## Introduction

 Bookmark this page

In this module, we'll take a look at the core information that you will need to understand before you delve into creating Web APIs, services, and consuming them.

We'll start with a very basic review of key concepts such as client/server and how the web works from the perspective of serving up information to the user. We'll provide some key definitions along the way that will help you understand the material later in the course.

The second lesson covers topics related the JSON data format and how we can serialize and deserialize the objects in our code into and out of JSON. This is necessary to transport the "objects" across the wire.

This module, like the others in this course, will close with a set of labs. Labs can be a tutorial lab where you will be guided through the necessary steps to complete a scenario that reinforces what you have learned in this module. The second lab, an assessment lab, is designed to have you complete a similar scenario, but without guidance, to allow you to apply your learning from the lessons. After completing the assessment lab, you will be given a series of questions that will be graded.

**What is Web API?**

 Bookmark this page

**What is the Web?**

**NOTE: This content is at a very basic level. You may skip this topic if you are already familiar with these concepts.**

Nowadays, networking and the Internet are very popular. Software running on different computers can "talk" to each other. The "languages" used by software to "talk" are **protocols**. You may have heard about protocols such as TCP, HTTP, FTP, etc.

The **Web** is a group of computers in a network, and they communicate with each other through HTTP protocol, typically. The Web can be big or small, for example:

* On an intranet of a company, if there are some computers that communicate with each other through HTTP and share documents via a web browser, that can be considered a SMALL web.
* On the Internet, billions of computers and devices communicate through HTTP, forming a BIG web. As a matter of fact, it is one of the largest things on the Earth - the World Wide Web.

**What is a Web Server**

Usually, when software is communicating across a network, the role of each software application may be different. If an application holds the **resources**, such as data or program logic, and can provide these resources to other applications, we refer to the application, and the computer, as a **server**. In contrast, if the application does not have the resources it needs, it has to request the resources from other applications through the network. We call this role a **client**.

If an application can provide data and/or computing power across a web, we call it a **web server**. Because the computers/devices on a web are connected by HTTP protocol, we can easily infer that a web server must provide/expose resources through HTTP protocol.

There are a variety of resources the web server can provide to the web clients, but we can categorize resources into just two categories - **data**and **computing power**. Data can be represented by many things on the server, such as an HTML page, a PNG image, an MP3/MP4 file or a bunch of records in a database. Computing power means the computing ability of a server. Servers can typically offer greater computing resources than a client may possess, depending on the computer hardware used.

The client can send parameters to the server through requests. After the server accepts the parameters and executes a computation of some sort, it returns the result back to the client through a response.

* If a web server provides data resources, and the data are in HTML format (we know a browser can render HTML content), we call this web server resource a **Website**.
* If a web server provides a data resource, and the data is serialized in XML, JSON or other formats, we call this a **Data Web Service** or **Data Service**.
* If a web server provides computing power resources, which means the web client can invoke the computing power on the web server remotely, and the web server will respond with the computing result (in proper format), we call this web server **Web Services** or **Web API**

As you may notice, regardless of the type of resources, there must be some result or data in the response to the client. The result data proves the existence of resources on the server, and we call these data the **representation** of resources.

**What is Web API**

API stands for Application Programming Interface. An API is critical for referencing and invoking members in class libraries. If you want to use the members in class libraries successfully, your code must comply with the API defined by the developer of those class libraries.

After referencing a class library, the application can easily call or utilize the functionality of the class library on a single computer. The real problem is when the application and class library are not on the same computer. How do we make a call to an API on a different computer than our client?

There are several generations of solutions for this problem, such as Remote Procedure Call (RPC), Common Object Request Broker Architecture (CORBA), etc. Eventually, developers grafted remote calls onto web servers. This results in the class libraries residing on the web server and exposing the API of class libraries through the HTTP protocol.

In this way, web servers centralize the computing power and publish it as web services, while the web clients consume the computing power remotely. Since the web clients are still calling the API of the class libraries but remotely through the web, we call this kind of web service, **Web API**. In short, if a web service acts as the wrapper of class libraries and just exposes remote computing ability, we call it Web API.

There are many styles and rules for designing these web APIs,

* Simple Object Access Protocol (SOAP) provides a convention for representing remote procedure calls and responses. If a web service adopts SOAP, we call this web service a **SOAP Web Service** or **SOAP Web API**.
* Representational state transfer (REST) is a set of architectural constraints (NOT a protocol or standard), for web application design. Since a web service is a kind of web application, we can apply REST constraints to the design of web services. Once a web service adopts REST concepts, we call it a **RESTful Web Service** or **RESTful Web API**.

