21- WCF -1 DOTNET

1. Create a WCF service will implement following methods

Define Operation Contract named SayHello should take name as argument and return the wishes (Good Morning or Good Afternoon or Good Evening) based on time

Define Operation Contract named TodayProgram should take name as argument and return the happy weekend or Enjoy Working day

Define two end point for TCP and HTTP

Use self-hosting to host the service, consume the service in ASP.NET MVC

Design web form with two button

First button should invoke using http request

Second button should invoke using TCP request

Service interface contract

[ServiceContract]

public interface IHelloService

{

/// <summary>

/// SayHello should take name as argument and return the wishes (Good Morning or Good Afternoon or Good Evening) based on time

/// </summary>

/// <param name="name">takes name as argument</param>

/// <returns>return the wishes based on time</returns>

[OperationContract]

string SayHello(string name);

/// <summary>

/// TodayProgram should take name as argument and return the Happy weekend or Enjoy Working day

/// </summary>

/// <param name="name">takes name as argument</param>

/// <returns>return greetings</returns>

[OperationContract]

string TodayProgram(string name);

}

Service Contract implementation

public class HelloService : IHelloService

{

/// <summary>

/// SayHello should take name as argument and return the wishes (Good Morning or Good Afternoon or Good Evening) based on time

/// </summary>

/// <param name="name">takes name as argument</param>

/// <returns>return the wishes based on time</returns>

public string SayHello(string name)

{

DateTime currentTime = DateTime.Now;

string returnResult = string.Empty;

try

{

if (currentTime.Hour >= 0 && currentTime.Hour < 12)

{

returnResult = $" Good Morning, {name}";

}

else if (currentTime.Hour >= 12 && currentTime.Hour < 18)

{

returnResult = $" Good Afternoon, {name}";

}

else if (currentTime.Hour >= 18)

{

returnResult = $" Good Evening, {name}";

}

}

catch (Exception ex)

{

returnResult = $"something went wrong ;( , {name} , Error Message = {ex.Message }";

}

return returnResult;

}

/// <summary>

/// TodayProgram should take name as argument and return the Happy weekend or Enjoy Working day

/// </summary>

/// <param name="name">takes name as argument</param>

/// <returns>return greetings</returns>

public string TodayProgram(string name)

{

DateTime currentTime = DateTime.Now;

string returnResult = string.Empty;

try

{

if (currentTime.DayOfWeek == DayOfWeek.Saturday || currentTime.DayOfWeek == DayOfWeek.Sunday)

{

returnResult = $" Happy weekend, { name}";

}

else

{

returnResult = $" Enjoy Working day, { name}";

}

}

catch (Exception ex)

{

returnResult = $"something went wrong ;( , {name} , Error Message = {ex.Message }";

}

return returnResult;

}

}

Self-hosted service in console app:

using (ServiceHost host = new ServiceHost(typeof(HelloService.HelloService)))

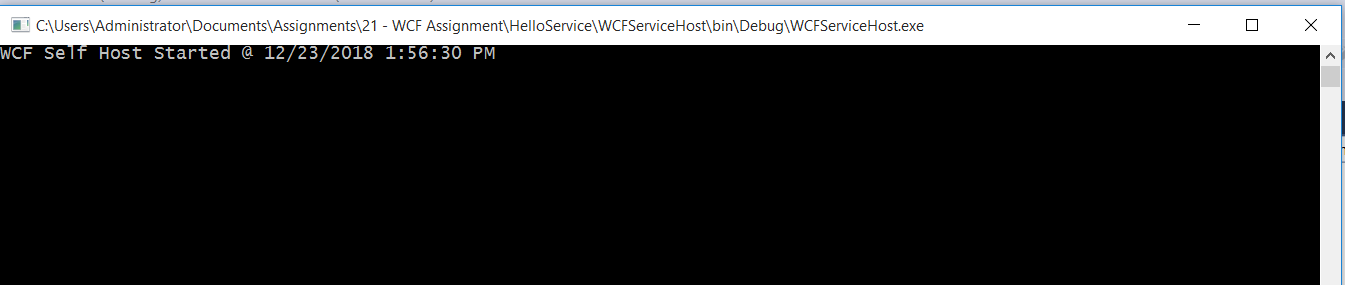
{

host.Open();

