

# Student projects for *Murach's ASP.NET Core MVC*

The projects in this document let your students apply the skills they'll learn as they progress through *Murach's ASP.NET Core MVC*. If you review these projects, you'll see that they represent different levels of difficulty. As a result, you can assign projects that are appropriate for the skill levels of your students. In addition, you can modify the projects to make them more or less challenging.

In the project name, the first number specifies the chapter that the student should complete before starting the project. For example, the student should complete chapter 2 before starting project 2-1 or 2-2, the student should complete chapter 3 before starting project 3-1, and so on.

Project guidelines	2
Project 2-1 Build the Price Quotation app	3
Project 2-2 Build the Tip Calculator app	4
Project 3-1 Style the Price Quotation app	5
Project 4-1 Build the Contact Manager app	7
Project 5-1 Debug the Tip Calculator app	10
Project 6-1 Build the FAQ app	11
Project 6-2 Update the FAQ app	14
Project 7-1 Build the MyWebsite app	15
Project 8-1 Build the Trips Log app	18
Project 9-1 Build the PIG game app	20
Project 10-1 Build the Tic Tac Toe app	22
Project 11-1 Build the Quarterly Sales app	24
Project 12-1 Update the Trips Log app	26
Project 13-1 Add paging, sorting, and filtering to the Quarterly Sales app	29
Project 14-1 Add DI and unit testing to the Contact Manager app	30
Project 15-1 Add tag helpers and view components to the Blackjack app	31
Project 16-1 Add authorization to the Quarterly Sales app	33

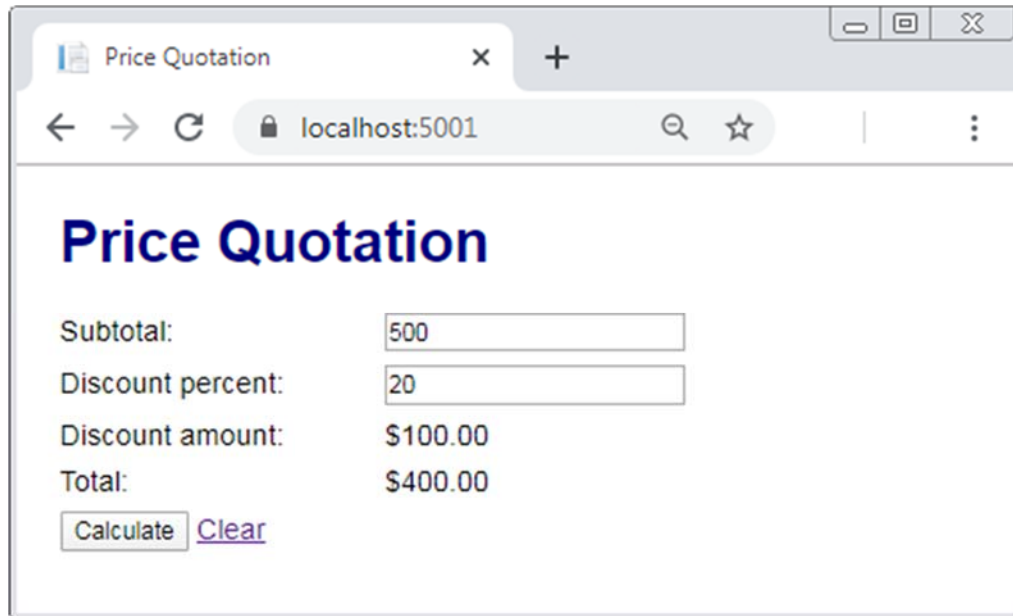
## Project guidelines

---

- If your instructor doesn't supply a starting point for a project, start your project from scratch or from a previous project that you've completed earlier.
- If you are doing a project in class with a time limit set by your instructor, complete as much of the project as you can in the time limit.
- Feel free to copy and paste code from the book apps or projects that you've already done into your solutions and to use your book as a guide to coding.
- When creating the filenames for your apps, please use the convention specified by your instructor. Otherwise, store the code in a folder named *first\_last\_app* where *first\_last* specifies your first and last name and *app* specifies the name of the app.
- When creating names for variables and functions, please use the guidelines and recommendations specified by *Murach's ASP.NET Core MVC* or by your instructor.

## Project 2-1 Build the Price Quotation app

For this project, you will build a single-page app like the one that's shown below.



The screenshot shows a web browser window with the title 'Price Quotation'. The address bar shows 'localhost:5001'. The page content includes a large blue heading 'Price Quotation'. Below the heading, there is a form with the following elements:

Subtotal:	<input type="text" value="500"/>
Discount percent:	<input type="text" value="20"/>
Discount amount:	\$100.00
Total:	\$400.00

At the bottom of the form, there are two buttons: 'Calculate' and 'Clear'.

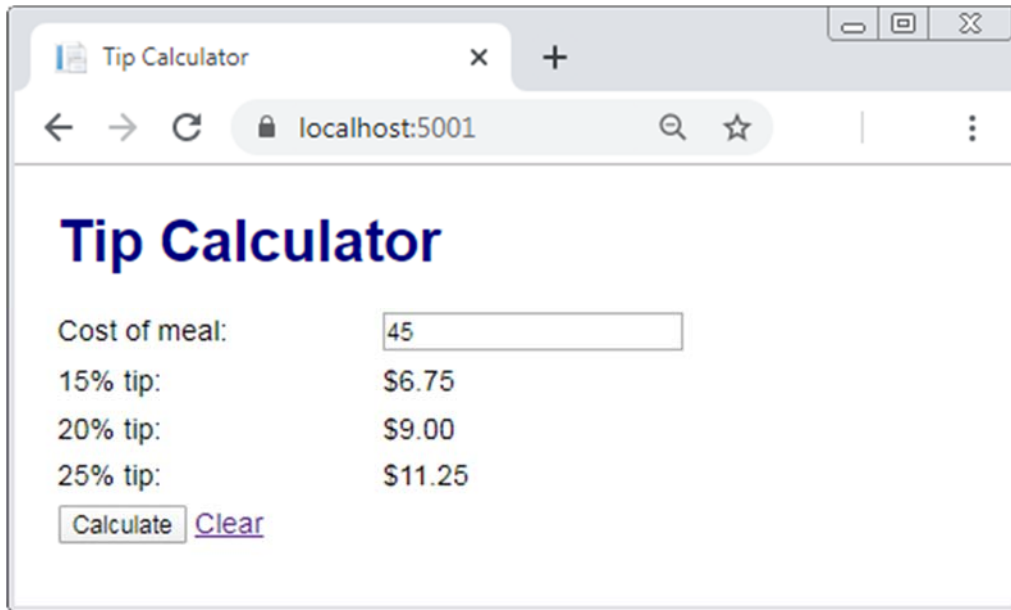
### Specifications

- When the app starts, it should display the Price Quotation page with no subtotal or discount percent, and it should set the discount amount and total to \$0.00.
- If the user enters a valid subtotal and discount percent and clicks the Calculate button, the app should calculate and display the discount amount and total.
- If the user enters invalid data and clicks the Calculate button, the app should display a summary of validation errors above the form.
- Here are the requirements for valid data:
  - The sales price is required and must be a valid number that's greater than 0.
  - The discount percent is required and must be a valid number from 0 to 100.
- If the user clicks the Clear link, the app should reset the form to how it was when the app first started.
- Use the MVC pattern. To do that, create a model class that stores the subtotal and discount percent and calculates the discount amount and total. Make sure to bind that model to the Razor view that displays the Price Quotation page shown above.
- Use a Razor layout to store the <html>, <head>, and <body> elements.
- Use a custom CSS style sheet to style the HTML elements so they appear as shown above.

## Project 2-2 Build the Tip Calculator app

---

For this project, you will build a single-page app like the one shown below.



### Specifications

- When the app starts, it should display an empty text box and tip amounts of \$0.00.
- If the user enters a valid meal cost and clicks the Calculate button, the app should calculate and display the tip amounts as shown above.
- If the user enters invalid data and clicks the Calculate button, the app should display a summary of validation errors above the form and \$0.00 for the tip amounts.
- The cost of the meal is required and must be a valid number that's greater than 0.
- If the user clicks the Clear link, the app should reset the page to how it was when the app first started.
- Use the MVC pattern. To do that, create a model class that stores the cost of the meal and includes a helper method for calculating the tip percentages.
- Use a Razor layout to store the <html>, <head>, and <body> elements.
- Use a custom CSS style sheet to style the HTML elements so they appear as shown above.

## Project 3-1 Style the Price Quotation app

For this project, you will use Bootstrap to improve the appearance of the app described in project 2-1. When you're done, the app should appear as shown below.

The image displays two browser windows showing the 'Price Quotation' application. Both windows are running on 'localhost:5001'.

**Desktop View (Top Window):**

- Header: A green banner with a white icon of a price tag and the text 'Price Quotation'.
- Form Fields:
  - Subtotal: Input field with a red error message 'Please enter a sale price.'
  - Discount percent: Input field with a red error message 'Please enter a discount percent.'
  - Discount amount: \$0.00
  - Total: \$0.00
- Buttons: 'Calculate' and 'Clear' buttons.

**Mobile View (Bottom Window):**

- Header: A green banner with a white icon of a price tag and the text 'Price Quotation'.
- Form Fields:
  - Subtotal: Input field with a red error message 'Please enter a sale price.'
  - Discount percent: Input field with a red error message 'Please enter a discount percent.'
  - Discount amount: \$0.00
  - Total: \$0.00
- Buttons: 'Calculate' and 'Clear' buttons.

## Specifications

- Use a horizontal form for small, medium and large (tablets, desktops, large desktops) screen sizes. For this form to lay out properly, you should:
  - For each row, use a `<div>` element that's assigned to the row and form-group classes.
  - Code each `<input>` element within a `<div>` element that's assigned to the correct col class.
  - Display a validation message in the third column.

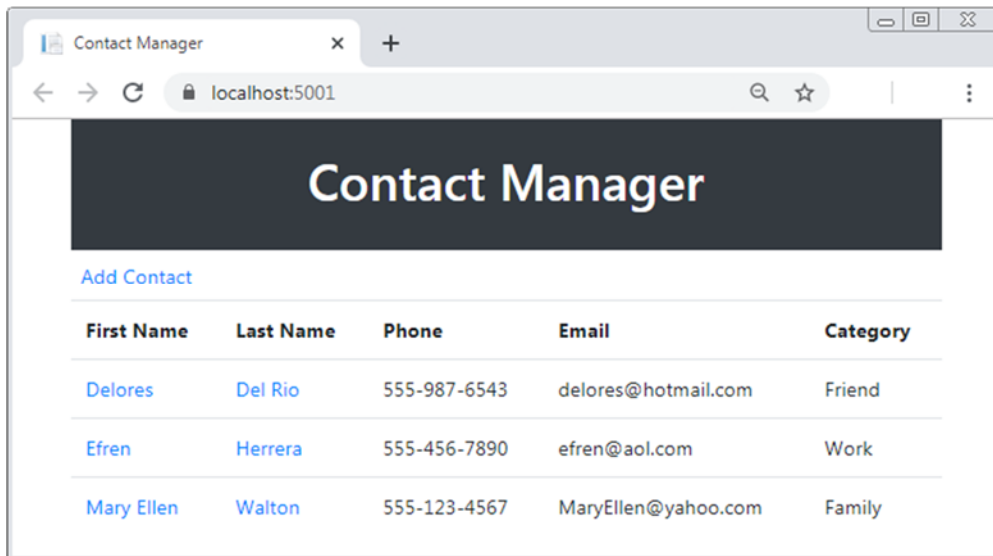
Note: In the first printing of this book, the code in figures 3-7 and 3-10 isn't correct. It should be coded as described by the corrections PDF that you can download from [murach.com](http://murach.com).

