WORKING WITH JSON

Lecturer: Le Thi Thu Lan

TEAM 4

- HUỲNH HOÀNG PHƯƠNG
- NGUYỄN QUỐC HUY

- NGUYỄN THANH QUANG DUY
- TRẦN MINH DUY

- QUÁCH HOÀNG ĐẠO
- HUỲNH ANH KIỆT

WHAT IS JSON

- JSON stands for JavaScript Object Notation
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand
- JSON data is written as name/value pairs, just like JavaScript object properties. A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

```
{
  "firstName":"John", "lastName":"Doe"
}
```

JSON SYNTAX RULES

1. Data is in name/value pairs
2. Data is separated by commas
3. Square brackets hold arrays
4. Curly braces hold objects

```
{
 "employees":
   [
     {"firstName":"John", "lastName":"Doe"},
     {"firstName":"Anna", "lastName":"Smith"},
     {"firstName":"Peter", "lastName":"Jones"}
   ]
}
```

JSON DATA TYPES

JSON supports mainly 06 data types: String, Number, Boolean, null, Object, and Array

String: JSON strings must be written in double quotes like C-language there are various special characters(Escape Characters) in JSON that you can use in strings such as \ (backslash), / (forward slash), b (backspace), n (new line), r (carriage return), t (horizontal tab), etc

Example:

{ "name":"Vivek" }

{ "city":"Delhi\/India" }

here \ / is used for Escape Character / (forward slash)

JSON DATA TYPES

Number: Represented in base 10. The octal and hexadecimal formats are not used

Example: { "age": 20 } , { "percentage": 82.44}

Boolean: This data type can be either true or false

Example: { "result" : true }

Null: It is just a define null value

Example: { "middlename":null }

Object: It is a set of name or value pairs inserted between {} (curly braces). The keys must be strings and should be unique and multiple key and value pairs are separated by a, (comma)

Syntax:

{ key : value, ...}

Example:

```
{
   "student":{ "name":"David", "age":20, "score": 50.05}
}
```

JSON DATA TYPES

Array: It is an ordered collection of values and begins with [(left bracket) and ends with] (right bracket). The values of array are separated by ,(comma) Syntax:

[value, ...]

Example:

```
{
  "collection" :  [
        {"id" : 101},
        {"id" : 102},
        {"id" : 103}
    ]
}
```

UNDERSTANDING JSON SERIALIZATION

The JSON (JavaScript Object Notation) serializer is fast and efficient,

JsonSerializer is used by ASP.NET Core 3

JsonSerializer (in the System.Text.Json namespace) is

The JSON serializer directly maps class property names to property names in JSON

UNDERSTANDING JSON SERIALIZATION

METHOD NAME	DESCRIPTION			
Deserialize(String, Type, JsonSerializerOptions)	**Parses the text representing a single JSON value into an instance of a specified type**	**v**	**v**	**v**
Deserialize(Utf8JsonReader, Type, JsonSerializerOptions)	Reads one JSON value (including objects or arrays) from the provided reader and converts it into an instance of a specified type	v	v	v
Deserialize<TValue>(String, JsonSerializerOptions)	Parses the text representing a single JSON value into an instance of the type specified by a generic type parameter.	v	v	v
Serialize(Object, Type, JsonSerializerOptions)	Converts the value of a specified type into a JSON string	v	v	v
Serialize(Utf8JsonWriter, Object, Type, JsonSerializerOptions)	Writes the JSON representation of the specified type to the provided writer	v	v	v
SerializeToUtf8Bytes(Object, Type, JsonSerializerOptions)	Converts a value of the specified type into a JSON string, encoded as UTF-8 bytes	v	v	v
		v	v	v

Tô hai hoặc nhiều ô, **nhấp chuột phải** rồi chọn "**Gộp ô**" để sắp xếp bảng theo nhu cầu của bạn!

CONTROLLING SERIALIZATION WITH ATTRIBUTES

ATTRIBUTE NAME	DESCRIPTION			
JsonIgnoreAttribute	**Prevents a property from being serialized or deserialized**			
JsonPropertyNameAttribute	Specifies the property name that is present in the JSON when serializing and deserializing. This overrides any naming policy specified by JsonNamingPolicy			
JsonExtensionDataAttribute	When placed on a property of type IDictionary<TKey,TValue>, any properties that do not have a matching member are added to that dictionary during deserialization and written during serialization			
JsonConverterAttribute	Converts an object or value to or from JSON			
JsonIncludeAttribute	Indicates that the member should be included for serialization and deserialization			

Jinsei no subete no mono ni wa hajimari to owari ga aru

CONTROLLING SERIALIZATION WITH ATTRIBUTES DEMO

```
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Serialization;

1 reference
public class Person
{
    1 reference
    public string FirstName { get; set; }

    [JsonIgnore]
    1 reference
    public string LastName { get; set; }

    [JsonPropertyName("age")]
    1 reference
    public int AgeInYears { get; set; }

    [JsonConverter(typeof(DateConverter))]
    1 reference
    public DateTime Birthdate { get; set; }
}

1 reference
public class DateConverter : JsonConverter<DateTime>
{
    0 references
    public override DateTime Read(
        ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        return DateTime.ParseExact(reader.GetString(), "yyyy-MM-dd", null);
    }

    0 references
    public override void Write(
        Utf8JsonWriter writer, DateTime value, JsonSerializerOptions options)
    {
        writer.WriteStringValue(value.ToString("yyyy-MM-dd"));
    }
}
```

```
public class DateConverter : JsonConverter<DateTime>
{
    0 references
    public override DateTime Read(
        ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        return DateTime.ParseExact(reader.GetString(), "yyyy-MM-dd", null);
    }

    0 references
    public override void Write(
        Utf8JsonWriter writer, DateTime value, JsonSerializerOptions options)
    {
        writer.WriteStringValue(value.ToString("yyyy-MM-dd"));
    }
}
```

```
        var person = new Person
        {
            FirstName = "John",
            LastName = "Doe",
            AgeInYears = 35,
            Birthdate = new DateTime(1988, 3, 15)
        };

        var options = new JsonSerializerOptions
        {
            IgnoreNullValues = true,
            WriteIndented = true
        };

        var json = JsonSerializer.Serialize(person, options);
        Console.WriteLine(json);
    }
}
```

```
{
  "FirstName": "John",
  "age": 35,
  "Birthdate": "1988-03-15"
}
```

JSON SERIALIZATION OPTIONS

PROPERTY NAME	DESCRIPTION
WriteIndented	**Gets or sets a value that defines whether JSON should use pretty printing. By default, JSON is serialized without any extra white space**
AllowTrailingCommas	Get or sets a value that indicates whether an extra comma at the end of a list of JSON values in an object or array is allowed (and ignored) within the JSON payload being deserialized
ReadCommentHandling	Gets or sets a value that defines how comments are handled during deserialization
PropertyNameCaseInsensitive	Gets or sets a value that determines whether a property's name uses a case-insensitive comparison during deserialization. The default value is false
ReferenceHandler	Configures how object references are handled when reading and writing JSON

JSON SERIALIZATION OPTIONS

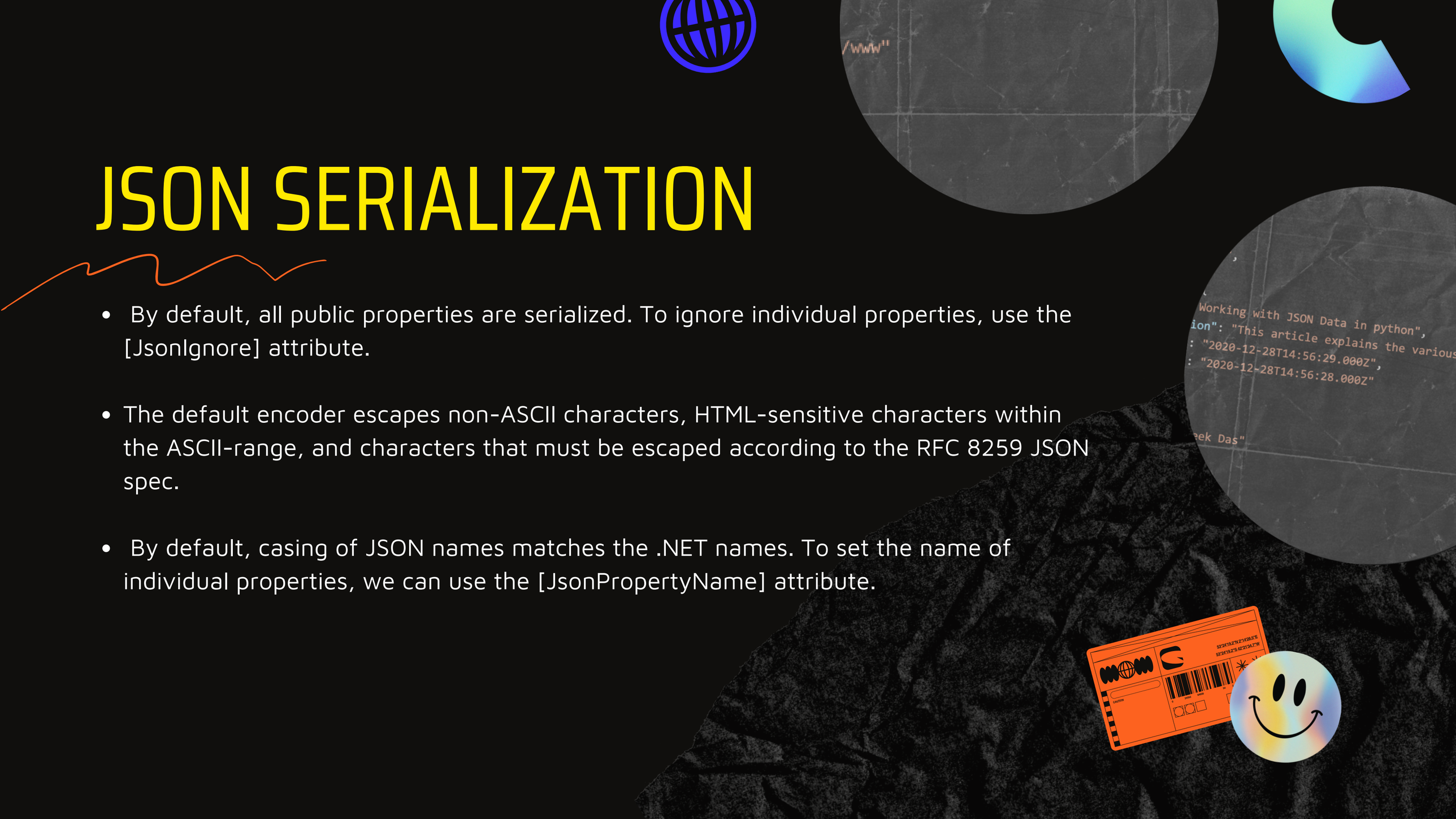
PROPERTY NAME	DESCRIPTION
PropertyNamingPolicy	Gets or sets a value that specifies the policy used to convert a property's name on an object to another format, such as camel-casing, or null to leave property names unchanged
DictionaryKeyPolicy	Gets or sets the policy used to convert a IDictionary key's name to another format, such as camel-casing
Encoder	Gets or sets the encoder to use when escaping strings, or null to use the default encoder
IgnoreNullValues	ets or sets a value that determines whether null values are ignored during serialization and deserialization. The default value is false.
IgnoreReadOnlyProperties	Determines whether read-only fields are ignored during serialization. A field is read-only if it is marked with the readonly keyword. The default value is false

JSON SERIALIZATION

- By default, all public properties are serialized. To ignore individual properties, use the [JsonIgnore] attribute.

- The default encoder escapes non-ASCII characters, HTML-sensitive characters within the ASCII-range, and characters that must be escaped according to the RFC 8259 JSON spec.

- By default, casing of JSON names matches the .NET names. To set the name of individual properties, we can use the [JsonPropertyName] attribute.



JSON DESERIALIZATION

- Property name matching is case-sensitive.
- By default, enums are supported as numbers. We can serialize enum names as strings.
- To allow comments in the JSON, we can set the. JsonSerializerOptions.ReadCommentHandling property to JsonCommentHandling.Skip
- The default maximum depth is 64.