**25. Introduction to WEB Dev.**

* Web development is the building and maintenance of websites
* It’s the work that happens behind the scenes to make a website look great, work fast and perform well with a seamless user experience.
* In .net there is ASP.NET, it’s a web application framework which allow us to build dynamic web sites.
* It’s a server side scripting language.

1. **ASP.Net Web Forms**

* Web forms are designed to make building web-based application easy.
* It is event driven development framework. It is used to develop application with powerful data access. It provides server side control and event to create web application.
* It is a part of the ASP.NET framework.

1. **MVC**

* The MVC model-view-controller is a design pattern used to decouple interface(view), data(model), and application logic(Controller)
* Using MVC pattern of websites, requests are routed to a controller that is responsible for working with model to perform action /or retrieve data.
* The controller chooses the view to display , and provides it with the model
* The view renders the final page, based on the data in the model.
* ASP.NET gives a powerful, pattern-based way to build dynamic websites using the MVC pattern that enables a clean sepration of concerns.

1. **Rest Web API**

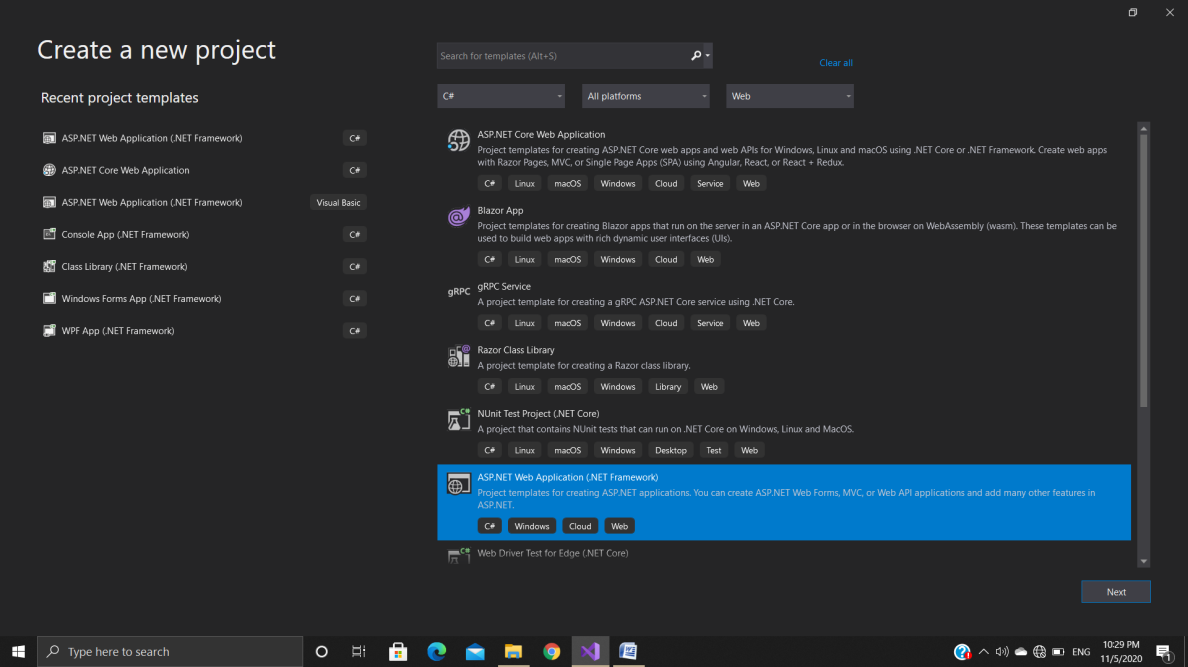
* ASP.NET web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices.
* ASP.NET makes it easy to build services that reach a broad range of clients, including browsers and mobile devices.
* It is an ideal platform for building RESTful applications on the .NET Framework.
* The term REST stands for Representational State Transfer.
* It is an architectural style that defines a set of rules in order to create web services.
* REST architecture is based on a few characteristics to be called it RESTful based services.