In much the same way that we simply call a local variable, "variable", and call a parameter variable, "parameter", we always simplify the name SOAP Web Service to Web Service, and the RESTful Web Service to Web API. This implies that most of the modern Web APIs tend to adopt the REST concept as their design principle.

So, what is REST? In the next module, we will give it a closer look.

**Lesson 2 Serialization and Deserialization**

**Data Serialization and Deserialization**

As we've seen in the demo, when the REST service sent a Product type object or array back to the client through an HTTP response, the object or array was converted to a string. In other words, we use a string to represent the object or array. Because the string is a series of characters that are actually sent one at a time, or in a serial communication style, we call this conversion **serialization**. In contrast, when the client received the string, which was a serialized object or array, it had to convert the string back to the object or array. Once the object or array is reconstructed into an object or array, the application can use it. Since the object or array is a small blob of memory and no longer a serial of characters, we call this reverse conversion **deserialization**.

Some important notes,

1. The serialized object (a string) is a **representation** of the object.
2. You can serialize the object to its representation with any encoding format, but two encoding formats are industry standard – XML and JSON. JSON format is more compact and widely supported by JavaScript/.NET libraries, it is the major serialization format of REST Web API.
3. When we serialize a source object to a string and deserialize the string to a target object, it is not necessary that the source object and the target object have the same data type. When we refer to type here, we are referring to the type of object that we are dealing with. As we've seen in the code, the source object’s type is a class called Product and the target object’s type is a class called JsonProduct. The only constraint is the source data type and the target data type must share the same set of properties.

**JSON and XML**

JSON and XML are two of the most popular serialization encoding formats. Given a class Product,

public class Product {

public int ID { get; set; }

public string Name { get; set; }

public double Price { get; set; }

}

and its instance,

var product = new Product { ID = 101, Name = "Bike", Price = 79.99 };

The JSON format serialization will look like,

{"ID":101,"Name":"Bike","Price":79.99}

The XML format serialization will look like,

<?xml version="1.0" encoding="utf-16"?>

<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<ID>101</ID>

<Name>Bike</Name>

<Price>79.99</Price>

</Product>

or (SOAP encoding standard)

<?xml version="1.0" encoding="utf-16"?>

<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" id="id1">

<ID xsi:type="xsd:int">101</ID>

<Name xsi:type="xsd:string">Bike</Name>

<Price xsi:type="xsd:double">79.99</Price>

</Product>

Obviously, the JSON format serialization is far more lightweight than XML format serialization. The benefits of lightweight include reduced processing time on both the server side and the client side, and the low network traffic requirements.

**Simple Rules of JSON Format**

If you are not familiar with JSON we'll present a summary here for your information. The JSON format is very easy to understand and master. You can find the definition at [http://www.json.org](http://www.json.org/), and there is only one web page to read through.

Here is a quick summary of the JSON format,

1. A JSON array is represented by a pair of brackets. A JSON array may contain one or more objects in it. Objects in a JSON array are separated by a comma.
2. A JSON object is represented by a pair of curly braces. A JSON object may contain one or more members in it. Members in a JSON object are separated by commas.
3. A JSON object member is a name/value pair. The name of a member is a double-quoted string. The value of a member self-describes its data type with its format. For example, 10 is an integer, 10.0 is a float, "10" is a string, etc.
4. The JSON object's member value can be either a simple literal or another object or array.

The example below shows an example of representing an object in JSON. Given class named Product,

public class Product {

public int ID { get; set; }

public string Name { get; set; }

public double Price { get; set; }

public ICollection<Product> Accessories { get; } = new List<Product>();

}

and a list of instances of the Product class. Here we create to instances called p1 and p2. Each instance is given an ID, a name, and a price. Within each instance, we add two accessories, which are in turn, other Product class instances. We are creating the objects in this manner to demonstrate the use of objects and then arrays in our JSON representation.

var p1 = new Product { ID = 1001, Name = "P1", Price = 100.1 };

p1.Accessories.Add(new Product { ID = 1002, Name = "P1A1", Price = 100.2 });

p1.Accessories.Add(new Product { ID = 1003, Name = "P1A2", Price = 100.3 });

var p2 = new Product { ID = 2001, Name = "P2", Price = 200.1 };

p2.Accessories.Add(new Product { ID = 2002, Name = "P2A1", Price = 200.2 });

p2.Accessories.Add(new Product { ID = 2003, Name = "P2A2", Price = 200.3 });

var products = new List<Product>();

products.Add(p1);

products.Add(p2);

the JSON serialization of the product list object is

[{"ID":1001,"Name":"P1","Price":100.1,"Accessories":[{"ID":1002,"Name":"P1A1","Price":100.2,"Accessories":[]},{"ID":1003,"Name":"P1A2","Price":100.3,"Accessories":[]}]},{"ID":2001,"Name":"P2","Price":200.1,"Accessories":[{"ID":2002,"Name":"P2A1","Price":200.2,"Accessories":[]},{"ID":2003,"Name":"P2A2","Price":200.3,"Accessories":[]}]}]

It's pretty hard to read unless you are a JSON veteran. But if you copy this string to a JSON formatter (such as <https://jsonformatter.curiousconcept.com/>), you will get an output that is easier to decipher, such as this. Note the square brackets [] denoting arrays while the curly braces {} denote objects.

