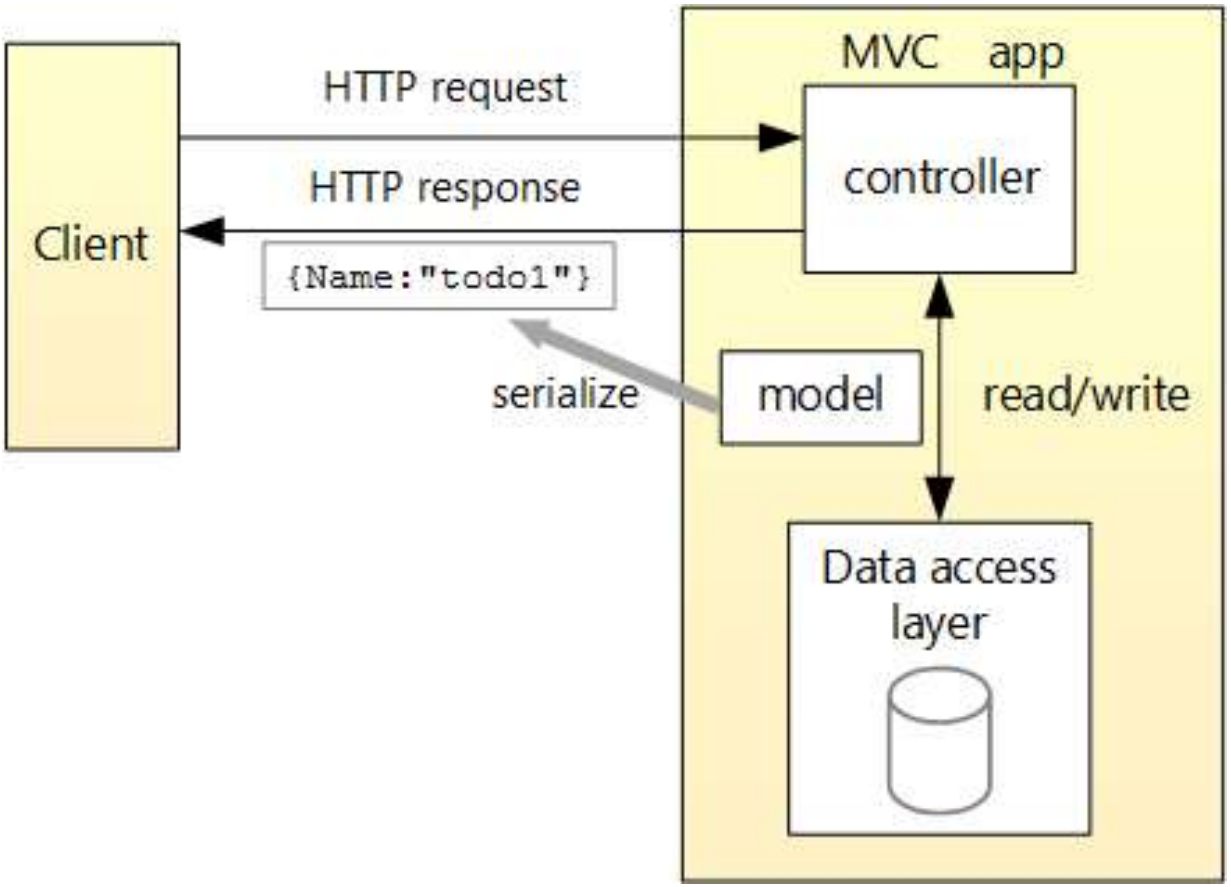
ASP.NET Core

Web API

Mohamed ELshafei


MVC app
HTTP request
HTTP response
Client
controller
{Name:"todo1"}
serialize
model
read/write
Data access
layer

Attributes

The Microsoft.AspNetCore.Mvc namespace provides attributes that can be used to configure the behavior of web API controllers and action methods.

[Route]: Specifies URL pattern for a controller or action.

[Bind] :Specifies prefix and properties to include for model binding.

[HttpGet]: Identifies an action that supports the HTTP GET action verb.

[Consumes] :Specifies data types that an action accepts.

[Produces]:Specifies data types that an action returns.

Binding source parameter inference

A binding source attribute defines the location at which an action parameter's value is found. The following binding source attributes exist:

[FromBody]: Request body

[FromForm]: Form data in the request body

[FromHeader]: Request header

[FromQuery]: Request query string parameter

[FromRoute]: Route data from the current request

[ExcludeFromDescription]： not be included in the generated API metadata.

Attribute routing with Http[Verb] attributes.

- Attribute routing can also make use of the *Http[Verb]* attributes such as *HttpPostAttribute*. All of these attributes can accept a route template.

```
[HttpGet("/products")]
public IActionResult ListProducts()
{
    // ...
}

[HttpPost("/products")]
public IActionResult CreateProduct(...)
{
    // ...
}
```

Token replacement in route templates

- For convenience, attribute routes support token replacement by enclosing a token in square-braces ([,]).
- No route Prefix.

```
[Route("[controller]/[action]")]
public class ProductsController : Controller
{
    [HttpGet] // Matches '/Products/List'
    public IActionResult List() {
        // ...
    }

    [HttpGet("{id}")] // Matches '/Products/Edit/{id}'
    public IActionResult Edit(int id) {
        // ...
    }
}
```

