Web API

Christen Zarif

outline

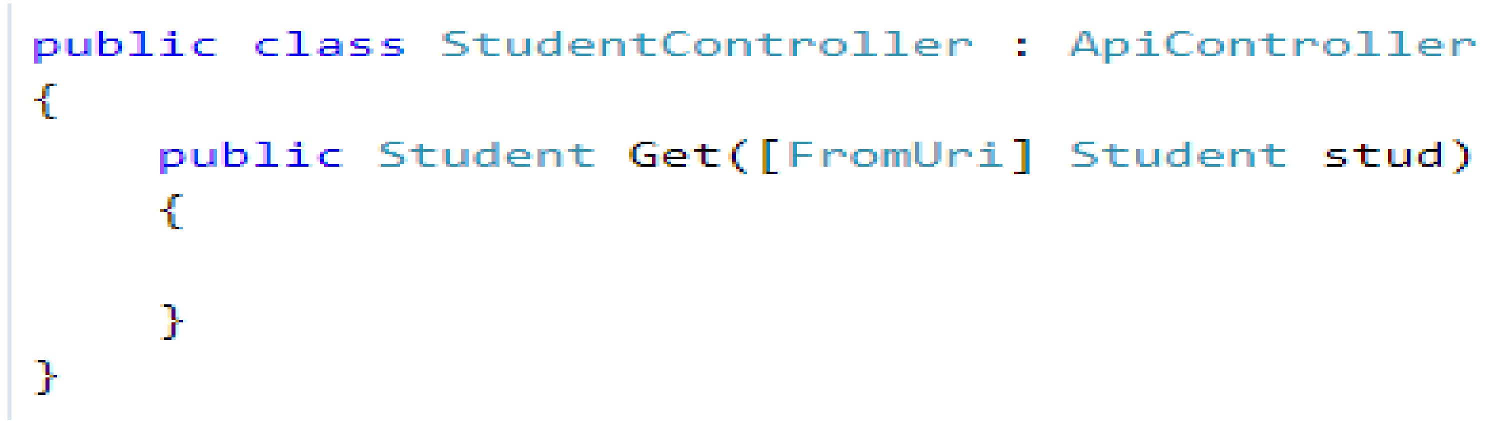
- Restful Constraint
- Web Api Parameter Binding
- DTO & IgnoreJson
- MIME Types
- Consumer(JS & .Net Client)
- CORS

Web API Parameter Binding(cont.)

- **[FromUri] and [FromBody]**
- You have seen that by default Web API gets the value of a **primitive** parameter from the **query string** and **complex** type parameter from the **request body**.
- But, what if we want to change this default behaviour?
- Use **[FromUri]** attribute to force Web API to get the value of complex type from the **query string** and
- **[FromBody]** attribute to get the value of primitive type from the **request body**, opposite to the default rules.

Web API Parameter Binding(cont.)

- **[FromUri] and [FromBody]**
- For example, consider the following Get method.



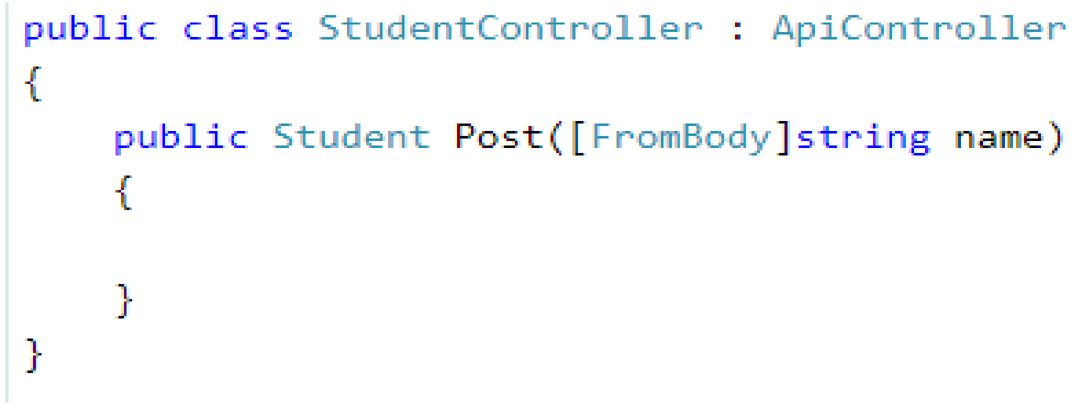
```
public class StudentController : ApiController
{
    public Student Get([FromUri] Student stud)
    {

    }
}
```

Web API Parameter Binding(cont.)