[

{"ID":1001, "Name":"P1", Price":100.1, "Accessories":[

{"ID":1002, "Name":"P1A1", "Price":100.2, "Accessories":[]},

{"ID":1003, "Name":"P1A2", "Price":100.3, "Accessories":[]}]

},

{"ID":2001, "Name":"P2", "Price":200.1, "Accessories":[

{"ID":2002, "Name":"P2A1", "Price":200.2, "Accessories":[]},

{"ID":2003, "Name":"P2A2", "Price":200.3, "Accessories":[]}]

}

]

For C# developers, please note that any collection in C# will be serialized to a JSON array. There are no other collection data structures (such as list and set) equivalents, in JSON.

**Web Server and Web Application**

There are various web server software applications like Internet Information Server (IIS), Apache Tomcat, NodeJS, Nginx, Jetty, and so on. The web server software plays the role of the container for web applications. It receives HTTP requests from clients, regardless whether the client is a smartphone or a desktop application. They then forward the information included in the requests to the web applications which are sitting in the container. After the web application takes actions based on the request and generates a proper result, such as a JSON string, the web server will wrap the result in an HTTP response and send the response back to the client. In short, a web server’s job is to accept requests and send responses back; the jobs of serialization, deserialization, and computing actions are performed by web applications, not by the web server.

As we mentioned previously, a website, a web service (web API) and a web data service are all web applications. There are many programming languages and frameworks for us to choose from to build web applications. For example,

* We can build a website or REST web API using C# and ASP.NET MVC, then host that website in IIS
* We can build all kinds of web services (but not websites) using C# and WCF, then host in IIS
* We can build a website, web service, or REST web API using Spring MVC and Java, then host in Tomcat
* We can build a website and REST web API using PHP, and host them in Tomcat or Nginx

Please note, not all web applications need web servers. For example, web applications created using Node or ASP.NET Core can be **self-hosted**, meaning they don't need the web server to accept the HTTP request and send back the HTTP response, they can do it themselves.

## Labs Overview

### Labs Overview

This module of the course includes two tutorial style labs and one self-assessment lab. Some simple lab configuration is required.

The tutorial labs in this module are designed to help you understand the client server model and also build base knowledge on how to serialize and deserialize objects in JSON.

The final lab in this module is a self-assessment lab. In this lab, you will apply what you've learned during the tutorial labs.

#### Tutorial Labs

1. Run a Client/Server System

In this lab, you will run a web server and a web client on a single computer and use the client to call the server. Although somewhat basic, it allows you to understand how the client-server system works.

1. Serialize and Deserialize with JSON

The goal of this lab is to help you understand that JSON format is supported by both C# and JavaScript, which makes the data exchange between different types of applications possible.

This lab also helps you learn how to use .NET Core Command Line (dotnet cli) to create, build and run applications.

## Configuration To Set Up The Lab Environment

 Bookmark this page

# **Build Development Environment**

## Overview

The purpose of this laboratory is to build the development environment for this course. There are four tools for us to download and install,

* .NET Core SDK
* Visual Studio Code
* Postman
* MySQL

## .NET Core SDK

Login your Windows, MacOS, or Linux system. Some of the below steps may require admin or root privilege. You can use Run As Administrator on a Windows computer or sudo on Mac/Linux computers.

Open your web browser and navigate to <https://aka.ms/edx-dev247x-ne>. The **SDK** download option should be selected by default.

Please note, the .NET Core Runtime will be installed with .NET Core SDK, so we don't need to download and install the .NET Core Runtime separately. The .NET Core Runtime download is for those who only want to run the .NET Core applications and don't want to develop/debug .NET Core applications.

Download the SDK by clicking the link that corresponds to your operating system platform. After the download finishes, double click the installer and follow the instructions.

To verify the installation, start the command line (on Windows) or terminal (on MacOS/Linux). Type the command dotnet --version and hit enter, if you see a version number in the output, the installation is successful.