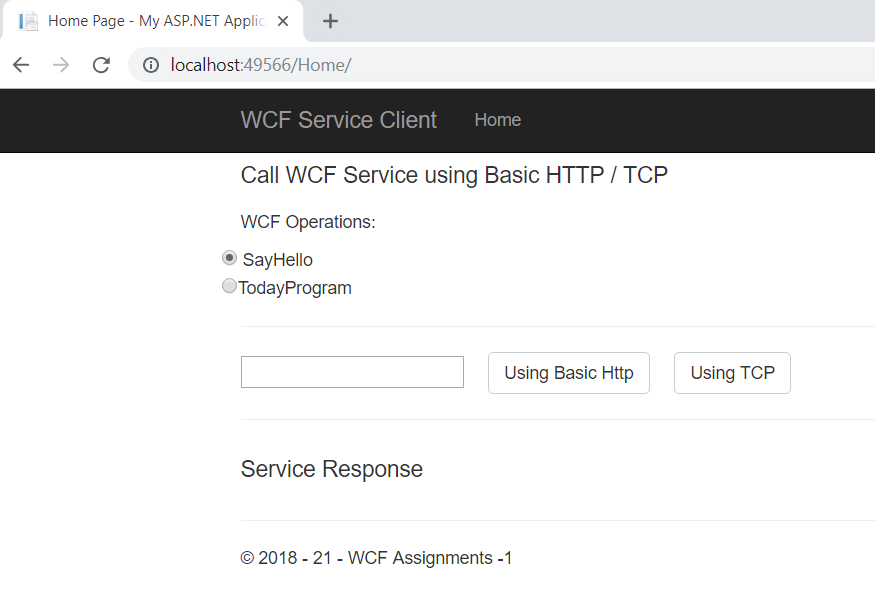
Console.WriteLine($"WCF Self Host Started @ {DateTime.Now}");

Console.ReadLine();

