ASP.NET WEB API CORE

Lesson 01: Introduction to .Net Core WebAPI

Capgemini

Lesson Objectives

- In this lesson we will cover the following
- Introduction to .Net Core WebAPI
- Introduction to Web Service with Demo
- Introduction to WCF Service with Demo
- Introduction to Web API
- Difference between Web Service, WCF Service and Web API
- Web API features
- HTTP Web Services
- Web API Introduction
- Middleware
- Web API core Routing

ASP.NET Core Overview

- ASP.NET Core is the new version of the ASP.NET web framework mainly targeted to run on .NET Core platform.
- ASP.NET Core is a free, open-source, and cross-platform framework for building cloud-based applications, such as web apps, IoT apps, and mobile backends.
- It is designed to run on the cloud as well as on-premises.
- Same as .NET Core, it was architected modular with minimum overhead, and then other more advanced features can be added as NuGet packages as per application requirement.
- This results in high performance, require less memory, less deployment size, and easy to maintain.

What are Web Services?

- **Web Services** are a standardized way for developing interoperable applications that enable an application to invoke a method of another application.

- These applications can be on the same computer or different computers. Web services use open standards and protocols like **HTTP**, **XML** and **SOAP**. Since these are open and well-known protocols, applications built on any platform can interoperate with web services.

- For example, a Java application can interoperate with a web service built using .NET. Similarly a web service built using Java can be consumed by a .NET application.

- The Web service consumers are able to invoke method calls on remote objects by using SOAP and HTTP over the Web.

- Webservice is language independent and Web Services communicate by using standard web protocols and data formats, such as

  - HTTP
  - XML
  - SOAP

What is SOAP?

SOAP (simple object access protocol) is a remote function calls that invokes method and execute them on Remote machine and translate the object communication into XML format. In short, SOAP are way by which method calls are translate into XML format and sent via HTTP.

What is WSDL?

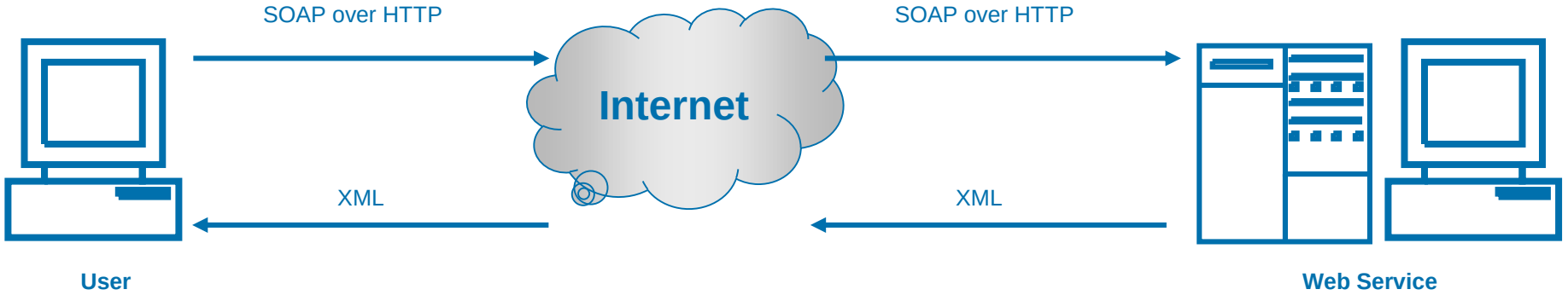
WSDL stands for Web Service Description Language, a standard by which a web service can tell clients what messages it accepts and which results it will return.

WSDL contains every detail regarding using web service and Method and Properties provided by web service and URLs from which those methods can be accessed and Data Types used.

What is UDDI?

UDDI allows you to find web services by connecting to a directory.

Process of Webservice



What is WCF?

