Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Бакалавърска програма
„Софтуерно инженерство”

Предмет: XML технологии за семантичен Уеб

Зимен семестър, 2022/2023 год.

Тема №32: „Каталог на ресторантите в България - 2“

Курсов проект

Автори:

Давид Петров, фак. номер 62596

Павел Атанасов, фак. номер 62555

януари, 2023 г.

София

Съдържание

1 Въведение (1 стр.)

Избраното от нас задание има за цел изготвянето на каталог на ресторантите в България по региони, базиран на XML документи с текстово и графично съдържание, описващо възможните характеристики на всеки един ресторант в страната.

Това е винаги актуална тема у нас поради динамичността на бранша - постоянно се появяват нови или вече съществуващи ресторанти прекратяват своята дейност, затова е необходим автоматизиран начин за представянето на тази динамична информация в актуален вид за нуждите на потребителите - искаме да автоматизираме представянето на данни за съществуващите ресторанти по региони у нас, като от сурови данни във входен формат директно получим стилизиран html документ, готов за представяне в браузър. Като цяло разработеното задание решава проблема по обработката на сурови данни до готов за презентация формат, като входните данни могат да идват от разнообразни източници, а финалният резултат вече може да се използва за по-широка дистрибуция в интернет.

Тази ключова трансформация постигаме, като първо получим входни данни в съответния формат (.csv или .xml файл), като при входен формат .csv сме добавили допълнителен скрипт, който превежда данните в .xml формат. Това е всъщност желания от нас входен файл, тъй като подлежи на семантична валидация (посредством XML Schema), както и удобна трансформация (посредством XSLT) до желания резултат. В самия процес на генерирането на краен .html документ, прилагаме и CSS стилизация върху DOM съдържанието му с цел по-приятна за окото визуализация на данните.

Именно върху детайлите на цялостния процес се фокусира и остатъкът от този документ, следващ грубо следната структура: кратък анализ на решението в точка 2, обсъждащ грубо работния процес на решението (от вход до очакван изход), както и структурата и типа на представеното съдържание. След това в точка 3 следва дълбок поглед върху етапите на обработката както и използваните инструменти / технологии / подходи. Приключваме с кратко резюме на начините, по които сме тествали коректността на изпълнението, и заключение от цялостната работа плюс идеи за бъдещо развитие.

2 Анализ на решението

2.1 Работен процес (2-3 стр.)

Грубо очертано, работният процес се състои от три стъпки:

1. получаваме входни данни в .xml формат, примерно restaurants.xml
   - добавили сме и поддръжка за входни данни в .csv формат, който се превежда чрез помощен скрипт на Python до очаквания от нас .xml файл;
2. .xml файлът се валидира посредством XML Schema, намираща се в .xsd файл със същото име, в случая restaurants.xsd, реферирaн от .xml файла;

3. след като е валидирано съдържанието, следва трансформация посредством XSLT скрипт, реферирaн от .xml документа, в случая restaurants.xsl, резултатът от която е .html файл;
4. генерираният HTML документ реферира style.css - файл с каскадни стилове за пригледно оформление на резултатното съдържание.

Входното съдържание представлява множество от ресторанти в България. Под “ресторант” разбираме конкретен обект (физическа сграда), за който имаме:

- Име: свободно текстово съдържание, представляващо комерсиалното наименование на съответния обект (ресторант)
- Тип: основната категория на ресторанта според вида на предлаганата храна / културна атмосфера / тема или друго. Това съдържание също може да е свободно, но по идея типът служи като външен ключ за категоризация, така че е препоръчително да е една дума, която лесно да може да се повтори при други ресторанти от същия тип.
- Регион: географски регион, в който се намира ресторантът. В данните, подбрани от нас, регион е равносилно на административна област в България, например София, Бургас, пр. Отново, няма функционално ограничение и съдържанието може да е свободен текст, но семантично това също е категоризационен ключ, по който можем да групираме ресторантите.
- Адрес: точен адрес на физическия обект без валидационни ограничения - свободен текст
- Рейтинг: дробно число по стандартна петобалната рейтингова система
- Изображение: примерно изображение, онаглеждаващо атмосферата в обекта (като цяло изображението също е свободен избор - според каквато цел е най-потребна)

Данните могат да идват от произволен източник (най-често интернет), като конкретно в нашето изпълнение сме черпили данни за ресторанти от следните сайтове:

- http://www.restaurant.bg
- http://www.restaurants.com
- http://zavedenia-sofia.com/restaurants.html

Входните данни сме извличали на ръка с демонстрационна цел, но, разбира се, при нужда от по-мащабно използване и извличането им може да се автоматизира.