- Use a vertical form on extra small screens (phones).
- Start the page with a jumbotron component that uses one of Bootstrap's context classes to set the background color. In addition, it should center the text and make it white.
- In the jumbotron, specify an `<h1>` element that displays the name of the app and a Font Awesome icon named `money-bill-alt`.
- Style the Calculate button and the Clear link as buttons and set their background color to the same color that's used by the jumbotron.
- Display the validation messages in red.
- Add margin and/or padding between elements as needed to improve the look of the form.
- Override the Bootstrap style for the `<h1>` element in the jumbotron so that its font size is 4 rem units.

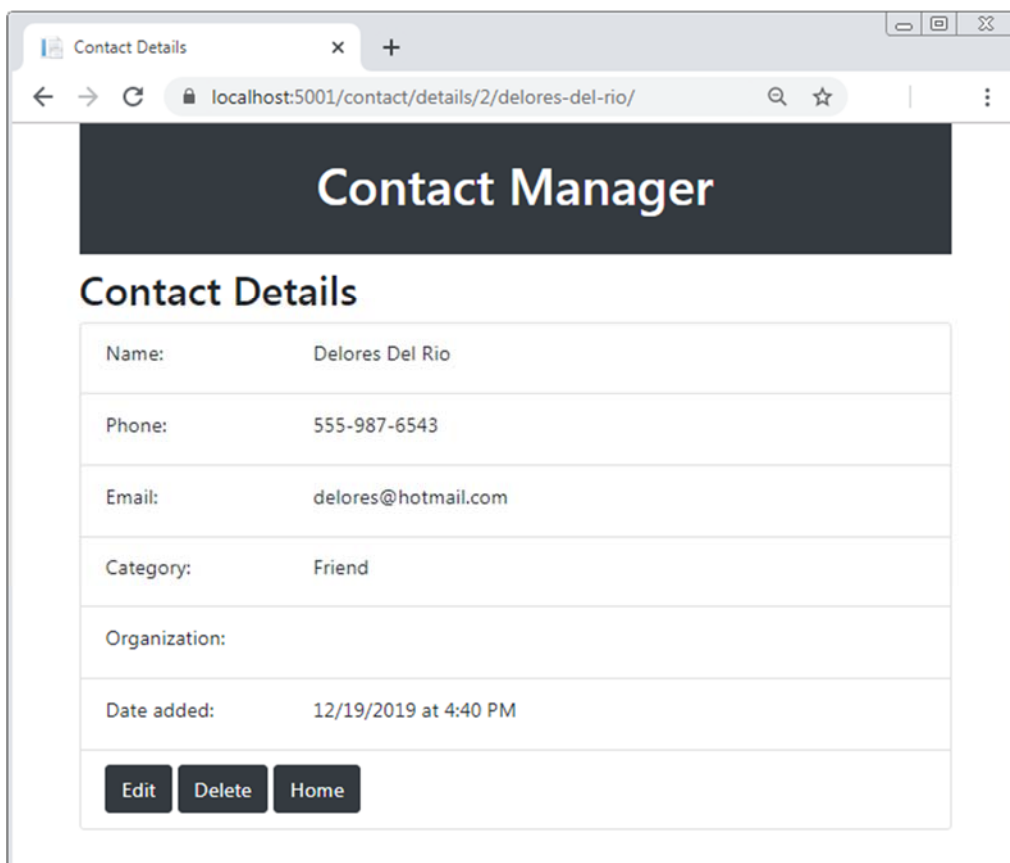
## Project 4-1 Build the Contact Manager app

For this project, you will build a multi-page, data driven app like the one that's shown below.

### The Home page



### The Details page



### The Add and Edit pages (both use the same view)

The image displays two browser windows side-by-side, both showing the 'Contact Manager' application. The left window is titled 'Add Contact' and shows a form with fields for 'First name', 'Last name', 'Phone', 'Email', 'Category' (a dropdown menu with 'select a category' selected), and 'Organization'. At the bottom are 'Add' and 'Cancel' buttons. The right window is titled 'Edit Contact' and shows the same form with pre-filled data: 'First name' is 'Delores', 'Last name' is 'Del Rio', 'Phone' is '555-987-6543', 'Email' is 'delores@hotmail.com', and 'Category' is 'Friend'. At the bottom are 'Edit' and 'Cancel' buttons. Both windows have a dark header bar with the text 'Contact Manager'.

### The Delete page

The image shows a browser window titled 'Delete Contact' for the 'Contact Manager' application. The URL is 'localhost:5001/contact/delete/2/delores-del-rio/'. The page features a dark header bar with the text 'Contact Manager'. Below the header, the title 'Delete Contact' is displayed, followed by a confirmation message: 'Do you really want to delete Delores Del Rio?'. At the bottom are 'Yes' and 'No' buttons.



## Specifications

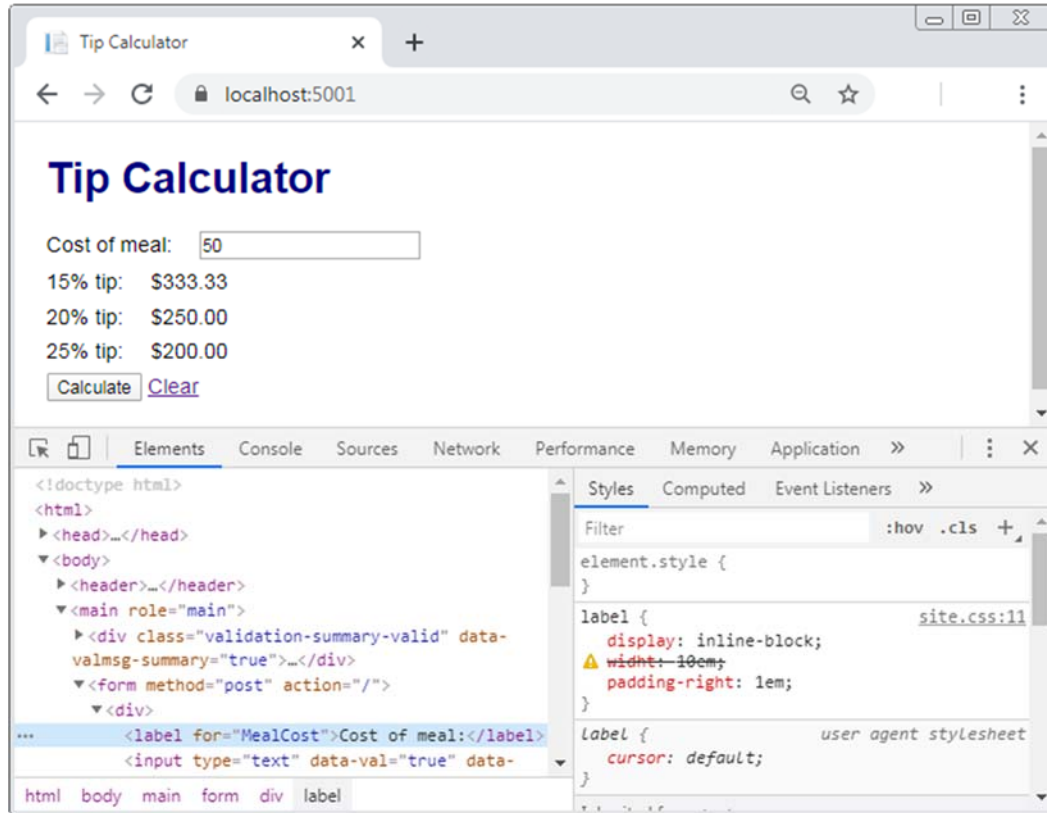
- When the app starts, it should display a list of contacts and a link to add a contact.
- If the user clicks the first or last name of a contact, the app should display the Detail page for that contact.
- The Details page should include buttons that allow the user to edit or delete the contact. Before deleting a contact, the app should display the Delete page to confirm the deletion.
- To reduce code duplication, the Add and Edit pages should both use the same view. This view should include a drop-down for Category values.
- The Add and Edit pages should *not* include the Date Added field that's displayed by the Details page. That field should only be set only by code when the user first adds a contact.
- If the user enters invalid data on the Add or Edit page, the app should display a summary of validation errors above the form.
- Here are the requirements for valid data:
  - The Firstname, Lastname, Phone, Email, and CategoryId fields are required.
  - The Organization field is optional.

Note: Since the CategoryId field is an int (see domain model specifications below), you can't use the Required validation attribute with it. However, you can use the Range attribute to make sure the value of CategoryId is greater than zero.

- If the user clicks the Cancel button on the Add page, the app should display the Home page.
- If the user clicks the Cancel button on the Edit page, the app should display to the Details page for that contact.
- The domain model classes for contacts and categories should use primary keys that are generated by the database.
- The Contact class should have a foreign key field that relates it to the Category class. It should also have a read-only property that creates a slug of the contact's first and last name that can be added to URLs to make them user friendly.
- Use EF Code First to create a database based on your domain model classes. Include seed data for the categories and one or more contacts.
- Use a Razor layout to store the <html>, <head>, and <body> elements.
- Use Bootstrap to style the views. If necessary, use a custom CSS style sheet to override Bootstrap classes.
- Use the default route with an additional route segment that allows an optional slug at the end of a URL.
- Make the app URLs lowercase with trailing slashes.

## Project 5-1 Debug the Tip Calculator app

For this project, you will debug an existing app that has errors. Your instructor should provide you with this app.



