ASP.NET Core Web API

Web API

- REST APIs provide at least the following operations:
  - GET
  - POST
  - PUT
  - DELETE

Web API Basic Functions

- Supports creating Restful services automatically
- Converts data list to Json format
- Attribute routing requirement
- Automatic HTTP 400 responses
- Binding source parameter inference
- Multipart/form-data request inference
- Problem details for error status codes

Creating end point using ASP Core

- ASP Core defines an API controller class with or without methods according to selected configuration.
- Decorates the class with the [ApiController] attribute. This attribute indicates that the controller responds to web API requests.
- Example attribute definition
  - [HttpGet]
  - [HttpGet("{id}")]

ASP. Core Prepared REST Calls

GET /api/todo	Get all to-do items	None	Array of to-do items
GET /api/todo/{id}	Get an item by ID	None	To-do item
POST /api/todo	Add a new item	To-do item	To-do item
PUT /api/todo/{id}	Update an existing item	To-do item	None
DELETE /api/todo/{id}	Delete an item	None	None

- **GET**: The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.
- **POST**: is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.
- **PUT**: requests that the enclosed entity be stored under the supplied Request-URI. Idempotent
- **DELETE:** requests that the origin server delete the resource identified by the Request-URI.

Created End Point by Web API

```
// GET: api/Todo
[HttpGet]
public async Task<ActionResult<IEnumerable<TodoItem>>> GetTodoItems()
{
    return await _context.TodoItems.ToListAsync();
}

// GET: api/Todo/5
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long id)
{
    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    return todoItem;
}
```

Calling Endpoint

- These methods implement two GET endpoints:

- GET /api/todo
- GET /api/todo/{id}
- Test the app by **calling** the two endpoints from a browser. For example:

- https://localhost:<port>/api/todo
- https://localhost:<port>/api/todo/1

Attribute routing requirement

- The ApiController attribute makes attribute routing a requirement.
- For example:
  - [Route("api/[controller]")]
  - [ApiController]
  - public class ValuesController : ControllerBase …

Annotation with ApiController attribute

- ASP.NET Core 2.1 introduces the [ApiController] attribute to denote a web API controller class. For example:

- [Route("api/[controller]")]
- [ApiController]
- public class ProductsController : ControllerBase

The return types

- The return type is ActionResult<T> type.
- ASP.NET Core automatically serializes the object to JSON and writes the JSON into the body of the response message.
- The response code for this return type is 200, assuming there are no unhandled exceptions. Unhandled exceptions are translated into 5xx errors.

Automatic HTTP 400 responses

- The ApiController attribute makes model validation errors automatically trigger an HTTP 400 response. Consequently, the following code is unnecessary in an action method:

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

Binding source parameter inference automatically

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

Attribute	Binding source
[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
[FromServices]	The request service injected as an action parameter

API controller action return types

- Specific type
- IActionResult
- ActionResult<T>

Specific type

- [HttpGet]
- public List<Product> Get() =>
-     _reposito

- Without known conditions to safeguard against during action execution, returning a specific type could suffice. The preceding action accepts no parameters, so parameter constraints validation isn't needed.ry.GetProducts();

IActionResult type

- The IActionResult return type is appropriate when multiple ActionResult return types are possible in an action. The ActionResult types represent various HTTP status codes. Any non-abstract class deriving from ActionResult qualifies as a valid return type. Some common return types in this category are BadRequestResult (400), NotFoundResult (404), and OkObjectResult (200).

Don’t have to convert data to json explicitly

```
public JsonResult Test()
  {
      var events = new List<Event>()
      {
          new Event() {EventId = 1},
          new Event() {EventId = 2},
          new Event() {EventId = 3}
      };


      var results = events.Select(e => new
      {
          OrderID = e.EventId
      }).ToList();

      return new JsonResult() { Data = results, JsonRequestBehavior = JsonRequestBehavior.AllowGet };
  }
```

Multiple get methods

- Different parameter

```
 [HttpGet("{city}/{country}")]
        public string GetAll(string city,string country)
        { … }
/controller/city
```

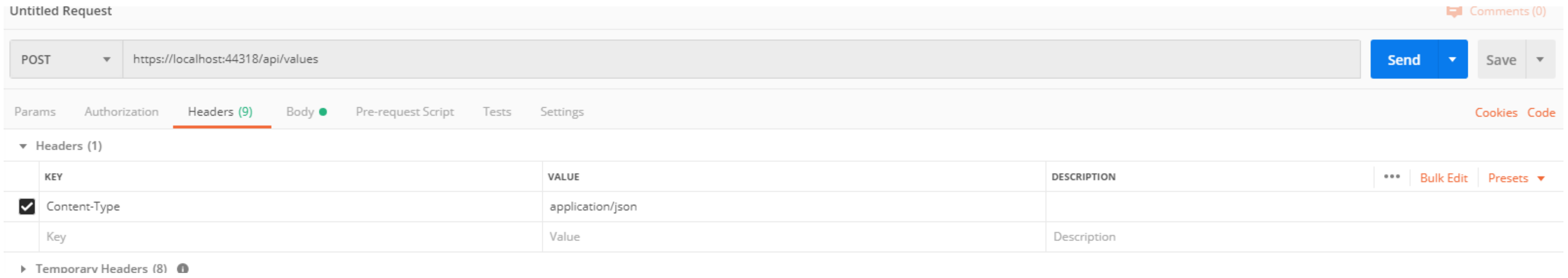
- if you want to have another Get() variation that has same signature to an existing Get().

```
 [Route("[action]/{country}")]
[HttpGet]
public IActionResult GetByCountry(string country)
{ ….
}
```

you need to explicitly specify the action name in the URL

http://www.binaryintellect.net/articles/9db02aa1-c193-421e-94d0-926e440ed297.aspx

Post Call fromPostman



POST https://localhost:44318/api/values

Headers (1)

	KEY	VALUE	DESCRIPTION
✓	Content-Type	application/json	
	Key	Value	Description

Questions