- **[FromUri] and [FromBody]**
- In the above example, **Get** method includes complex type parameter with [**FromUri**] attribute. So, Web API will try to get the value of Student type parameter from the query string.
- For example, if an HTTP GET request **http://localhost:xxxx/api/student?id=1&name=steve** then Web API will create Student object and set its id and name property values to the value of id and name query string

Web API Parameter Binding(cont.)

- **[FromUri] and [FromBody]**



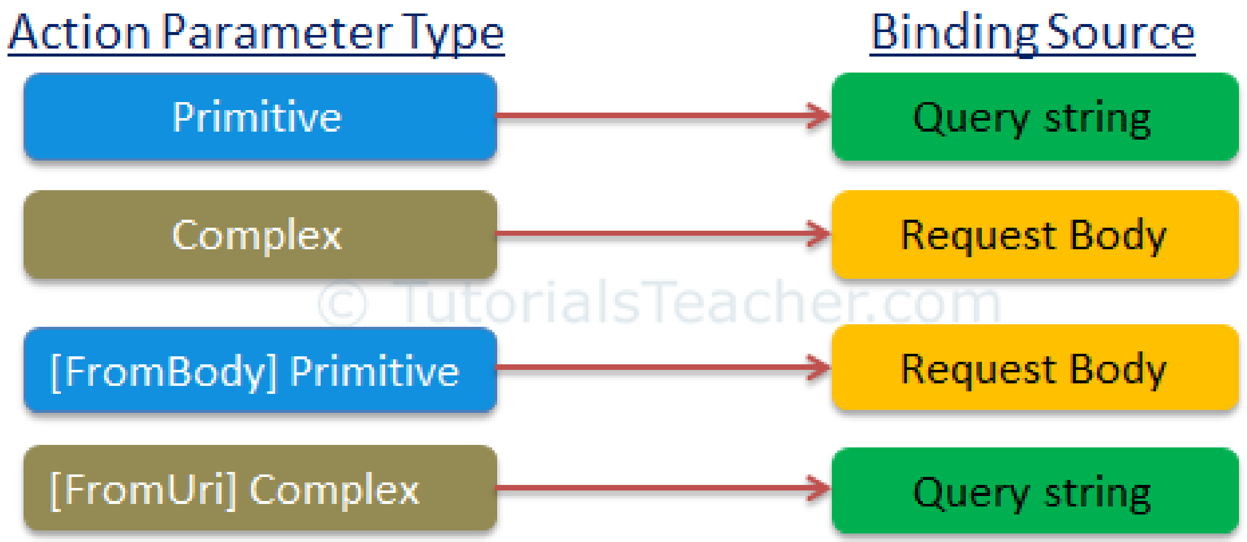
```
public class StudentController : ApiController
{
    public Student Post([FromBody]string name)
    {

    }
}
```

- Here you must pass the object in the **body request.**

Web API Parameter Binding(cont.)

- **[FromUri] and [FromBody]**

The following figure summarizes parameter binding rules.



Web API Parameter Bindings

DATA TRANSFER OBJECT (DTO)