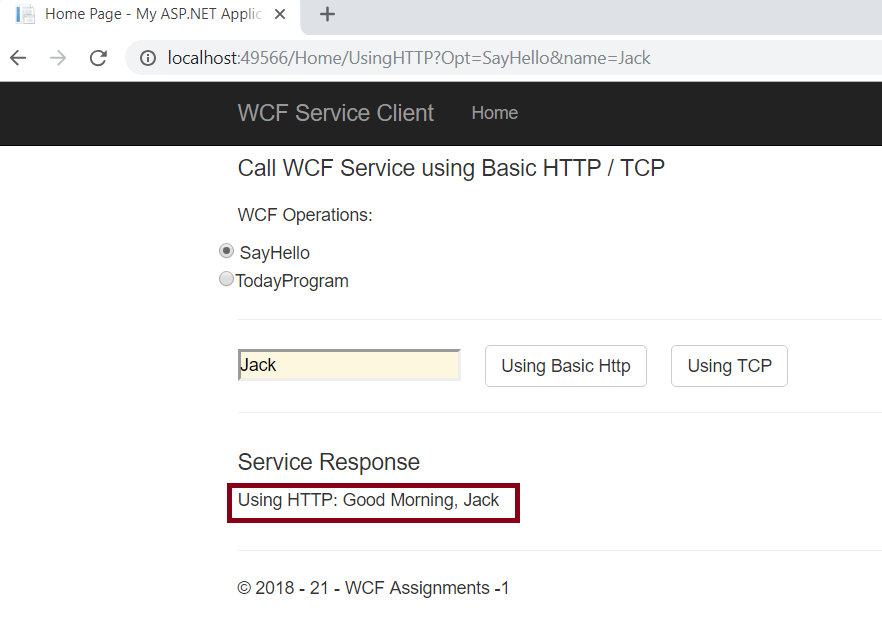
}



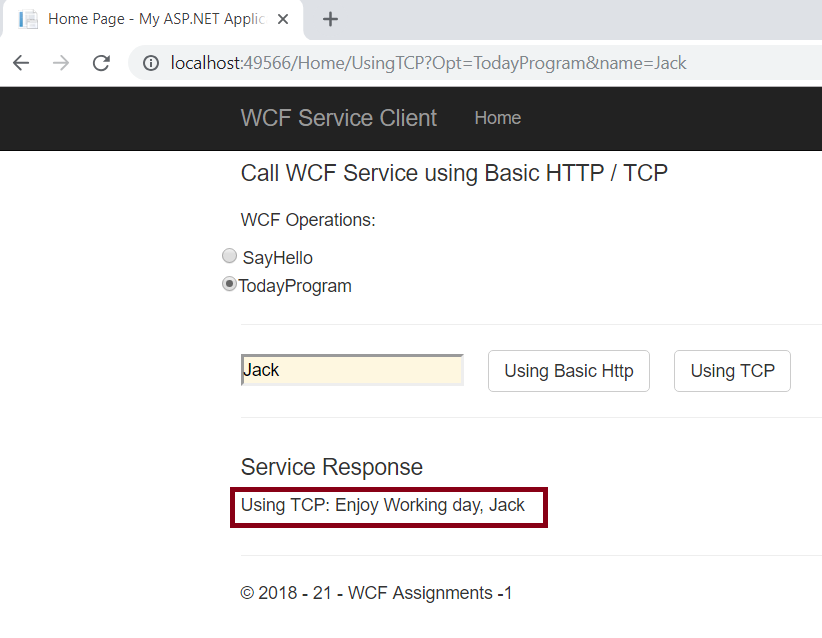
Created MVC client application to consume above created service



Called SayHello operation:



Call TodayProgram:



1. Job Openings

Create a WCF service will implement following methods

Define Operation Contract Opening Jobs return the list of openings.

Define Operation Contract Opening Jobs by Role return the list of openings on specified role given in the parameter.

Service Contract:

[ServiceContract]

public interface IJobOpenings

{

[OperationContract]

List<Jobs> GetAvailbleJobList();

[OperationContract]

List<Jobs> GetJobsByRole(string role);

}

Data Contract:

[DataContract]

public class Jobs

{

[DataMember]

public string Category { get; set; }

[DataMember]

public string Role { get; set; }

[DataMember]

public string Title { get; set; }

[DataMember]

public string Location { get; set; }

}

Service Implementation:

public class JobOpenings : IJobOpenings

{

private List<Jobs> \_availbleJobsList = new List<Jobs>();

public JobOpenings()

{

\_availbleJobsList = new List<Jobs>();

\_availbleJobsList.Add(new Jobs

{

Category ="IT",

Role="DBA",

Title="DB Administrator",

Location="Chicago"

});

\_availbleJobsList.Add(new Jobs

{

Category = "IT",

Role = "Full Stack Developer",

Title = "Associate",

Location = "Plano"

});

\_availbleJobsList.Add(new Jobs

{

Category = "IT",

Role = "Full Stack Developer",

Title = "VP",

Location = "Portland"

});

\_availbleJobsList.Add(new Jobs

{

Category = "Mechanical",

Role = "Operator",

Title = "Trainee",

Location = "Dallas"

});

\_availbleJobsList.Add(new Jobs

{

Category = "Mechanical",

Role = "Designer",

Title = "Team Lead",

Location = "Dallas"

});

}

/// <summary>

/// return available list of jobs

/// </summary>

/// <returns></returns>

public List<Jobs> GetAvailbleJobList()

{

return \_availbleJobsList;

}

/// <summary>

/// Filter List for requested Job role

/// </summary>

/// <param name="role"></param>

/// <returns></returns>

public List<Jobs> GetJobsByRole(string role)

{

var filteredList = \_availbleJobsList

.Where(j => j.Role.ToLower()