## Visual Studio Code

Open your web browser and navigate to <https://aka.ms/edx-dev274x-vs>. Find the Visual Studio Code download link on the page (the page layout changes from time to time). The smart download will provide the installer against your operating system. Click the download link, wait for the download finish.

After the download finishes, run the installer for your operating system and follow the instructions. For Windows users, on the **Select Additional Tasks** page, we suggest selecting the following options:

* Add "Open with Code" action to Windows Explorer file context menu
* Add "Open with Code" action to Windows Explorer directory context menu
* Add to PATH (available after restart)

After the installation finishes, launch Visual Studio Code.

* On the main menu, click View->Extensions, the EXTENSIONS panel shows up.
* In the panel find the C# extension for Visual Studio Code (at the top).
* Install the extension and restart Visual Studio Code.

Now your Visual Studio Code installation is complete and ready for C# development.

Most of the coding gurus like to zoom the editor in and out with the mouse wheel. If you prefer to use this functionality:

* click File->Preferences->Settings
* add "editor.mouseWheelZoom": true to the right panel (custom configurations).

## Postman

Open your web browser and navigate to <https://www.getpostman.com/>. On the home page, choose and download the installer appropriate for your operating system.

After the download finishes, run the installer and follow the instructions.

After the installation completes, Postman will start automatically. You have to login to use Postman.

To verify the installation:

* in the right panel (Builder), keep the GET selected
* input http://www.google.com in the Enter request URL text box, then click Send button. You should see response content in the panel.
* If the **Status:** is 200 OK Postman is working correctly.

## MySQL

### Install on Windows

To run on Windows, MySQL needs some dependencies. Before installing MySQL, please download and install these two Microsoft official Visual C++ runtime libraries:

* [Microsoft Visual C++ 2013 Runtime](https://aka.ms/edx-dev247x-vs1)
* [Microsoft Visual C++ 2015 Runtime](https://aka.ms/edx-dev247x-vs2)

Open your web browser and navigate to <https://dev.mysql.com/downloads/mysql/>. Click the Download button for the version you want to install. Note, the installer itself is x86 32bit, but it can help you install the x64 MySQL server on your machine.

* On the Download MySQL Installer page, click the Download button for either the online installer (the small size one) or the offline installer (the large size one).
* After downloading the installer, double click the installer and follow the instructions.
* On the **Choosing a Setup Type** page, select **Custom** option, then click Next button.
* On the **Select Products and Features**, select the components listed below and move them to the installation list use the right arrow button,
  + MySQL Servers -> MySQL Server -> MySQL Server 5.7 -> MySQL Server 5.7.18 - X64
  + Applications -> MySQL Workbench -> MySQL Workbench 6.3 -> MySQL Workbench 6.3.9 - X64
  + MySQL Connectors -> Connector/NET -> Connector/NET 6.9 -> Connector/NET 6.9.9 - X86
  + Documentation -> Samples and Examples -> Samples and Examples 5.7 -> Samples and Examples 5.7.18 - X86

Follow the instructions, when the installer asks for the password for root, enter a strong password. Please note, the examples of this course use a weak password of 123456 but please don't use such a simple password in a production environment.

Follow the rest of the instructions in the installation wizard. On the final screen, you can check **Start MySQL Workbench after Setup** on last page or start the MySQL Workbench manually after the installation is completed. Once you have started MySQL Workbench, login with the root password and verify that the sakila and world databases exist in the SCHEMAS section.

In the Query 1 editor window, input the SQL query below:

USE sakila;

SELECT \* FROM actor;

USE world;

SELECT \* FROM country;

Click the execute button (the lightning icon), if there are no errors and you see the query results, MySQL is successfully installed.

### Install on Mac OS

Please reference the following URL for installation instructions for the Mac OS:

* <https://dev.mysql.com/doc/mysql-osx-excerpt/5.7/en/>
* <https://dev.mysql.com/doc/refman/5.7/en/osx-installation.html>

### Install on Linux

Please reference the following URL for installation instructions for Linux:

* <https://dev.mysql.com/doc/refman/5.7/en/linux-installation.html>

## To Install Visual Studio Code

### To Install Visual Studio Code

Visual Studio Code is lightweight and is compatible with most available hardware and platform versions.

#### Mac OS X

1. [Download Visual Studio Code](https://aka.ms/edx-dev247x-vscode) for Mac OS X.
2. Double-click on the downloaded archive to expand the contents.
3. Drag Visual Studio Code.app to the Applications folder, making it available in the Launchpad.
4. Add Visual Studio Code to your Dock by right-clicking on the icon, and choosing Options, Keep in Dock.

