Software Development Tools & Practices

PUSL2020

Level 2

“API Development + Revision”

Dr. Rasika Ranaweera (ranaweera.r@nsbm.ac.lk | +94 11 544 6126)

Faculty of Computing | NSBM Green University

API

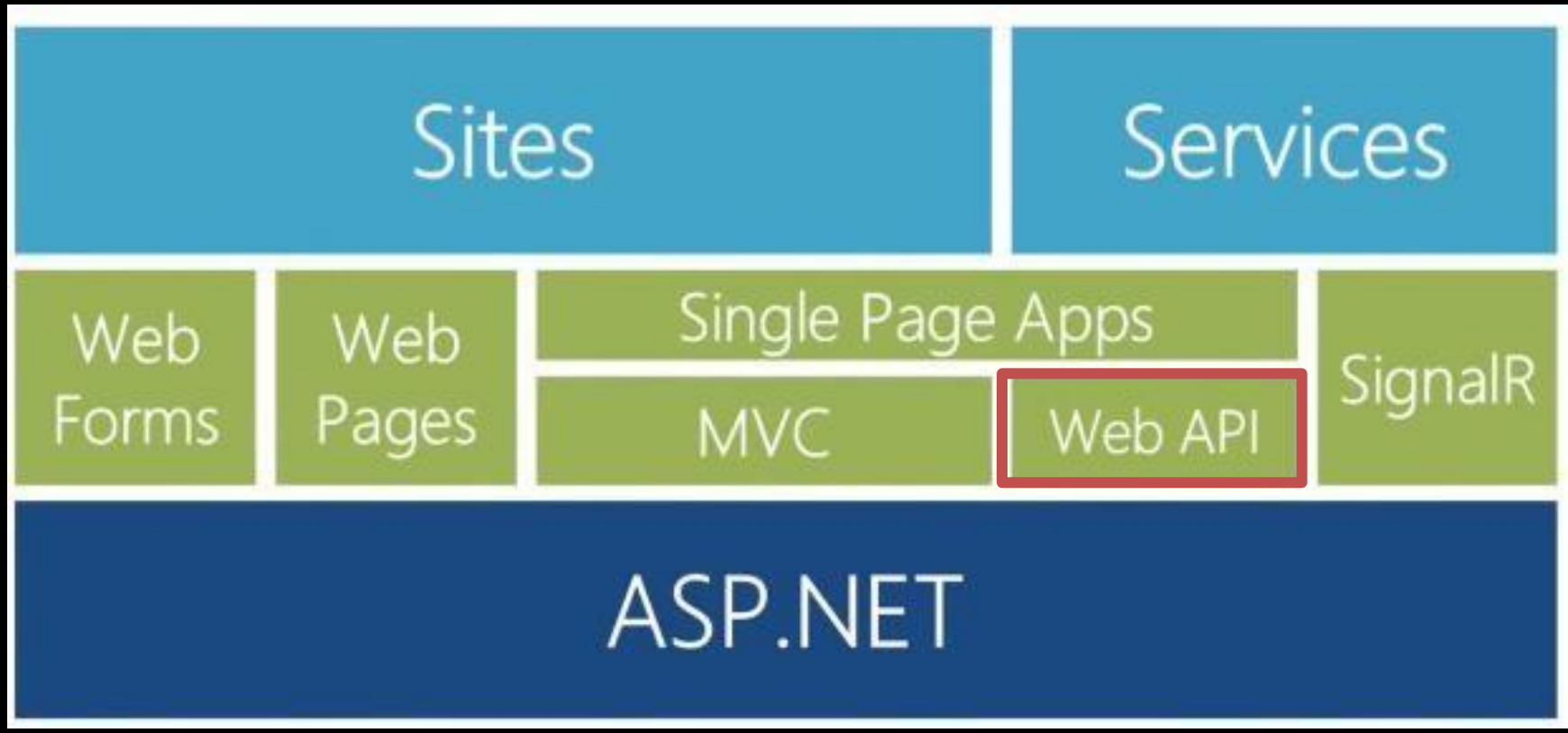
- Application Programming Interface
- Source code interface
  - For library or OS
  - Provides services to a program
- At its base, like a header file
  - But, more complete

```
#include <iostream>
#include <string>

using namespace std;

void prompt(string name[3], int grade[3]);
void init(int index[3]);
void sort(string name[3], int index[3]);
void output(string name[3], int grade[3], int index[3]);
```

ASP.NET


Sites
Services
Web Forms
Web Pages
Single Page Apps
MVC
Web API
SignalR
ASP.NET

ASP.NET WEB API

A Framework for creating HTTP Services that can reach a broad range of clients including browsers and mobile devices

What?

