**WebAPI HandsOn Questions**

**1.Explain the concept of RESTful web service, Web API & Microservice and Features of REST architecture - Representational State Transfer, Stateless, Messages, Concept of Microservice, Difference between WebService & WebAPI, Not restricted to send XML as response REST stands for Representational State Transfer.**

* REST is an architectural model and design for server network applications. The most common application of REST is the World Wide Web itself, which used REST as a basis for HTTP development.
* REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

**HTTP methods:**

Following four HTTP methods are commonly used in REST based architecture.

* **GET** − Provides a read only access to a resource.
* **POST** − Used to create a new resource.
* **DELETE** − Used to remove a resource.
* **PUT** − Used to update a existing resource or create a new resource.

**RESTFul web services:**

* A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.
* Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

**Web APIs:**

* **API** stands for **Application Programming Interface**. It is a collection of communication conventions and subroutines used by various programs to communicate between them.
* A developer can utilize different API apparatuses to make its program simpler and less complex. Likewise, an API encourages the developers with a proficient method to build up their product programs. Thus, in simple terms, an API determines how programming segments ought to associate with one another. It is a set of protocols and schedules, and its reactions are returned as JSON or XML in data.

**Difference between Web Services and APIs:**

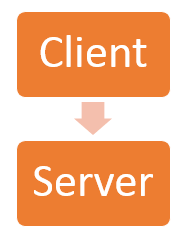
|  |  |
| --- | --- |
| **Web Services** | **Web API** |
| 1.Web services are a type of API, which must be accessed through a network connection. | 1.APIs are application interfaces, implying that one application can communicate with another application in a standardized manner. |
| 2.Web service is used for REST, SOAP and XML-RPC for communication. | 2.API is used for any style of communication. |
| 3.All Web services are APIs. | 3.APIs are not web services. |
| 4.It doesn’t have lightweight design, needs a SOAP convention to send or receive data over the system. | 4.It has a light-weight architecture furthermore, useful for gadgets which have constrained transmission capacity like smart phones. |
| 5.It provides supports only for the HTTP protocol. | 5.It provides support for the HTTP/s protocol: URL Request/Response Headers, and so on. |
| 6.It is not open source, however, can be devoured by any customer that comprehends xml. | 6.It is an open source and also ships with .NET framework. |
| 7.Web service supports only XML. | 7.API supports XML and JSON. |
| 8.Web Services can be hosted on IIS. | 8.Web API can be hosted only on IIS and self. |

**What is MicroServices?**

* The term microservices portrays a software development style that has grown from contemporary trends to set up practices that are meant to increase the speed and efficiency of developing and managing software solutions at scale.
* Microservices is more about applying a certain number of principles and architectural pattern as architecture. Each microservice lives independently, but on the other hand, also all rely on each other.
* All microservices in a project get deployed in production at their own pace, on-premise on the cloud, independently, living side by side.
* The REST architecture is based on a few characteristics which are elaborated below. Any RESTful web service has to comply with the below characteristics in order for it to be called RESTful. These characteristics are also known as design principles which need to be followed when working with RESTful based services.

1. **RESTFul Client-Server**

* This is the most fundamental requirement of a REST based architecture. It means that the server will have a RESTful web service which would provide the required functionality to the client. The client send's a request to the web service on the server. The server would either reject the request or comply and provide an adequate response to the client.

[](https://cdn.guru99.com/images/3-2016/032316_0816_RESTfulWebS3.png)

1. **Stateless**

* The concept of stateless means that it's up to the client to ensure that all the required information is provided to the server. This is required so that server can process the response appropriately. The server should not maintain any sort of information between requests from the client. It's a very simple independent question-answer sequence. The client asks a question, the server answers it appropriately. The client will ask another question. The server will not remember the previous question-answer scenario and will need to answer the new question independently.

