Module 13

Performance and Communication

Module Overview

- Implementing a Caching Strategy
- Managing State
- Two-Way Communication

Lesson 1: Implementing a Caching Strategy

- Why Use Caching?
- Cache Tag Helper
- Demonstration: How to Configure Caching
- The Data Cache
- Distributed Cache

Why Use Caching?

Caching:

- Helps improve the performance of a web application by reducing the time needed to process a webpage
- Helps increase the scalability of a web application by reducing the workload on the server
- Can be customized to retain an appropriate life time and priority for the data being cached

Cache Tag Helper

- One way to cache in an ASP.NET Core MVC application is by using the **cache** tag helper
- **cache** tag helper attributes:
  - enabled
  - priority
  - Expiration attributes: expires-on, expires-after, expires-sliding
  - vary-by attributes: vary-by-query, vary-by-cookie, vary-by-route, vary-by-user, vary-by-header, vary-by

Cache Tag Helper Example

```
<cache>
    @for (int i = 0; i < DateTime.Now.Second; i++) {
        <div>Number of seconds</div>
    }
</cache>

<cache vary-by="@Model.Id">
    <div>
        @Model.Name
    </div>
    <div>
        @Model.Price
    </div>
</cache>
```

Demonstration: How to Configure Caching

In this demonstration, you will see how to:

–Use a **cache** tag helper

–Use a **vary-by** attribute to create a cache per loaded product

The Data Cache

To cache data, you can use the **IMemoryCache** service

- –Register the service in the **ConfigureServices** method
- –Inject the service throughout the components of your application
- –Use the **Set** method to store data in the cache
- –Use the **TryGetValue** method to retrieve data from the cache

Data Cache Example

```
public IActionResult Index() {
   List<Product> products;

   if (!_memoryCache.TryGetValue(PRODUCT_KEY, out products)) {
      products = _productService.GetProducts();
      MemoryCacheEntryOptions options = new MemoryCacheEntryOptions();
      options.SetPriority(CacheItemPriority.Low);
      options.SetSlidingExpiration(new TimeSpan(6000));
      _memoryCache.Set(PRODUCT_KEY, products, options);
   }

   return View(products);
}
```

Distributed Cache

Distributed cache:

- –Stores shared cache data across multiple users and servers
- –Can be configured to work with both SQL and Redis
- –Is managed by using the **IDistributedCache** interface to cache information in components such as controllers
- –Is managed by using a **distributed-cache** tag helper alongside the **name** attribute in views

Lesson 2: Managing State

- Why Store State Information?
- State Storage Options
- Configuring Session State
- Demonstration: How to Store and Retrieve State Information
- Using the HTML5 Web Storage API

Why Store State Information?

Using states:

- –Creates a continuity between multiple different requests
- –Allows identifying specific users and using user specific logic
- –Is required for handling authentication
- –Allows developers to overcome weaknesses of using a stateless protocol

State Storage Options

State Storage:

- Allows websites to maintain a more coherent continuous experience
- Involves client-side session management techniques such as:
  - Hidden fields
  - Cookies
  - Query strings
- Involves server-side session management techniques such as:
  - TempData
  - HttpContext.Items
  - Cache
  - Dependency Injection
  - Session state

TempData Example

```
public IActionResult Index()
{
   object tempDataValue = TempData["myKey"];

   if (tempDataValue != null)
   {
      return Content("TempData exists!" + tempDataValue);
   }

   TempData["myKey"] = "Temporary Value";
   return Content("TempData does not exist!");
}
```

Configuring Session State

```
public IActionResult Index() {
   int? visitorCount = HttpContext.Session.GetInt32(VISIT_COUNT_KEY);
   if (visitorCount.HasValue) {
      visitorCount++;
   } else {
      visitorCount = 1;
   }
   HttpContext.Session.SetInt32(VISIT_COUNT_KEY, visitorCount.Value);
   return Content(string.Format("Number of visits:{0}", visitorCount));
}
```

Demonstration: How to Store and Retrieve State Information

In this demonstration, you will see how to:

–Configure an ASP.NET Core application to use session state

–Retrieve values from the **HttpContext.Session** property

–Store values in the **HttpContext.Session** property

Using the HTML5 Web Storage API

- Types of storage:
  - Local storage – Persists until removed and is shared between tabs
  - Session storage – Exists for a single tab and removed when it is closed
- Functions which exist in both local storage and session storage:
  - Get – Retrieves a stored value for a key
  - Set – Stores a chosen value for a key
  - Remove – Removes a saved value for a key

Local Storage Example

```
var storage_key = "num_of_visits";
var numberOfVisitsString = localStorage.getItem(storage_key);
var numberOfVisits = 1;
if (numberOfVisitsString) {
    numberOfVisits = parseInt(numberOfVisitsString) + 1;
}
alert("The page has been visited “ + numberOfVisits + " times");
if (numberOfVisits >= 5) {
    localStorage.removeItem(storage_key);
} else {
    localStorage.setItem(storage_key, numberOfVisits);
}
```

Lesson 3: Two-Way Communication

- The Web Sockets Protocol
- Using SignalR
- Demonstration: How to Use SignalR
- Additional SignalR Settings

The Web Sockets Protocol

Characteristics of web sockets:

- –W3C provides the WebSocket protocol to ensure that browsers support WebSockets as part of the HTML5 implementation
- –WebSockets facilitate two-way communication between client and server systems
- –WebSockets eliminate the need to re-create requests multiple times
- –WebSockets function in a similar manner as traditional network sockets

Using SignalR

- SignalR enables two-way communications between client and server
- Server Side:
  - Configure SignalR in Startup class
  - Define hubs
- Client Side:
  - Use SignalR Client Library
  - Connect to hubs by using JavaScript

SignalR Hub Example

```
using Microsoft.AspNetCore.SignalR;

public class MyChatHub : Hub
{
  public async Task MessageAll(string sender, string message)
  {
    await Clients.All.SendAsync("NewMessage", sender, message);
  }
}
```

Demonstration: How to Use SignalR

In this demonstration, you will see how to:

- –Configure SignalR in an ASP.NET Core application
- –Add a SignalR hub
- –Connect to a SignalR hub from a client
- –Register for calls from the hub
- –Call hub methods from the client

Additional SignalR Settings

- Hub members:
  - –Properties: **Clients**, **Context**, **Groups**
  - –Methods: **OnConnectedAsync**, **OnDisconnectedAsync**
- Serializing messages:
  - –JSON
  - –MessagePack
- Configuring connection:
  - –Server-side: **HandshakeTimeout**, **KeepAliveInterval**, **SupportedProtocols**, **EnableDetailedErrors**
  - –Client-side: **transport**, **serverTimeoutInMilliseconds**

Additional SignalR Settings Example

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddSignalR(hubOptions =>
  {
    hubOptions.HandshakeTimeout=TimeSpan.FromSeconds(30);
    hubOptions.KeepAliveInterval=TimeSpan.FromSeconds(50);
  }
  ).AddMessagePackProtocol();
  services.AddMvc();
}
```

Lab: Performance and Communication

- Exercise 1: Implementing a Caching Strategy
- Exercise 2: Managing State
- Exercise 3: Two-Way Communication

Estimated Time: 60 minutes

Lab Review

- A member of your team added a product to the database. However, when he looks in the browser he can’t see this product. Can you explain to him why?
- A member of your team changed the Configure method in the Startup class, so calling to the UseMvc middleware occurs before calling the UseSignalR middleware. Can you explain to him what is the impact of his change?

Module Review and Takeaways

- Review Question
- Common Issues and Troubleshooting Tips