.Contains(role.ToLower()))

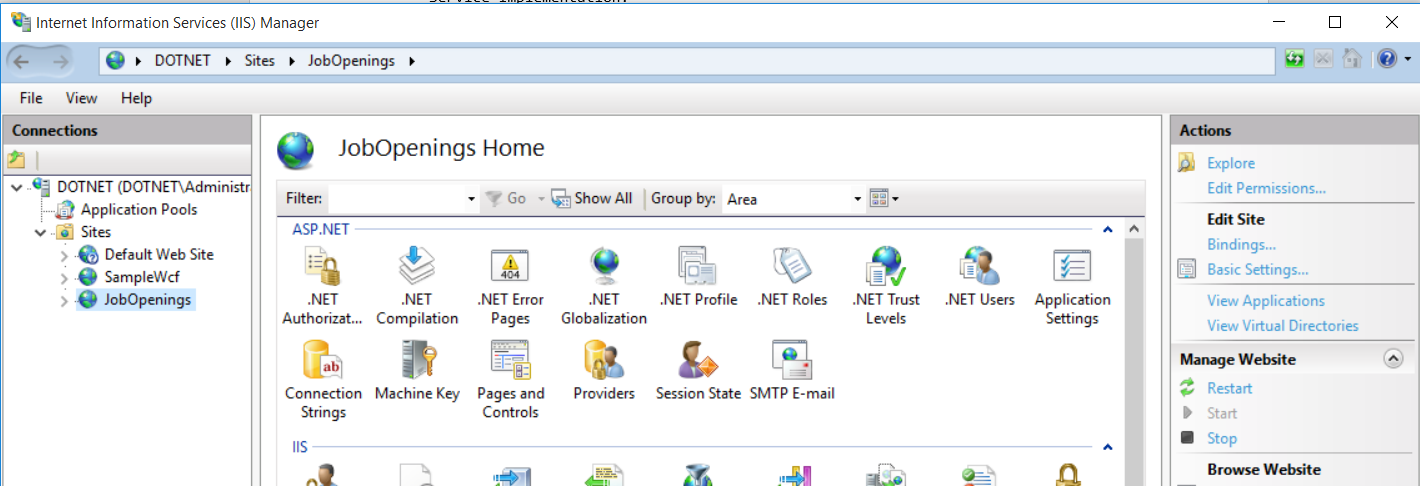
.ToList();

return filteredList;

