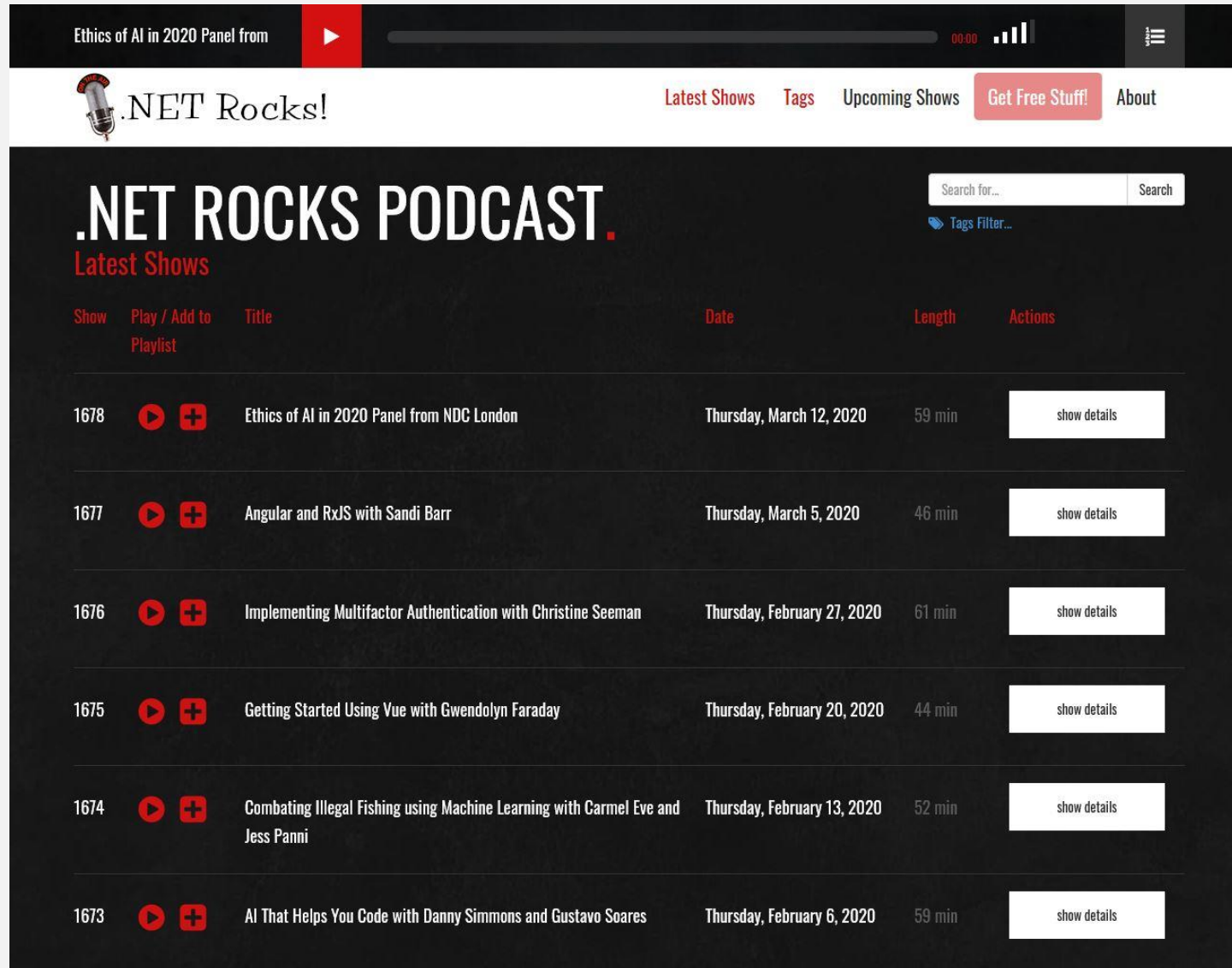
Most of the MVC features I use for the *SportsStore* application have their own chapters later in the book. Rather than duplicate everything here, I tell you just enough to make sense for the example application and point you to the other chapter for in-depth information.

I will call out each step needed to build the application, so that you can see how the MVC features fit together. You should pay particular attention when I create views. You will get some odd results if you do not follow the examples closely.