#### Linux

1. Download Visual Studio Code for your distribution, [.deb](http://go.microsoft.com/fwlink/?LinkID=760868) for Debian-based distributions such as Ubuntu or [.rpm](http://go.microsoft.com/fwlink/?LinkID=760867) for Red Hat-based distributions, such as Fedora or CentOS.
2. Install the package through a graphical user interface package manager by double-clicking on the package file, or through the command line:

bash # For .deb sudo dpkg -i.deb

# For .rpm (Fedora 21 and below) sudo yum install.rpm

# For .rpm (Fedora 22 and above) sudo dnf install.rpm

1. Visual Studio Code should now be available to run through the launcher or the command line by running code.

Tip: Run code in any folder to start editing files in that folder.

#### Windows

1. [Download Visual Studio Code](https://aka.ms/edx-dev247x-vscode1) for Windows.
2. To launch the setup process, double-click **VSCodeSetup.exe**.

By default, Visual Studio Code is installed in the "C:\Program Files (x86)\Microsoft VS Code" folder location (for a 64-bit machine). The setup process should only take about a minute.

**Note:** .NET Framework 4.5 is required for Visual Studio Code. If you are using Windows 7, please ensure [.NET Framework 4.5](https://aka.ms/edx-dev247x-net) is installed.

For more detailed instructions and tips, visit the full Microsoft Visual Studio Code Installation Instruction guide [here](https://code.visualstudio.com/Docs/editor/setup).

## .NET Core Versions

### Update

About two months after the release of this course, Microsoft shipped a new version of .NET Core platform - .NET Core 2.0. Although there are some breaking changes of .NET Core 2.0, these changes won't impact the content of this course.

But, since the old .NET Core SDK download link always gives you the latest version of .NET Core SDK, as a new .NET Core developer, you may miss the chance to download the .NET Core SDK 1.x for the labs of this course. Thus, you may run into some issue when you build and run the labs.

Below are some solutions you can take to make the labs run without issues.

###### Solution 1: Keep Using .NET Core SDK 1.x

Go to the .NET Framework download page, (https://www.microsoft.com/net/download/windows). Scroll down to the download link table of .NET Core. Then, download the \*\*x64 Installer (.exe)\*\* of the \*\*Long Term Support (v1.x)\*\* .NET Core SDK.

Start the command line (or terminal on macOS), execute

dotnet --version

to see if the version number matches the one you installed. If the version number is higher than you installed, for example, 2.0.3, that means a newer version is installed already, and you have to remove it. On Windows, you can uninstall the higher version from Control Panel. On macOS, you just delete the higher version from the Applications folder.

###### Solution 2: Rewrite the Code

If you already installed .NET Core SDK 2.x, don't worry, follow the instructions to write the code from scratch, you won't have any issue. The only thing you should pay attention to is - when you execute the

dotnet add package

command to add references/dependencies for your application, please go to nuget.org, (https://www.nuget.org) to check the package version for .NET Core 2.x.

For example, the package MySql.Data.EntityFrameworkCore version  7.0.7-m61  is a package widely referenced by the labs. If you check this package on NuGet, its latest version is 8.0.9-dmr , which is for .NET Core 2.x. That means you should execute the command

dotnet add package MySql.Data.EntityFrameworkCore --version 8.0.9-dmr

to add the package to your application.

(\*\*Tips\*\*: On the top of the page, click the \*\*.NET CLI\*\* tab you will get this command.)

By the way, when you create a web API application or a web application using dotnet new CLI command, a package named Microsoft.AspNetCore.All has been added to the project automatically. You will see

<ItemGroup>  
  <PackageReference Include="Microsoft.AspNetCore.All" Version="2.x.x" />  
</ItemGroup>

in the .csproj file. That means, you no longer need to add any ASP.NET Core package manually. In another words, you no longer need to execute a command such as

dotnet add package Microsoft.AspNetCore.StaticFiles

###### Solution 3: Update the Lab Projects

Sometimes we want to run the finished lab project to see the running result first. In this case, if you've already installed .NET Core SDK 2.x and don't want to downgrade to .NET Core 1.x, you have to update the lab projects.