1. Client-Server
2. Stateless
3. Cacheable
4. Uniform Interface.

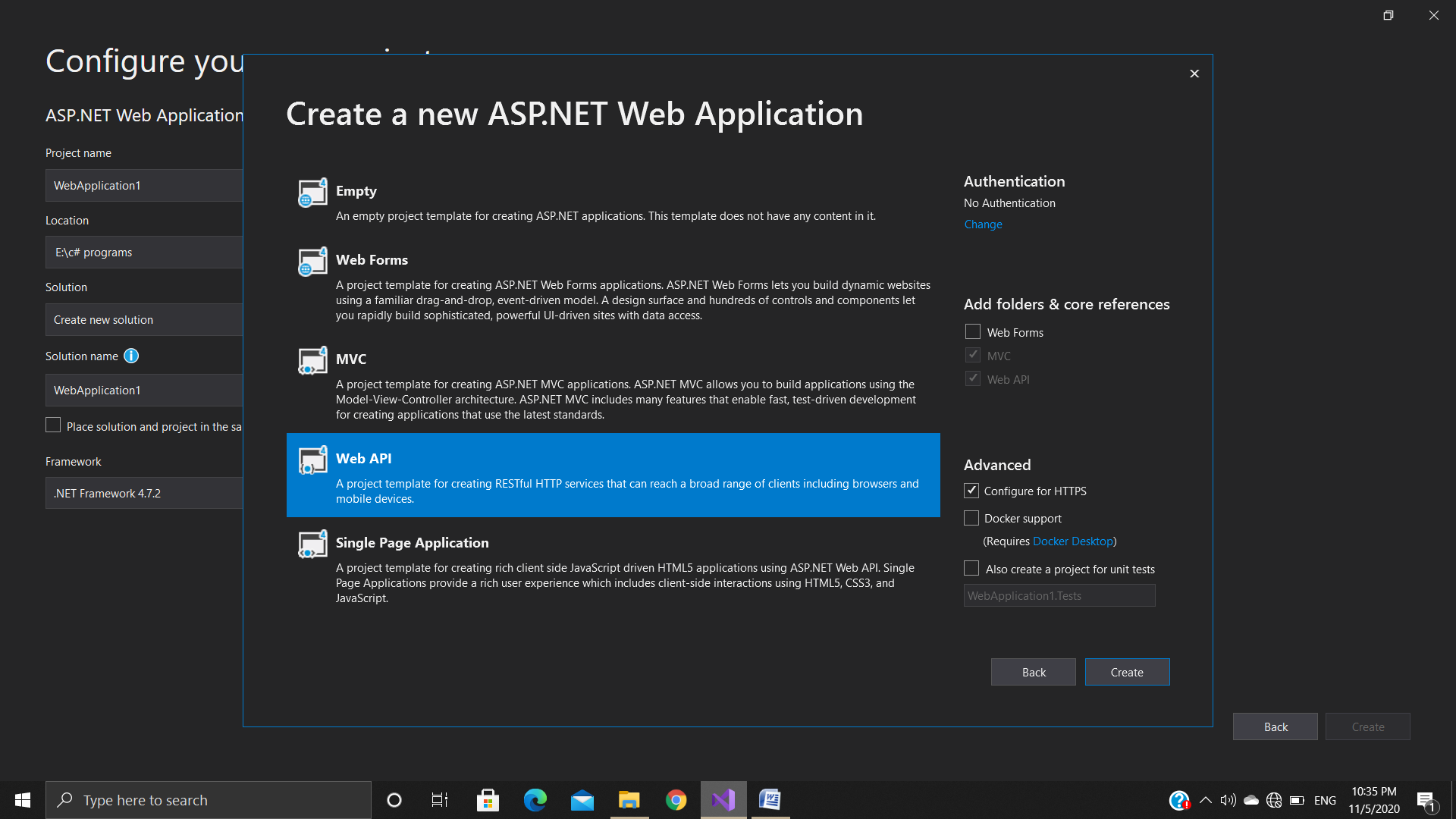
**26. Start with project**

1. **Create New Web API Project**

* Visual Studio includes Web API template which creates a new Web API project and includes all necessary references to get started.
* For this we need to open visual studio and click on to the create a new project, This will open popup as below.



* In this as a language select the C# and as a project type select the web.
* From rest options select ASP.NET Web Application template
* Then enter the project name and then click to create
* It will open the Template list as shown below.



* Select the Web API template notice that this has also selected the MVC and Web API checkbox.
* It means it will add all necessary folders and references for both MVC and Web API.
* Then click to the create button it will create a new Web API project.

1. **Setting up infrastructure**

* In the solution explorer of the project there is availability of some files and folders.
* At there , one of the file named Global.asax, Which includes Appliction-Start method from where execution of our application started.
* That firstly checks the routes in the method
* Under The App-Start folder there is two files RoutrConfig.cs which is the route file of the MVC and another is a WebApiConfig.cs which is the route file of the Web API project.That shows routes related Web API.
* By this Routes Controllers knows that hoe to navigate .
* There is also availability f Controller , Model and views folder.
* Under the controller folder there is a availability of both controllers Web API and MVC controllers.
* Values controller is a Web API controller which inherits from API Controller and available in System.web.HTTP namespace.
* Home Controller is from MVC Controller and available in System.web.MVC namespace.

1. **Create Controller, Model**

**Controller**

* Web API controllers handles incoming HTTP requests and send response back to the caller.
* We will add a controller that can return either a list of products or single product specified by id.
* Web API Controller is a class which can be created under the Controllers folder or any other folder which is our project’s root folder.
* The name of a controller class must end with “Controller” and it must be derived from System.Web.Http.ApiController class.
* There is a Values Controllers class added by Visual Studio by default when we created a new web API project.
* If we want to add our own controller then we can add it by Right click on the controller folder then select Add then select Controller.
* Methods name of the controllers can be same as HTTP verb name(Get, Post, Put, Delete) or it can be start with HTTP verb with any suffix or we can apply Http verb attributes to method.

**Model**

* A model is an object that represents the data in our application.
* It contains the properties.
* For adding the model first right clicks on the model folder and then selects a class.
* Then your required properties into that.

1. **Parameters (From URI, From Body)**

* Methods in Web API controller can have one or more parameters of different types.
* It can be either primitive type or complex type.
* Web API binds action methods with url’s query string or with request body depending on the parameter type.
* If parameter type is primitive type then it sets the value of a parameter from the query string and if the parameter type is complex then web api tries to get the value from request body by default.
* But if we want to change the default behavior then we can use this concepts of [FromUri] and [FromBody] attributes.
* Use [FromUri] attribute to force web API to get the value of complex type from the query string.
* And use [FromBody] to get the value of primitive type from request body, opposite to the default rules.
* For Example

Public class StudentController : ApiController

{

Public Student Post([FromUri] Student stud)

{

}

}

* We have applied [FromUri] attribute woth student parameter although that is complex type but because we apply [FromUri] attribute so, now web API will extract the value from the query string instead of request body.
* Public class StudentController : ApiController

{

Public Student Post([FromBody] Student stud)

{

}

}

* Same as before By appliying [FromBody] attribute we can take the value of primitive data type from the request body instead of the query string.

1. **Serialization**

* In c# serialization is the process of converting object into byte stream so that it can be saved in to memory , file or database or any persistence storage location.
* We can store object into memory with serilization and also retrieve it again when needed.
* The reverse of serialization is called deserialization.
* Together this process allows the data to be stored and transformed.
* It is an technology that enables an object to be converted into a stream of data so they can be easily passes across the system or machine.
* This format should be understandable by both end of a communication channel.
* It is used by the web services , remoting for transmitting data between server and a client.
* The namespace of serialization contain Iformatter interface which contain the methods serialize and de\_serialize that can used to save and load data to and from a stream.
* In order to implement serilization in .NET ,we basically require a stream and a formatter.
* Stream act as a container for serializes object.
* Formatter is used to serialize these objects onto the stream.
* The namespace used for serialization is are:

System.Runtime.Serilization

System.Xml.Serilization

System.Text.Json

**Advs of using serilization:**

* Ability to transmit data across the network in cross-platform compatible format.
* Also for saving it in a persistent or non-proprietary format.
* Passing an object from one application to another.
* Passing an object through a firewall as an XML string.
* .NET allows the following serilization techniques.