- A fully supported and extensible framework for building HTTP based endpoints
- Built on the top of ASP.NET
  - Mostly ASP.NET Routing
- Released with ASP.NET MVC4
  - Not linked to MVC – you can use alone, with MVC4 or you can use with ASP.NET web forms
- Also includes a new HTTP Client

Introduction, Routing, Attribute Routing

Attribute Routing

- Convention-based routing makes it hard to support certain URI patterns that are common in RESTful APIs.
- E.g. Customers have orders, movies have actors, books have authors
- It's natural to create URIs that reflect these relations
  - **/customers/1/orders**

- Attribute routing, is trivial to define a route for this URL.
- Simply by adding an [attribute] to the controller action

```
[Route("customers/{customerId}/orders")]
public IEnumerable<Order> GetOrdersByCustomer(int customerId) { ... }
```

Override the route prefix

Use a **tilde (~)** on the method attribute to override the route prefix:

```
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET /api/authors/1/books
    [Route("~/api/authors/{authorId:int}/books")]
    public IEnumerable<Book> GetByAuthor(int authorId) { ... }

    // ...
}
```

Route prefix parameters

route prefix can include parameters:

```
[RoutePrefix("customers/{customerId}")]
public class OrdersController : ApiController
{
    // GET customers/1/orders
    [Route("orders")]
    public IEnumerable<Order> Get(int customerId) { ... }
}
```

Route constraints

Route constraints let you restrict how the parameters in the route template are matched. The general syntax is "**{parameter:constraint}**"

```
[Route("users/{id:int}"]
public User GetUserById(int id) { ... }

[Route("users/{name}"]
public User GetUserByName(string name) { ... }
```

Optional URI Parameters and Default Values

Make a URI parameter optional by adding a question mark to the route parameter. If a route parameter is optional, you must define a default value for the method parameter.

```
public class BooksController : ApiController
{
    [Route("api/books/locale/{lcid:int?}")]
    public IEnumerable<Book> GetBooksByLocale(int lcid = 1033) { ... }
}
```

[OR]

```
public class BooksController : ApiController
{
    [Route("api/books/locale/{lcid:int=1033}")]
    public IEnumerable<Book> GetBooksByLocale(int lcid) { ... }
}
```

Supported constraints

Constraint	Description	Example
alpha	Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)	{x:alpha}
Bool	Matches a Boolean value.	{x:bool}
Datetime	Matches a DateTime value.	{x:datetime}
Decimal	Matches a decimal value.	{x:decimal}
double	Matches a 64-bit floating-point value.	{x:double}
float	Matches a 32-bit floating-point value.	{x:float}
guid	Matches a GUID value.	{x:guid}

C#

Int	Matches a 32-bit integer value.	{x:int}
length	Matches a string with the specified length or within a specified range of lengths.	{x:length(6)} {x:length(1,20)}
Long	Matches a 64-bit integer value.	{x:long}
max	Matches an integer with a maximum value.	{x:max(10)}
maxlength	Matches a string with a maximum length.	{x:maxlength(10)}
min	Matches an integer with a minimum value.	{x:min(10)}
minlength	Matches a string with a minimum length.	{x:minlength(10)}
range	Matches an integer within a range of values.	{x:range(10,50)}
regex	Matches a regular expression.	{x:regex(^\d{3}-\d{3}-\d{4}$)}

ACTION RESULTS IN WEB API

WEB API Controller Action

- Web API uses different mechanisms to create the **HTTP response**

Return type	How Web API create the response
Void	Return empty 204 (No Content)
HttpResponseMessage	Convert directly to an HTTP response message
IHTTPActionResult	Call ExecuteAsync to create an HTTPResponseMessage, then convert to an HTTP response message
Other type	Write the serialized return value into the response body; return 200(OK)

VOID

- If the return type is void, Web API simply returns an empty HTTP response with status code **204 (NO CONTENT)**

```
// POST: api/Test
0 references
public void Post
 ([FromBody]string value)
{
}
```

```
HTTP/1.1 204 No Content
Server: Microsoft-IIS/8.0
Date: Mon, 27 Jan 2014 02:13:26 GMT
```