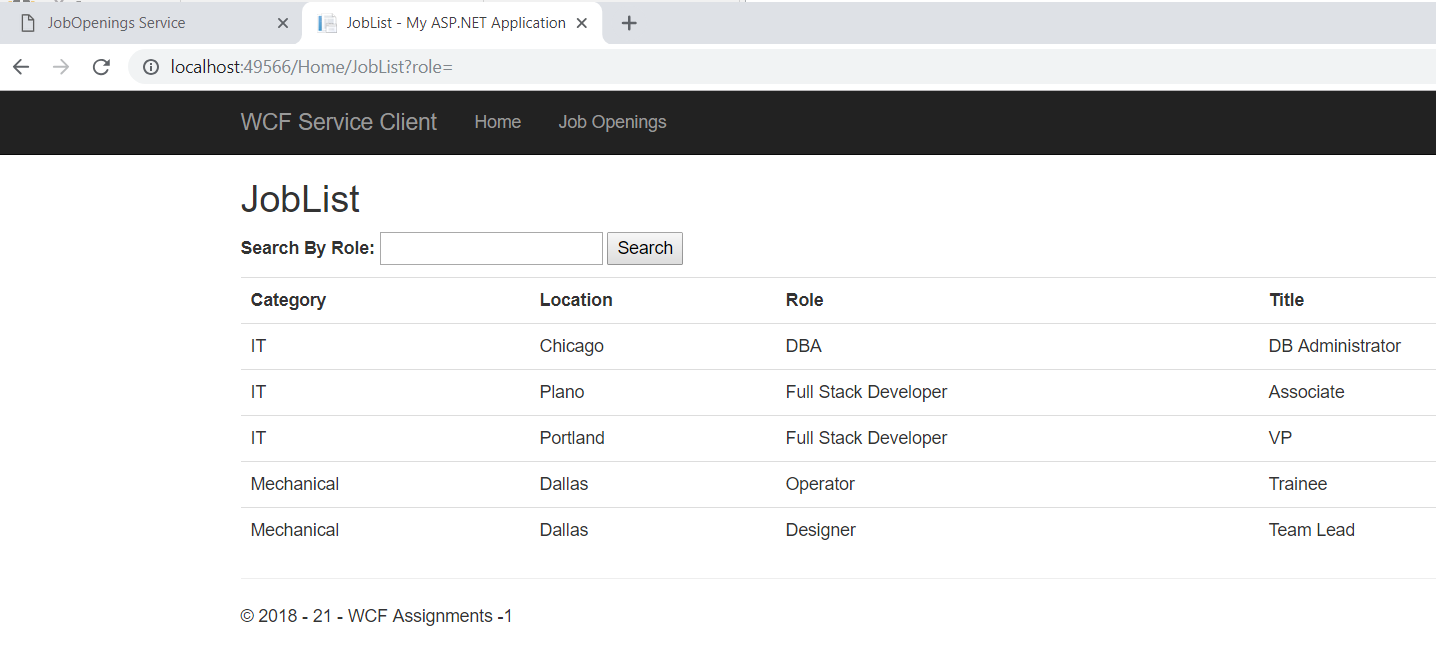
}

}

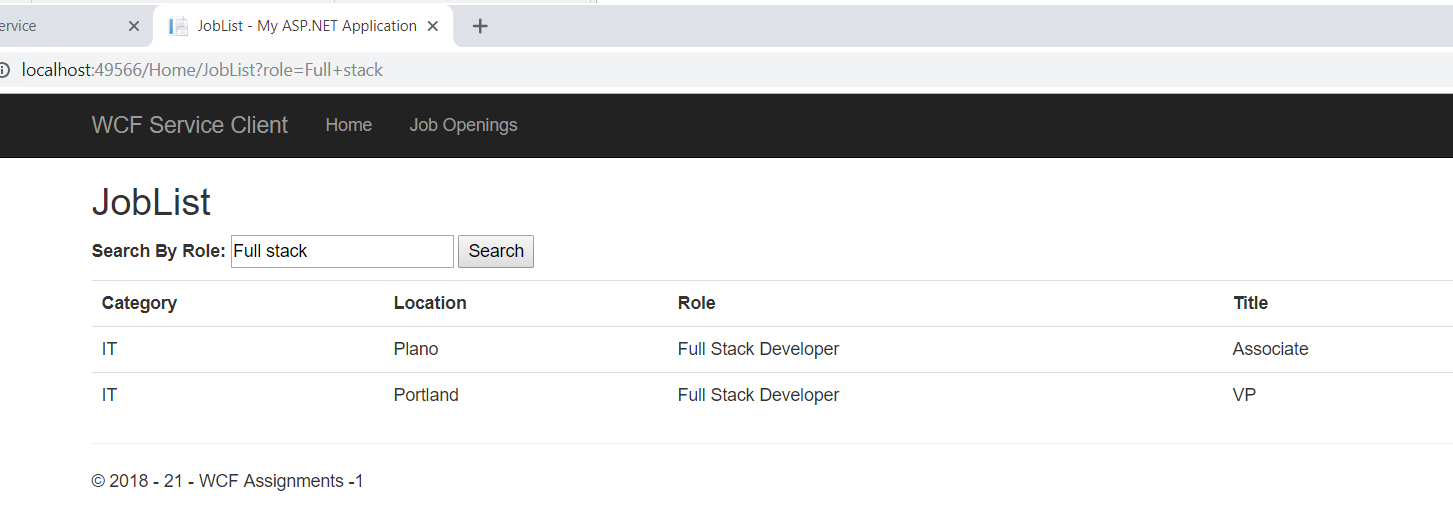
Deployed service on IIS



Added web page to display available list of jobs and also given facility to filter the list by Role



Filter the jobs by Role



1. Create a WCF service to depict simple calculator with addition, subtraction features and deploy and consume the service.