1. **Cache**

[](https://cdn.guru99.com/images/3-2016/032316_0816_RESTfulWebS4.png)

The Cache concept is to help with the problem of stateless which was described in the last point. Since each server client request is independent in nature, sometimes the client might ask the server for the same request again. This is even though it had already asked for it in the past. This request will go to the server, and the server will give a response. This increases the traffic across the network. The cache is a concept implemented on the client to store requests which have already been sent to the server. So if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information. This saves the amount of to and fro network traffic from the client to the server.

**4.Layered System**

The concept of a layered system is that any additional layer such as a middleware layer can be inserted between the client and the actual server hosting the RESTFul web service (The middleware layer is where all the business logic is created. This can be an extra service created with which the client could interact with before it makes a call to the web service.). But the introduction of this layer needs to be transparent so that it does not disturb the interaction between the client and the server.

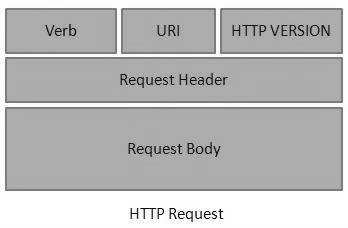
**5.Interface/Uniform Contract**

This is the underlying technique of how RESTful web services should work. RESTful basically works on the HTTP web layer and uses the below key verbs to work with resources on the server

* POST - To create a resource on the server
* GET - To retrieve a resource from the server
* PUT - To change the state of a resource or to update it
* DELETE - To remove or delete a resource from the server

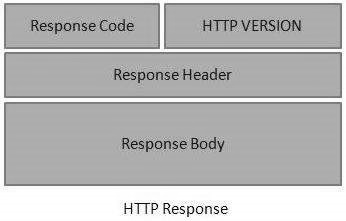
2.Explain what is HttpRequest & HttpResponse

HTTP Request

An HTTP Request has five major parts −

* **Verb** − Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.
* **URI** − Uniform Resource Identifier (URI) to identify the resource on the server.
* **HTTP Version** − Indicates the HTTP version.
* **Request Header** − Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by the client, format of the message body, cache settings, etc.
* **Request Body** − Message content or Resource representation.

HTTP Response

An HTTP Response has four major parts −

* **Status/Response Code** − Indicates the Server status for the requested resource. For example, 404 means resource not found and 200 means response is ok.
* **HTTP Version** − Indicates the HTTP version.
* **Response Header** − Contains metadata for the HTTP Response message as keyvalue pairs. For example, content length, content type, response date, server type, etc.
* **Response Body** − Response message content or Resource representation.

3. List the types of Action Verbs

\* HttpGet, HttpPost, HttpPut, HttpDelete - Meaning of action verbs and how that should be declared as attributes for Web API

The following table lists the usage of HTTP methods:

| Http method | Usage |
| --- | --- |
| GET | To retrieve the information from the server. Parameters will be appended in the query string. |
| POST | To create a new resource. |
| PUT | To update an existing resource. |
| HEAD | Identical to GET except that server do not return the message body. |
| OPTIONS | It represents a request for information about the communication options supported by the web server. |
| DELETE | To delete an existing resource. |
|  |  |
|  |  |

public class StudentController : Controller

{

public ActionResult Index() // handles GET requests by default

{

return View();

}

[HttpPost]

public ActionResult PostAction() // handles POST requests by default

{

return View("Index");

}

[HttpPut]

public ActionResult PutAction() // handles PUT requests by default

{

return View("Index");

}

[HttpDelete]

public ActionResult DeleteAction() // handles DELETE requests by default

{

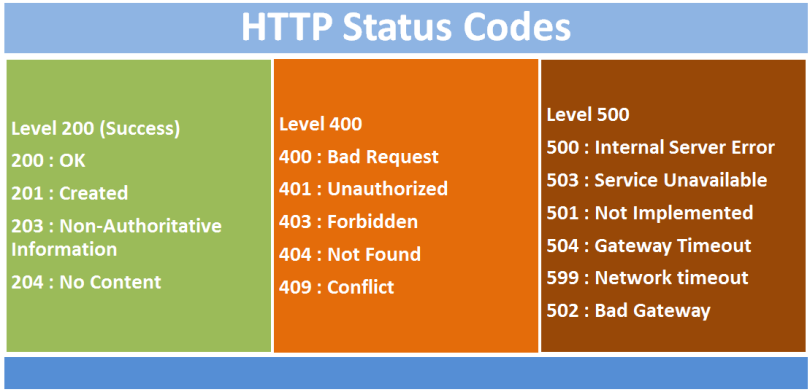
return View("Index");