1. Binary serilization

2. XML and SOAP serilization

3. JSON serilization

**Binary serialization:**

* It is mechanism which writes the data to the output stream such as it can be used to re-construct the objectautomatically.
* Binary refers to that the necessary information that is required to create object is saved onto the storage media.
* It also preserves the instance identity.
* In other word the binary serialization the entire object state is saved where in XML only some of the object data is saved.
* Advantage

It is faster and even more powerful in the sense that ist provides support for complex object , read only properties.

* Disadvantage

It is not easily portable to another platform.

**XML serialization:**

* It converts only the public fields and properties of object or parameters and return the values of methods into an XML stream.
* XML serialization result in strongly typed classes with public properties
* Fields that is converted in serial for storage purpose or transport purpose.
* Besacuse XML is an open source standard , XML stream can be processed by any application as need.
* Implementing XML serialization in .NET is quite simple.
* The basic class we need is xmlserializer for both serialization and de-serialization.
* It much slower compare to binary serialization.

**JSON serialization:**

* As the name suggest it is used to convert object in to the JSON stream format.
* We can convert object in JSON stream format and also can get back that object from JSON stream using the concept of the serialization and de-serialization.
* The quickest method of converting between JSON text and a .NET object is using the jsonSerializer.
* The jsonserializer converts .NET object into their JSON equivalent and back again by mapping the .NET object property names to the JSON property names and copies the values for you.

1. **Routing**

* Web API routing routes an incoming HTTP requests to a particular action method on a web API controller.
* There is Two Type of Routing

1. Convention- based routing

* In 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.
* When we create Web API project using Web API template It also added WebApiConfig class in the App\_Start folder with default route.
* In that config.MapHttpAttributeRoutes() enables attribute routing.

1. Attribute routing

* Attribute routing uses [Route()] attribute to define routes.
* The Route attribute can be applied on any controller or action method.
* Attribute routing gives us more control over the URIs than convention based routing.
* We can use both Routing in a single Web API project.

1. **Config**

* We can configure web API to customize the behavior of web API hosting infrastructure and component such as routes, formatters, filters etc.
* Web API configuration process starts when the application starts.
* In Asp.NET application, configure Web API by calling GlobalConfiguration.Configure in the Application\_Start method.
* The configure() method requires the callback method where Web API has been configured in code. By default this is the static WebApiConfig.Register() method.
* WebApiConfig.Register mthod includes a parameter of HttpConfiguration type which is then used to configure the Web API.
* The ASP.NET Web Application project template automatically sets up the configuration code.
* The project template create a file named WebApiConfig.cs inside the App\_Start folder.
* This code file defines the delegates where we should put our Web API configuration code.

**PROGRAMS**

1. **Web API Example(With attribute routing and parameter binding)**

**Model(Employee.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace FirstWebAPI.Models

{

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**Controller(EmployeeDataController)**

using FirstWebAPI.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace FirstWebAPI.Controllers

{

public class EmployeeDataController : ApiController

{

static List<Employee> employees = new List<Employee>()

{

new Employee() {Id = 1, Name= "Tom"},

new Employee() {Id = 2, Name= "Rock"},

new Employee() {Id = 3, Name= "Peter"}

};

public IEnumerable<Employee> Get()

{

return employees;

}

public Employee Get(int id)

{

return employees.FirstOrDefault(s => s.Id == id);

}

[Route("api/employeeData/{id}/courses")]

public IEnumerable<string> GetStudentCourses([FromUri]int id)

{

if (id == 1)

return new List<string>() { "C#", "Android", "ASP.net" };

else if (id == 2)

return new List<string>() { "Web API", "Java", "HTML" };

else

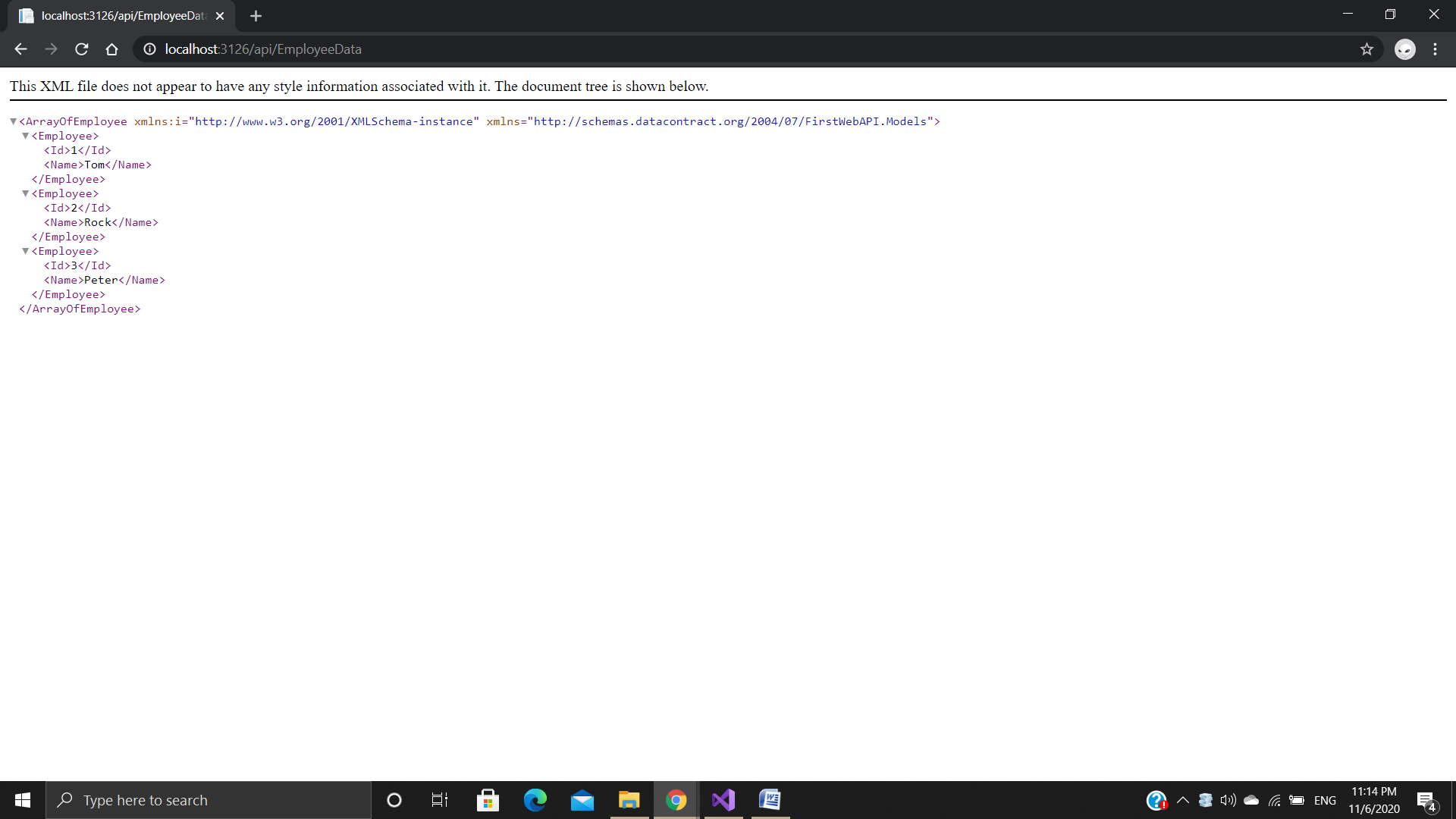
return new List<string>() { "CSS", "PHP", "JavaScript" };