Както беше и за нас по-удобно, ще допуснем, че тази информация е налична в .csv файл, като например:

```
Name                    |Type    |Region          |Address                    |Rating

SASA Fashion Food       |Sushi   |Sofia           |bul. Cherni vrah 100       |4.0
SUSHI EXPRESS           |Sushi   |Varna           |Centar, bul. Praga 20      |4.9
Srabski Restorant BEST  |Srabska|Sofia            |bul. Nikolaj J. Vapcarov 1 |4.4
Franco Pica             |Pizza   |Burgas          |Sofia, Ul. Han Asparuh 37  |4.2
Kapitan Kuk             |Morska |Sofia           |pl. Narodno Sabranie 4     |4.0
Zlatna srabska skara    |Srabska|Stara Zagora     |bul. Nikola Petkov 17      |4.8
Trite Shterki           |Bulgarska|Varna         |bul. Dondukov 60A          |3.9
```

Следва обработка от .py скрипта, за да получим целевия .xml документ, който служи за същински вход занапред. Структурата на .xml файла е разгледана по-подробно в следващата точка, затова няма да се повтаряме тук.

Забелязваме, че в посочения на горното изображение .csv файл липсва информацията за изображението и това е така по дизайн. Тъй като скриптът е по-скоро помощен, по презумпция той генерира линкове към изображения в папка `images/restaurantX.jpg`, където X е поредния номер на ресторанта във входния файл. Конкретните URI се считат като указан вход в самия .xml файл.

След като вече имаме съдържанието си в .xml документ, следва валидация. Избрали сме XML Schema като валидационен инструмент вместо DTD заради по-широкия набор от функционалности, но пък за сметка на това сега имаме нужда от валидиращ парсър. Понеже парсърите на браузърите не поддържат валидиране, използваме външен инструмент за валидация: https://www.liquid-technologies.com/online-xsd-validator

Следва трансформация посредством XSLT, чийто краен резултат е стилизиран HTML документ, готов за визуализация в браузър:

Restaurant Catalogue

Name	Type	Region	Address	Rating	Image
SUSHI EXPRESS	Sushi	Varna	Centar, bul. Praga 20	4.9	
Zlatna srabska skara	Srabska	Stara Zagora	bul. Nikola Petkov 17	4.8	
Srabski Restorant BEST	Srabska	Sofia	bul. Nikolaj J. Vapcarov 1	4.4	
Franco Pica	Pizza	Burgas	Ul. Han Asparuh 37	4.2	
SASA Fashion Food	Sushi	Sofia	bul. Cherni vrah 100	4.0	

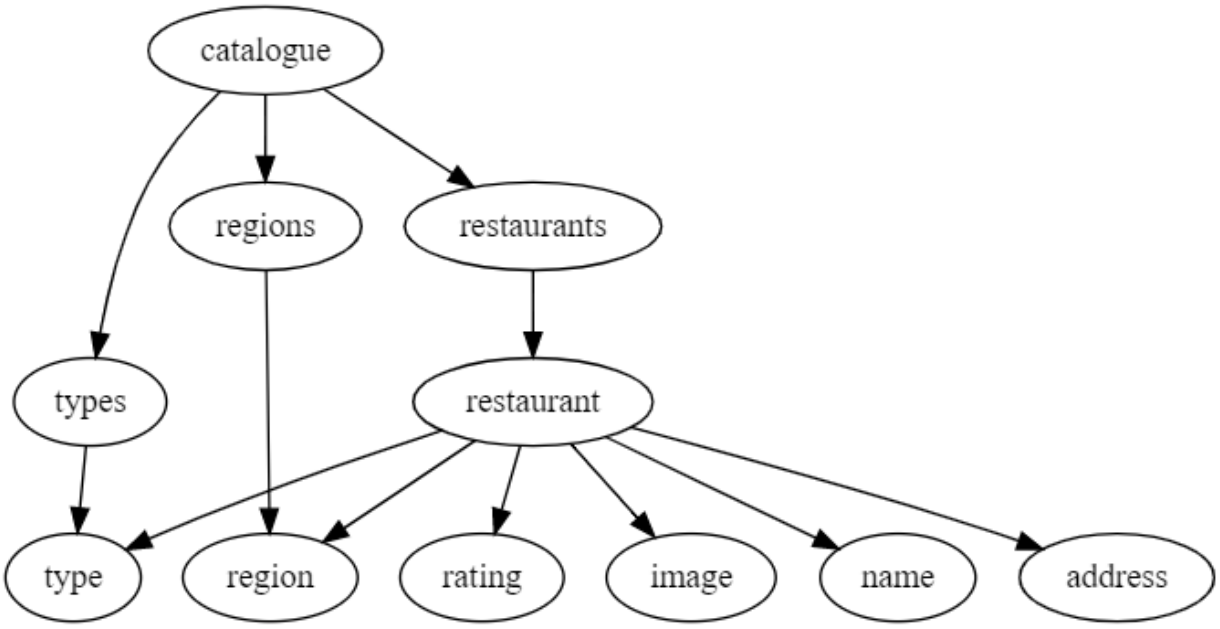
2.2 Структура на съдържанието (2-4 стр.)