Below is a .csproj file of a .NET Core 1.x web API application,

<Project Sdk="Microsoft.NET.Sdk.Web">  
  <PropertyGroup>  
    <TargetFramework>netcoreapp1.1</TargetFramework>  
  </PropertyGroup>  
  <ItemGroup>  
    <Folder Include="wwwroot\" />  
  </ItemGroup>  
  <ItemGroup>  
    <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />  
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />  
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />  
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1" />  
    <PackageReference Include="MySql.Data.EntityFrameworkCore" Version="7.0.7-m61" />  
  </ItemGroup>  
</Project>

Here are the steps to update:

1. Update the value of <TargetFramework> element to netcoreapp2.0  
2. Merge all Microsoft.AspNetCore.\* packages  
3. Update the version of other packages to the latest (check on nuget.org, (https://www.nuget.org/))

After the update, the content of the .csproj file looks as below:

<Project Sdk="Microsoft.NET.Sdk.Web">  
  <PropertyGroup>  
    <TargetFramework>netcoreapp2.0</TargetFramework>  
  </PropertyGroup>  
  <ItemGroup>  
    <Folder Include="wwwroot\" />  
  </ItemGroup>  
  <ItemGroup>  
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.3" />  
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="2.0.0" />  
    <PackageReference Include="MySql.Data.EntityFrameworkCore" Version="8.0.9-dmr" />  
  </ItemGroup>  
</Project>

At last, remove the .vscode folder from the project folder. Then, open the project folder with Visual Studio Code, switch to the \*\*Debug\*\* panel. You should see there's a red dot on the \*gear\* icon. Click the icon to re-generate the .vscode folder and files in the folder.

Now, run the lab application, there should not be issues. If you still see issues, please leave messages on the course web page, we will help you to solve the problems.

## Task 1 Run Client Server Single Machine

# **Run a Client-Server System on Single Machine**

## Overview

In this lab, you will run a web server and a web client on a single computer and use the client to call the server. While a very basic scenario, this allows you to understand or regain familiarity with how the client-server system works. Plus, it helps you learn how to use .Net Core to create server and client app.

## Configuration

Get the starter files for this task from the GitHub repo, <https://github.com/MicrosoftLearning/edX_DEV247X_Build_WebAPIs_Using_ASP.NET.git>. It is recommended that you clone the entire repo to your computer for easier access to the required files in an offline scenario.

Create a folder on your computer as a working directory. Give it a name of your choosing but we suggest starting with a WebAPI folder name. Then create a subfolder or directory for Module1.

For this task, locate the **Run a Client-Server System on a Single Machine**folder. Copy the WebServer and WebClient directories and their entire contents to your Module1 working directory.

## Start the Web Server

Run the command line (on Windows) or Terminal (on MacOS/Linux), enter the **WebServer** directory. Execute the commands below, pressing ENTER after each command line:

dotnet restore

dotnet build

dotnet run

If you see the output below, the server is started:

Now listening on: http://localhost:5000

Application started. Press Ctrl+C to shut down.

## Start the Web Client

Open a new console or command window. Enter the **WebClient** directory, execute the commands below:

dotnet restore

dotnet build

dotnet run http://localhost:5000

If you see the output below, the client is started:

Acceptable input: integer | ALL | EXIT

## Interaction of Client-Server

In the WebClient command line window, try the listed input options below:

* Enter an integer between 0 and 3: you will get the data of the rocket that has this ID
* Enter an integer not in the range of 0 to 3: you will see "Nothing to print."
* Enter ALL (case insensitive): you get the data list of all rockets
* Enter EXIT (case insensitive): the client says "Thank You!" then stops.

You can now go back to the console for the server and press CTRL + C to stop the server.

## Task 2 Run Client Server Separate Machine

# **Run a Client/Server Environment on Separate Computers (Optional)**

In the event that you would like to test the Server/Client concept using two separate computers, follow the steps outline below.

* Run the WebServer application on a different computer.
* Copy the WebServer folder contents to the other computer.

## Start the Web Server

Run the command line (on Windows) or Terminal (on MacOS/Linux), enter the **WebServer** directory. Execute the commands below, pressing ENTER after each command line:

dotnet restore

dotnet build

dotnet run

If you see the output below, the server is started:

Now listening on: http://localhost:5000

Application started. Press Ctrl+C to shut down.

* Get the IP address of the server computer, for example 192.168.1.101.
* Switch back to your client computer
  + When you start the WebClient application, execute the commands below (replace the IP listed here with the one for your server computer that you gathered above:

dotnet restore

dotnet build

dotnet run http://192.168.1.101:5000

If there is no network firewall blocking the client access to the server, you will see the same output as in Task 1 before.

**Task 1 Serialize Deserialize C#**

 Bookmark this page

C# Serialization and Deserialization

Let's go through the serialization and deserialization of C# first.

**Scaffolding the Application**

Start the command line (on Windows) or Terminal (on MacOS/Linux). Navigate to the path to where you want to create the C# project. Execute the commands below:

dotnet new console --name SeDes

cd SeDes

dotnet add package NewtonSoft.Json

dotnet restore

dotnet build

dotnet run

If you see the Hello World! on the screen, congratulations! The .NET Core C# console application scaffolding is created, and you added the class library reference to NewtonSoft.Json successfully. .NET Core uses NuGet to manage the dependencies for your application. That means if NewtonSoft.Json has further dependencies to be added, the NuGet package manager will pull them to your local machine automatically from the proper sources.