}

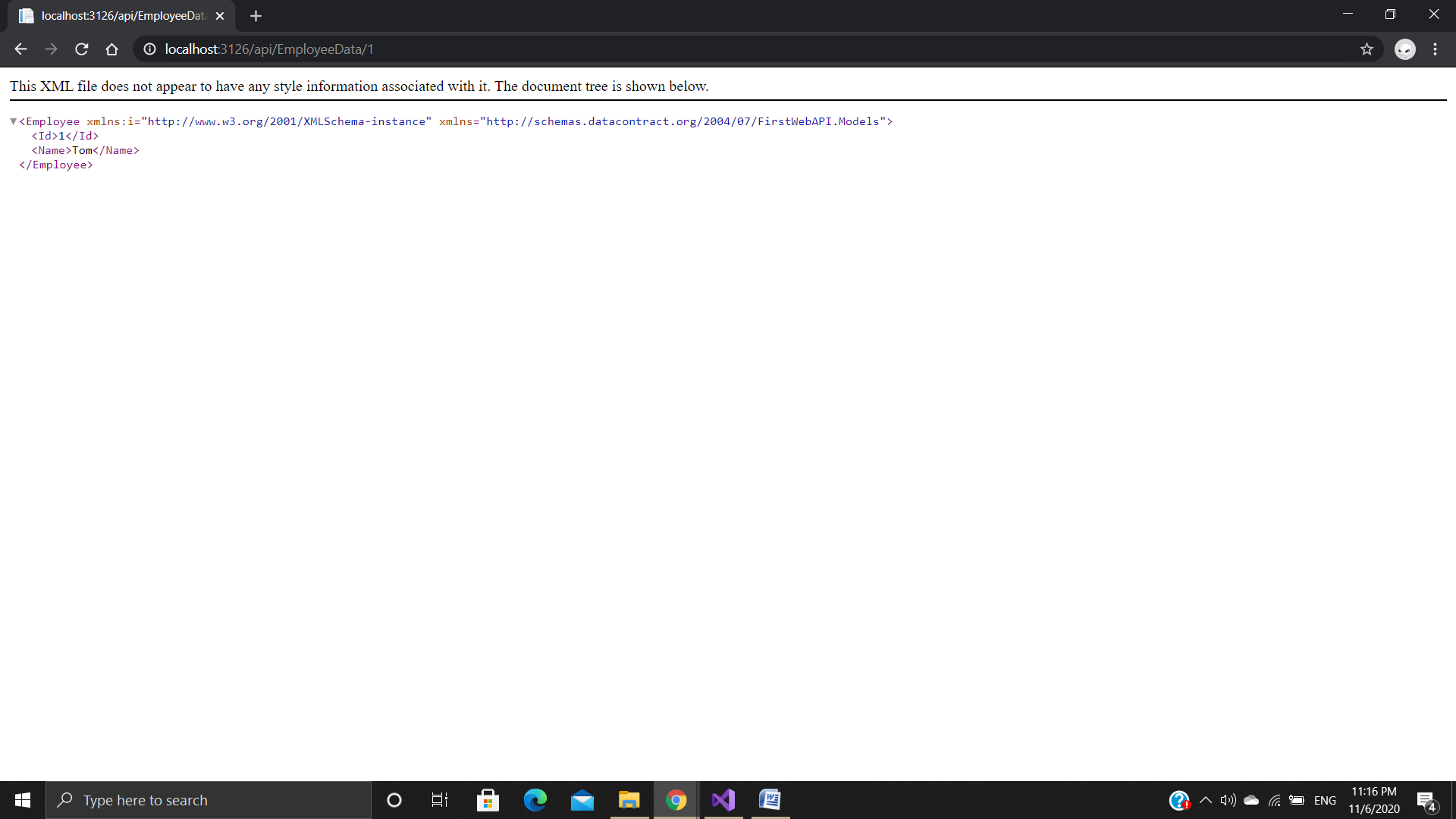
}

}

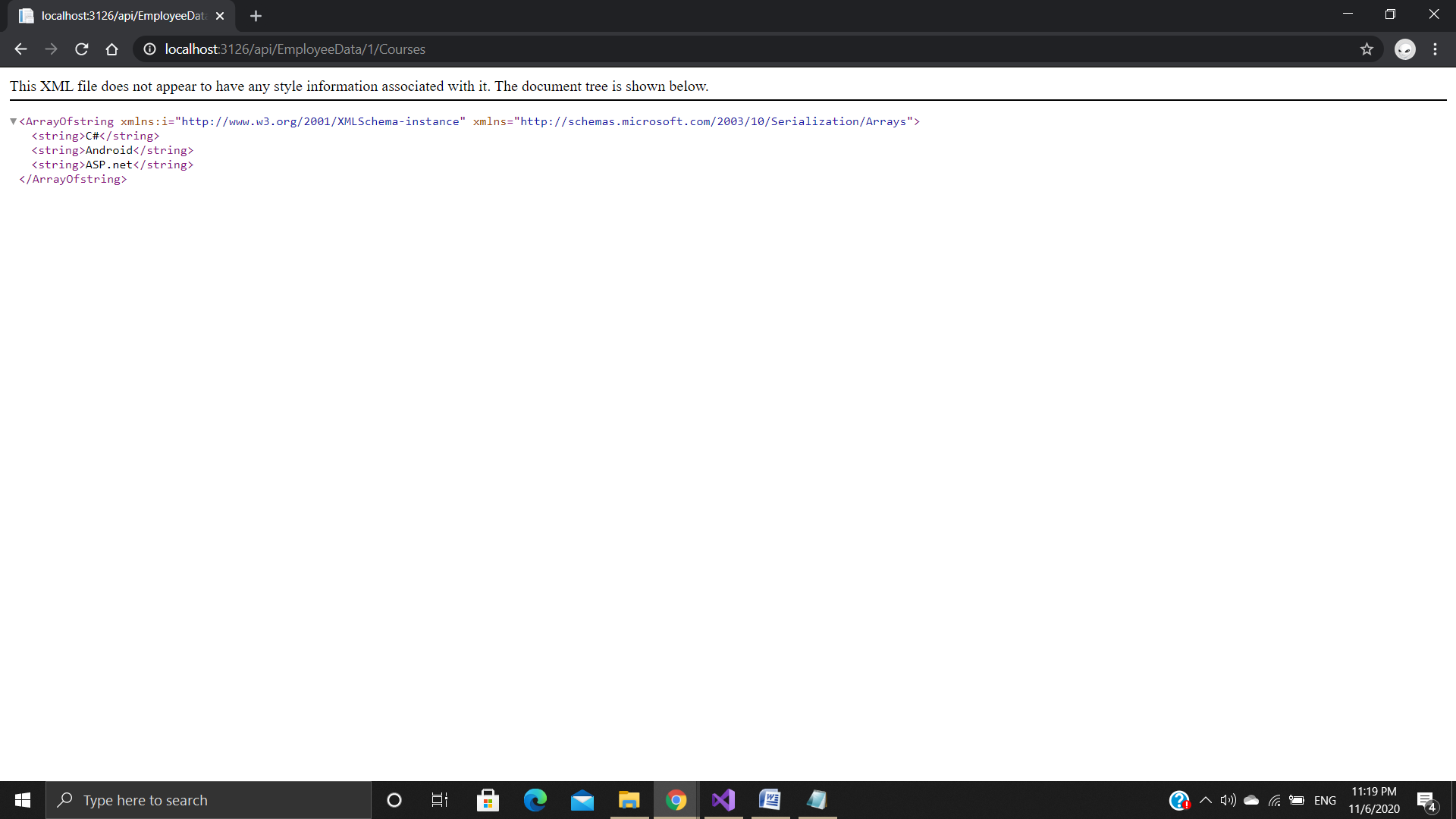
**Output**

****

**By ID**

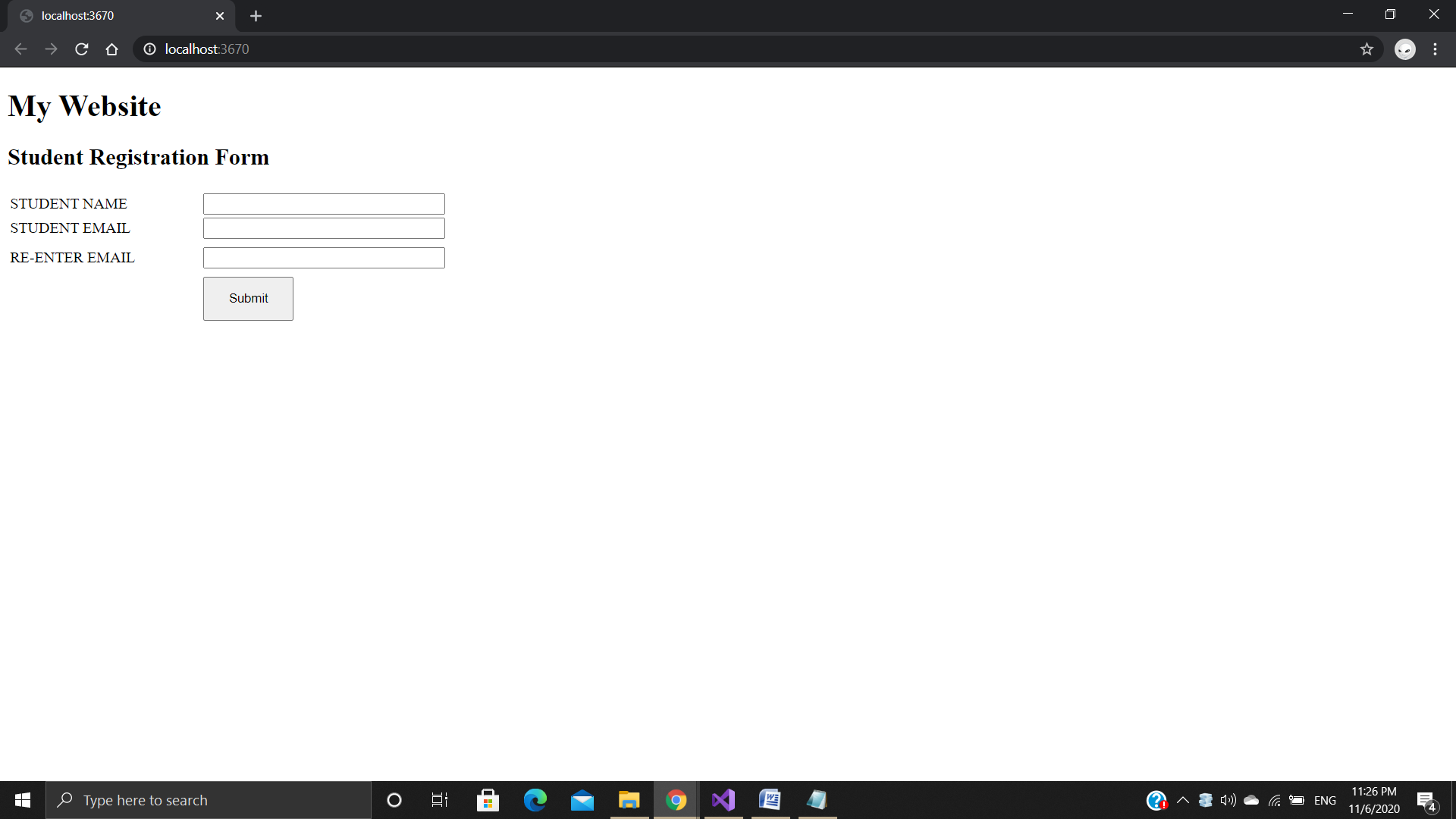