- WCF as a programming platform that is used to build Service-Oriented applications.
- Windows Communication Foundation is basically a unified programming model for developing, configuring and deploying distributed services.
- Microsoft has unified all its existing distributed application technologies (e.g. MS Enterprise Services, ASMX web services, MSMQ, .NET Remoting etc) at one platform i.e. WCF.
- Different clients can interact with same service using different communication mechanism. This is achieved by using service endpoints.
- A single WCF service can have multiple endpoints.
- So, developer will write code for service once and just by changing configuration (defining another service endpoint), it will be available for other clients as well.

Features of WCF

I. Service Orientation

II. Interoperability

III. Multiple Message Patterns

IV. Service Metadata

V. Data Contracts

VI. Multiple Transports and Encodings

VII. Reliable and Queued Messages Security

VIII. Durable Messages

Introduction to Web API

- The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc.

- It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view.

- It is like a web service or WCF service but the exception is that it only supports HTTP protocol.

- ASP.NET Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices.

- It is an ideal platform for building RESTful applications on the .NET Framework.

Introduction to WEB API Contd…

- ASP.NET Web API makes it easy to create a set of web services that can respond to browser requests by using simple HTTP verbs such as GET, POST, and DELETE

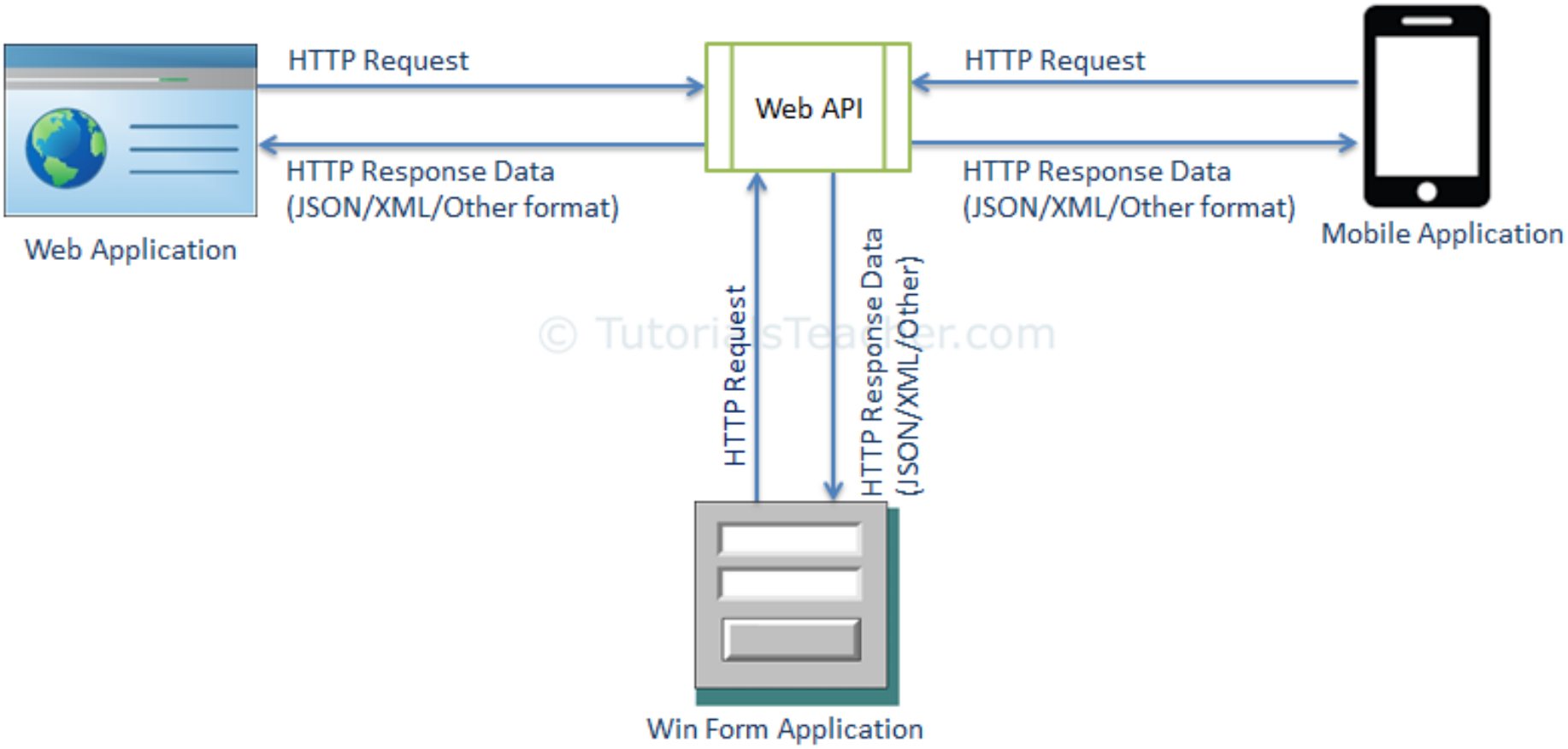
- Using the Web API, you can build the back-end web services that a client-specific web application can call