Service Contract

[ServiceContract]

public interface ICalculator

{

[OperationContract]

int Addition(int firstNo, int SecondNo);

[OperationContract]

int Substraction(int firstNo, int SecondNo);

[OperationContract]

int Multiplication(int firstNo, int SecondNo);

[OperationContract]

decimal Division(int firstNo, int SecondNo);

}

Service implementation

public class Calculator : ICalculator

{

public int Addition(int firstNo, int SecondNo)

{

return firstNo + SecondNo;

}

public int Substraction(int firstNo, int SecondNo)

{

return firstNo - SecondNo;

}

public int Multiplication(int firstNo, int SecondNo)

{

return firstNo \* SecondNo;

}

public decimal Division(int firstNo, int SecondNo)

{

if (SecondNo == 0 || firstNo == 0)

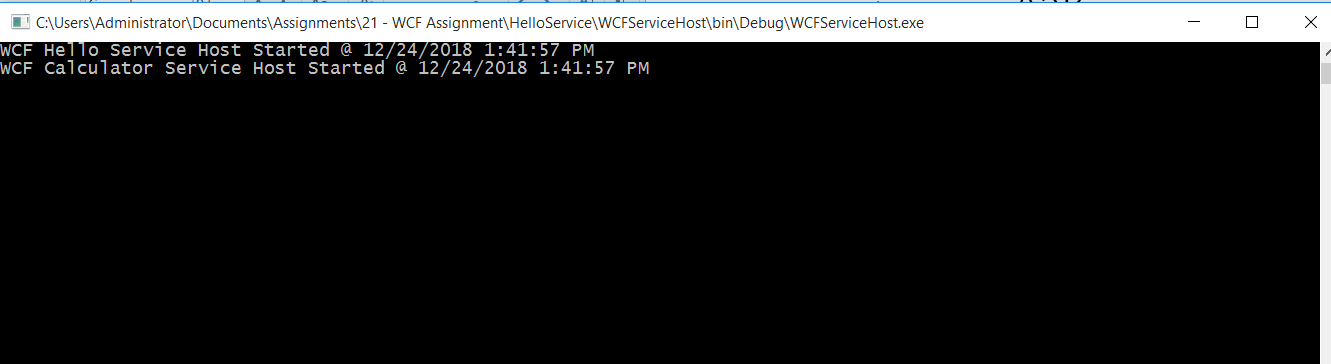
return 0;

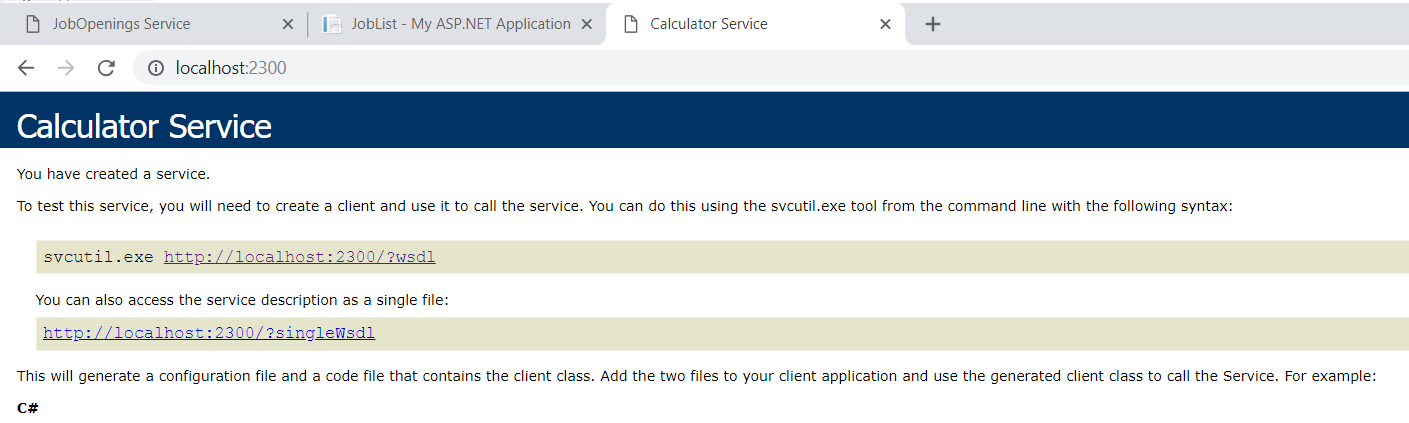
return firstNo / SecondNo;