HttpResponseMessage

- **HttpResponseMessage** return type, converts the return value directly into an HTTP response message using the properties of the **HttpResponseMessage** object to populate the response.
- The option gives you a lot of control over the response message

```
0 references
public class TestExampleController : ApiController
{
    0 references
    public HttpResponseMessage Get()
    {
        HttpResponseMessage response = Request.CreateResponse
          (HttpStatusCode.OK, "value");
        response.Content = new StringContent("Hello Syed Awase Khirni",
          System.Text.Encoding.Unicode);
        response.Headers.CacheControl = new
          System.Net.Http.Headers.CacheControlHeaderValue()
        {
            MaxAge = TimeSpan.FromMinutes(20)
        };
        return response;
    }

}
```

IHttpActionResult

- Introduced in Web API2
- *IHttpActionResult* Interface defines an HttpResponseMessage factory
- Advantages of using IHttpActionResult
  - Simplifies unit testing webapi controllers
  - Moves common logic for creating HTTP responses into separate classes
  - Makes the intent of the controller action clearer, by hiding the low-level details of constructing the response

Other Return Types

- Web API uses a media formatter to serialize the return value. Web API writes the serialized value into the response body.
- A disadvantage of this approach is that you cannot directly return an error code e.g. 404.
- You can only throw an **HttpResponseException** for error codes.

```
public class ProductsController : ApiController
{
    public IEnumerable<Product> Get()
    {
        return GetAllProductsFromDB();
    }
}
```

Parameter Binding in ASP.NET WEB API

- When a WEB API calls a method on a controller, it must set values for the parameters, a process called binding.

Rules of binding

- **Simple types** – include .NET primitive types (int, bool, double
..+ TimeSpan, DateTime, Guid, decimal and String)
- **Complex types** –Web API tries to read the value from the message body, using a media-type formatter

```
HttpResponseMessage Put(int id, Product item) { ... }
```

Simple type

Complex Type

FromURI

- To force WEB API to read a complex type from the URI
- Add the **[FromUri]** attribute to the parameter.

```
0 references
public class GeoPointController : ApiController
{
    0 references
    public HttpResponseMessage Get([FromUri] GeoPoint location)
    {
        HttpResponseMessage response = Request.CreateResponse
          (HttpStatusCode.OK, "value");
        string str = location.Latitude + " " + location.Longitude;
        response.Content = new StringContent(str,
          System.Text.Encoding.Unicode);
        response.Headers.CacheControl = new
          System.Net.Http.Headers.CacheControlHeaderValue()
        {
            MaxAge = TimeSpan.FromMinutes(20)
        };
        return response;
    }
}
```

```
public class GeoPoint
{
    1 reference
    public double Latitude { get; set; }
    1 reference
    public double Longitude { get; set; }
}
```

http://localhost/api/geopoint/?Latitude=47.678558&Longitude=122.130989

JSON (JavaScript Object Notation)

- A lightweight text based data-interchange format(data format)
- Completely language independent
- Based on a subset of the JavaScript Programming Language
- Easy to understand, manipulate and generate

NOT

- Overly Complex
- A “document” format
- A markup language
- A programming language

JSON Data Types

- Integer, real, scientific
- Strings (wrapped in double quotes)
- No octal or hex
- No NaN or Infinity – Use **null** instead
- Booleans: true or false
- Null: A value that specifies nothing or no value
- Objects: Unordered key/value pairs wrapped in { }
- Arrays: Ordered key/value pairs wrapped in [ ]

JSON Example

```
var employeeData = {
  "employee_id": 1236937,
  "name": "Jeff Fox",
  "hire_date": "1/1/2013",
  "location": "Norwalk, CT",
  "consultant": false,
  "random_nums": [ 24,65,12,94 ]
};
```

XML (Extensible Markup Language)

- Structure of document
- XML document is viewed as tree
- Elements divide the document into its constituent parts
- Elements are organized into a hierarchy

An Example XML Document

```
<!-- \Prolog"    -->
<?xml version="1.0"?>
<!DOCTYPE Greeting SYSTEM "wel.dtd">
<!-- End of Prolog -->
<!-- Content-->
<Greeting>
      <from>Instructor</from>
      <greetings>Welcome to XML</greetings>
      <to>CSIB 313 Class</to>
      <myGreetings>
            <greetings>Welcome to XML</greetings>
            <greetings>Welcome to XML</greetings>
      </myGreetings>
</Greeting>
```

JSON vs. XML

Similarities

- Plain text formats
- “Self-describing“ (human readable)
- Hierarchical (Values can contain lists of objects or values)

Difference

- JSON is lighter and faster than XML
- JSON uses typed objects. All XML values are type-less strings and must be parsed at runtime.
- JSON has less syntax, no semantics
- JSON properties are immediately accessible to JS code

Thank You!