DTO

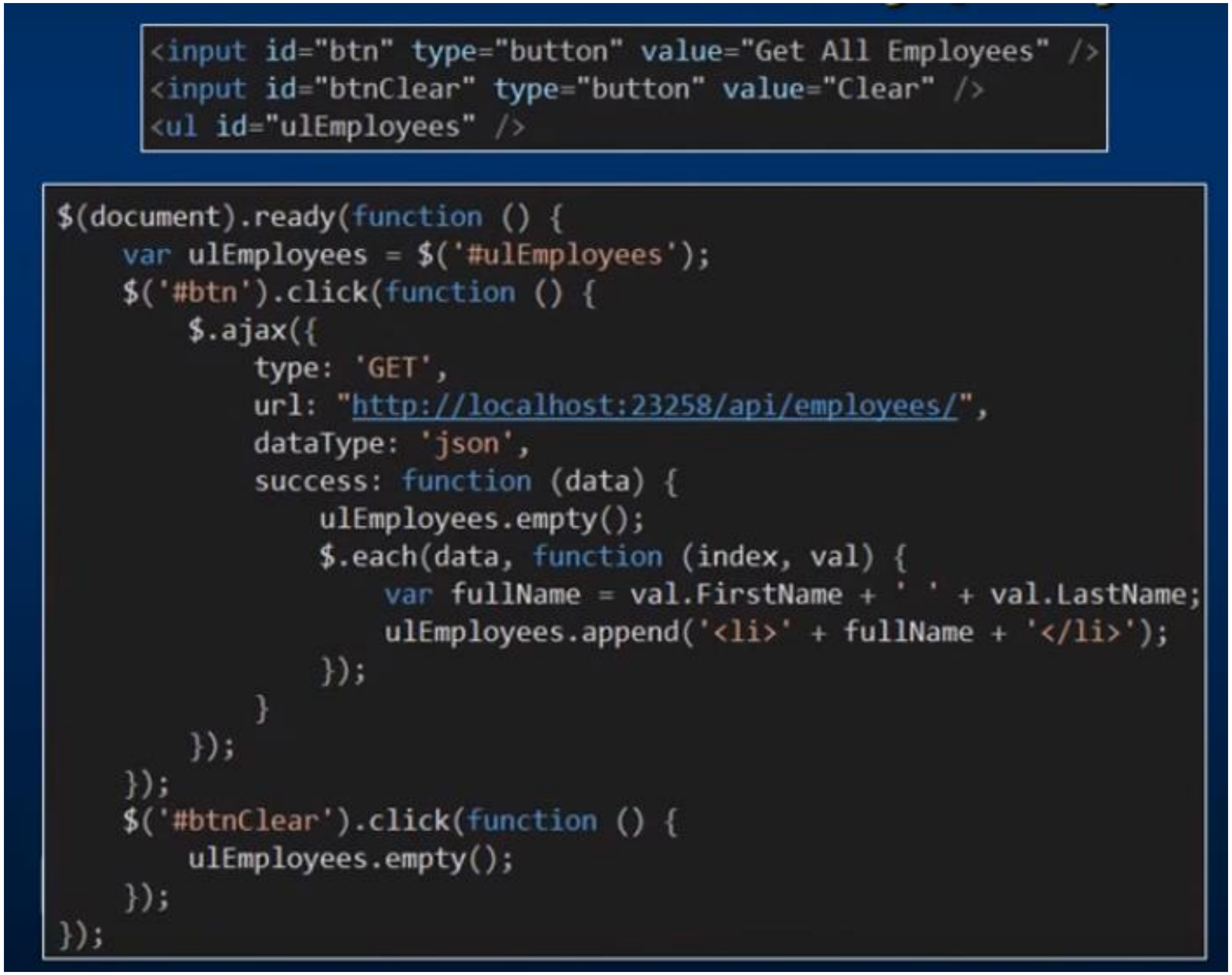
- DTOs being used in the service layer to return data back to the presentation layer.
- The biggest advantage of using DTOs is decoupling clients from your internal data structures

Why DTO

- Remove circular references (see previous section).
- Hide particular properties that clients are not supposed to view.
- Omit some properties in order to reduce payload size.
- Flatten object graphs that contain nested objects, to make them more convenient for clients.
- Avoid "over-posting" vulnerabilities.
- Decouple your service layer from your database layer.