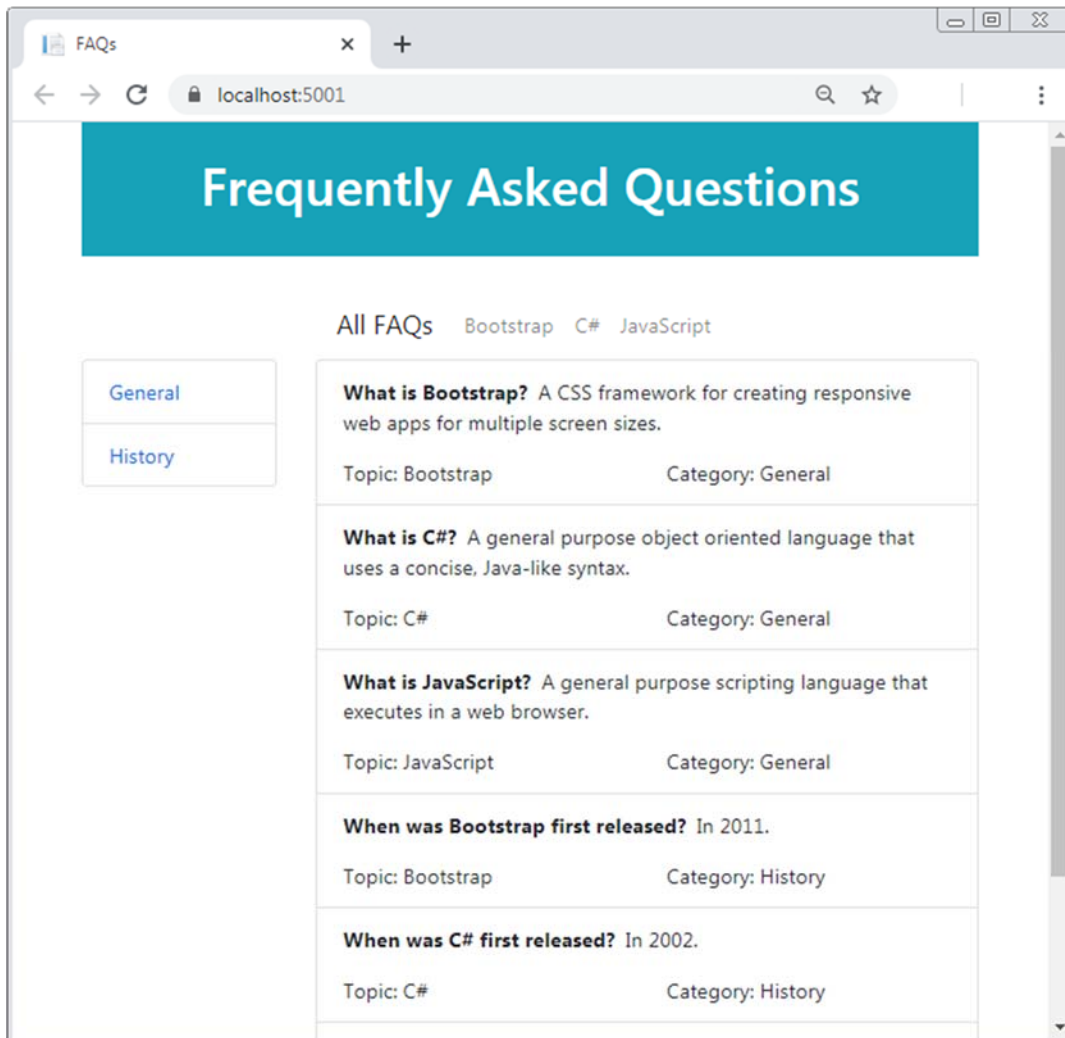
### Specifications

- Your instructor should provide you with a folder named 5-1\_TipCalculator that contains the Tip Calculator app that you will debug for this project.
- Use the Error List window to find and correct any syntax errors that display when you try to run the app.
- Use the Internal Server Error page and its stack trace to find and correct any unhandled exceptions occur when you run the app.
- Set a breakpoint in the model class and step through its code to find the error that leads to the app calculating an incorrect tip amount.
- Use the Developer Tools (as shown above) to find and correct the CSS issue that prevents the label elements from displaying with the expected width.

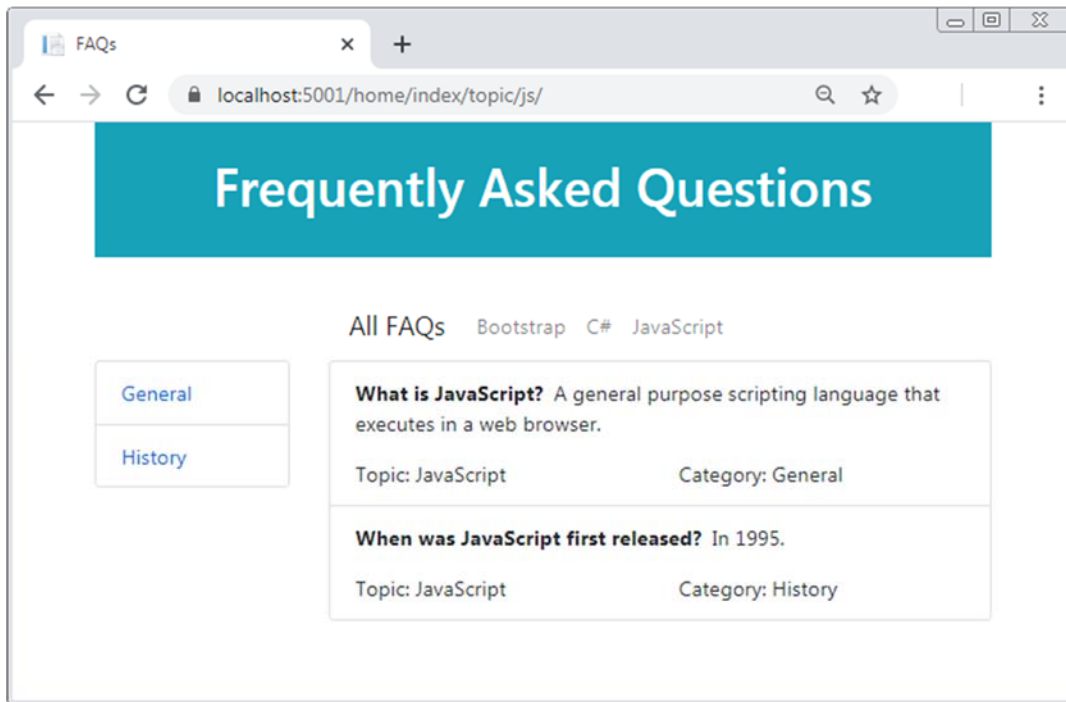
## Project 6-1 Build the FAQ app

For this project, you will build a multi-page, data driven app with multiple routes like the one shown below.

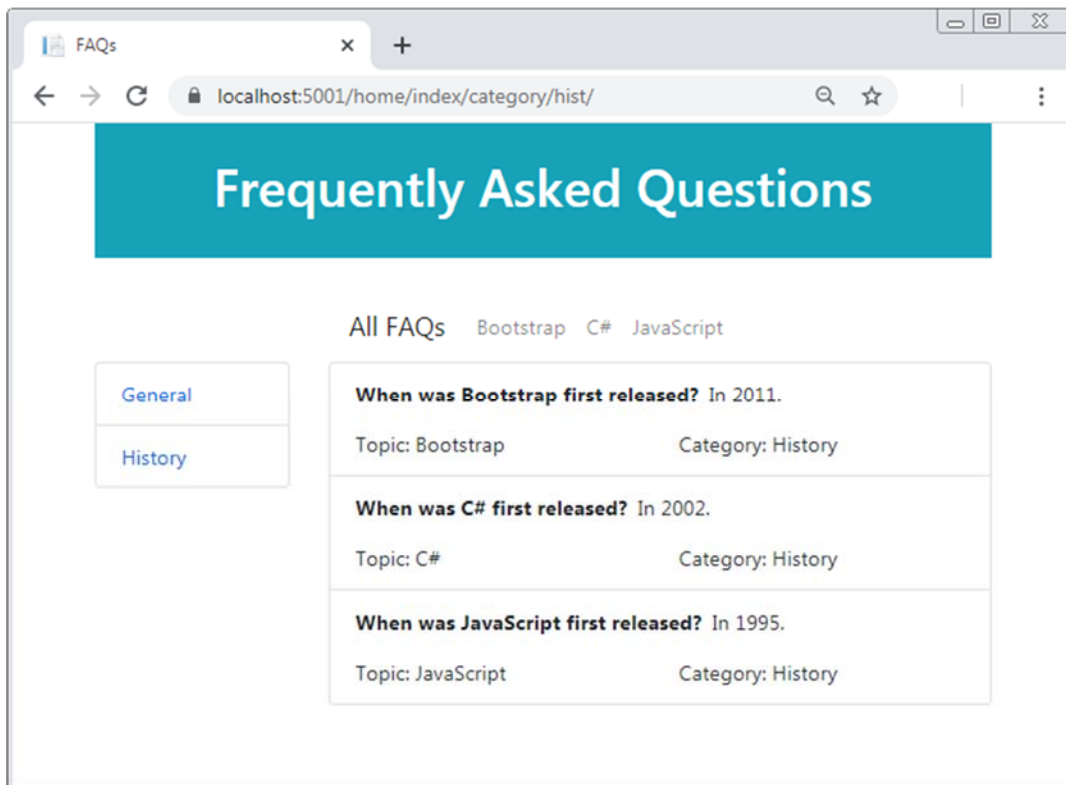
### The Home page on initial load



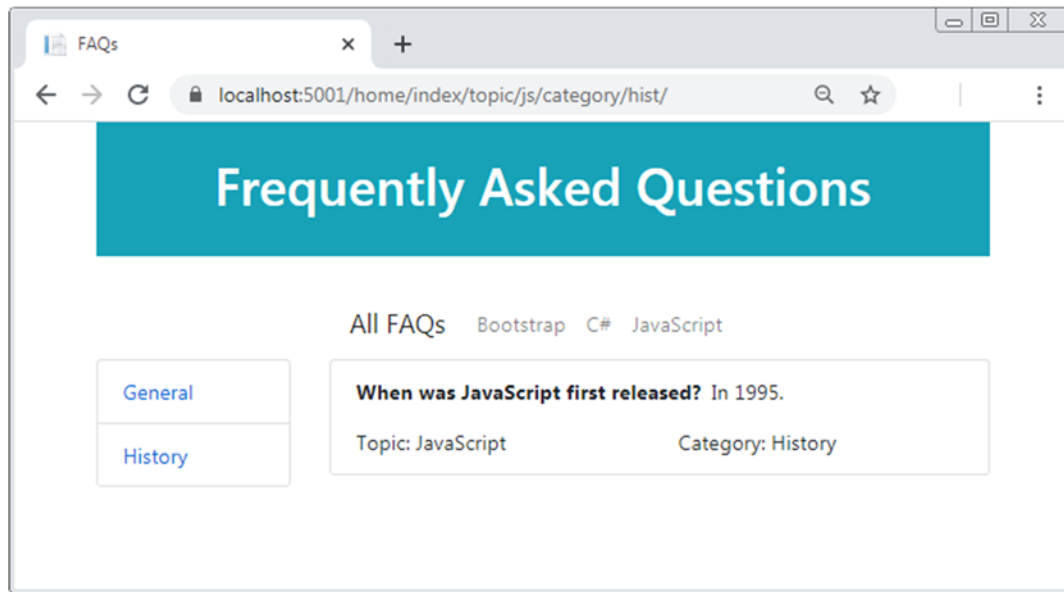
### The Home page with a topic selected



### The Home page with a category selected



## The Home page with a topic and category selected



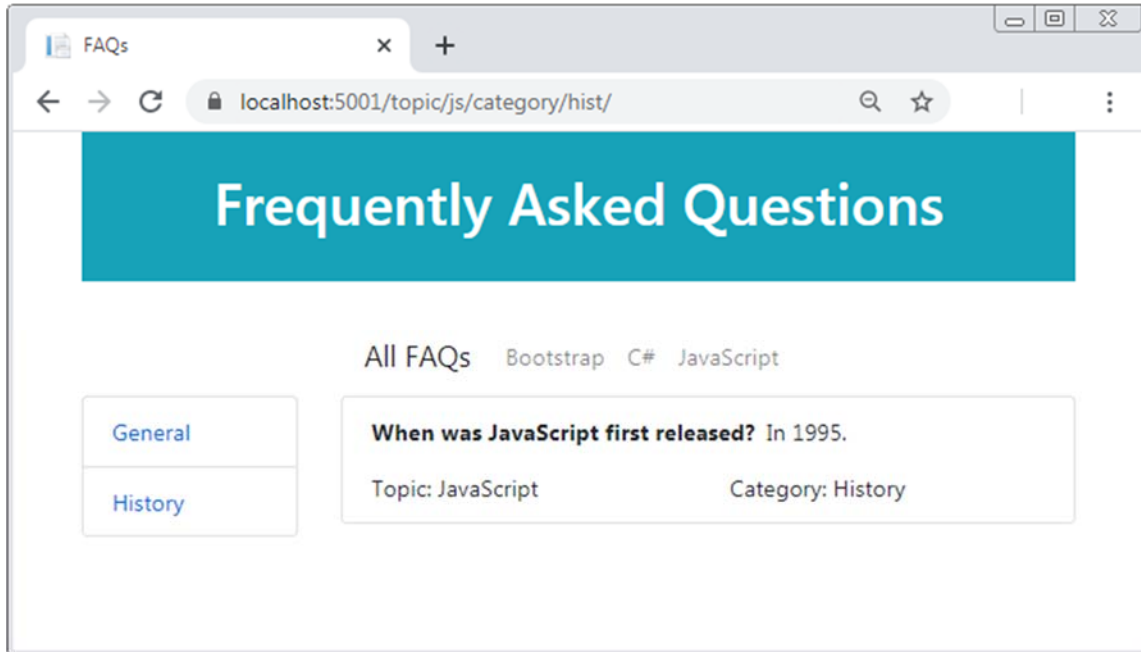
## Specifications

- When the app starts, it should display a list of all frequently asked questions (FAQs).
- The Home page should have a Bootstrap navbar across the top with links that filter the FAQs by topic. One of the links should clear any filtering and display all the FAQs.
- The Home page should have a Bootstrap list group to the left of the list of FAQs with links that filter the FAQs by category.
- Here's how the filtering should work:
  - On initial load, or when the user clicks the All FAQs link, the app should display all the FAQs.
  - When the user clicks a category link, the app should filter FAQs by category. If the user subsequently clicks a topic link, the FAQs will be filtered by topic and by category.
  - When the user clicks a topic link, the app should filter FAQs by topic. If the user subsequently clicks a category link, the app should filter the FAQs by topic and by category.
- Use custom routes to create the links for filtering. Use static segments to distinguish the topic route from the category route.
- The FAQ model class should have a primary key that's generated by the database.
- The Topic and Category model classes should have primary keys that are strings. Use primary key values that will be user friendly in a URL.
- The FAQ class should have foreign key fields that relate it to the Topic and Category classes.
- Use EF Code First and migrations to create a database based on your model classes. Include seed data for the topics, categories, and FAQs. (You can copy the data shown above, or you can create your own topics, categories, and FAQs.)
- The URLs for the app should be lowercase with trailing slashes.

## Project 6-2 Update the FAQ app

For this project, you will update the app from project 6-1 to use attribute routing.

### The Home page with a topic and category selected



### Specifications

- The URLs for the app should no longer include the “home/index” segments.
- Here are the URLs the app should use:

On initial load or no filtering (All FAQs):  
`https://localhost:5001`

When filtering by topic:  
`https://localhost:5001/topic/js/`

When filtering by category:  
`https://localhost:5001/category/hist/`

When filtering by topic and category:  
`https://localhost:5001/topic/js/category/hist/`

Note: To make this work, you need to apply multiple attributes to the `Index()` action method of the Home controller, including an attribute for the default route. Be sure to apply the attributes in the correct order, from most to least specific.

## Project 7-1 Build the MyWebsite app

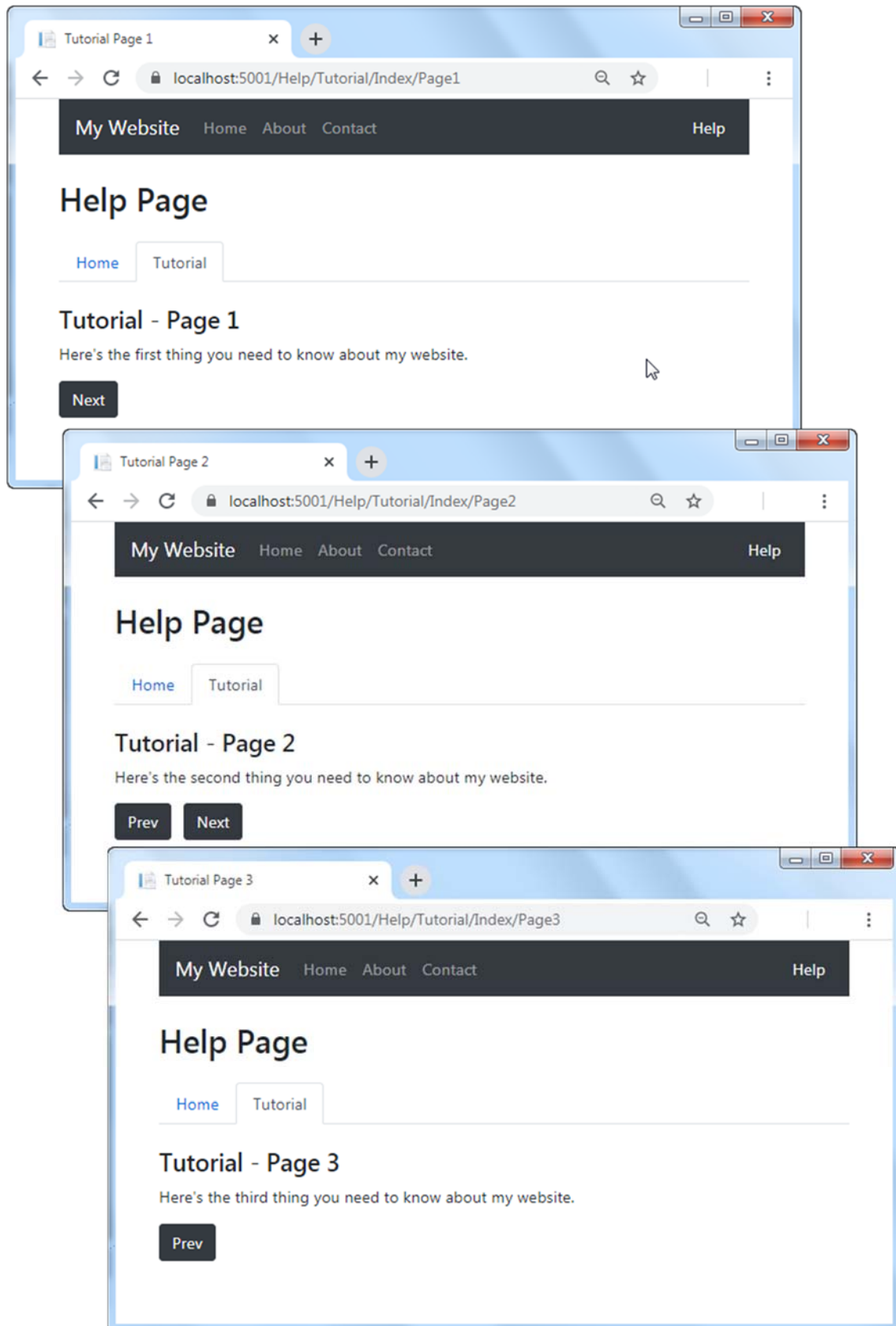
---

In this project, you'll build a multi-page web app with multiple views, nested layouts, layout sections, and a Help area. It will have the directory structure and user interface that's shown below.

### The directory structure

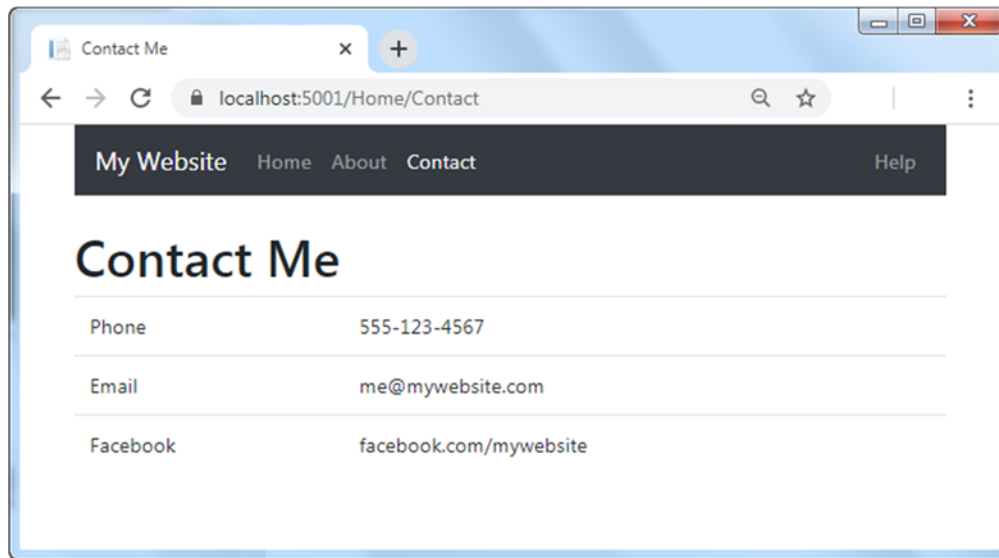
```
/Areas
  /Help
    /Controllers
      /HomeController.cs
      /TutorialController.cs
    /Views
      /Home
        /Index.cshtml
      /Shared
        /_HelpLayout.cshtml
      /Tutorial
        /Page1.cshtml
        /Page2.cshtml
        /Page3.cshtml
      /_ViewImports.cshtml
      /_ViewStart.cshtml
/Controllers
  /HomeController.cs
/Views
  /Home
    /About.cshtml
    /Contact.cshtml
    /Index.cshtml
  /Shared
    /_Layout.cshtml
  /_ViewImports.cshtml
  /_ViewStart.cshtml
```