- Building Web application by using client specific HTML Pages and the WEB API is an alternative to using ASP.NET MVC.

- Web API enable developers to obtain business information by using REST , without creating complicated XML request such as SOAP

- Web API uses URL in requests and obtains results in the JSON format

WEB API PROCESS



Web Service v/s WCF v/s Web API

- Web Service
  - It is based on SOAP and return data in XML form.
  - It support only HTTP protocol.
  - It is not open source but can be consumed by any client that understands xml.
  - It can be hosted only on IIS.
- WCF
  - It is also based on SOAP and return data in XML form.
  - It is the evolution of the web service(ASMX) and support various protocols like TCP, HTTP, HTTPS, Named Pipes, MSMQ.
  - The main issue with WCF is, its tedious and extensive configuration.
  - It is not open source but can be consumed by any client that understands xml.
  - It can be hosted with in the applicaion or on IIS or using window service.

Web Service v/s WCF v/s Web API

Web API

- This is the new framework for building HTTP services with easy and simple way.
- Web API is open source an ideal platform for building REST-ful services over the .NET Framework.
- Unlike WCF Rest service, it use the full features of HTTP (like URIs, request/response headers, caching, versioning, various content formats)
- It also supports the MVC features such as routing, controllers, action results, filter, model binders, IOC container or dependency injection, unit testing that makes it more simple and robust.
- It can be hosted with in the application or on IIS.
- It is light weight architecture and good for devices which have limited bandwidth like smart phones.
- Responses are formatted by Web API's MediaTypeFormatter into JSON, XML or whatever format you want to add as a MediaTypeFormatter.

What are the advantages of Web Services?

- **Interoperability**: Web services are accessible over network and runs on HTTP/SOAP protocol and uses XML/JSON to transport data, hence it can be developed in any programming language. Web service can be written in java programming and client can be PHP and vice versa.
- **Reusability**: One web service can be used by many client applications at the same time.
- **Loose Coupling**: Web services client code is totally independent with server code, so we have achieved loose coupling in our application.
- **Deployment**: Easy to deploy and integrate, just like web applications
- **Versioning**: Multiple service versions can run at the same time.

Introduction to WEB API

- The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc.
- It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view.
- It is like a web service or WCF service but the exception is that it only supports HTTP protocol.
- ASP.NET Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It is an ideal platform for building RESTful applications on the .NET Framework.

Introduction to WEB API Contd…

- Web API enable developers to obtain business information by using REST , without creating complicated XML request such as SOAP
- Web API uses url in requests and obtains results in the JSON format

Middleware

- It can also control how our application looks when there is an error, and it is a key piece in how we authenticate and authorize a user to perform specific actions.
- Middleware are software components that are assembled into an application pipeline to handle requests and responses.
- Each component chooses whether to pass the request on to the next component in the pipeline, and can perform certain actions before and after the next component is invoked in the pipeline.
- Request delegates are used to build the request pipeline. The request delegates handle each HTTP request.
- Each piece of middleware in ASP.NET Core is an object, and each piece has a very specific, focused, and limited role.
- Ultimately, we need many pieces of middleware for an application to behave appropriately.