WEB API CONSUMERS

Call Web API from JQurey



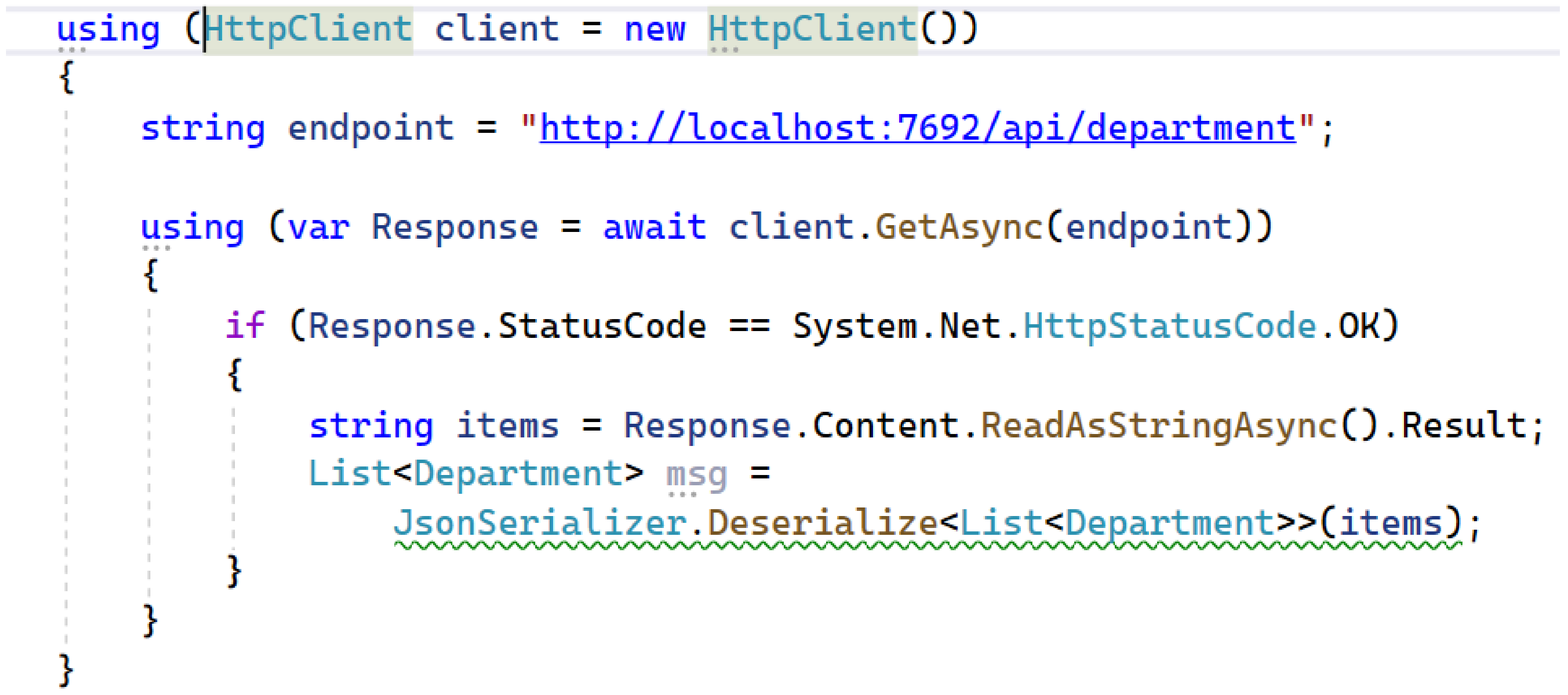
```
<input id="btn" type="button" value="Get All Employees" />
<input id="btnClear" type="button" value="Clear" />
<ul id="ulEmployees" />
```

```
$(document).ready(function () {
    var ulEmployees = $('#ulEmployees');
    $('#btn').click(function () {
        $.ajax({
            type: 'GET',
            url: "http://localhost:23258/api/employees/",
            dataType: 'json',
            success: function (data) {
                ulEmployees.empty();
                $.each(data, function (index, val) {
                    var fullName = val.FirstName + ' ' + val.LastName;
                    ulEmployees.append('<li>' + fullName + '</li>');
                });
            }
        });
    });
    $('#btnClear').click(function () {
        ulEmployees.empty();
    });
});
```