Корен на .xml файла е елемент <catalogue>, който има три деца:

- <types>, съдържащ енумерация от всички типове ресторанти в базата данни - 5 на брой;
- <regions>, съдържащ енумерация от всички региони на ресторантите - 4 на брой;
- <restaurants>, съдържащ списък с произволна дължина от елементи <restaurant> - 7 на брой

Всеки <restaurant> елемент съдържа по един дъщерен елемент за всеки свой компонент.

Йерархична диаграма (генерирана чрез dot скрипт и сайта GraphViz) за обобщен преглед на структурата:



Обобщаващ вариант на целия .xml файл с входни данни изглежда така (елементите, които не носят нова пояснаваща информация за цялостната структура са свити с цел олекотяване на визията):

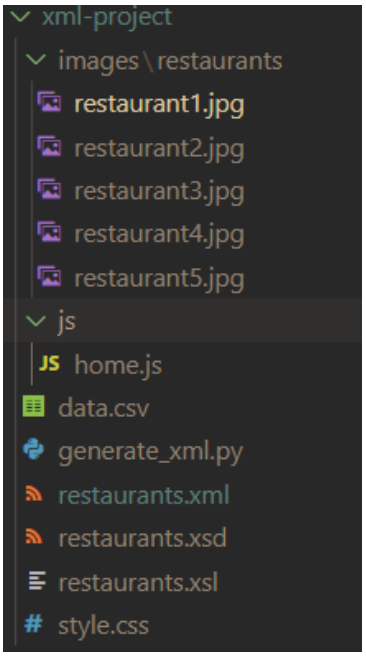
```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="restaurants.xsl" type="text/xsl"?>

<!DOCTYPE catalogue [
  <!NOTATION jpeg SYSTEM "image/jpeg">
  <!ENTITY restaurant1_image SYSTEM "./images/restaurants/restaurant1.jpg" NDATA jpeg>···
]>

<catalogue>
  <types>
    <type id="sushi">Sushi</type>···
  </types>
  <regions>
    <region id="sofia">Sofia</region>···
  </regions>
  <restaurants>
    <restaurant id="restaurant1">
      <name>SASA Fashion Food</name>
      <type idref="sushi"/>
      <region idref="sofia"/>
      <address>bul. Cherni vrah 100</address>
      <rating>4.0</rating>
      <image source="restaurant1_image"/>
    </restaurant>
    <restaurant id="restaurant2">···
  </restaurants>
</catalogue>

```

А цялостното съдържание на проекта като файлове и разположението им изглежда така:



2.3 Тип и представяне на съдържанието (1-2 стр.)

Съдържанието на проекта е цялостно два типа - текстово и графично.

Текстовото съдържание представлява данните за ресторантите, които компактно сме побрали в единствен .csv файл (преди преработка със скрипта), а иначе е директно закодирано като съдържание на елементите във входния .xml файл (Parsed Character DATA (PCDATA)). Това са всички данни освен изображението на ресторанта.