Middleware

- Let us now assume that we want to log information about every request into our application.
- In that case, the first piece of middleware that we might install into the application is a logging component.
- This logger can see everything about the incoming request, but chances are a logger is simply going to record some information and then pass along this request to the next piece of middleware.

- Middleware is a series of components present in this processing pipeline.
- The next piece of middleware that we've installed into the application is an authorizer.

Middleware

- An authorizer might be looking for specific cookie or access tokens in the HTTP headers.
- If the authorizer finds a token, it allows the request to proceed. If not, perhaps the authorizer itself will respond to the request with an HTTP error code or redirect code to send the user to a login page.
- But, otherwise, the authorizer will pass the request to the next piece of middleware which is a router.
- A router looks at the URL and determines your next step of action.
- The router looks over the application for something to respond to and if the router doesn't find anything to respond to, the router itself might return a 404 Not Found error.
- We set up the middleware in ASP.NET using the Configure method of our Startup class.
- Inside the Configure() method, we will invoke the extension methods on the IApplicationBuilder interface to add middleware.

Middleware

- There are two pieces of middleware in a new empty project by default –
  - IISPlatformHandler
  - Middleware registered with app.Run

- IISPlatformHandler allows us to work with Windows authentication.

- It will look at every incoming request and see if there is any Windows identity information associated with that request and then it calls the next piece of middleware.

- The next piece of middleware in this case is a piece of middleware registered with app.Run.

- The Run method allows us to pass in another method, which we can use to process every single response.

- Run is not something that you will see very often, it is something that we call a terminal piece of middleware.

- Middleware that you register with Run will never have the opportunity to call another piece of middleware, all it does is receive a request, and then it has to produce some sort of response.

Middleware

- You also get access to a Response object and one of the things you can do with a Response object is to write a string.
- If you want to register another piece of middleware after app.Run, that piece of middleware would never be called because, again, Run is a terminal piece of middleware.
- It will never call into the next piece of middleware.
- This RuntimeInfoPage is a middleware that will only respond to requests that come in for a specific URL.
- If the incoming request does not match that URL, this piece of middleware just lets the request pass through to the next piece of middleware.
- The request will pass through the IISPlatformHandler middleware, then go to the UseRuntimeInfoPage middleware.
- It is not going to create a response, So it will go to our app.Run and display the string.

Web Api Routing

Web API supports two types of routing:

- Convention-based Routing
- Attribute Routing

Convention-based Routing

- In the convention-based routing, Web API uses route templates to determine which controller and action method to execute.
- At least one route template must be added into route table in order to handle various HTTP requests.

Attribute Routing

- Attribute routing is supported in Web API 2. As the name implies, attribute routing uses [Route()] attribute to define routes. The Route attribute can be applied on any controller or action method.

- In order to use attribute routing with Web API, it must be enabled in WebApiConfig by calling config.MapHttpAttributeRoutes() method

Convention Based Routing

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Enable attribute routing
        config.MapHttpAttributeRoutes();

        // Add default route using convention-based routing
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Attribute Based Routing

```
public class EmployeeController : ApiController
{
    [Route("api/Employee/names")]
            public IEnumerable<string> Get()
    {
            return new string[] { "student1", "student2" };
    }
}
```

Route attribute defines new route "api/Employee/names" which will be handled by the Get() action method of EmployeeController. Thus, an HTTP GET request http://localhost:1234/api/employee/names will return list of Employee names.

Web Api Core Routing

- ASP.NET Core controllers use the Routing middleware to match the URLs of incoming requests and map them to actions.