## The three Tutorial views in the Help area





## The Contact view

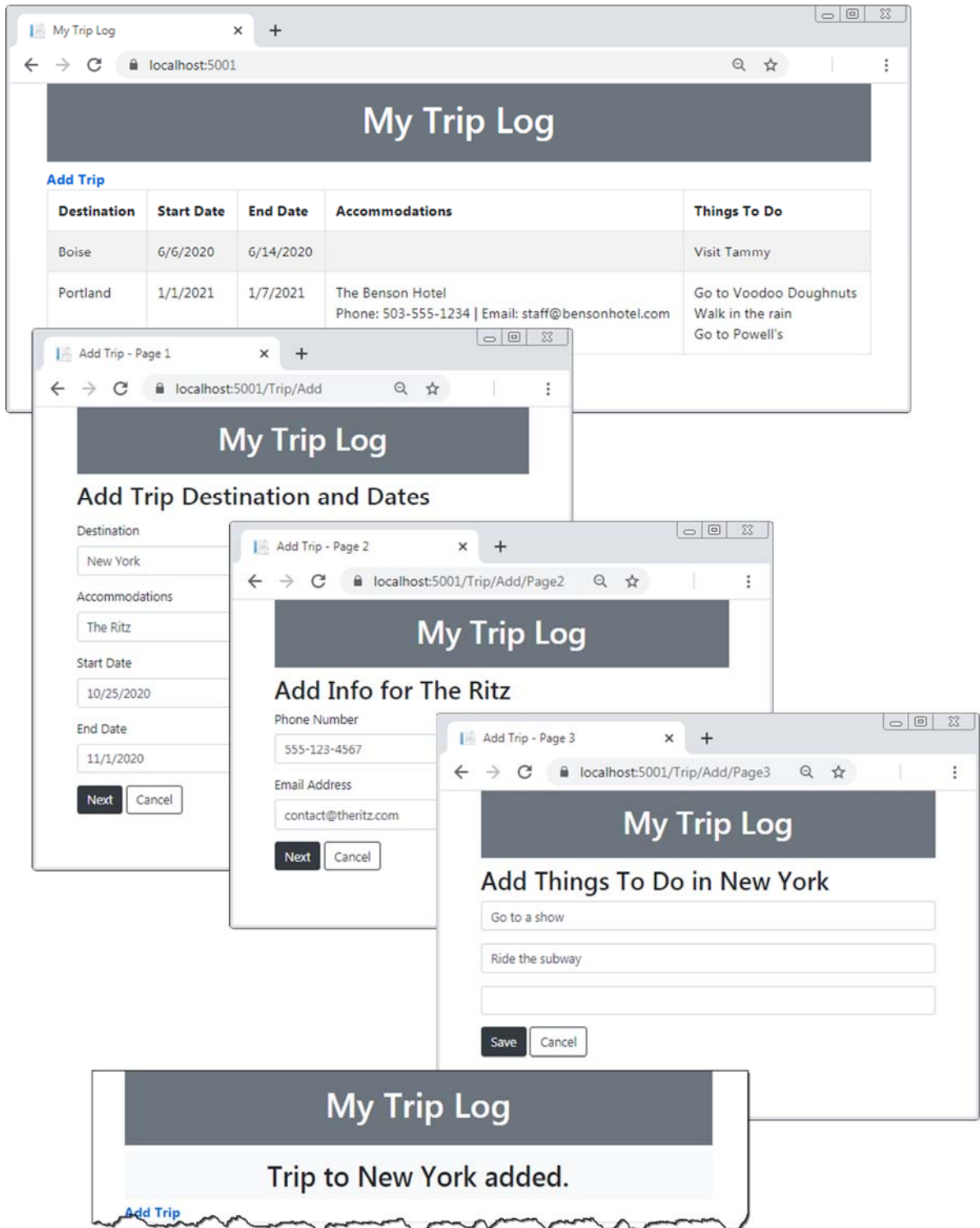


## Specifications

- Create an app with the controllers and views shown above. Review chapter 6 for how to set up routing for an area. The app doesn't need model classes, so you don't need to create a Models folder.
- Use a Razor layout to store the `<html>`, `<head>`, and `<body>` elements. Use a Razor view start file to apply this layout to the views in the app, and a Razor view imports file to add using statements.
- The layout should have a Bootstrap navbar and mark the current link in the navbar as active.
- The layout should have a section within a `<header>` element that's required. Each view that uses the layout should use that section to add an `<h1>` element with an appropriate message for that view.
- Use a nested Razor layout in the Help area that uses the main layout. Use a view start file to apply this nested layout to the views in the Help area, and a view imports file to add using statements.
- The nested layout should have Bootstrap navigation links styled as tabs, and it should mark the current tab as active.
- The default area should have Home, Contact, and About pages. The Contact view should get a collection of contact data from the Home controller and display that data to the user.
- The Help area should have Home and Tutorial pages. The `Index()` action method of the Tutorial controller should determine which view file to use based on the value of the id segment of the route.

## Project 8-1 Build the Trips Log app

For this project, you will build a data-driven web app that uses view models to pass data from controllers to views, the ViewBag object to pass data from views to the layout, and TempData to persist data between requests.

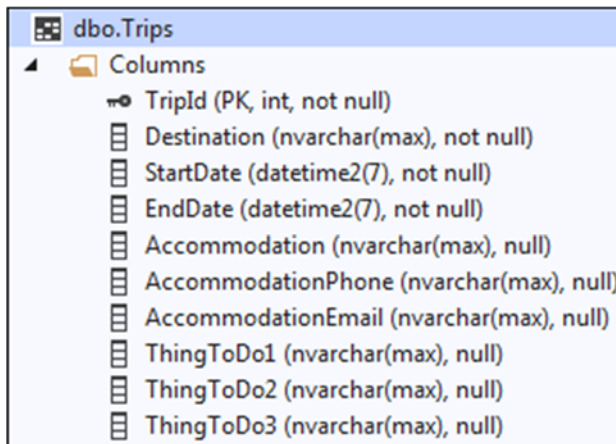


## Specifications

- When the app starts, it should display trip data and an Add Trip link.
- The layout for this web app should display a banner at the top of the page, and a subhead below it. However, the subhead should only display if there's text in the ViewData property for it.
- When the user clicks the Add Trip link, a three page data entry process should start.
- On the first page, the user should enter data about the trip destination and dates. The Accommodations field on this page should be optional, and the rest should be required.
- The second page should only display if the user entered a value for the Accommodations field on the first page. On this page, the user should enter data about the accommodations. The accommodation value the user entered should display in the subhead, and the fields should be optional.
- On the third page, the user should enter data about things to do on their trip. The destination the user entered on the first page should display in the subhead, and the fields should be optional.
- When the user clicks the Next button on the first or second page, the web app should save the data posted from the page in TempData. Use this data to get the user entries to display in the subheads as needed, but make sure any data that needs to be saved to the database persists in TempData.
- When the user clicks the Save button on the third page, the web app should save the data posted from the page and the data in TempData to the database. Then, the Home page should display with a temporary message that the trip has been added.
- When the user clicks the Cancel button on any of the Add pages, the data in TempData should be cleared and the Home page should display. You can use this statement to clear the data:

```
TempData.Clear();
```

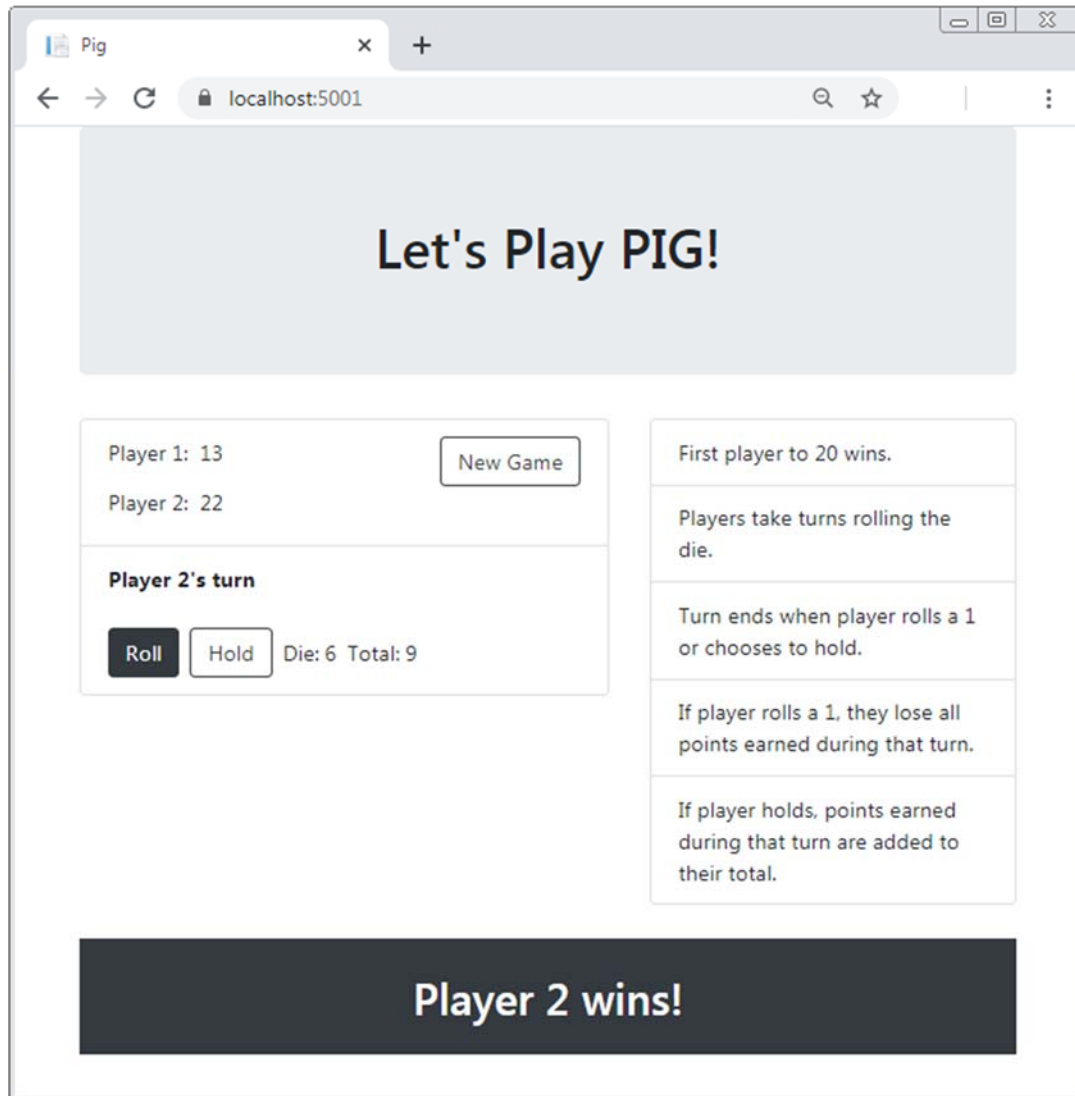
- To keep things simple, store all fields for the trip in a single table like this:



dbo.Trips	
Columns	
PK	TripId (PK, int, not null)
	Destination (nvarchar(max), not null)
	StartDate (datetime2(7), not null)
	EndDate (datetime2(7), not null)
	Accommodation (nvarchar(max), null)
	AccommodationPhone (nvarchar(max), null)
	AccommodationEmail (nvarchar(max), null)
	ThingToDo1 (nvarchar(max), null)
	ThingToDo2 (nvarchar(max), null)
	ThingToDo3 (nvarchar(max), null)

## Project 9-1 Build the PIG game app

For this project, you will build a single-page app (SPA) that uses session state to persist data between requests.