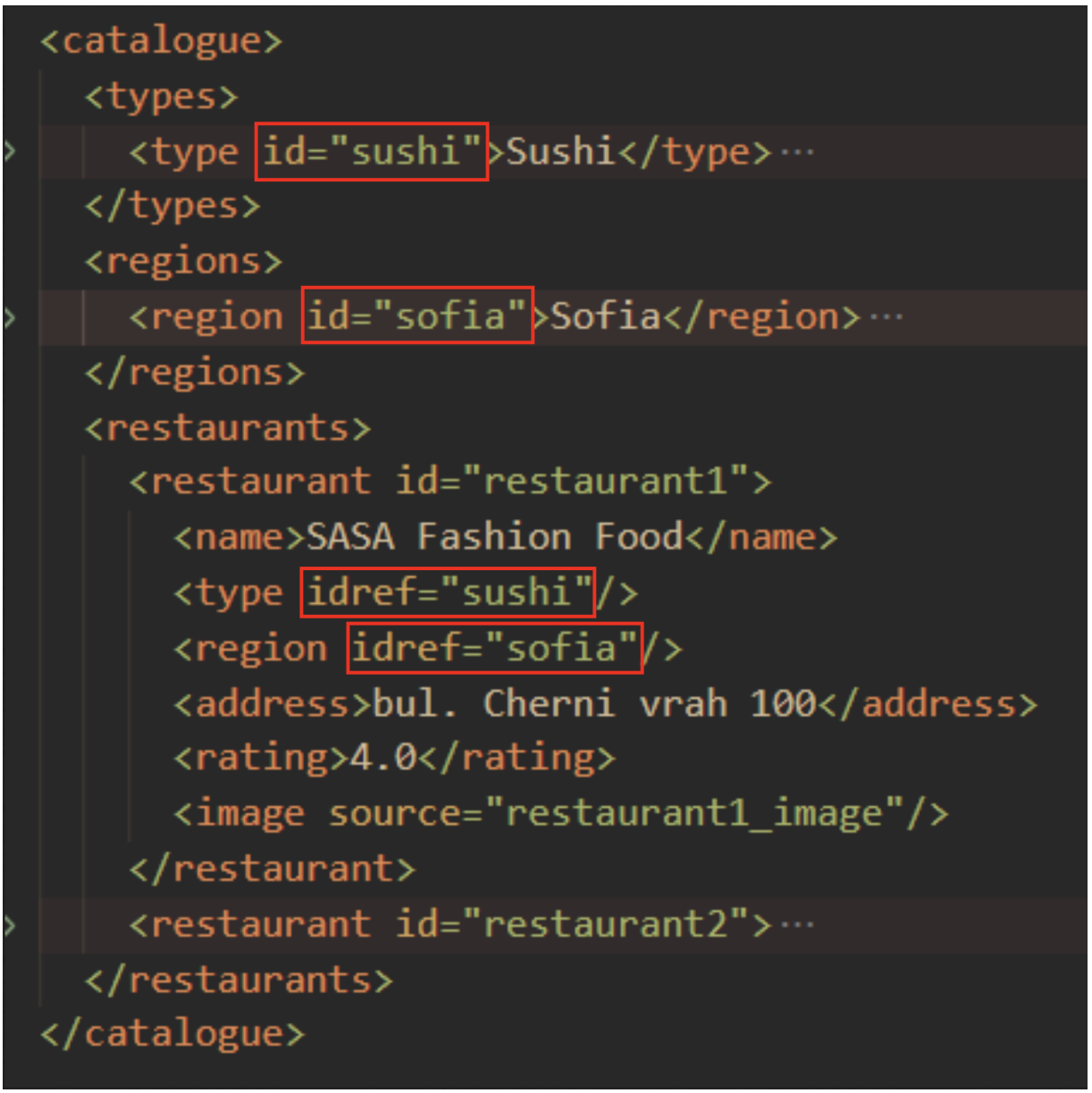
Графичното съдържание са снимките за ресторантите. Това са `.jpeg` файлове, разположени в папката `./images/restaurants/` относително спрямо разположението на входния .xml файл. Всеки ресторант има по една снимка, именувани идиоматично с restaurantX.jpg за X от 1 до броя ресторанти (в случая 7). Снимките са със сравнително малък размер, всяка от тях е в интервал от 50 до 130 KB. Източник на изображенията е сайтът: https://zavedenia-sofia.com/restaurants.html

Изображенията са енкодирани посредством частни външни XML единици (entities) за не-XML съдържание (т.е. unparsed, бинарно). Разбира се, техния тип сме оказали с предварително дефинирана вътрешна частна нотация JPEG.

3 Дизайн (4-5 стр.)

Самият xml документ

Структурата на данните в xml документа бе описано пò в детайли в точка 2.2, но тук ще отбележим няколко ключови момента по заданието. Както казахме, типовете и регионите са представени като отделни множества от възможни стойности, всяка от които има свой уникален идентификатор (атрибут id). След това, в самите restaurant елементи, съответния тип и регион за всеки ресторант сме моделирали като елементи, реферирaщи съответно елементите от първоначално въведеното крайно множество от типове/региони. Общо взето сме ползвали id/idref подход.



Друг ключов момент от дизайна на съдържанието е въвеждането на binary data в документа (изображението за всеки ресторант) като отделно entity, както е описано в 2.3:

Важно е да отбележим също, че единиците с не-XML съдържание могат да се реферират само от атрибут на елемент, понеже парсърът не може да ги обработи.

Валидация

Валидираме чрез XML schema, намираща се във файла restaurants.xsd, реферирана с пространството от имена по подразбиране без префикс от root елемент в .xml файла:

```
<catalogue
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="restaurants.xsd">
```

Самата валидация започва като дефинира руут елемент с комплексен тип “catalogueType”, в чийто обхват дефинираме key/keyref елементи за валидиране на връзките ресторант <-> тип и ресторант <-> регион:

```
<xs:element name="catalogue" type="catalogueType">

  <!-- possible types are stored within the @id attributes
    of the elements types/type-->
  <xs:key name="typeKey">
    <xs:selector xpath="types/type"/>
    <xs:field xpath="@id"/>
  </xs:key>
  <!-- we expect the type element within each restaurant to refer
  to a valid type id -->
  <xs:keyref name="typeKeyref" refer="typeKey">
    <xs:selector xpath="restaurants/restaurant/type"/>
    <xs:field xpath="@idref"/>
  </xs:keyref>

  <xs:key name="regionKey">
    <xs:selector xpath="regions/region"/>
    <xs:field xpath="@id"/>
  </xs:key>
  <xs:keyref name="regionKeyRef" refer="regionKey">
    <xs:selector xpath="restaurants/restaurant/region"/>
    <xs:field xpath="@idref"/>
  </xs:keyref>
</xs:element>
```

Следва дефиниция на самия catalogueType, в който очакваме три елемента стриктно в този ред: types, regions, restaurants:

```
<xs:complexType name="catalogueType">
  <xs:sequence>
    <xs:element ref="types"/>
    <xs:element ref="regions"/>
    <xs:element ref="restaurants"/>
  </xs:sequence>
</xs:complexType>
```

Следва дефиницията на тези три елемента, като първите два (types и regions) са доста подобни. Ще поясним за types: в него очакваме неограничено множество от type елементи, всеки от които има точно един атрибут id от текстов тип и текстово съдържание. Аналогично, при regions очакването е за списък от region елементи със същите свойства. Общото между елементите сме абстрахирали в комплексен тип с името `idAndPCDataOnly`:

```
<xs:element name="types">
  <xs:complexType>
    <xs:sequence>
      <xs:element
        name="type" minOccurs="1" maxOccurs="unbounded"
        type="idAndPCDataOnly"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="regions">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="region" minOccurs="1" maxOccurs="unbounded"
      type="idAndPCDataOnly"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="idAndPCDataOnly">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Остава дефиницията на restaurants елемента като комплексен тип, който на свой ред очаква поредица от поне един ресторант. Елементът ресторант сме изнесли в отделен комплексен тип с името `restaurantType`, в който вече дефинираме конкретната последователност от елементи, които очакваме:

- name от тип произволен текст (xs:string)

- type от тип idRefOnlyType (отделен тип, изискващ празен елемент с единствен атрибут idref)
- region от тип idRefOnlyType
- address от тип произволен текст (xs:string)
- image от тип, съдържащ единствен атрибут source

```
<xs:element name="restaurants">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="restaurant" type="restaurantType"
        minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="restaurantType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="type" type="idRefOnlyType"/>
    <xs:element name="region" type="idRefOnlyType"/>
    <xs:element name="address" type="xs:string"/>
    <xs:element name="rating" type="xs:float"/>
    <xs:element name="image">
      <xs:complexType>
        <xs:attribute name="source" type="xs:ENTITY" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="idRefOnlyType">
  <xs:attribute name="idref" type="xs:string" use="required"/>
</xs:complexType>
```

Трансформация до HTML

Трансформацията извършваме посредством XSLT stylesheet, който генерира финален HTML документ. В него съдържанието е представено като html таблица, с реферирan CSS sheet (повече за него в следващата точка) и един JS скрипт, намиращ се в js/home.js, който позволява динамично пресортиране на таблицата по избрана от потребителя нова колона.

Три от тях са свързани със логиката за сортиране:

- sortOn задава ключа, по който ще сортираме (име на елемент от ресторант)
- sortDataType задава типа на данните, по които ще сортираме (в нашето решение, това е text или number)
- sortOrder задава реда на подредбата (ascending / descending)

Четвъртият параметър е булев флаг, който указва дали това е първото извършване на трансформацията. Този параметър е нужен поради решението, което сме измислили, за динамично пренареждане на съдържанието във финалния документ. Цялостната идея е такава, че на първо зареждане съдържанието е статично, но в самия html документ реферираме JS файл, който позволява при кликане на произволен хедър от таблицата да презаредим отново само таблицата със съдържание с новата сортировка (подадени новите параметри към XSLT). Повече по темата ще обобщим накрая, сега нека се насочим към самата трансформация.

На първо време дефинираме XSL ключ с цел да индексираме всички елементи с наличен такъв по техния id атрибут. Следва шаблон, обработващ корена на входния документ, като:

- ако това е първа трансформация, се извиква именувания шаблон “generate_entire_html”, който генерира цял HTML документ
- ако това не е първа трансформация, се извиква именувания шаблон “generate_sorted_table”, който генерира само <table> елемента със съдържание