- I will also be referencing Chapter 7 of the MVC Core textbook in order to setup the online store
- The textbook uses a Sports Store as an example, and I will tailor the details to reflect a Podcast Store



# EXAMPLE PODCAST WEBSITE

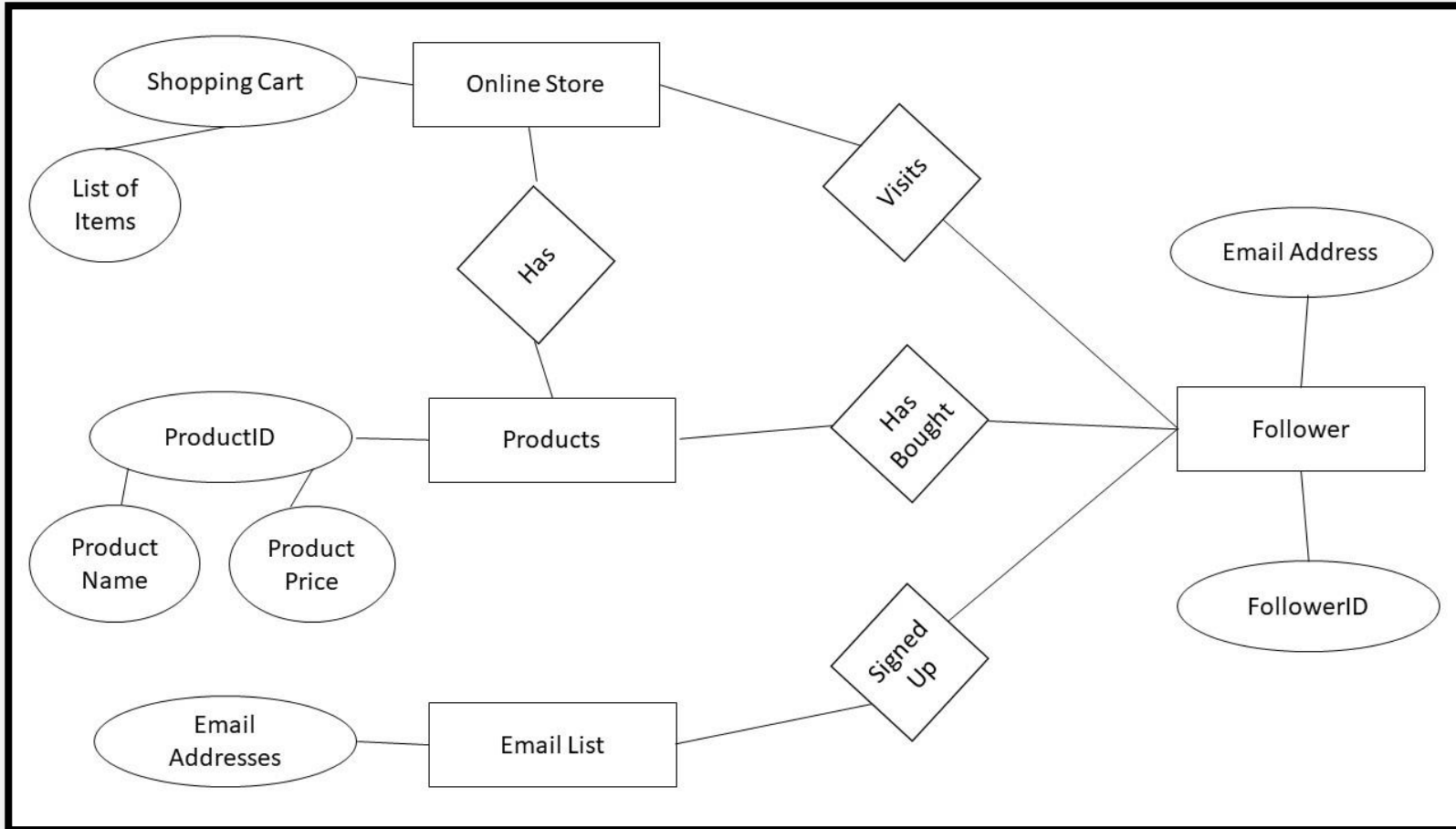


- An example podcast website would look like the photo on the left
- At the top would be a button to play the current episode
- On the bottom would be additional buttons to play previous episodes
- The menu bar would have links to the “About” page and the “Store” page

# PROJECT PLAN

- Step 1: Create a new ASP.NET Core Web Application project
- Step 2: Add Controllers, Views, and Model
- Step 3: Link to a database
- Step 4: Add content
- Step 5: Use Razor, HTML, and CSS to improve the user interface
- Step 6: Publish website using Azure