JQuery

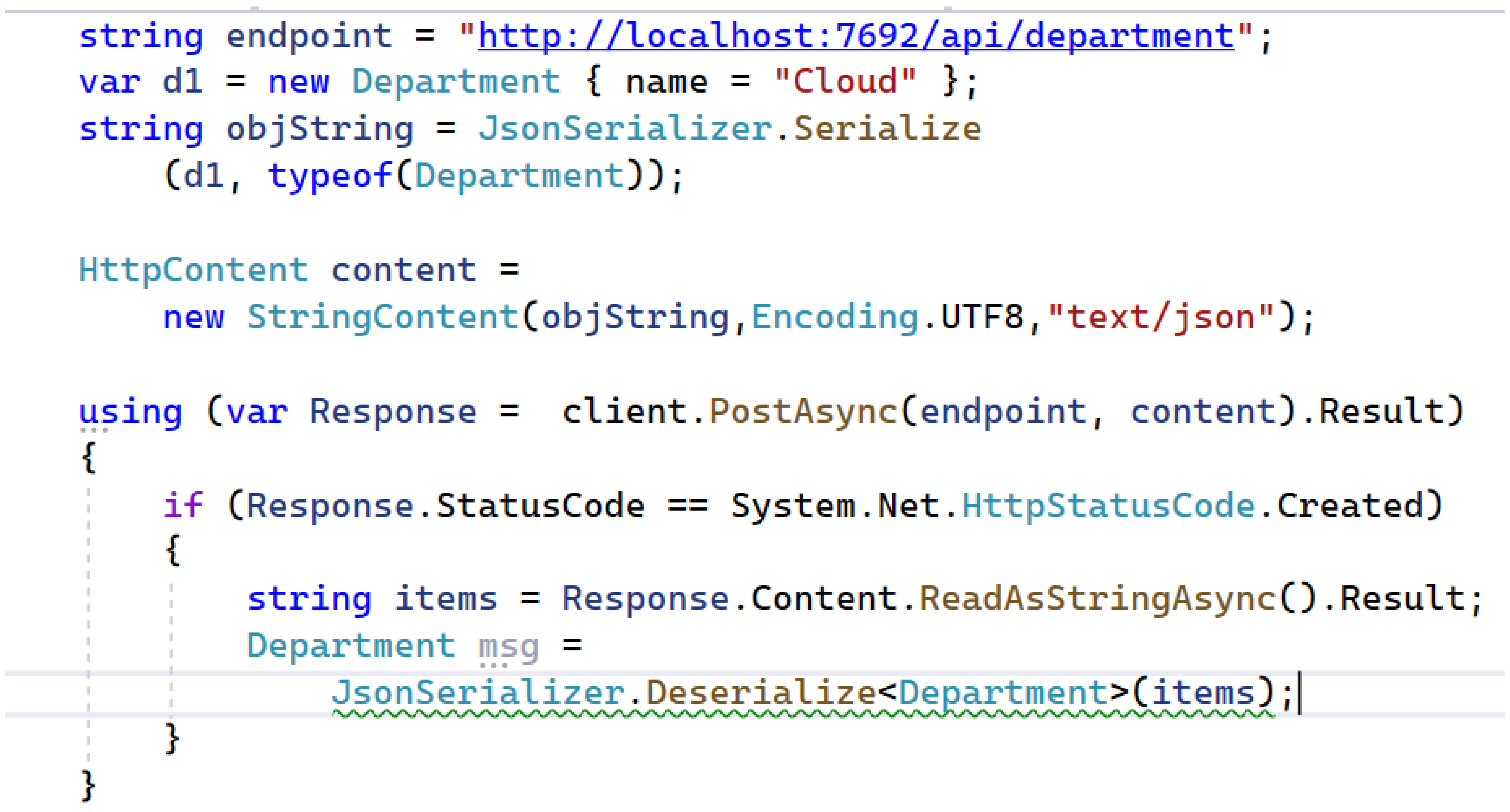
- Calling ASP NET Web API service in a cross domain using jQuery ajax