### Specifications

- When the app starts, or when the user clicks New Game, the scores for both players, the value for the die, and the total for the turn should all be zero.
- When it's their turn, a player can click Roll to roll the die or Hold to end their turn and add the point total for that turn to their score.
- When a player clicks Roll, the app should generate and display a random number between 1 and 6 (see below). If the number is 1, the point total for the turn should be set to zero, and that player's turn should end. Otherwise, the number rolled should be added to the point total for the turn, and the player's turn should continue.
- When a player clicks Hold, the app should add their points for that turn to their score and then end the turn. If the score is greater than the number of points needed to win, the app should end the game and display the name of the winner.

## Specifications (continued)

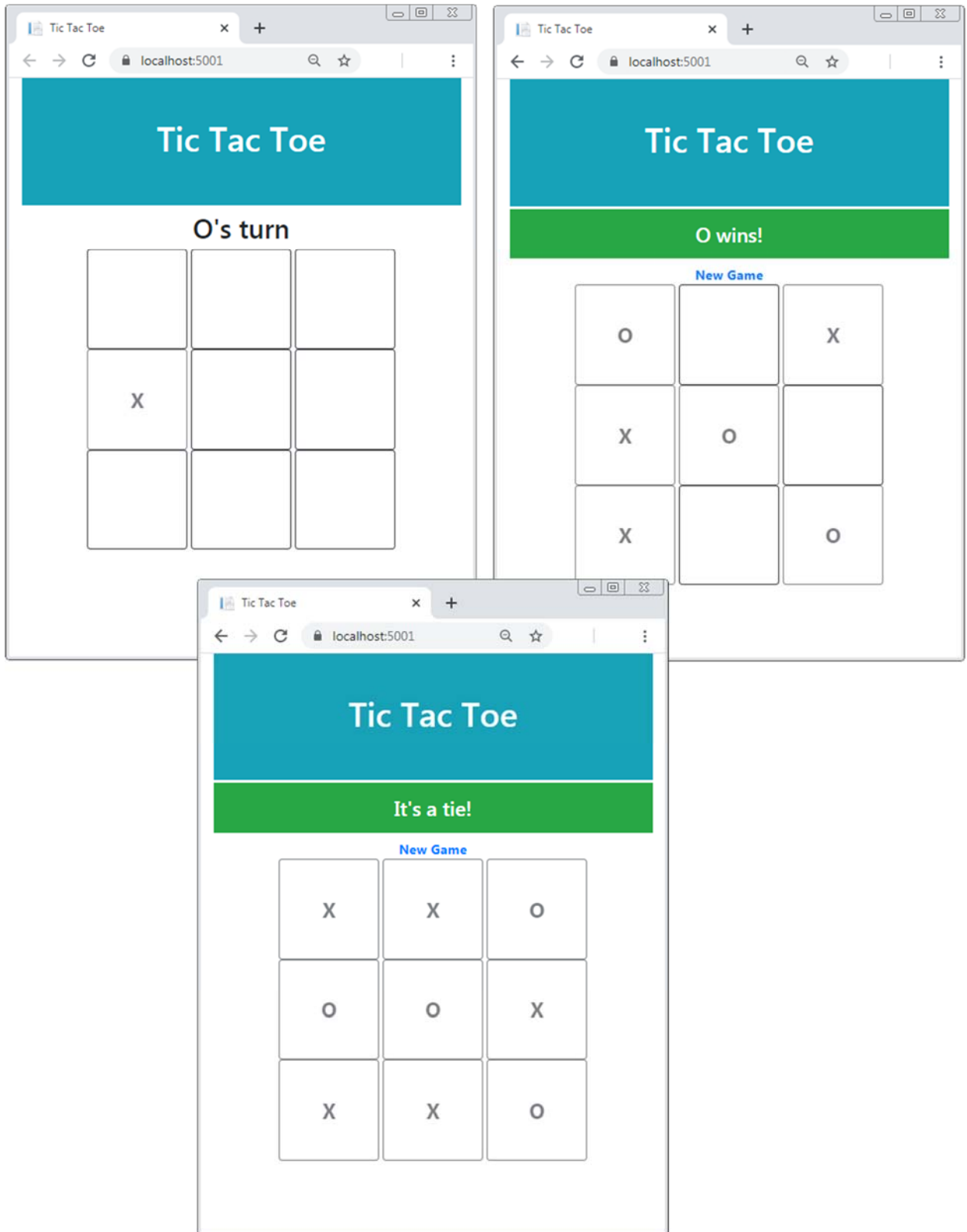
- Clicking the browser's Refresh button should only redisplay the page, not post the previous button click again.
- Use the Random class in the System namespace to generate a random number like this:

```
Random rand = new Random();  
int roll = rand.Next(1, 7);
```

- Use session state to store game data between requests.

## Project 10-1 Build the Tic Tac Toe app

For this project, you will build a single-page app that uses model binding to pass data between controllers and views.

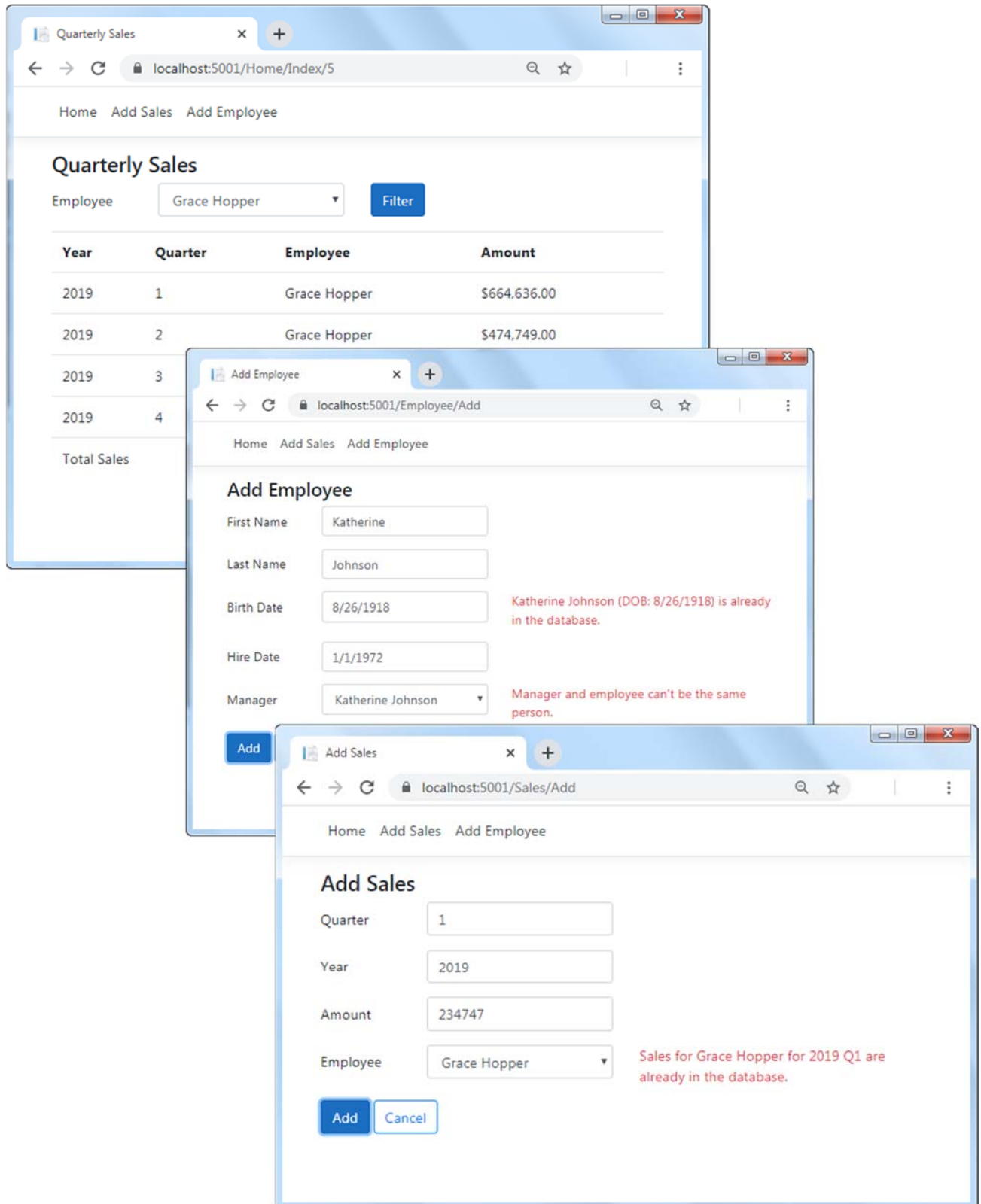


## Specifications

- When the app starts, it should display nine `<button>` elements, styled as above to appear as a 3-x-3 grid. Above the grid, an `<h2>` element should say that it's X's turn.
- When the user clicks a button, the page should post to the server, where the X or O for that button is saved.
- When the page re-loads, the clicked `<button>` element should be marked X or O and should be disabled. In addition, clicking the browser's Refresh button should redisplay the page, not post the previous click again.
- After each turn, the `<h2>` element above the grid should say whose turn it is.
- When there are three Xs or three Os in a line (vertically, horizontally, or diagonally), the game is over and the player with the three marks in a line wins. When this happens, the app should say who won and replace the `<h2>` element with a New Game link.
- When there are no more blank cells but there's no winner, then the game ends in a tie. When this happens, the app should notify the user and replace the `<h2>` element with a New Game link.
- When the user clicks the New Game link, a new grid of `<button>` elements should display. Above the grid, an `<h2>` element should say that it's X's turn.

## Project 11-1 Build the Quarterly Sales app

In this project, you'll validate the data a user enters in a data-driven web app.





## Specifications

- When the app starts, it should display quarterly sales data and provide a drop-down that allows a user to filter by employee.
- The web app should have a page to enter a new employee, and a page to enter new sales data. To keep things simple, this app doesn't provide edit or delete capabilities.
- Here are the requirements for valid data for a new employee:
  - First name, last name, date of birth, date of hire, and manager are required.
  - Date of birth and date of hire must be valid dates that are in the past.
  - Date of hire must not be before 1/1/1995, the date the company was founded.
  - A new employee may not have the same first name, last name, and date of birth of an employee that's already in the database.
  - A new employee may not have a manager with the same first name, last name, and date of birth. That is, employee and manager may not be the same person.
- Here are the requirements for valid data for a new sales amount:
  - Quarter, year, amount, and employee are required.
  - Quarter must be between 1 and 4.
  - Year must be after the year 2000.
  - Amount must be greater than 0.
  - New sales data may not have the same quarter, year, and employee as sales data that's already in the database.
- The web app should include the jQuery validation libraries so validation happens on the client, as a convenience to users. However, for security, all validation should take place on the server as well. To keep things simple, you can make your custom validation attributes only validate on the server. Thus, they will only fire after all client-side validation has passed.
- To make it possible to enter a new employee, you can edit the context file so it provides some seed data so the database includes at least one employee that you can select as a manager. For example, you might want to add data for an employee something like this:

```
new Employee
{
    EmployeeId = 1,
    Firstname = "Ada",
    Lastname = "Lovelace",
    DOB = new DateTime(1956, 12, 10),
    DateOfHire = new DateTime(1995, 1, 1),
    ManagerId = 0 // has no manager
}
```

## Project 12-1 Update the Trips Log app

In this project, you'll update the Trips Log app from project 8-1 so that it uses multiple related entities, rather than a single Trip entity.