**Edit Project in IDE**

We use Visual Studio Core as the IDE in this course. If you are a user of Visual Studio, there is a significant difference between VS and VS Code in terms of how they manage a project. To open a project in Visual Studio, you open the solution file (.sln) or project file (.csproj). Visual Studio Code is a *Editor Style* IDE, when you open a project in VS Code, it opens the **folder** which contains the project.

For this project, you have at least three ways to open it in VS Code:

* To open your project in VS Code, type code . in the command line and hit enter. The code.exe is the executive file of VS Code and its path was registered in the system variables when you installed it. The . is the relative path of current folder.
* In the Windows Explorer, navigate to the project folder, right click the folder and click **Open with Code**. Or you can open the project folder in Windows Explorer and right click any blank area, then click **Open with Code**.
* Start the VS Code, use main menu's File | Open Folder... to open the project folder.

After opening the project folder, the **EXPLORER** panel expands by default, and you can see the project folder name on the toggle-expandable tab. Click the **New File** icon and input file name **Rocket.cs** then hit enter, VS Code will open this file for you. If the VS Code suggests you to add configurations and restore libraries, click **Yes** on the pop up message. You will typically see this because this is the first time VS Code recognizes you are editing a C# project.

Unlike Visual Studio, which generates most of the boilerplate code for us, we have to edit the code from scratch in VS Code. Add the code below to the Rocket.cs file:

namespace SeDes {

public class Rocket {

public int ID { get; set; }

public string Builder { get; set; }

public string Target { get; set; }

public double Speed { get; set; }

}

}

We are creating a simple class like this because this *model* has properties that are frequently used types - **int**, **double** and **string**.

Now, open the Program.cs in VS Code and import the namespace of JSON utility classes:

using Newtonsoft.Json;

Then add these two static method members to the Program class:

// Serialize a Rocket array to JSON string

public static string DoSerialization() {

Rocket[] rockets = {

new Rocket{ ID = 0, Builder = "NASA", Target = "Moon", Speed=7.8},

new Rocket{ ID = 1, Builder = "NASA", Target = "Mars", Speed=10.9},

new Rocket{ ID = 2, Builder = "NASA", Target = "Kepler-452b", Speed=42.1},

new Rocket{ ID = 3, Builder = "NASA", Target = "N/A", Speed=0}

};

var json = JsonConvert.SerializeObject(rockets);

return json;

}

// Deserialize a JSON string back to a Rocket array

public static void DoDeserialization(string json) {

var rockets = JsonConvert.DeserializeObject<Rocket[]>(json);

foreach (var r in rockets) {

System.Console.WriteLine($"ID:{r.ID} Builder:{r.Builder} Target:{r.Target} Speed:{r.Speed}");

}

}

Finally, edit the Main method as below:

static void Main(string[] args) {

var json = DoSerialization();

System.Console.WriteLine(json);

System.Console.WriteLine("================");

DoDeserialization(json);

}

**Run the Application**

To run this application, use Debug | Start Debugging (F5) in main menu. Or, click the Debug icon on the left vertical tool bar switch to the debug panel, then click the green triangle icon of .NET Core Launch (console).

If the VS Code configuration is not correct or if you receive application build errors, you can also build and run from the command prompt. Switch back to the command prompt window and execute the following commands:

* dotnet build
* dotnet run

Without error, you will see:

[{"ID":0,"Builder":"NASA","Target":"Moon","Speed":7.8},{"ID":1,"Builder":"NASA","Target":"Mars","Speed":10.9},{"ID":2,"Builder":"NASA","Target":"Kepler-452b","Speed":42.1},{"ID":3,"Builder":"NASA","Target":"N/A","Speed":0.0}]

================

ID:0 Builder:NASA Target:Moon Speed:7.8

ID:1 Builder:NASA Target:Mars Speed:10.9

ID:2 Builder:NASA Target:Kepler-452b Speed:42.1

ID:3 Builder:NASA Target:N/A Speed:0

Here are a couple of points to keep in mind:

* To be a good JSON reader is very important to the RESTful Web API developers. You should recognize it is an array in the pair of bracket, all JSON field names are quoted, the field values are self descriptive on their type - int values don't have fractional parts, double values have fractional parts even if it's zero, string values are always quoted.
* The source type of serialization and the target type of deserialization are not required to be the same. The only requisite is that these two types share the same set of properties. This helps reduce the coupling between the web client and web services.

**Bonus**

To experience the loose-coupling between serialization and deserialization, please add the class below to the project:

namespace SeDes {

public class UFO {

public string Target { get; set; }

public double Speed { get; set; }

}

}

and update the DoDeserialization method to:

public static void DoDeserialization(string json) {

var ufos = JsonConvert.DeserializeObject<UFO[]>(json);

foreach (var ufo in ufos) {

System.Console.WriteLine($"Target:{ufo.Target} Speed:{ufo.Speed}");

}

}

Rebuild and run the application, you will see the new output like:

[{"ID":0,"Builder":"NASA","Target":"Moon","Speed":7.8},{"ID":1,"Builder":"NASA","Target":"Mars","Speed":10.9},{"ID":2,"Builder":"NASA","Target":"Kepler-452b","Speed":42.1},{"ID":3,"Builder":"NASA","Target":"N/A","Speed":0.0}]

================

Target:Moon Speed:7.8

Target:Mars Speed:10.9

Target:Kepler-452b Speed:42.1

Target:N/A Speed:0

**Source Code**

All the source code shows in this lab (including the bonus) is in the zip file.

## Task 2 Serialize Deserialize JS

 Bookmark this page

## JavaScript Serialization and Deserialization

Nowadays, JavaScript can be used to program either server side applications or client side applications. In this example, we will look at the client side JavaScript application which runs in the web browser.

### Create a HTML Scaffolding

Open a plain text editor, such as VS Code, Notepad, Atom or Sublime. Create an empty file, add the HTML content below and save it as **App.html**.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>SeDes</title>

<script>

// Your code will be here

</script>

</head>

<body>

</body>

</html>

### Add the JavaScript Code

Add the JavaScript code to the body of <script>...</script> element.

var rockets = [

{ ID: 0, Builder: "NASA", Target: "Moon", Speed: 7.8 },

{ ID: 1, Builder: "NASA", Target: "Mars", Speed: 10.9 },

{ ID: 2, Builder: "NASA", Target: "Kepler-452b", Speed: 42.1 },

{ ID: 3, Builder: "NASA", Target: "N/A", Speed: 0.0 }];

// Serialization

var jsonStr = JSON.stringify(rockets);

document.write(jsonStr);

document.write('<br/>');

document.write("========================");

document.write('<br/>');

// Deserialization

var objects = JSON.parse(jsonStr);

for (var i in objects) {

document.write("ID:" + objects[i].ID + " Builder:" + objects[i].Builder + " Target:" + objects[i].Target + " Speed:" + objects[i].Speed);

document.write('<br/>');

}

### Run the JavaScript Application

To run the JavaScript application is very easy, just double click the App.html and open it in your local web browser. You will see the text content below shown on the web page:

[{"ID":0,"Builder":"NASA","Target":"Moon","Speed":7.8},{"ID":1,"Builder":"NASA","Target":"Mars","Speed":10.9},{"ID":2,"Builder":"NASA","Target":"Kepler-452b","Speed":42.1},{"ID":3,"Builder":"NASA","Target":"N/A","Speed":0}]

========================

ID:0 Builder:NASA Target:Moon Speed:7.8

ID:1 Builder:NASA Target:Mars Speed:10.9

ID:2 Builder:NASA Target:Kepler-452b Speed:42.1

ID:3 Builder:NASA Target:N/A Speed:0

Note the difference between the JavaScript object code in JavaScript and the JSON string.

## Lab01 Self Assessment Lab

# **Serialization and Deserialization**

## Overview

The purpose of this lab is to help students understand JSON as a universal data exchange format that can be used to transfer data between applications written in different programming languages. For example, the developer can serialize the data object to a JSON string in a C# application then deserialize that JSON string back to a data object in a JavaScript application or vice versa.

## Prerequisites

* .NET Core CLI 1.0.1 installed
* Visual Studio Code installed

## Task 1

Given the following C# class

class Product

{

public int ID { get; set; }

public string Name { get; set; }

public double Price { get; set; }

}

1. Write a .NET Core C# application
2. Create a Product type object
3. Serialize the object to JSON string
4. Deserialize the JSON string back to a Product type object.

## Task 2

Given the following JavaScript object

var p1 = { id: 101, name: 'Product-101', price: 99 };

1. Create a JavaScript application within a simple HTML page
2. Serialize the data object to a JSON string
3. Deserialize the JSON string back to a data object

## Task 3

Since the C# object and JavaScript share the same set of properties, their JSON serialization are exchangeable. Can you prove this assumption? Your task here is to serialize an object in C# and then deserialize that string back to a JavaScript object.

### Hint

* You can create your C# application to serialize the object and then copy and paste that string into your JavaScript application.

Once completed, answer the assessment questions to test your knowledge and completeness of this assessment lab.