Routes templates:

- Are defined in startup code or attributes.
- Describe how URL paths are matched to actions.
- Are used to generate URLs for links. The generated links are typically returned in responses.
- Actions are either conventionally-routed or attribute-routed. Placing a route on the controller or action makes it attribute-routed.

Set up conventional route:

Startup.Configure typically has code similar to the following when using conventional routing:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Conventional routing in .NET Core

Conventional routing is used with controllers and views.

The default route:

- endpoints.MapControllerRoute(name: "default", pattern: "{controller=Home}/{action=Index}/{id?}");

- The first path segment, {controller=Home}, maps to the controller name.

- The second segment, {action=Index}, maps to the action name.

- The third segment, {id?} is used for an optional id. The ? in {id?} makes it optional. id is used to map to a model entity.

Using this default route, the URL path:

/Products/List maps to the ProductsController.List action.

/Blog/Article/17 maps to BlogController.Article and typically model binds the id parameter to 17.

Multiple conventional routes

Multiple conventional routes can be added inside UseEndpoints by adding more calls to MapControllerRoute and MapAreaControllerRoute. Doing so allows defining multiple conventions, or to adding conventional routes that are dedicated to a specific action, such as:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(name: "blog",
                pattern: "blog/{*article}",
                defaults: new { controller = "Blog", action = "Article" });
    endpoints.MapControllerRoute(name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Web Api Core Routing

- Attribute routing requires more input to specify a route.
- The conventional default route handles routes more succinctly.
- However, attribute routing allows and requires precise control of which route templates apply to each action.
- With attribute routing, the controller and action names play no part in which action is matched, unless token replacement is used.
- Route templates applied to an action that begin with / or ~/ don't get combined with route templates applied to the controller.
- The following keywords are reserved route parameter names when using Controllers or Razor Pages:

  - action
  - area
  - controller
  - handler
  - page

Web Api Core Routing

- Using page as a route parameter with attribute routing is a common error. Doing that results in inconsistent and confusing behavior with URL generation.
- The special parameter names are used by the URL generation to determine if a URL generation operation refers to a Razor Page or to a Controller.

- ASP.NET Core has the following HTTP verb templates:

- [HttpGet]
- [HttpPost]
- [HttpPut]
- [HttpDelete]
- [HttpHead]
- [HttpPatch]

ASP.NET Core has the following route templates:

- All the HTTP verb templates are route templates.
- [Route]

- Each action contains the [HttpGet] attribute, which constrains matching to HTTP GET requests only.

Web Api Core Routing

- The Get action includes the "{id}" template, therefore id is appended to the "api/[controller]" template on the controller.
- The methods template is "api/[controller]/"{id}"".
- Therefore this action only matches GET requests of for the form /api/test2/xyz,/api/test2/123,/api/test2/{any string}, etc.
- The GetIntProduct action contains the "int/{id:int}") template.
- The :int portion of the template constrains the id route values to strings that can be converted to an integer.
- A GET request to /api/test2/int/abc:
  - Doesn't match this action.
  - Returns a 404 Not Found error.
- The GetInt2Product action contains {id} in the template, but doesn't constrain id to values that can be converted to an integer. A GET request to /api/test2/int2/abc:
  - Matches this route.
  - Model binding fails to convert abc to an integer. The id parameter of the method is integer.
  - Returns a 400 Bad Request because model binding failed to convertabc to an integer.

Demo

- Demo:-
- Implementing in Asp.NET Web service Program.
- Implementing in Asp.NET WCF Program.
- Implementing in Asp.NET Web API core Program.
- Implementing in Asp.NET Web API core Routing Program.

Summary

- In this lesson you have learnt about:

  - Introduction to Web Service
  - Introduction to WCF Service
  - Introduction to Web API
  - Difference between Web Service, WCF Service and Web API
  - Web API features
  - HTTP Web Services
  - Web API Introduction
  - Middleware
  - Web API core Routing