}

}

4. List the types of HttpStatusCodes used in WebAPI

\* Ok, InternalServerError, Unauthorized, BadRequest - All thru the action result types



5. Demonstrate creation of a simple WebAPI - With Read, Write actions

\*Structure of a web api - Controller & its inheritance from ApiController, Action verbs, Action method

namespace EmpWebApi.Controllers {

[Route("api/[controller]")]

[ApiController]

public class EmployeeController : ControllerBase {

IEmpRepository empRep;

public EmployeeController(IEmpRepository ier) {

empRep = ier;

}

[HttpGet]

public ActionResult<List<EMPL>> GetAll() {

return Ok(empRep.GetAllEmployees());

}

[HttpGet("{id}")]

public ActionResult<EMPL> GetOne(int id) {

try {

return Ok(empRep.GetEmployeeById(id));

}

catch(EmpException ex) {

return NotFound(ex.Message);

}

}

[HttpPost]

public ActionResult Insert(EMPL emp) {

empRep.InsertEmployee(emp);

return Created($"api/Employee/{emp.EID}", emp);

}

[HttpPut("{id}")]

public ActionResult Update(int id, EMPL emp) {

empRep.UpdateEmployee(id, emp);

return Ok(emp);

}

[HttpDelete("{id}")]

public ActionResult Delete(int id) {

empRep.DeleteEmployee(id);

return Ok();

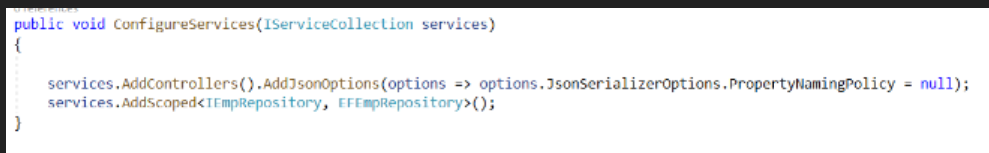
}

}

6. Explain the types of Configuration files of WebAPI

\*Startup.cs with depdency injection, appSettings.json, launchSettings.json, Explain Route.config & WebAPI.config in .Net 4.5 framework

**Startup.cs With Dependency Injection:**



**Configuring Dependencies:**



**3.Routeconfig.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

namespace MvcApplicationDemo

{

public class RouteConfig

    {

public static void RegisterRoutes(RouteCollection routes)

        {

            routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

            routes.MapRoute(

                name: "Default",

                url: "{controller}/{action}/{id}",

                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

            );

        }

    }

}

**4.WebApiconfig.cs**

Web API supports code based configuration. It cannot be configured in web.config file. We can configure Web API to customize the behaviour of Web API hosting infrastructure and components such as routes, formatters, filters, DependencyResolver, MessageHandlers, ParamterBindingRules, properties, services etc

public class Startup

{

    public void Configuration(IAppBuilder appBuilder)

    {

        HttpConfiguration config = new HttpConfiguration();

        config.Routes.MapHttpRoute(

            name: "DefaultApi",

            routeTemplate: "api/{controller}/{id}",

            defaults: new { id = RouteParameter.Optional }

        );

        appBuilder.UseWebApi(config);

    }

}

**Appsettings.json**

The appsettings.json file is an application configuration file used to store configuration settings such as database connections strings, any application scope global variables, etc.

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

**Launch Settings.json**

{

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:17323",

"sslPort": 0

}

},

"profiles": {

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"AirlinesMvcApp": {

"commandName": "Project",

"dotnetRunMessages": "true",

"launchBrowser": true,

"applicationUrl": "http://localhost:5000",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}