Module 10

Implementing
Web APIs

Module Overview

- Introducing Web APIs
- Developing a Web API
- Calling a Web API

Lesson 1: Introducing Web APIs

- HTTP Services
- HTTP Messages
- Status Codes
- Introduction to Web API
- What is a Web API?

HTTP Services

- HTTP is a first class application protocol
- An HTTP URI has the following basic structure:

$$\underbrace{\texttt{http}}_{\text{Schema}}\texttt{://}\underbrace{\texttt{blueyonder.com}}_{\text{Host}}\texttt{:}\underbrace{\texttt{8080}}_{\text{Port}}\underbrace{\texttt{/travelers}}_{\text{Absolute Path}}\texttt{?}\underbrace{\texttt{id=1}}_{\text{Query}}$$

- HTTP defines a set of methods or verbs that add action-like semantics to requests

HTTP Services (Continued)

- REST is an architectural style that was developed in parallel to HTTP
  - REST is used to add important capabilities to a service

- Media types are used in HTTP to express message format
  - Client and server need to agree on the message format they exchange

HTTP Messages

An HTTP request message:

```
GET http://localhost:4392/travelers/1 HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US,en;q=0.7,he;q=0.3
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows
NT 6.2; WOW64; Trident/6.0)
Accept-Encoding: gzip, deflate
Host: localhost:4392
DNT: 1
Connection: Keep-Alive
```

An HTTP Response Message

An HTTP response message:

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/11.0.0.0
Date: Tue, 13 Nov 2012 18:05:11 GMT
X-AspNet-Version: 4.0.30319
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: application/json; charset=utf-8
Content-Length: 188
Connection: Close

{"TravelerId":1,"TravelerUserIdentity":"aaabbbccc","FirstName":“John"
,"LastName":“Doe","MobilePhone":"555-555-5555","HomeAddress":“123
Main Street","Passport":"AB123456789"}
```

Status Codes

- Status codes describe the result of the server’s attempt to process the request
- Status codes are constructed from a three-digit integer and a description called reason phrases
- HTTP has five different categories of status codes:
  - 1xx – Informational
  - 2xx – Success
  - 3xx – Redirection
  - 4xx – Client Error
  - 5xx – Server Error

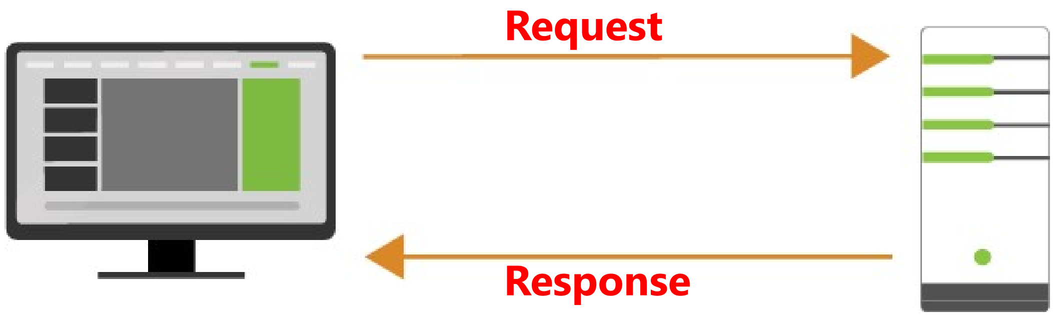
Introduction to Web API

- For a long time the .NET Framework did not have a first-class framework for building HTTP services
- The need for developing HTTP services justified creating a new framework
- In February 2012, ASP.NET Web API was released
- In June 2016, ASP.NET Core Web API was released

What is a Web API?

Web API:

- Helps create REST-style APIs
- Enables external systems to use the business logic implemented in your application
- Uses URLs in requests and helps obtain results
- Is ideal for mobile application integration



Lesson 2: Developing a Web API

- Using Routes and Controllers
- RESTful Services
- Action Methods and HTTP Verbs
- Binding Parameters to Request Message
- Control the HTTP Response
- Data Return Formats
- Demonstration: How to Develop a Web API

Using Routes and Controllers

Obtaining information by using Web API:

```
[Route("api/[controller]")]
public class HomeController : ControllerBase
{
    public string Get()
    {
        return "Response from Web API";
    }
}
```

RESTful Services

Characteristics of a REST Service:

- Can be called to retrieve business information from the server
- Can create, update, and delete information in a database through HTTP operations
- Uses URLs to uniquely identify the entity that it operates on
- Uses HTTP verbs to identify the operation that the application needs to perform. The HTTP verbs include:
  - **GET**
  - **POST**
  - **PUT**
  - **DELETE**

A Rest Service Example

```
[Route("api/[controller]")]
public class CustomerController : ControllerBase
{
    public IEnumerable<Customer> Get()
    { }
    public void Post(Customer item)
    { }
    public void Put(int id, Customer item)
    { }
    public void Delete(int id)
    { }
}
```

Action Methods and HTTP Verbs

You can use the following attributes to control the mapping of HTTP requests (HTTP verb+URL) to actions in the controller:

- The **HttpGet**, **HttpPut**, **HttpPost**, or **HttpDelete** attributes
- The **AcceptVerbs** attribute
- The **ActionName** attribute

Action Methods and HTTP Verbs Example

```
[Route("api/[controller]")]
public class HomeController : ControllerBase
{
    [HttpGet("Some")]
    public string SomeMethod()
    {
        return "SomeMethod was invoked";
    }

    [HttpGet("Other")]
    public string OtherMethod()
    {
        return "OtherMethod was invoked";
    }
}
```

Binding Parameters to Request Message

- An action that gets two parameters:

```
[Route("api/[controller]")]
public class HomeController : ControllerBase
{
    [HttpGet("{id}/{name}")]
    public string Get(int id, string name)
    {
        return "id: " + id + ", name: " + name;
    }
}
```

- This action method is chosen when sending a GET request by using the **api/Home/1/Mike** path

Using the ApiController Attribute

Use the **ApiController** attribute to target conventions on the controller:

```
[Route("api/[controller]")]
[ApiController]
public class CustomerController : ControllerBase
{
    [HttpPost]
    public void Post(Customer item)
    { }
}
```

Control the HTTP Response

HTTP responses use status codes to express the outcome of the request processing

```
public IActionResult Get(string id)
  {
      if (_items.ContainsKey(id) == false)
          return NotFound();

      return Ok(_items[id]);
  }
```

Return ActionResult<T>

Returning **ActionResult<T>** enables you to return a specific type or an object which inherits from **ActionResult**

```
public ActionResult<T> Get(string id)
{
    if (_items.ContainsKey(id) == false)
        return NotFound();

    return _items[id];
}
```

Data Return Formats

```
public IEnumerable<string> Get()
{
    return new string[] { "value1", "value2" };
}
```

```
["value1","value2"]
```

```
<ArrayOfstring>
    <string>value1</string>
    <string>value2</string>
</ArrayOfstring>
```

Demonstration: How to Develop a Web API

In this demonstration, you will learn how to:

- Add a Web API controller
- Add actions to a Web API controller
- Call Web API from a browser
- Control the data return format

Lesson 3: Calling a Web API

- Calling Web APIs by Using the javascript fetch API
- Demonstration: How to call Web APIs by Using the javascript fetch API
- Calling Web APIs by using Server-Side Code
- Working with Complex Objects
- Demonstration: How to Call Web APIs by Using Server-Side Code

Calling Web APIs by Using jQuery Code

- You can use the javascript fetch API to generate an HTTP request from a browser to a Web API

- You can use **JSON.stringify** in the **data** parameter of the **fetch** function to serialize the JavaScript objects into JSON objects

Calling the Web API Get method by using fetch:

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

Calling the Web API Post method by using jQuery

Calling the Web API Post method by using fetch:

```
 async function postData(url = '', data = {}) {
  // Default options are marked with *
  const response = await fetch(url, {
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    headers: {
      'Content-Type': 'application/json’
    },
    body: JSON.stringify(data) // body data type must match
"Content-Type" header
  });
  return response.json(); // parses JSON response into native
JavaScript objects
}
```

Demonstration: How to call Web APIs by Using jQuery code

In this demonstration, you will learn how to:

- Add a Web API controller with Get and Post methods
- Call the Web API Get method by using fetch
- Call the Web API Post method by using fetch

Calling Web APIs by using C#

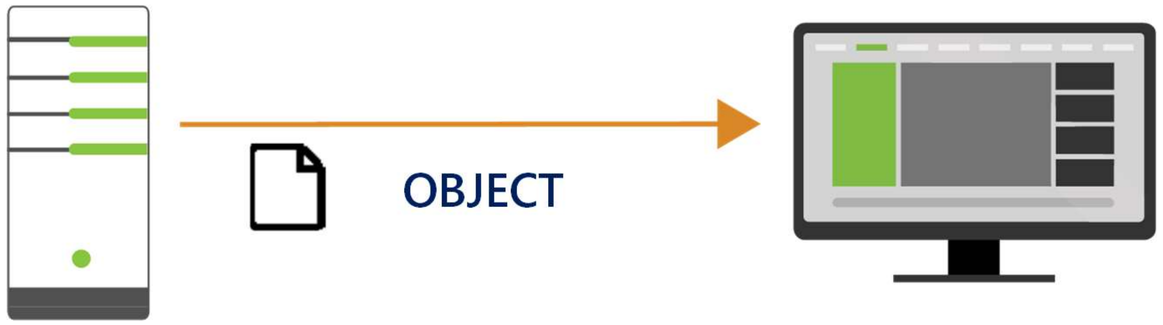
To call Web APIs by using C#:

- Add code to initialize the **HttpClient** class
- Add code to create requests by using **GetAsync**, **PostAsync**, **PutAsync** and **DeleteAsync**

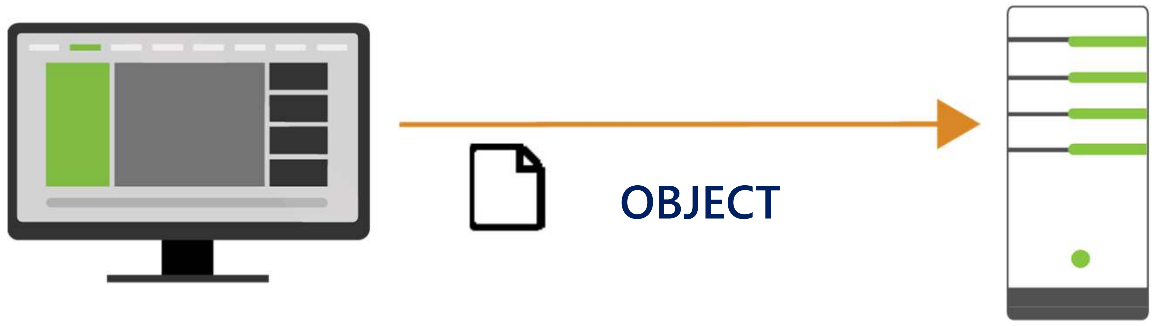
```
HttpClient client = _httpClientFactory.CreateClient();
client.BaseAddress = new Uri("http://localhost:[port]");
HttpResponseMessage response =
        client.GetAsync("api/Values/key1").Result;
```

Working with Complex Objects

- Returning complex objects from a Web API



- Passing complex objects to a Web API



Get a Complex Object from a Web API

```
HttpResponseMessage response =
       await httpClient.GetAsync("api/Person");
if (response.IsSuccessStatusCode) {
    Person person =
       await response.Content.ReadAsAsync<Person>();
    return Content(person.Name);
} else {
    return Content("An error has occurred");
}
```

Pass a Complex Object to a Web API

```
Entry entry =
       new Entry() { Key = "key3", Value = "value3" };
HttpResponseMessage response =
       await httpClient.PostAsJsonAsync("api/Values", entry);
if (response.IsSuccessStatusCode) {
    return Content("succedded");
} else {
    return Content("An error has occurred");
}
```

Demonstration: How to Call Web APIs by Using C#

In this demonstration, you will learn how to:

- Register and use the **IHttpClientFactory** service
- Call a Web API Get method by using the **HttpClient** class
- Call a Web API Post method by using the **HttpClient** class

Lab: Implementing Web APIs

- Exercise 1: Adding Actions and Calling them by using a browser
- Exercise 2: Calling a Web API by Using C#
- Exercise 3: Calling a Web API by Using fetch

Estimated Time: 60 minutes