Multiple Routes

```
[Route("[controller]")]
public class ProductsController : Controller
{
   [Route("")]     // Matches 'Products'
   [Route("Index")] // Matches 'Products/Index'
   public IActionResult Index()
}
```

```
[Route("Store")]
[Route("[controller]")]
public class ProductsController : Controller
{
   [HttpPost("Buy")]     // Matches 'Products/Buy' and 'Store/Buy'
   [HttpPost("Checkout")] // Matches 'Products/Checkout' and 'Store/Checkout'
   public IActionResult Buy()
}
```

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
   [HttpPut("Buy")]      // Matches PUT 'api/Products/Buy'
   [HttpPost("Checkout")] // Matches POST 'api/Products/Checkout'
   public IActionResult Buy()
}
```

Specifying attribute route optional parameters, default values, and constraints

- {id:int} integar constraint.
- {id:int?} optional Parameter
- {id:int=3} Default Value

```
[HttpPost("product/{id:int}")]
public IActionResult ShowProduct(int id)
{
    // ...
}
```

Ignore individual properties

- To ignore individual properties, use the [JsonIgnore] attribute.
- You can specify conditional exclusion by setting the [JsonIgnore] attribute's Condition property. The JsonIgnoreCondition enum provides the following options:

✓ **Always** - The property is always ignored. If no Condition is specified, this option is assumed.

✓ **WhenWritingDefault** - The property is ignored on serialization if it's a reference type null, a nullable value type null, or a value type default.

✓ **WhenWritingNull** - The property is ignored on serialization if it's a reference type null, or a nullable value type null

```
public class Forecast
   {
      [JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingDefault)]
      public DateTime Date { get; set; }

      [JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]
      public string? Summary { get; set; }
   }
```

Reference Loop Handling

- The problem was occurring because in .NET Core 3 they change little bit the JSON politics. Json.Net is not longer supported and if you want to used all Json options, you have to download this Nuget:

Microsoft.AspNetCore.Mvc.NewtonsoftJson.

```
services.AddControllers().AddNewtonsoftJson(x => x.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore);
```

Data Transfer Object (DTO)

Right now, our web API exposes the database entities to the client. The client receives data that maps directly to your database tables. However, that's not always a good idea. Sometimes you want to change the shape of the data that you send to client.

For example, you might want to:

- Remove circular references .
- 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.
- Decouple your service layer from your database layer.

To accomplish this, you can define a data transfer object (DTO). A DTO is an object that defines how the data will be sent over the network.

ASP.NET Core API Pagination

```
[HttpGet]
public IActionResult Get([FromQuery] int page = 1, [FromQuery] int pageSize = 10)
{
    var query = _articles.Tolist();

    var totalCount = query.Count();
    var totalPages = (int)Math.Ceiling((double)totalCount / pageSize);

   query = query.Skip((page - 1) * pageSize).Take(pageSize);

    return Ok(query);
}
```

Enable Cross-Origin Requests (CORS)

- Define string variable as Cors policy in ***start*** class

```
var MyAllowSpecificOrigins = "_myAllowSpecificOrigins";
```

- Register ***AddCors*** in ***ConfigureServices*** method

```
builder.services.AddCors(options =>
        {
           options.AddPolicy(MyAllowSpecificOrigins,
           builder =>
           {
              builder.AllowAnyOrigin();
              builder.AllowAnyMethod();
              builder.AllowAnyHeader();
           }); });
```

- Add ***UseCors*** Middleware in ***Configure*** method.

```
app.UseCors(MyAllowSpecificOrigins);
```

JQeury client

```
function GetAllEmployees() {
        jQuery.support.cors = true;
        $.ajax({
            url: 'http://localhost:8080/API_SVC/api/EmployeeAPI',
            type: 'GET',
            dataType: 'json',
            success: function (data) {
                WriteResponse(data);
            },
            error: function (x, y, z) {
                alert(x + '\n' + y + '\n' + z);
            }
        });
    }
```

JQeury client

```
function AddEmployee() {
    jQuery.support.cors = true;
    var employee = {
        ID: $('#txtaddEmpid').val(),
        EmpName: $('#txtaddEmpName').val(),
        EmpDepartment: $('#txtaddEmpDep').val(),
        EmpMobile: $('#txtaddEmpMob').val()
    };

    $.ajax({
        url: 'http://localhost:8080/API_SVC/api/EmployeeAPI',
        type: 'POST',
        data:JSON.stringify(employee),
        contentType: "application/json;charset=utf-8",
        success: function (data) {
            WriteResponse(data);
        },
        error: function (x, y, z) {
            alert(x + '\n' + y + '\n' + z);
        }
    });
}
```

JQeury client

```
function DeleteEmployee() {
    jQuery.support.cors = true;
    var id = $('#txtdelEmpId').val()

    $.ajax({
        url: 'http://localhost:8080/API_SVC/api/EmployeeAPI/'+id,
        type: 'DELETE',
        contentType: "application/json;charset=utf-8",
        success: function (data) {
            WriteResponse(data);
        },
        error: function (x, y, z) {
            alert(x + '\n' + y + '\n' + z);
        }
    });
}
```

Lab

- Use ITI DB
- Create web API to manage ITI department and students.
- Create DTO to return student data with department name and supervisor name .
- Create DTO to return department data with number of student in this department .
- Limit add student to receive and send data in JSON format only;
- Apply Pagination and searching on student data
- Configure your API to Allow CORS .
- Apply auto mapping
- Test cors from any web consumer”js”