Call GetMethod



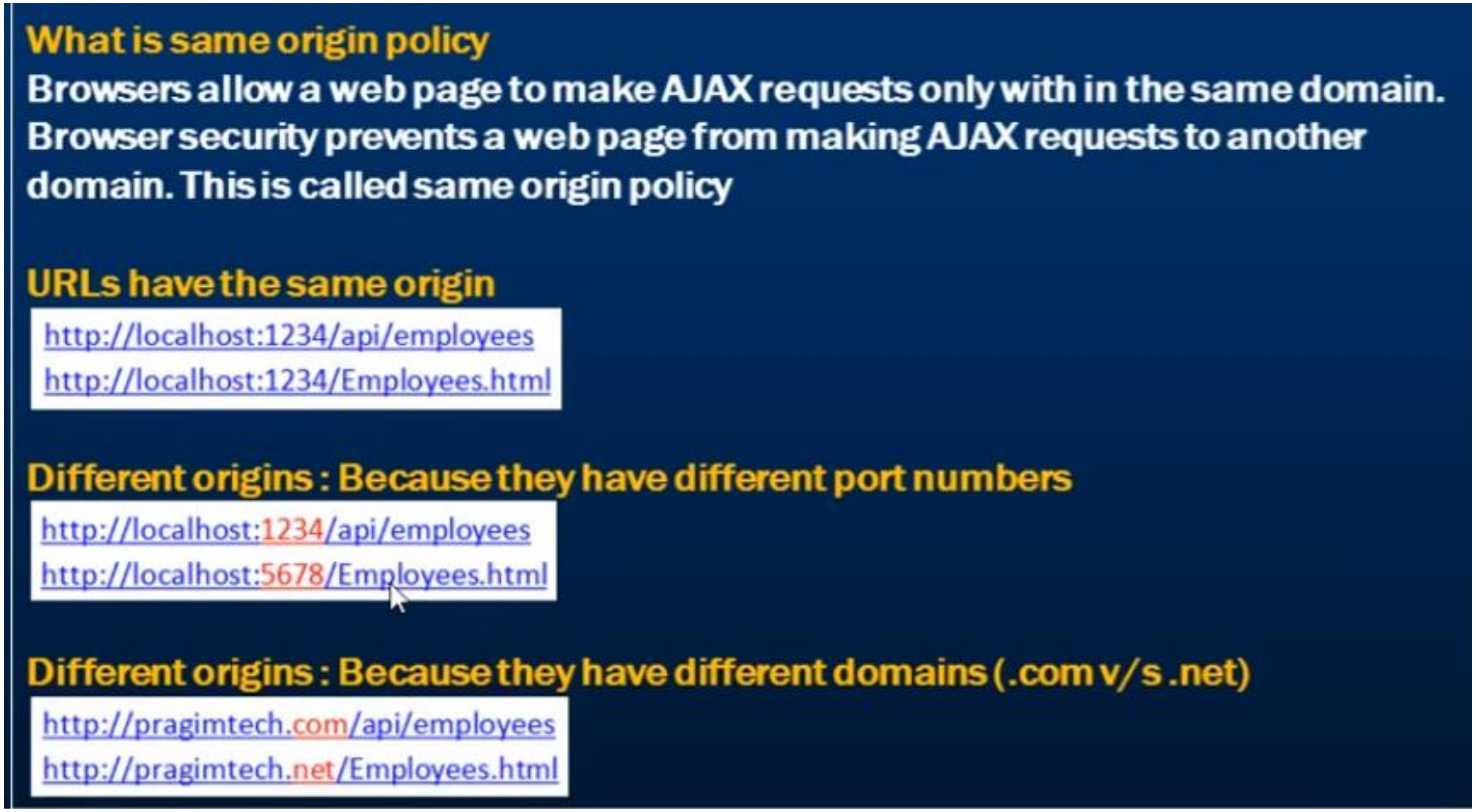
```
using (HttpClient client = new HttpClient())
{
    string endpoint = "http://localhost:7692/api/department";

    using (var Response = await client.GetAsync(endpoint))
    {
        if (Response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            string items = Response.Content.ReadAsStringAsync().Result;
            List<Department> msg =
                JsonSerializer.Deserialize<List<Department>>(items);
        }
    }
}
```

Call Post Method



```
string endpoint = "http://localhost:7692/api/department";
var d1 = new Department { name = "Cloud" };
string objString = JsonSerializer.Serialize
    (d1, typeof(Department));

HttpContent content =
    new StringContent(objString,Encoding.UTF8,"text/json");

using (var Response =  client.PostAsync(endpoint, content).Result)
{
    if (Response.StatusCode == System.Net.HttpStatusCode.Created)
    {
        string items = Response.Content.ReadAsStringAsync().Result;
        Department msg =
            JsonSerializer.Deserialize<Department>(items);
    }
}
```