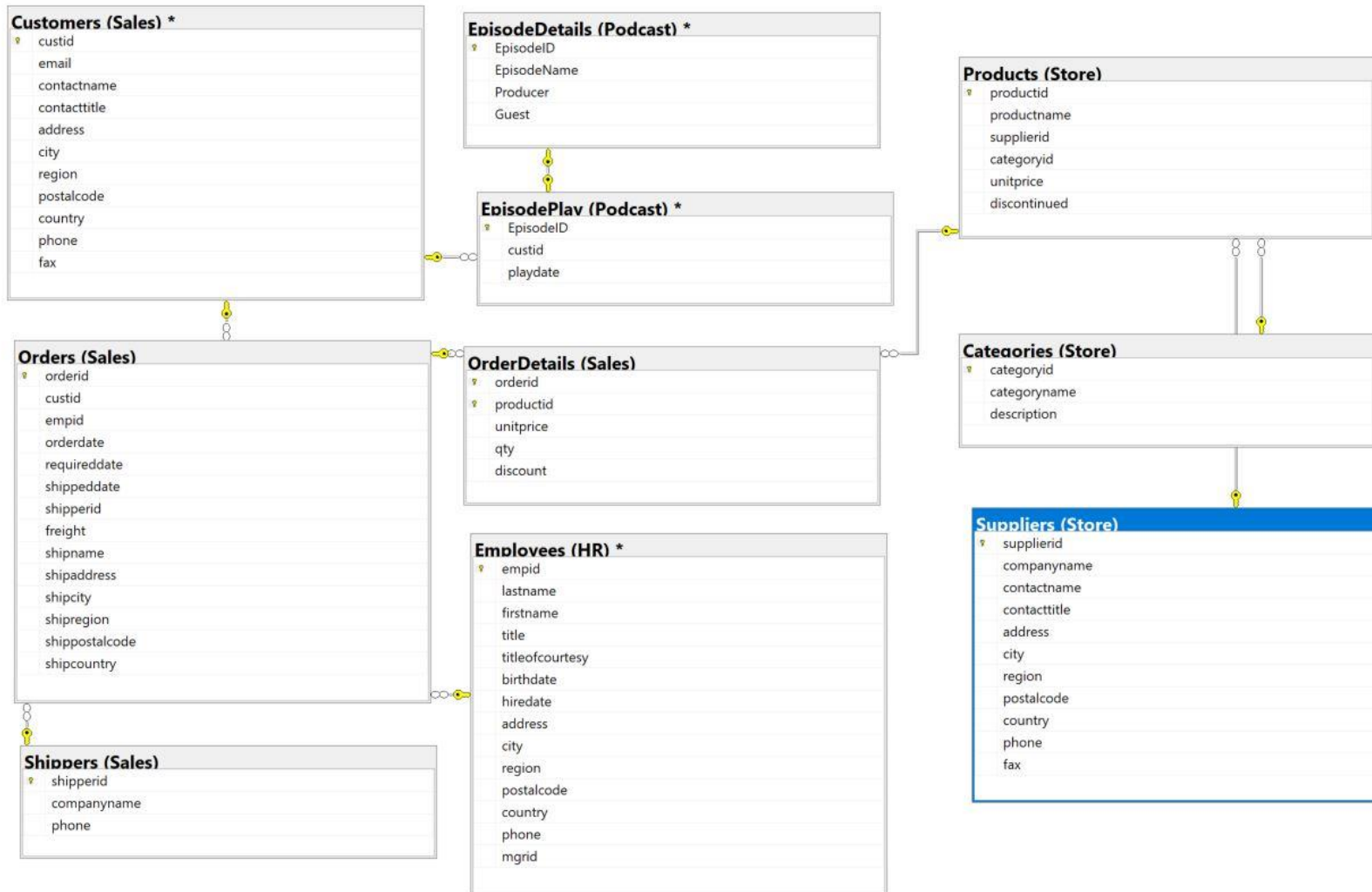
# INFORMATION ARCHITECTURE



- My original concept for the information architecture was very basic
- A follower of the podcast could create an account by signing up for the email list
- The follower could also visit the online store and fill their shopping cart with items
- The online store would be linked to a products table that contained information pertaining to all of the products that we sell