```
5   <xsl:param name="firstLoad">true</xsl:param>
6   <xsl:param name="sortOn">rating</xsl:param>
7   <xsl:param name="sortDataType">number</xsl:param>
8   <xsl:param name="sortOrder">descending</xsl:param>
9
10  <!-- define a key to index all elements by their id -->
11  <xsl:key name="elementById" match="*" use="@id" />
12
13  <!-- start processing of root -->
14  <xsl:template match="/">
15    <xsl:choose>
16      <!-- if that's the first load of the document,
17        then do the full document rendering -->
18      <xsl:when test="$firstLoad = 'true'">
19        <xsl:call-template name="generate_entire_html"/>
20      </xsl:when>
21      <!-- otherwise, if that's a reload with new sorting,
22        regenerate the table of restaurant contents only -->
23      <xsl:otherwise>
24        <xsl:call-template name="generate_sorted_table"/>
25      </xsl:otherwise>
26    </xsl:choose>
27  </xsl:template>
28
29  <!-- the template to generate the entire html document -->
30  <xsl:template name="generate_entire_html">
```

Ще опишем първо накратко случая с първото извикване, в който генерираме целия документ чрез “generate_entire_html” шаблона. Това е по-скоро boilerplate, но няколкото важни неща са:

- референцията към файла style.css, съдържащ CSS stylesheet-а за документа
- зареждаме JQuery, което ползваме допълнително за по-удобно селектиране от HTML DOM-а
- зареждаме настоящите параметри за сортиране както идват от XSLT скрипта в глобални JS променливи
- зареждаме скрипта js/home.js за динамично сортиране. Няма да навлизаме в детайли по него, тъй като е допълнителна функционалност.

Следва по-съществения случай: генерирането на <table> елемент с подредените данни.

Първо започваме с извикването на шаблон, генериращ заглавния ред на таблицата. Той е достатъчно интересна тема сам по себе си, затова ще го разгледаме в детайли по-долу. След това обхождаме всички ресторанти чрез xsl:for-each, като за всеки ресторант ще генерираме по един ред за таблицата. Именно тук, в първия дъщерен елемент на xsl:for-each, се случва и сортирането чрез xsl:sort, като тук идва параметъра sortOn. Семантично предвид структурата на входния XML файл искаме да можем да сортираме по един от два начина:

- ако $sortOn е елемент, който няма idref атрибут, сортираме по неговото съдържание;
- ако $sortOn е елемент с idref (при нас това е или type, или region), то искаме да сортираме по съдържанието на реферирания елемент, който извличаме именно с въведения в началото на трансформационния скрипт elementById ключ.

Семантиката на останалите два параметъра $sortOrder и $sortDataType е очевидна - предназначени са именно за употреба като съответните атрибути на xsl:sort елемента.

Следва и генерирането на самите клетки от таблицата, като това правим чрез прилагане на темплейти върху дървото с корен настоящия <restaurant> елемент.

- Тук е важно да отбележим и параметъра mode. Кодът работи и без него, но сме го сложили, за да обособим прилагането на шаблоните тук като изрична стъпка с оглед на евентуално бъдещо разширение на трансформацията - за да се избегнат конфликти.

Кодът за описаното дотук изглежда така:

```
<!-- a template to generate a sorted HTML table element -->
<xsl:template name="generate_sorted_table">
  <table>
    <!-- first generate the header row -->
    <xsl:call-template name="generate_header_row"/>

    <!-- then iterate over the restaurant elements from the XML data hierarchy -->
    <xsl:for-each select="catalogue/restaurants/restaurant">
      <!-- and sort them in either of two ways:
        - if the $sortOn element has no @idref attribute, then its content
        - if the $sortOn element has an @idref, then by the content of the element it refers to
      -->
      <xsl:sort select="*[name(.)=$sortOn and not(@idref)] | key('elementById', *[name(.)=$sortOn]/@idref)"
                order="{$sortOrder}"
                data-type="{$sortDataType}" />
      <tr>
        <!-- apply the necessary template from this level of the tree in order to
          generate the respective <td> element for each data component (chld of <restaurant>)

          This also works without the specified mode, but we put it there explicitly
          to specify this particular step of the transformation
          with regard to future extensibility of this xsd file
        -->
        <xsl:apply-templates select="*" mode="generate-td"/>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>
```

Нека сега преминем към шаблоните, които генерират отделните <td> елементи. Семантично при нас това са три случая (подредени от най-конкретен шаблон към най-общ):

- ако елементът е конкретно <image>, то използваме XPath функцията unparsed-entity-uri, за да извлечем идентификатора от самото entity, която се реферира от @source атрибутът на елемента;
- ако елементът има @idref атрибут, то искаме да извлечем съдържанието от реферирания елемент (отново, това е потребно за type и region);
- във всички други случаи просто копираме съдържанието на елемента.

```
<!-- if the element is an image element, then we want an <img> tag -->
<xsl:template match="image" mode="generate-td">
  <td>
    <img src="{unparsed-entity-uri(@source)}" alt="{../name}" />
  </td>
</xsl:template>

<!-- if the element has an idref attribute, then we want to exctract
the content of the referred element -->
<xsl:template match="*[@idref]" mode="generate-td">
  <td>
    <xsl:value-of select="key('elementById', @idref)" />
  </td>
</xsl:template>

<!-- otherwise (least specific), we just copy the content as plain text -->
<xsl:template match="*" mode="generate-td">
  <td>
    <xsl:value-of select="." />
  </td>
</xsl:template>
```

Нека сега обърнем внимание на любимата ни част от трансформацията: шаблонът за генериране на заглавния ред. Тук решихме да навлезем малко по-дълбоко във възможностите на XSLT и финалното решение придоби следния вид:

Започваме, като в локална променлива “header_keys” дефинираме в отделно редицата от заглавните колони, като за всяка сме указали еднократно тук дали искаме по нея да може да се сортира или не, и то като явни XML елементи. След това използваме функцията ext:node-set, заредена от разширение на схемата ни: `xmlns:ext="http://exslt.org/common"`, за да трансформираме чистия текст до node-set, по който ще итерираме с xsl:for-each. На всяка стъпка в итерацията:

- дефинираме локална за тялото на цикъла променлива displayValue, в която просто взимаме стойността на текущия key елемент и първата и буква заменяме с главна;
- проверяваме дали по key може да се сортира
    - ако да, то проверяваме дали настоящия глобален параметър sortOn съпвада с този ключ, за да визуализираме текущата сортировка със сладки стрелкички в съответната посока (това се случва с отделен малък шаблон - “show_sorting_caret”).
      След това като съдържание на съответната заглавна клетка в реда генерираме линк / котва / <a> елемент, който при натискане извиква JS функция-та newSort с параметър ключа за сортиране от съответната колона. По-детайлен преглед на тази функция можем да направим при желание.

- ако не, то просто като съдържание на съответната заглавна клетка в реда извеждаме displayValue като чист текст.

```
<!-- the template to generate the header row -->
<xsl:template name="generate_header_row">
  <!-- we define a custom set of header keys
    marking for each header its corresponding element from the XML <restaurant>'s children
    and whether this column should be sortable or not
  -->
  <xsl:variable name="header_keys">
    <key sortable="true">name</key>
    <key sortable="true">type</key>
    <key sortable="true">region</key>
    <key sortable="false">address</key>
    <key sortable="true">rating</key>
    <key sortable="false">image</key>
  </xsl:variable>

  <tr>
    <!-- then we iterate over all header keys -->
    <xsl:for-each select="ext:node-set($header_keys)/key">
      <!-- map the first character to upper case as display value -->
      <xsl:variable name="displayValue"
        select="concat(translate(substring(., 1, 1), 'abcdefghijklmnopqrstuvwxyz', 'ABCDEFG
      <th>
        <xsl:choose>
          <!-- if this header should be sortable -->
          <xsl:when test="@sortable='true'">
            <!-- render a cute graphical caret to visualize whether
            this is the current header we're sorting by and in which order -->
            <xsl:variable name="caret">
              <xsl:call-template name="show_sorting_caret">
                <xsl:with-param name="header_key" select="." />
              </xsl:call-template>
            </xsl:variable>
```

```
            <!-- and visualize the header displayValue with surrounding carets if necessary -->
            <xsl:value-of select="$caret" />
            <!-- also, for sortable columns, we wrap the displayValue in a link to a
              javascript function to sort by it dynamically on client demand
              setting the necessary html attributes for the js function to work-->
            <a sort-on="{.}" href="javascript:newSort('{.}')">
              <xsl:value-of select="$displayValue" />
            </a>
            <xsl:value-of select="$caret" />

          </xsl:when>
          <!-- otherwise, if this header is not sortable, we simply display as plain text -->
          <xsl:otherwise>
            <xsl:value-of select="$displayValue" />
          </xsl:otherwise>
        </xsl:choose>
      </th>
    </xsl:for-each>
  </tr>
</xsl:template>
```

И отделно, малкия шаблон за генериране на стрелкичките, “short_sorting_caret”:

```
<!-- a shorthand template to generate the carets on sorted columns
  if necessary -->
<xsl:template name="show_sorting_caret">
  <xsl:param name="header_key"/>
  <xsl:if test="$sortOn = $header_key">
    <xsl:if test="$sortOrder = 'ascending'">&#x25B4;</xsl:if>
    <xsl:if test="$sortOrder = 'descending'">&#x25BE;</xsl:if>
  </xsl:if>
</xsl:template>
```

4 Тестване (2-3 стр.)

Голяма част от тестването се случваше по време на разработката в съответната интегрирана среда за разработка (IDE): VSCode (Давид) и Vim (Павел), със съответните приставки:

- XML Language server за проверка дали XML документът е добре структуриран, с валиден синтаксис, а и като цяло помага при самото му набиране и модификации;
- XSD Validator, за проверка на валидността на документа спрямо реферираната XSD схема в реално време. Поради недостиг във функционалността на приставките обаче все пак основно се придържахме към редовно тестване в посочения в началото на документа онлайн валидатор.

Самият краен резултат от трансформацията и въобще цялостната визуализация на крайния HTML документ сме тествали по два начина:

- директно отваряне на .xml файла в браузър, което обаче изисква модификация в настройките за сигурност по подразбиране поради директния достъп до локални файлове;
- чрез HTTP сървър от Python, удобно и бързо стартиран с терминалната команда:
  `py -m http.server 8080 --directory ./XML/project`

Браузърите, с които сме тествали решението, са Chrome, Firefox, и Edge, като единствено във FireFox имахме проблем с функцията "`unparsed-entity-uri()`", която не се поддържа от вградения трансформатор. Резултатът навсякъде беше консистентен с показаното на снимката в точка 2.1.

5 Заключение и възможно бъдещо развитие (1-2 стр.)

В заключение смятаме, че се получи едно издържано решение според всички установени добри практики за работата с указаните XML технологии. Основно поучително бяха прозренията относно широката функционалност на XSLT, както и валидационните похвати DTD vs. XSD. Основни предимства на избраните от нас XSLT и XSD е богатия набор от възможности за въвеждане на стриктни изисквания (валидация) на доста свободния формат XML, както и многообразните начини за трансформирането му до други формати. За сметка на това обаче като основен недостатък и на двете технологии може да се подчертае утежненият им синтаксис и сравнително трудното свикване с тях / навлизането в детайлите им. Добре е, все пак, че има доста ресурси (литература и примери) в интернет.

Тук е моментът да споменем и две алтернативи на моменти от сегашното ни решение, на които отделихме доста време, но в крайна сметка бяха неуспешни идеи:

- Искахме да поддържаме изцяло динамично сортиране, като идеята ни първоначално беше всичко да се зарежда от входна точка външен .html файл, служещ като статичен темплейт, а само <table> елементът да се генерира от XSLT. Затова обаче трябваше да минем през XSLTParser класа от JavaScript, който не поддържа функцията "`unparsed-entity-uri()`", и реално тук набиващ се на очи недостатък беше, че графичните данни (изображенията) не се показваха изобщо. Това все още се забелязва при динамичното ни сортиране.
- При самото (статично) сортиране в XSLT скрипта имахме идеята да преобразуваме XML документа на две стъпки, като:
  - Първо заменим всички дъщерни на restaurant елементи type и region директно със съдържанието на реферираните от тях;
  - След това направим чиста сортировка по вече получените съдържания на елементите;

За целта искахме да “композираме” два отделни шаблона (първо заменяне, после останалата обработка), което се оказа доста трудно със средствата на XSLT. Но това има общо с цялостната философия на трансформатора.

След сблъсъка с основните недостатъци на използваните технологии, със сигурност можем да посочим някои възможни алтернативи:

- JSON вместо XML като по-компактен и по-лесно четим формат с не по-малко поддръжка и популярност;
- Python вместо XSLT като по-общоприложим език за работа с данни в произволни формати.

Като цяло посока за бъдещо развитие със сигурност би била добавянето на още разнообразни функционалности (като филтриране, групиране по регион / тип, добавяне на отделна страница с повече информация за всеки ресторант и т.н.), вероятно по-силната намеса на JavaScript за динамизация на потребителското изживяване, както и по-добра стилизация.

6 Разпределение на работата

Гледали сме да съобразим равномерно разпределение на работата, като трябва да се има предвид, че все пак през повечето време екипът е бил в редовна комуникация за обсъждане на взетите решения. Разпределението на задачите в общи линии изглежда така:

- съставяне на структурата на входния XML файл - съвместно;
- извличане на данни - Павел;
- помощен скрипт за преход от .csv към установения .xml формат - Давид;
- изготвяне на XSD валидационна схема - основно Давид, съвместни модификации;
- изготвяне на XSLT трансформация - основно Павел, съвместни модификации;
- допълнителен JS скрипт за динамично сортиране - Павел;
- CSS stylesheet - Павел;
- финално почистване, оптимизиране и крайна реитерация на детайли от решението - Давид;
- изготвяне на документация - съвместно.

Разбира се, през цялото време екипът се стиковаше редовно и работеше по паралелни копия на проекта, за да поддържаме синхронизацията на ниво.

7 Използвани литературни източници и Уеб сайтове

1. Различни секции от w3schools на тема XML: https://www.w3schools.com/xml/default.asp
2. O’Reilly’s XSLT book: https://www.oreilly.com/library/view/xslt/0596000537/
3. O’Reilly’s XML Schema book: https://www.oreilly.com/library/view/xml-schema/0596002521/
4. Многобройни теми от Stack Overflow: https://stackoverflow.com/

5. GraphViz: https://graphviz.org/
6. Лекционните материали от курса, както и материалите от занятията на семинар