****

**By Using Attribute Routing Route**

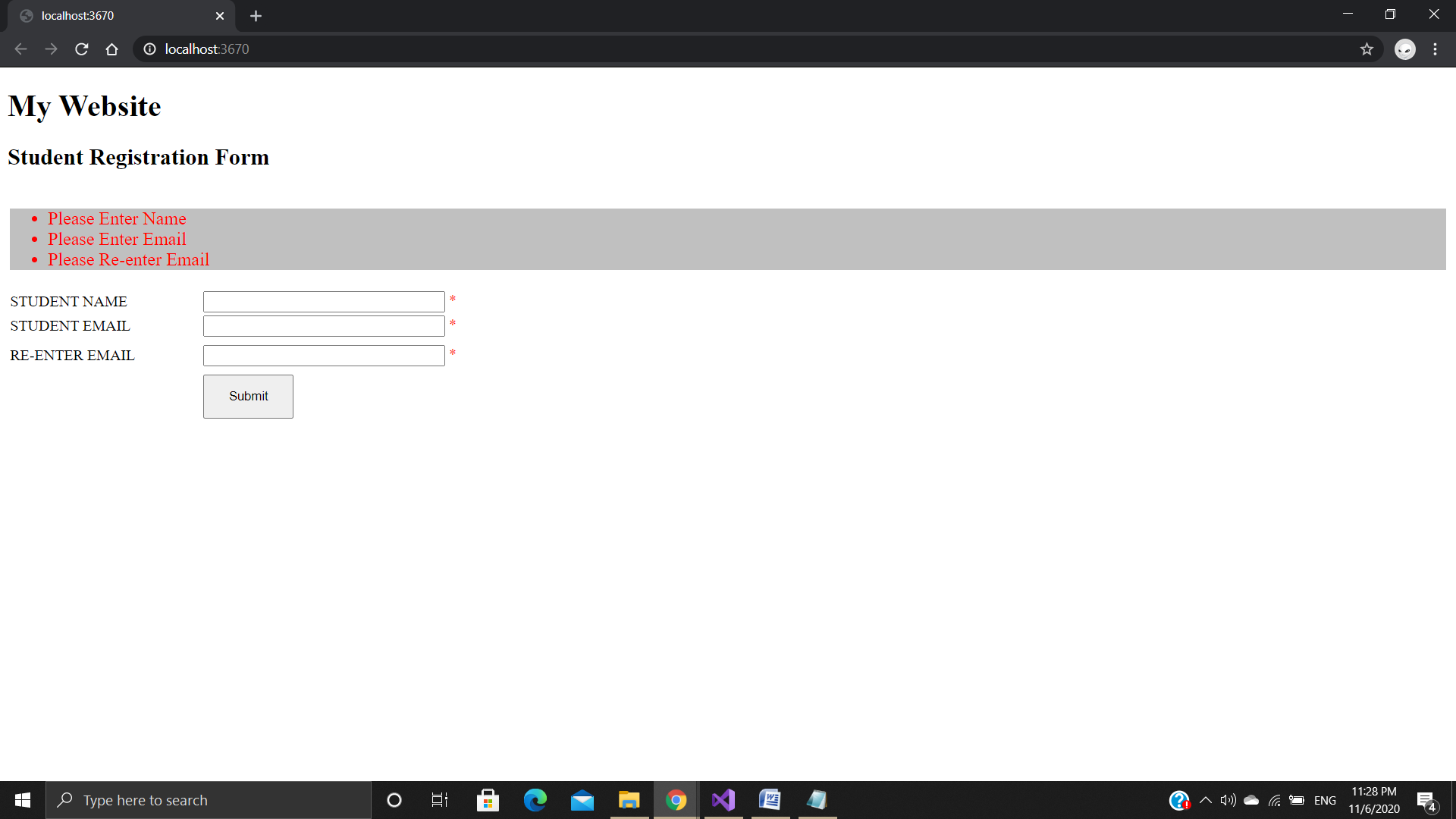
****

1. **Web Form Example**

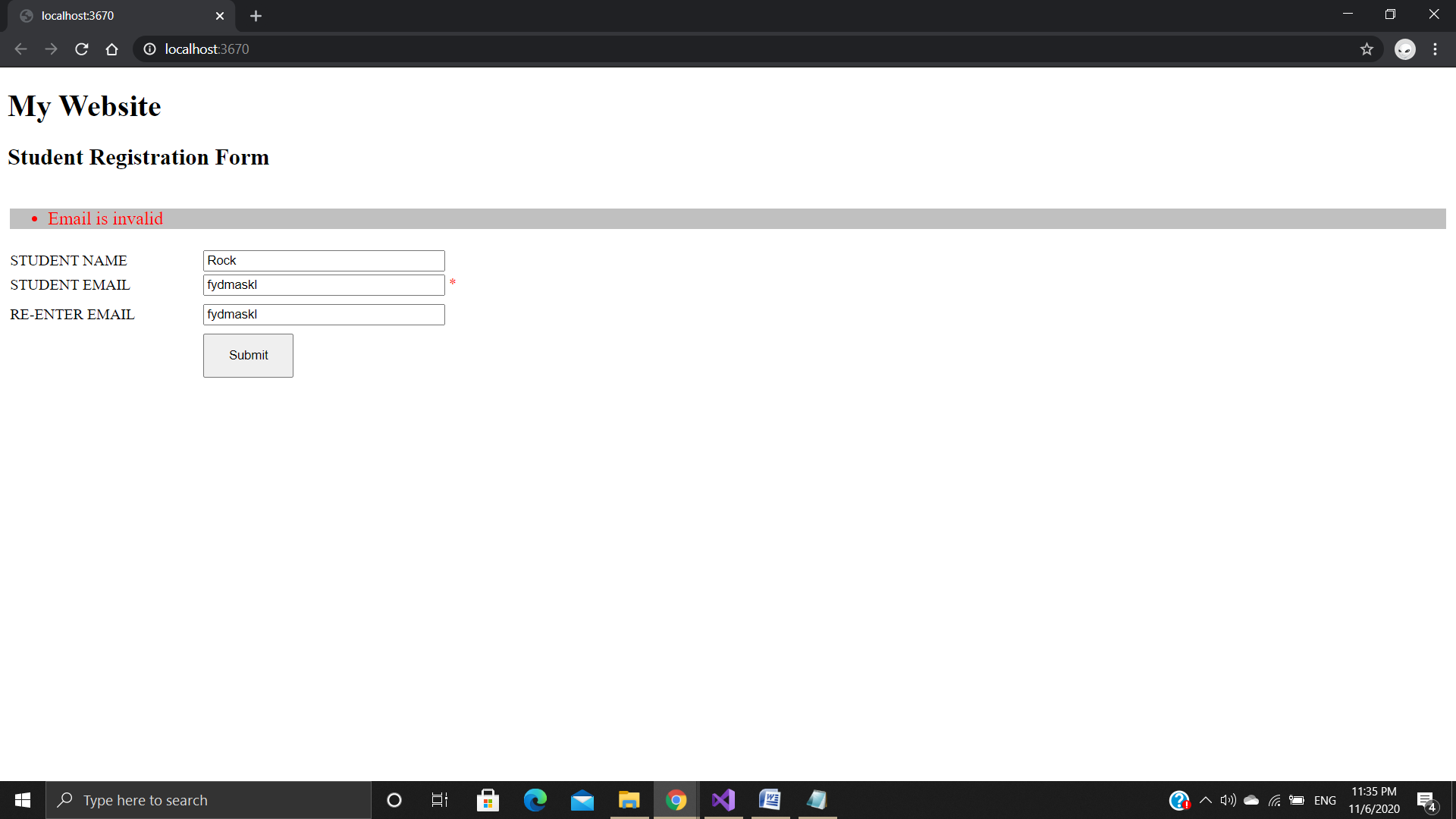
**Output of the Web Form**

****

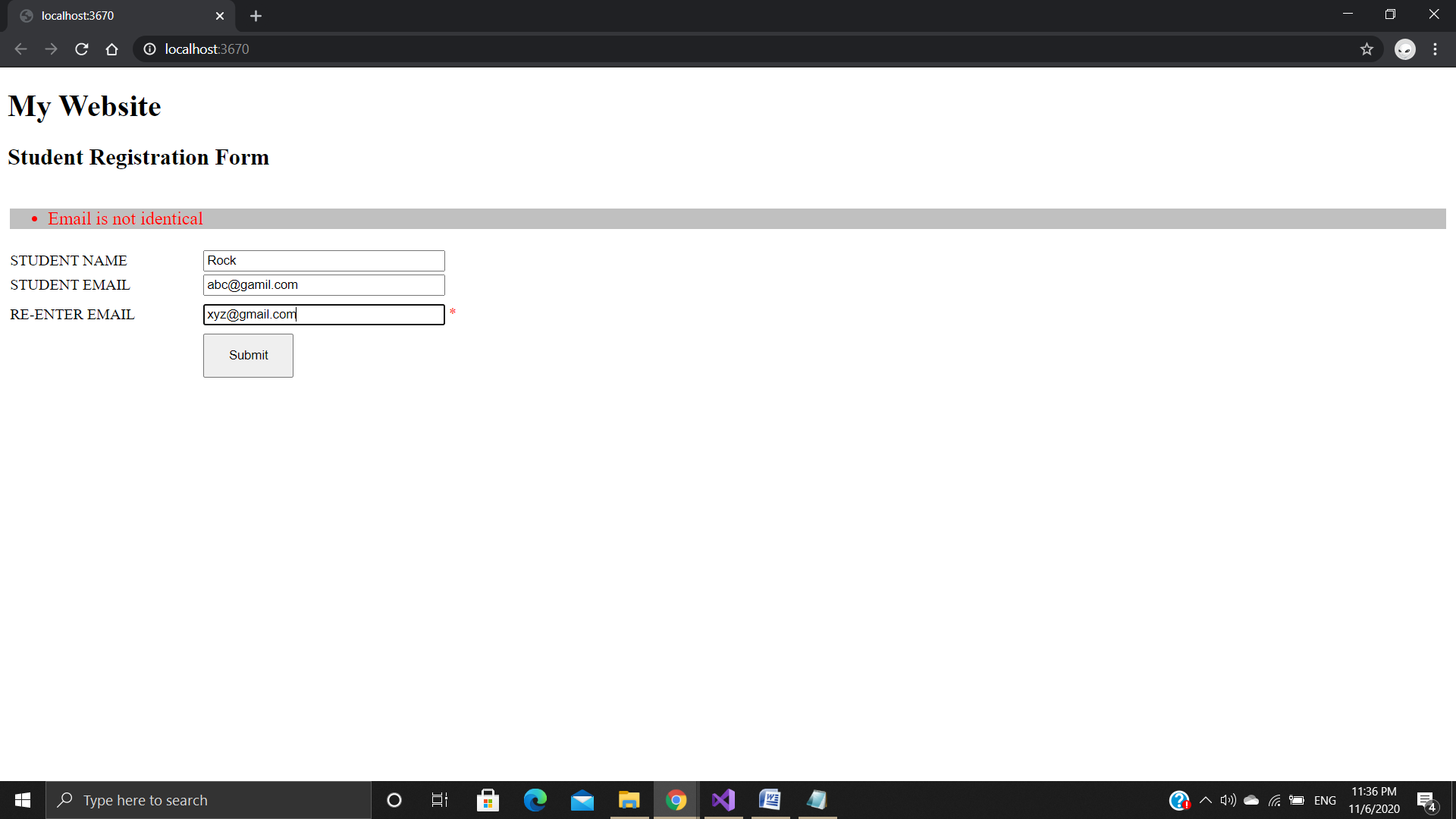
**Requirement validator**

****

**Regular Expression Validator**

****

**Compare Validator**

****

1. **MVC Example**

**Controller (HomeController.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace First\_mvc\_Demo.Controllers

{

public class HomeController : Controller

{

// GET: Home

public ActionResult Index()

{

return View();

}

}

}

**Views (index.cs.html)**

@{

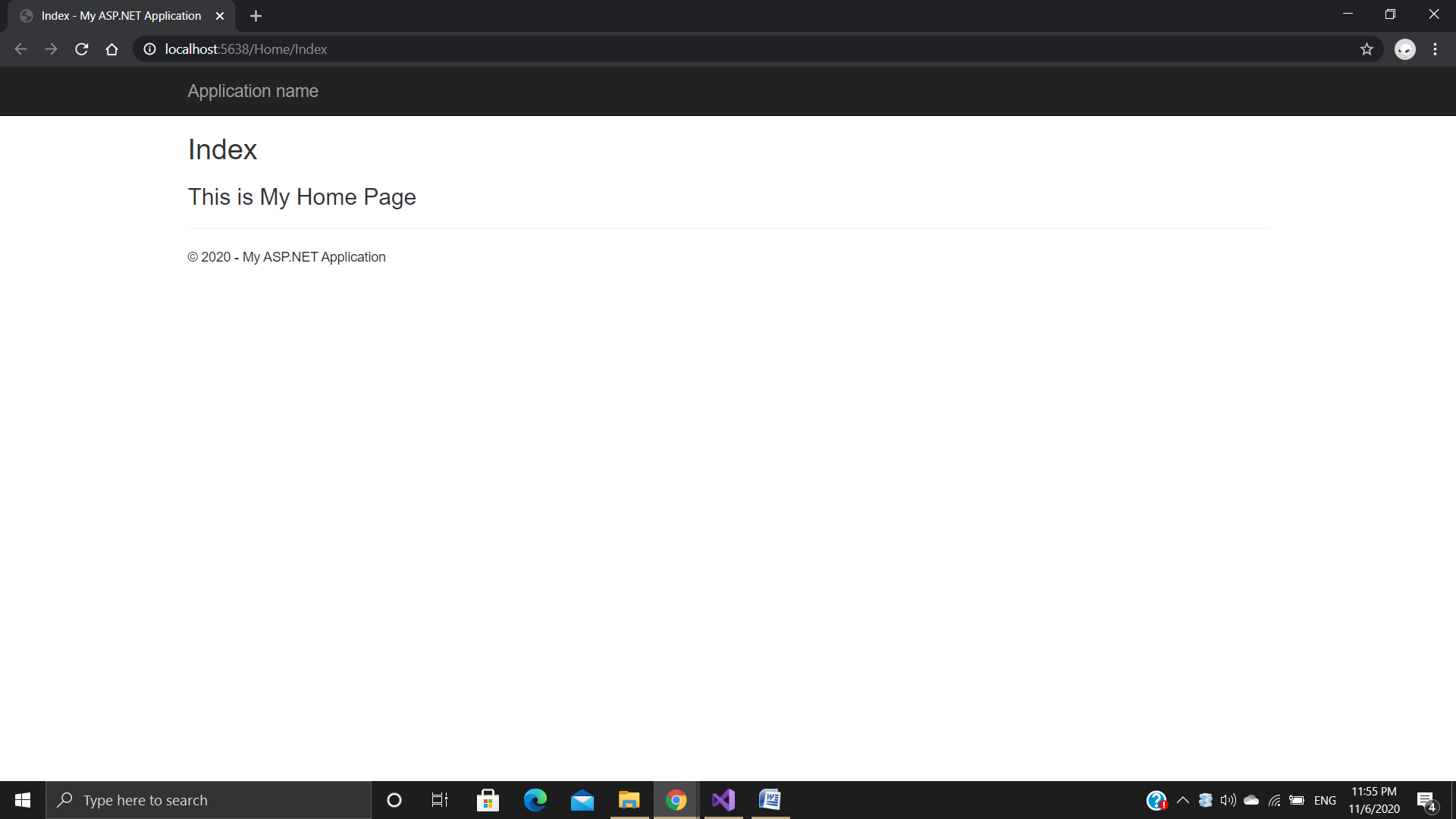
ViewBag.Title = "Index";

}

<h2>Index</h2>

<h3> This is My Home Page </h3>

**Output**

****