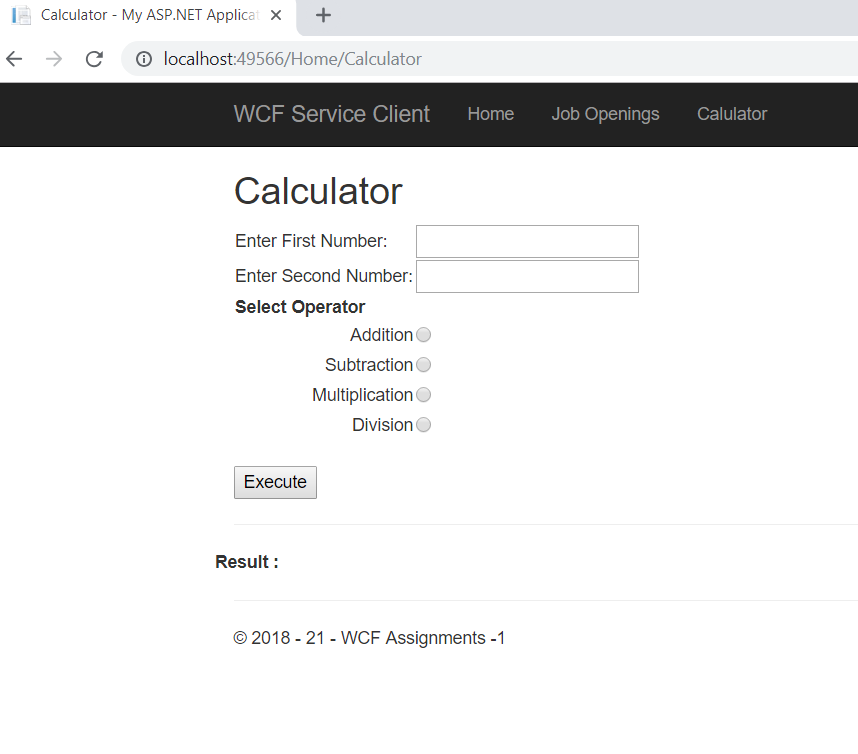
}

}

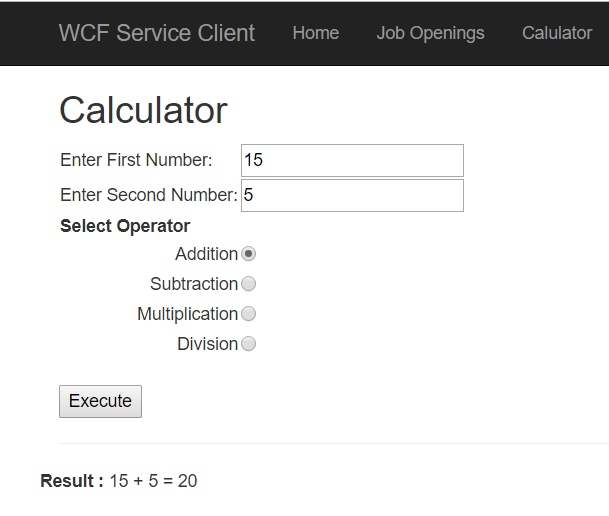
Self-hosted







Addition:



Multiplication:

