



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ ПО КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

КОМПЮТЪРНО И СОФТУЕРНО ИНЖЕНЕРСТВО

Програмиране за разпределени среди

Курсова работа

,, Функционалност, специфична за JSON формата,
която няма директен аналог в XML“

Съставил: Ангел Любомиров Стойнов

Факултетен номер: 121222150

Група: 40

Съдържание

Детайлно задание	3
JSON	4
XML	5
Специфична функционалност разглеждана в курсовата работа.....	6
Разработка	7
Обща характеристика	7
Файлова структура	7
JsonFormat.cs.....	8
IncorrectXmlFormat.cs	9
XsdFormat.cs	10
XmlFormat.cs.....	12
Цитирани източници.....	13

Детайлно задание

Функционалност, специфична за JSON формата, която няма директен аналог в XML.
Функционалността е по ваш избор.

- a) Намерете функционалност свързана с JSON , която няма директен аналог в XML
 - a. не че не може да се направи с XML, но не може под същата форма
- b) Намерете готов JSON файл или създайте подходящ JSON файл с достатъчно данни, върху който може да се демонстрира функционалността (от т.а).
- c) Създайте програма (на среда .Net), която да изпълнява функционалността (от т.а) върху демонстрационния файл (от т.b)
- d) Създайте еквивалентен XML файл на демонстрационния файл (от т.2)
- e) Създайте програма (на среда .Net), която да изпълнява функционалността (от т.а) върху еквивалентния файл (от т.e)
- f) Документирайте разликата и ефекта от разликата. Документирате всичко необходимо за да изпълните предните точки.

JSON

JSON (JavaScript Object Notation) е опростен формат за обмяна на данни. Той е базиран на едно подмножество на езика за програмиране **JavaScript**, Standard ECMAScript 262 3rd Edition - от декември 1999 г. **JSON** има текстов формат, напълно независим от реализацията на езика, но използва конвенции, които са познати на програмистите на C-подобни езици, включително **C, C++, C#, Java, JavaScript, Perl, Python**, и много други. Тези свойства правят **JSON** идеален формат за обмяна на данни.

JSON се състои от две структури:

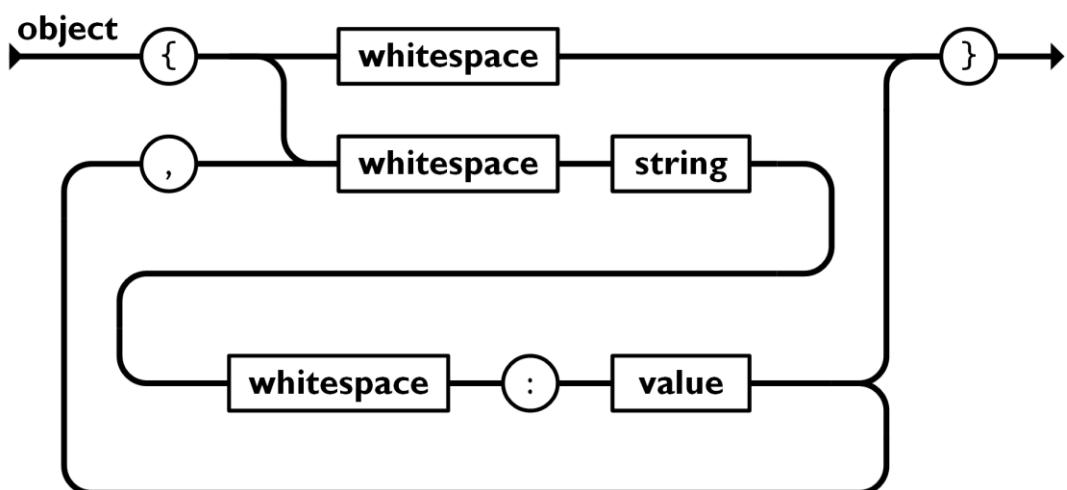
- Колекция от двойки име/стойност (key/pair).
- Подреден списък от стойности.

Обектът започва с **U+007B** ({) и се приключва с **U+007D** (}). Ключовете са в **U+0022** ("") и завършват с **U+003A** (:). Отделните записи за разделени с **U+002C** (,) [1]. Показано на фигура 1 и 2.

Една стойност може да бъде string (низ), число, boolean, null, обект или масив [2]. Тези структури могат да бъдат вложени. [3] [4]

```
1   {
2     "name": "Angel",
3     "age": 22,
4     "universityStudent": true
5   }
```

Фигура 1, примерен JSON файл.



Фигура 2, структура на **JSON** файл. [2]

XML

Extensible Markup Language (XML) представлява универсален формат за описание и съхранение на данни, който улеснява обмена на информация между различни компютърни системи — като уеб сайтове, бази данни и външни приложения.

Благодарение на предварително дефинираните синтактични правила, XML осигурява лесно и надеждно предаване на данни през всякакъв тип мрежа, тъй като получателят може точно да интерпретира съдържанието според тези правила. [5]

XML файловете могат да съдържат метаданни, които описват структурата, значението или контекста на самите данни. XML често се използва като основа за различни комуникационни и авторизиращи протоколи, като например **SAML**, **SOAP** и други. [6] [7]

Тези протоколи използват XML за формализиране и сигурен обмен на данни между приложения и системи в уеб среда. XML е доста по-вербозен отколкото **JSON**. Синтаксисът на XML е доста подобен на **HTML** (фиг. 3). Състои се от отварящи и затварящи тагове, които не са предварително дефинирани. Те се създават от програмиста. Таговете могат да съдържат вложени (дъщерни) елементи, което позволява изграждане на йерархична структура на данните. [5]

Всеки XML документ започва с декларация, която указва версията и кодирането, например [4]:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <person>
3      <name>Angel</name>
4      <age>22</age>
5      <universityStudent>true</universityStudent>
6  </person>
```

Фигура 3, синтаксис на **XML**. Всички тагове са създадени от програмиста и не са стандартно дефинирани от самия markup language.

Специфична функционалност разглеждана в курсовата работа

В основата си **XML** не поддържа примитивни типове, всеки елемент се разглежда и третира като **string** (низ) [5]. За да бъде уточнен примитивен тип (**number**, **boolean** и т.н.) е необходимо програмно да се добави атрибута **type** (фиг. 4) [8]. Това става възможно с използването на **XSD – XML Schema Definition**, който описва структурата и типовете данни (фиг. 5) [9] [10]. Това допълнително ще утежни файла, особено, ако той е дълъг и комплексен. Въпреки това **XML** ще може да постигне ниво на типизация, подобно на **JSON** [4] [11], но не по подразбиране, а чрез допълнителна схема или логика на приложението.

Друго решение би било, използване на десериализация, то е част от **.NET** средата, но това означава стойностите на **.xml** файла да бъдат облечени (wrap) в **C#** клас.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <root>
3      <name type="string">Angel</name>
4      <age type="number">22</age>
5      <universityStudent type="boolean">true</universityStudent>
6  </root>
```

Фигура 4, примерен начин за дефиниране на тип в **XML**.

```
<?xml version="1.0"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">

    <xss:element name="note">
        <xss:complexType>
            <xss:sequence>
                <xss:element name="to" type="xss:string"/>
                <xss:element name="from" type="xss:string"/>
                <xss:element name="heading" type="xss:string"/>
                <xss:element name="body" type="xss:string"/>
            </xss:sequence>
        </xss:complexType>
    </xss:element>

</xss:schema>
```

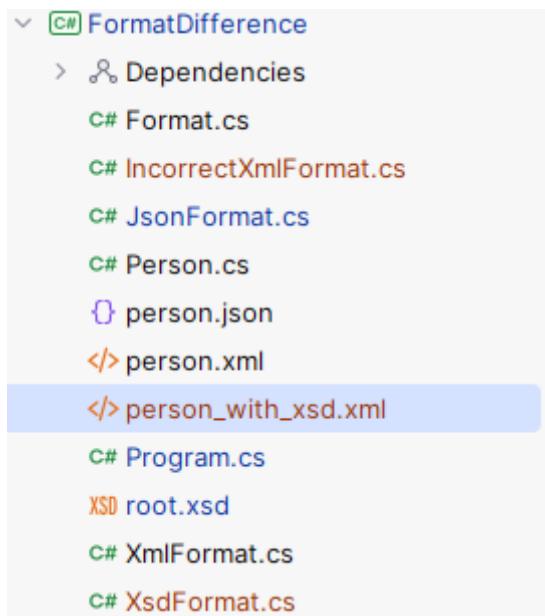
Фигура 5, примерен **XSD** използван за дефиниране на тип.

Разработка

Обща характеристика

За демонстрация на посочената функционалност е разработено конзолно .NET приложение. Целта му е да покаже различията на **JSON** и **XML** формата. Съдържа примери как може да се имплементира подобна функционалност на **XML**. Всички **.xml** и **.json** файлове са създадени предварително и могат да бъдат намерени в solution.

Файлов структура



Фигура 6, структура на приложението.

JsonFormat.cs

```
5 ^o public class JsonFormat : Format
6 {
7     private const string JsonPath = "person.json";
8
9 ^o     [1 usage Angel L Stoynov]
10    public override void Display()
11    {
12        var path = Path.Combine(AppContext.BaseDirectory, JsonPath);
13        if (!File.Exists(path))
14        {
15            Console.WriteLine("File not found.");
16            return;
17        }
18
19        var json :string = File.ReadAllText(path);
20
21        using var doc :JsonDocument = JsonDocument.Parse(json);
22        JsonElement root = doc.RootElement;
23
24        foreach (var prop :JsonProperty in root.EnumerateObject())
25        {
26            var value :JsonElement = prop.Value;
27            Console.WriteLine($"{prop.Name} => {value.ValueKind}");
28        }
29    }
}
```

Фигура 7, JsonFormat.cs класа

JSON притежава вградена типова система, която разпознава автоматично различните примитивни стойности — низове, числа, булеви стойности и null. Дори без десериализация, библиотеката **System.Text.Json** може да идентифицира всеки тип чрез свойството **ValueKind**.

Прочитаме **.json** файла (фиг. 8) типовете съвпадат с очакваните (фиг. 9).

```
1   {
2       "name": "Angel",
3       "age": 22,
4       "universityStudent": true
5 }
```

Фигура 8, използваният **.json** файл за изпълнението на JsonFormat.cs

```
Name: Angel Type: System.String
Age: 22 Type: System.Int32
Status: True Type: System.Boolean
```

Фигура 9, изходните данни, които се визуализират при изпълнението на JsonFormat.cs

IncorrectXmlFormat.cs

```
public class IncorrectXmlFormat
{
    private const string XmlPath = "person.xml";

    [1 usage]
    public static void XmlDifference()
    {
        XDocument xmlDocument = XDocument.Load(XmlPath);
        var root: XElement? = xmlDocument.Root;

        var name = root.Element("name").Value; // "Angel"
        var age :string = root.Element("age").Value; // "10"
        var universityStatus :string = root.Element("universityStudent").Value; // true

        Console.WriteLine($"\\nBefore proper deserialization - XML TYPES:");
        Console.WriteLine($"name: {name.GetType()}"); // String
        Console.WriteLine($"age: {age.GetType()}"); // String
        Console.WriteLine($"status: {universityStatus.GetType()}"); // String
    }
}
```

Фигура 10, IncorrectXmlFormat.cs класа

Отново се прочита файл, но този път файлът е **XML** (фиг. 11). Основната разлика е изходните данни. Има съществената разлика, типовете не са очакваните. Вместо да се визуализира System.String, System.Int32, System.Boolean, получените стойности са три пъти System.String (фиг. 12).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <person>
3      <name>Angel</name>
4      <age>22</age>
5      <universityStudent>true</universityStudent>
6  </person>
```

Фигура 11, показва използваният .xml файл за изпълнението на XmlFormat.cs

```
Before proper deserialization - XML TYPES:
name: System.String
age: System.String
status: System.String
```

Фигура 12, полученият резултат е неочекван, различава се по типовете.

XsdFormat.cs

```
6 ^o public class XsdFormat : Format
7 {
8     private const string XmlPath = "person_with_xsd.xml";
9     private const string XsdPath = "root.xsd";
10
11 ^o     □ 2 usages
12     {
13         □ 1 usage
14         public override void Display()
15         {
16             var xml:string = Path.Combine(AppContext.BaseDirectory, XmlPath);
17             var xsd:string = Path.Combine(AppContext.BaseDirectory, XsdPath);
18
19             var schema = new XmlSchemaSet();
20             schema.Add(targetNamespace:"", schemaUri: xsd);
21
22             var settings = new XmlReaderSettings()
23             {
24                 Schemas = schema,
25                 ValidationType = ValidationType.Schema
26             };
27             settings.ValidationEventHandler += ValidationCallback;
28
29             using var reader = XmlReader.Create(xml, settings);
30             while (reader.Read()) { }
31             Console.WriteLine("XML is valid according to the XSD schema.");
32         }
33
34     }
35     □ 1 usage
36     private static void ValidationCallback(object? sender, ValidationEventArgs e)
37     {
38         Console.WriteLine($"Validation error: {e.Message}");
39     }
40 }
```

Фигура 13, XsdFormat.cs класа

Ще използваме **schema** (фиг. 14), за да валидирате съдържанието на **XML** файла (фиг. 15). Ако всичко е наред, ще се визуализира ред 28, но ако има проблем, например при тип **number** е въведено “two”, тогава ще се изпълни статичният метод **ValidationCallback**.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
3      <xss:element name="root">
4          <xss:complexType>
5              <xss:sequence>
6                  <xss:element name="name">
7                      <xss:complexType>
8                          <xss:simpleContent>
9                              <xss:extension base="xs:string">
10                             <xss:attribute name="type" type="xs:string" use="required"/>
11                         </xss:extension>
12                     </xss:simpleContent>
13                 </xss:complexType>
14             </xss:element>
15             <xss:element name="age">
16                 <xss:complexType>
17                     <xss:simpleContent>
18                         <xss:extension base="xs:integer">
19                             <xss:attribute name="type" type="xs:string" use="required"/>
20                         </xss:extension>
21                     </xss:simpleContent>
22                 </xss:complexType>
23             </xss:element>
24             <xss:element name="universityStudent">
25                 <xss:complexType>
26                     <xss:simpleContent>
27                         <xss:extension base="xs:boolean">
28                             <xss:attribute name="type" type="xs:string" use="required"/>
29                         </xss:extension>
30                     </xss:simpleContent>
31                 </xss:complexType>
32             </xss:element>
33         </xss:sequence>
34     </xss:complexType>
35 </xss:element>
36 </xss:schema>

```

Фигура 14, **XSD – schema**, която дефинира три основни типа и **attribute - type**, който е задължително да бъде използван в **.xml** файла.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:noNamespaceSchemaLocation="root.xsd">
4      <name type="string">Angel</name>
5      <age type="number">22</age>
6      <universityStudent type="boolean">true</universityStudent>
7  </root>

```

Фигура 15, **.xml** файл, който използва горепосочената **schema**. По този начин се дефинират **primitive** типове данни.

XmlFormat.cs

```
7 ^o 2 usages  Angel L Stoynov
8 public class XmlFormat : Format
9 {
10     private const string XmlPath = "person.xml";
11 ^o 1 usage  Angel L Stoynov
12     public override void Display()
13     {
14         var path = Path.Combine(ApplicationContext.BaseDirectory, XmlPath);
15         if (!File.Exists(path))
16         {
17             Console.WriteLine("File not found.");
18             return;
19         }
20         var xmlContent:string = File.ReadAllText(path);
21         using var stringReader = new StringReader(xmlContent);
22
23         var xmlSerializer = new XmlSerializer(typeof(Person));
24         var person = (Person)xmlSerializer.Deserialize(stringReader);
25
26         if (person == null)
27         {
28             Console.WriteLine("Person is null.");
29             return;
30         }
31
32         Console.WriteLine("\nName: " + person.name + " Type: " + person.name.GetType());
33         Console.WriteLine("Age: " + person.age + " Type: " + person.age.GetType());
34         Console.WriteLine("Status: " + person.universityStudent + " Type: " + person.universityStudent.GetType());
35     }
36 }
```

Фигура 16, XmlFormat.cs класа

За да постигнем десериализация ще създаден клас Person.cs, който ще приеме стойностите на .xml файла. За целта е използван **XmlSerializer**, който успява да извърши автоматична програмна типизация.

Важно е да се отбележи, че това не е свойство на **XML**, а на **.NET** средата, която извършва типова конверсия въз основа на дефинирания модел.

Цитирани източници

- [1] „Unicodes,“ [Онлайн]. Available: <https://unicodes.jesselane.com/>.
- [2] „json,“ [Онлайн]. Available: <https://www.json.org/json-en.html>.
- [3] „ECMA-404,“ December 2017. [Онлайн]. Available: https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf.
- [4] D. Crockford, „rfc4627,“ Network Working Group , July 2006. [Онлайн]. Available: <https://www.ietf.org/rfc/rfc4627.txt>.
- [5] „what-is-xml,“ Amazon AWS, [Онлайн]. Available: <https://aws.amazon.com/what-is/xml/>.
- [6] „SAML,“ webopedia, [Онлайн]. Available: <https://www.webopedia.com/definitions/saml/>.
- [7] K. Lane, „SOAP API,“ Postman, 28 June 2023. [Онлайн]. Available: <https://blog.postman.com/soap-api-definition/>.
- [8] „storing-primitives,“ stackoverflow, [Онлайн]. Available: <https://stackoverflow.com/questions/50183989/storing-primitives-in-xml>.
- [9] „XSD,“ microsoft, [Онлайн]. Available: [https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ms764635\(v=vs.85\)](https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ms764635(v=vs.85)).
- [10] „XSD Schema,“ w3schools, [Онлайн]. Available: https://www.w3schools.com/xml/schema_intro.asp.
- [11] „JSON primitives,“ Yugabyte, [Онлайн]. Available: https://docs.yugabyte.com/preview/api/ysql/datatypes/type_json/primitive-and-compound-data-types/.