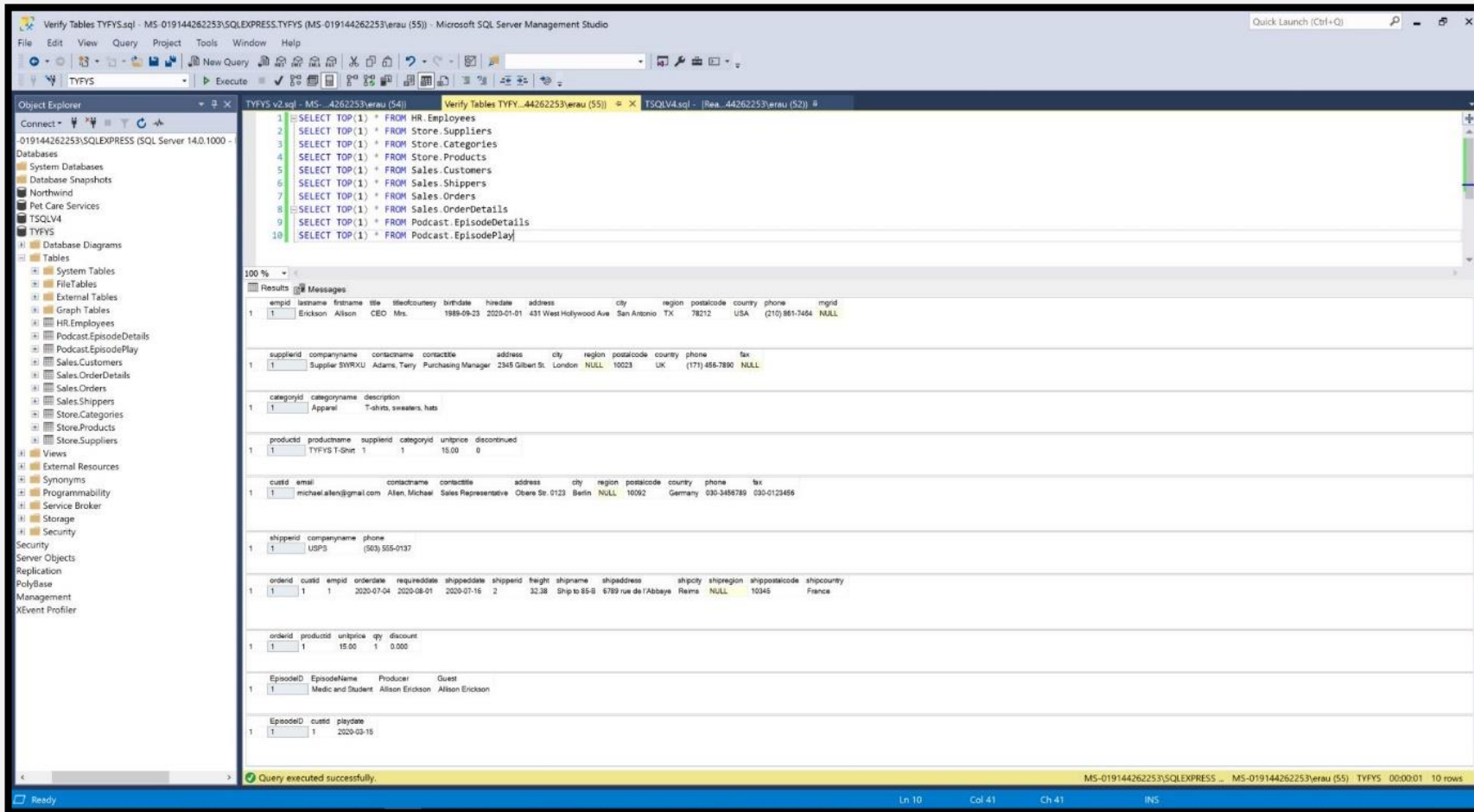


# DATABASE DESIGN



- I decided to model the TSQLV4 database and then tailor it for the podcast
- I decided to keep all of the TSQLV4 tables even though some weren't in my original information architecture
- I added two tables: EpisodeDetails and EpisodePlay
- EpisodeDetails lists all of the information for each episode
- EpisodePlay gets updated every time a listener plays an episode

# DATABASE IMPLEMENTATION



- I populated all of the tables with sample data in order to test the database
- My biggest challenge was updating tables if primary keys were already assigned
- For example, in EpisodeDetails I used varchar(10) for the podcast name.
- But the first episode's name contained 29 characters
- I found it was easier to edit the tables using the GUI in Object Explorer rather than updating my original code

# QUESTIONS