CORS- Cross Domain Ajax



What is same origin policy

Browsers allow a web page to make AJAX requests only with in the same domain. Browser security prevents a web page from making AJAX requests to another domain. This is called same origin policy

URLs have the same origin

http://localhost:1234/api/employees
http://localhost:1234/Employees.html

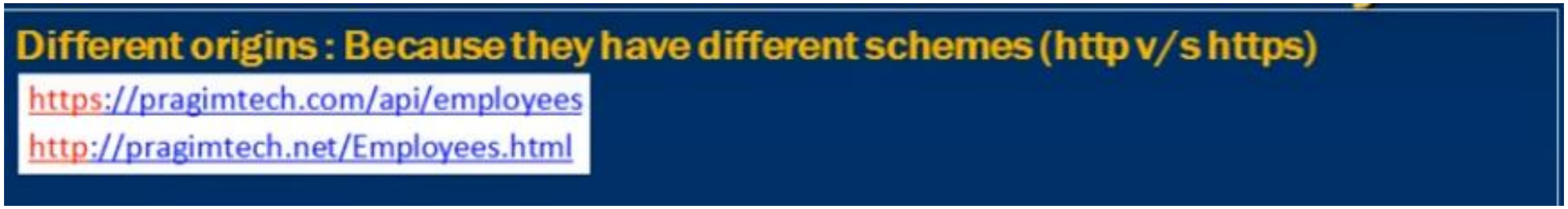
Different origins : Because they have different port numbers

http://localhost:1234/api/employees
http://localhost:5678/Employees.html

Different origins : Because they have different domains (.com v/s .net)

http://pragimtech.com/api/employees
http://pragimtech.net/Employees.html

To Allow CORS



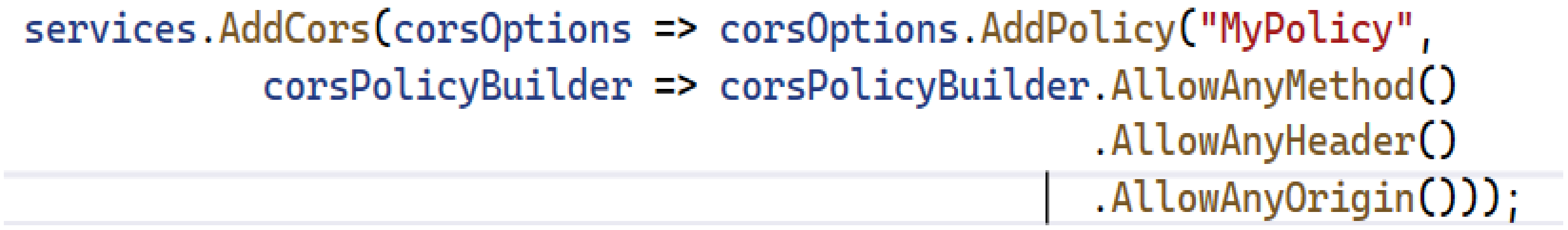
Different origins : Because they have different schemes (http v/s https)

https://pragimtech.com/api/employees

http://pragimtech.net/Employees.html

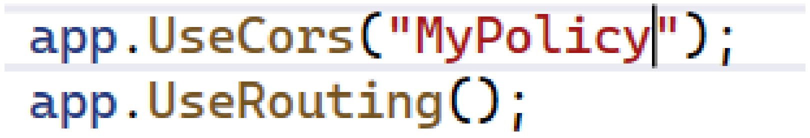
In startup class

- ConfigureService add this method



```
services.AddCors(corsOptions => corsOptions.AddPolicy("MyPolicy",
            corsPolicyBuilder => corsPolicyBuilder.AllowAnyMethod()
                                                  .AllowAnyHeader()
                                                  .AllowAnyOrigin()));
```

- Configure Method



```
app.UseCors("MyPolicy");
app.UseRouting();
```