### The Trips page

My Trip Log					
<a href="#">Add Trip</a>					
Destination	Start Date	End Date	Accommodations	Things To Do	
Olympic Peninsula	7/2/2020	7/9/2020	Camping	Swimming Hiking	<a href="#">Delete</a>
New York	10/25/2020	11/1/2020	Staybridge Suites Phone: 555-123-9876 Email: contact@staybridgesuites.com	Ride the subway Go to Grand Central Station	<a href="#">Delete</a>
Whitefish, Montana	2/15/2021	2/21/2021	The Lodge at Whitefish Lake Email: thelodge@whitefish.com	Walk in the snow Ice skate Cross country skiing	<a href="#">Delete</a>

### The Add Trip page

## My Trip Log

### Add Trip Destination and Dates

Destination [Manage destinations](#)

Whitefish, Montana

Accommodations [Manage accommodations](#)

The Lodge at Whitefish Lake

Start Date

2/15/2021

End Date

2/21/2021

[Next](#)
[Cancel](#)

## The Activities page

### My Trip Log

#### Add Things To Do in Whitefish, Montana

Hold down the CTRL key to select multiple items

Cross country skiing

Go to Grand Central Station

Hiking

Ice skate

Ride the subway

Swimming

Walk in the snow

Save

Cancel

## The Manage page

### My Trip Log

#### Manage Destinations, Accommodations, and Activities

##### Add

Destination

Accommodation Name

Accommodation Phone

Accommodation Email

Activity

Add

##### Delete

Destination

Accommodation

Activity

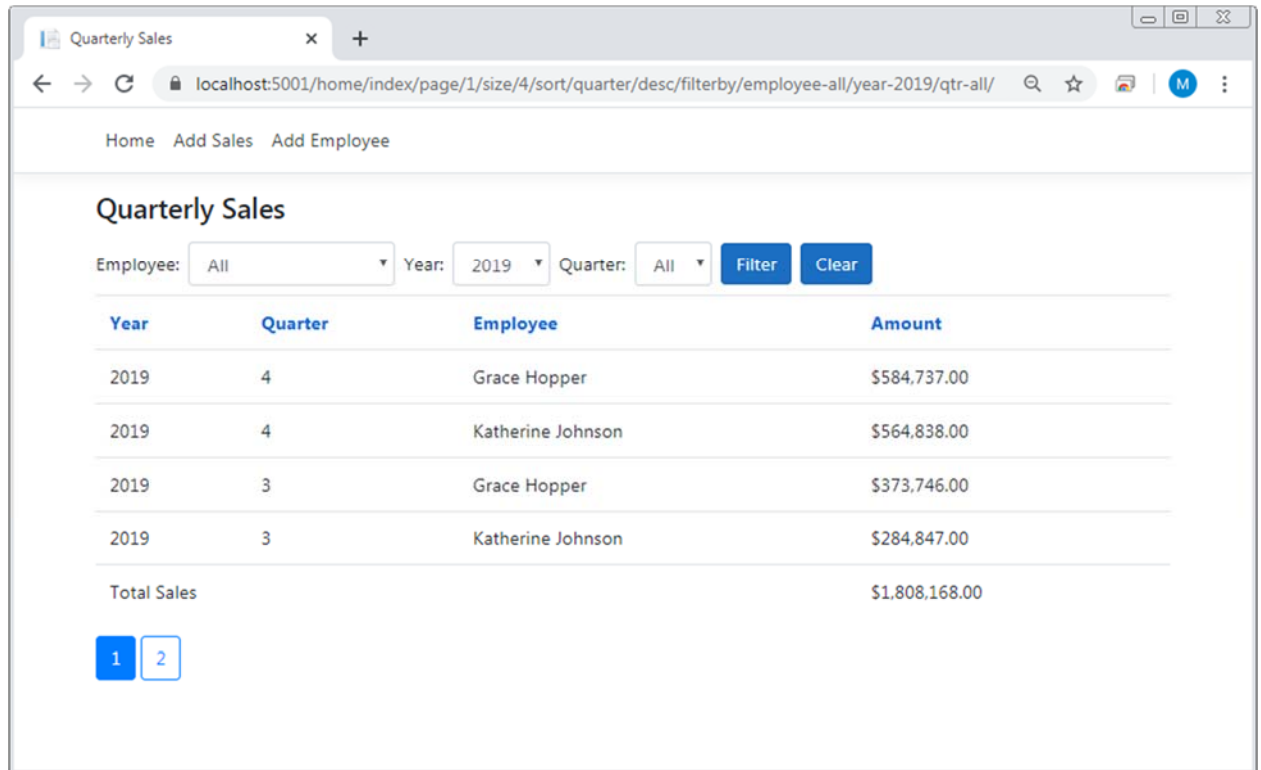
Delete

## Specifications

- There should be a Destination entity that has a one-to-many relationship with the Trip entity. That is, a trip can only have one destination, but a destination can be associated with more than one trip. The foreign key in the Trip entity should be configured so that attempting to delete a destination that's associated with a trip throws an error.
- There should be an Accommodation entity that has a one-to-many relationship with the Trip entity. That is, a trip can only have one accommodation, but an accommodation can be associated with more than one trip. The foreign key in the Trip entity should be configured so that when you delete an accommodation that's associated with a trip, the app sets the accommodation for that trip to null.
- There should be an Activity entity that has a many-to-many relationship with the Trip entity. That is, a trip can have more than one activity, and an activity can be associated with more than one trip.
- Use Entity Framework (EF) Code First and migrations to create a new database named TripLog2 that maps to these entities.
- Make sure to encapsulate your EF Core code in its own data layer.
- Update the Trips page so it allows users to delete a trip.
- Update the New Trip page so it allows users to select the destination and accommodation from drop-downs, rather than entering them in text boxes.
- Update the Activities page so the user can select one or more activities from a <select> element that allows multiple selections.
- Add a Manager page that provides a way for users to add and delete destinations, accommodations, and activities.
- Provide a way to access the Manager page from the Add Trip page.

## Project 13-1 Add paging, sorting, and filtering to the Quarterly Sales app

In this project, you'll update the Quarterly Sales app from project 11-1 so the table that displays sales data allows for paging, sorting, and filtering.



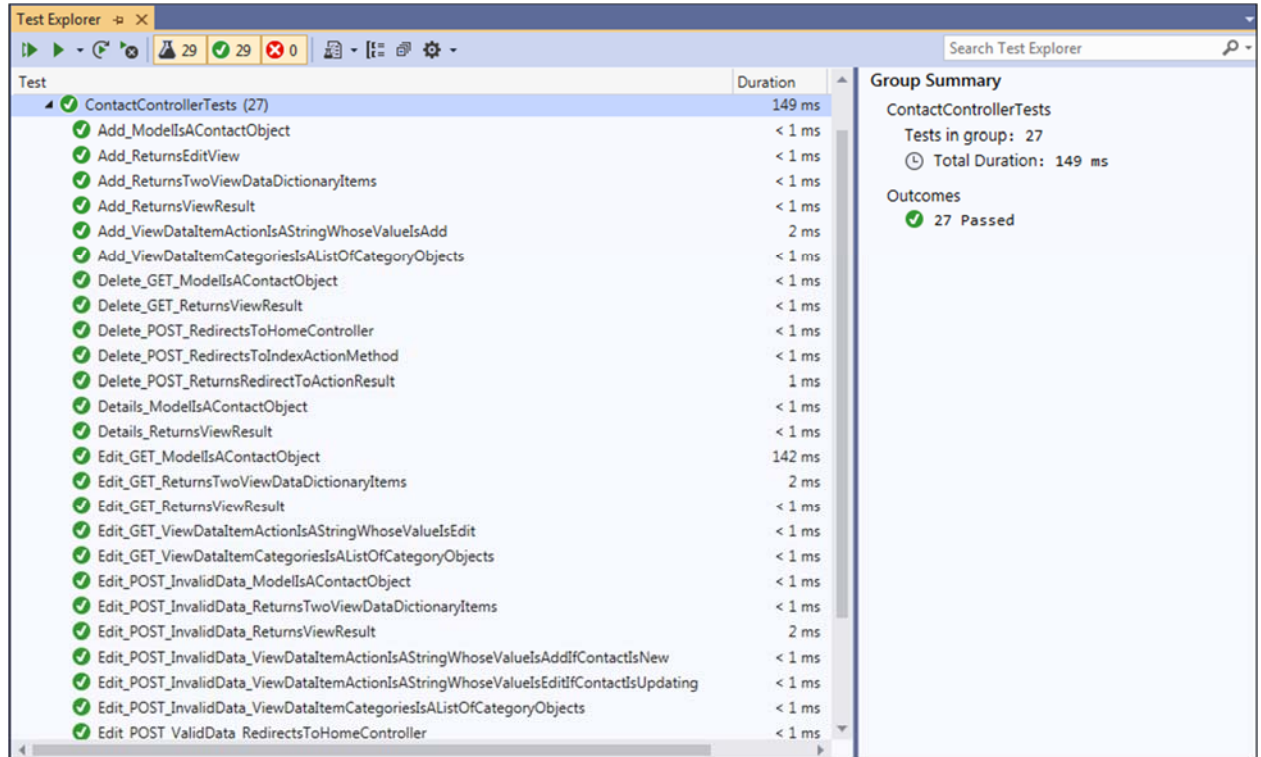
### Specifications

- When the app starts, it should display a table of sales data with the following features:
  - Drop-downs that allow a user to filter sales by employee, year, and/or quarter.
  - Links in the table header that allow a user to sort sales by employee, year, quarter, or amount.
  - The first time a user clicks on a sorting link, the data should sort in ascending order. If the user continues clicking on that link, the sorting should toggle between descending and ascending order.
  - Display a page of sales records, rather than all the sales records, and provide links below the table that allow the user to navigate through the pages.
- The app should have a user-friendly URL that shows the page and page size, sort order and sort direction, and what values are being filtered by.
- Note: The Bookstore app from chapter 13 uses a `FilterPrefix` class to add user-friendly prefixes to the filter segments. However, it's simpler to use static content in the route to achieve the same result. For example, this app can provide filter segments with a static route like this:

```
.../filterby/employee-{employee}/year-{year}/qtr-{quarter}
```

## Project 14-1 Add DI and unit testing to the Contact Manager app

In this project, you'll update the Contact Manager app from project 4-1 so that it uses dependency injection. Then, you'll add a test project and write some unit tests.

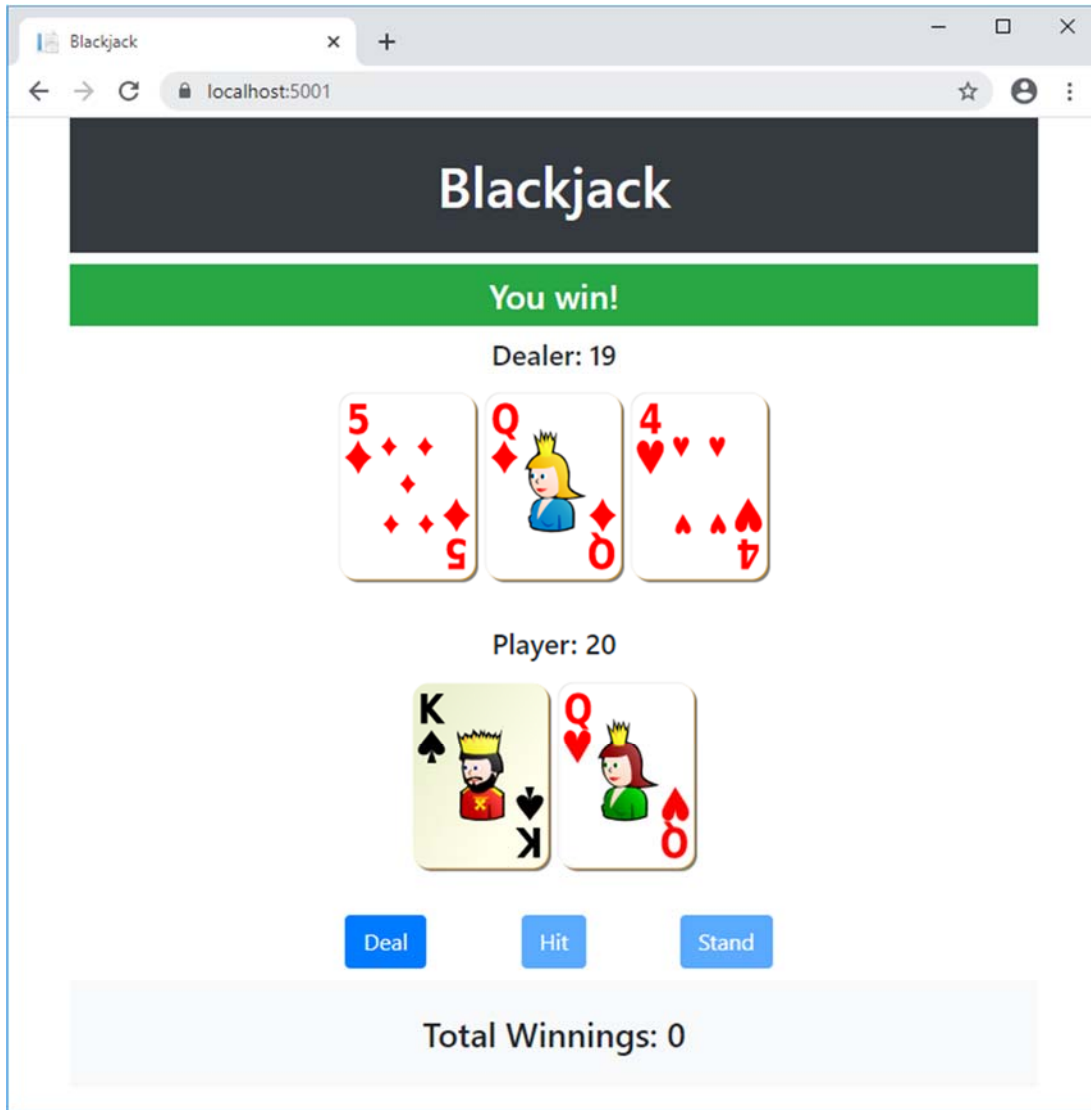


### Specifications

- The solution should have two projects, the ContactManager project and a test project.
- The Home controller and the Contact controller in the ContactManager project should receive repository or unit of work objects by dependency injection.
- The unit tests in the test project should test the action methods of the controllers in the ContactManager project.
- The unit tests should use the Moq framework to create any fake repository objects they need.
- Use your best judgement to code unit tests that thoroughly test the project.

## Project 15-1 Add tag helpers and view components to the Blackjack app

In this project, you'll use tag helpers and view components to simplify the views of a Blackjack web app. Your instructor should provide you with this app.



### Specifications

- Your instructor should provide you with a folder named 15-1\_BlackJack that contains the Blackjack app that you need to update for this project.
- Run the Blackjack app and play a few hands to get a sense of how the game works. (Note: This is a simple version of Blackjack that doesn't include some Blackjack rules such as insurance, splitting, and doubling down.)
- If the app displays giant cards that aren't sized as shown above, you need to clear the CSS cache for your browser. To do that with Chrome, you can press Ctrl+F5.

## Specifications (continued)

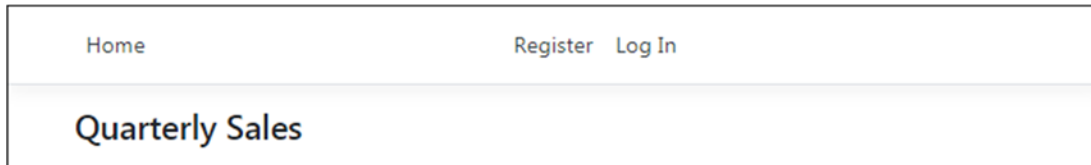
- Review the code for the app, particularly the Razor code blocks and conditional code in the layout and Home/Index views.
- The layout should use a tag helper to display the notification message to the user only if there's a message in TempData. In addition, the tag helper should vary the background color of the notification, based on the background value in TempData (green for a win, red for a loss, and so on).
- The layout should use a view component to get the player's current winnings from session and display them in the footer.
- When you're done updating the layout, there should be no Razor code blocks or conditional statements in the layout.
- The Home/Index view should use a tag helper display the headings that appear above the dealer's and player's hands. If there are any cards in the player's hand, the heading should display the total for the hand. The heading for the dealer's hand should only display the total for the hand when all the cards are showing.
- The Home/Index view should use a view component to show the hand of the dealer and the player. The view component should accept a Hand object from the Index view and pass it along to the partial view. The partial view should use the Hand object to create and display the card images.
- The Home/Index view should use a tag helper to provide the Deal, Hit, and Stand buttons. The tag helper should create the submit buttons and the forms that contain them. In addition, the tag helper should accept data from the Index view indicating the action method to post to and whether the button should be disabled.
- When you're done updating the Home/Index view, it shouldn't contain any Razor code blocks or conditional statements, except the Razor code block that sets the ViewBag.Title property.
- Note: the images of playing cards in this web app are from [pixabay.com](https://pixabay.com), which provides royalty-free stock images.



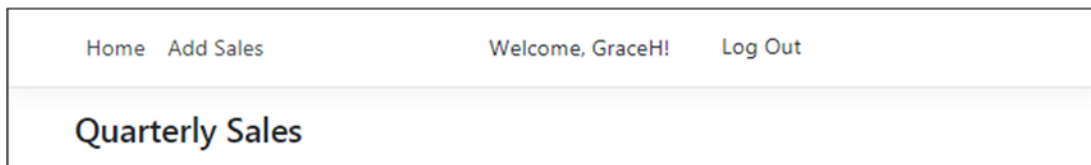
## Project 16-1 Add authorization to the Quarterly Sales app

In this project, you'll update the Quarterly Sales app from project 11-1 or project 13-1 to use authentication and authorization.

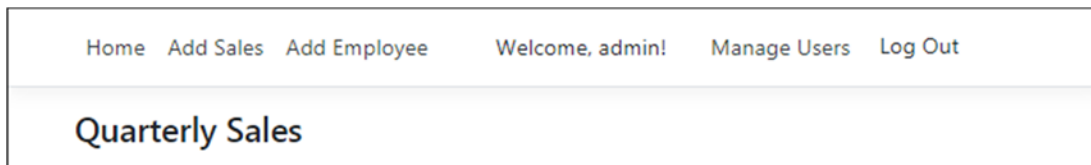
### The navbar for an anonymous user



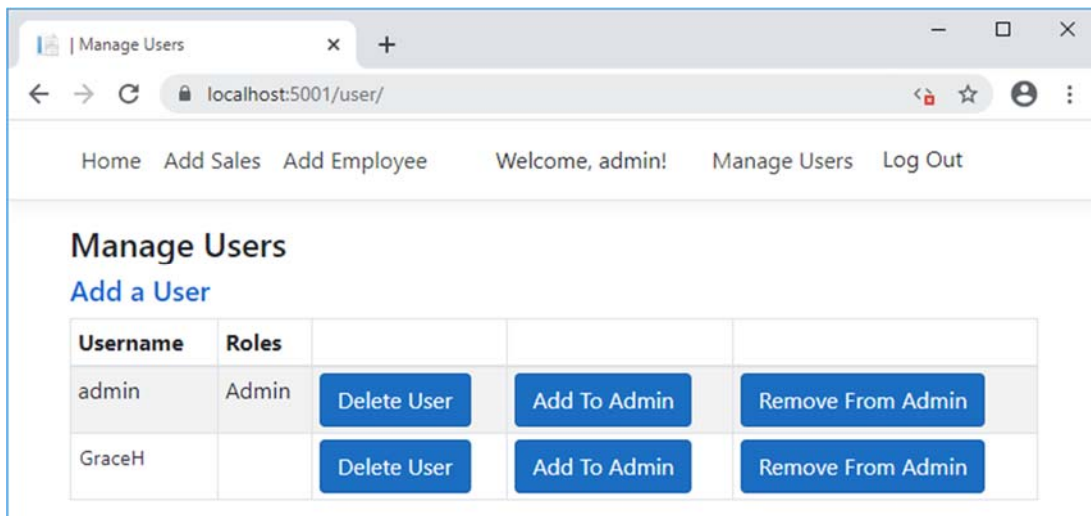
### The navbar for an authenticated user not in the Admin role



### The navbar for an authenticated user in the Admin role



### The Manage Users page



## Specifications

- The app should provide a way for users to register and log in.
- When registering, the user must provide a username and a password. Here's the password criteria:
  - Passwords must be at least 6 characters.
  - Passwords must have at least one uppercase character (A-Z).

## Specifications (continued)

- Anonymous users can view the sales data on the main page but aren't authorized to add sales or employees.
- Authenticated users are authorized to add sales, but only authenticated users in the Admin role are authorized to add employees.
- On startup, the app should create an Admin user named "admin" with a password of "P@ssw0rd".
- For anonymous users, the navbar should provide a Register link and a Log In link.
- For authenticated users, the navbar should display the user's name and provide a Log Out link. In addition, the navbar should display a link to the Add Sales page.
- For authenticated users in the Admin role, the navbar should display links to the Add Employee page and the Manage Users page. (Note: To check whether a user is in a role, you can use the `IsInRole()` method of the view's User property.)
- The Manage Users page should allow a user in the Admin role to add and delete users, and assign users to and remove them from the Admin role.