Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

2D vizualizace výsledků hierarchicky strukturovaných testů

Plzeň 2022 Hana Hrkalová

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Hana HRKALOVÁ**
Osobní číslo:	**A19B0064P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**2D vizualizace výsledků hierarchicky strukturovaných testů**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se se systémem SquashTM a jeho výstupními formáty. Dále se seznamte se způsoby zobrazování výsledků testů v projektu TbUIS a prozkoumejte i další možnosti vizualizací výsledků testů. Popište možnosti rozbalování a skrývání informací pomocí HTML a JavaScriptu.
2. Navrhněte aplikaci, která umožní ve 2D vizualizovat jak závislosti požadavků získaných exportem ze SquashTM na případech užití, tak i výsledky proběhlých testů na případech užití. Při návrhu klaďte důraz na budoucí pravděpodobnou rozšiřitelnost aplikace i na možnost použití různých vstupních datových formátů.
3. Navrženou aplikaci realizujte jako desktopovou aplikaci v programovacím jazyce Java. Aplikace bude spustitelná jak ve CLI, tak i v GUI režimu. Pro aplikaci vytvořte sadu automatizovaných testů s odpovídajícím pokrytím.
4. S využitím zkušeností z předchozích bodů navrhněte a realizujte další aplikaci, která umožní 2D vizualizaci výsledků z datových zdrojů popisujících závislosti jiných (principiálně obecných) hierarchicky strukturovaných entit. Formáty těchto datových zdrojů navrhněte.
5. Funkčnost vytvořené aplikace z bodu 3) ověřte na úplné datové sadě z předmětu KIV/OKS. Pro aplikaci z bodu 4) připravte nejméně dvě odlišné sady reprezentativních dat, pomocí nichž se prokáže její funkčnost.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Doc. Ing. Pavel Herout, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **4. října 2021**
Termín odevzdání bakalářské práce: **5. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 14. října 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 5. května 2022

Hana Hrkalová

Poděkování

Tímto bych ráda poděkovala svému vedoucímu bakalářské práce Doc. Ing. Pavlu Heroutovi, Ph.D. za odborné vedení, pomoc a cenné rady při zpracování této práce.

Abstract

This bachelor thesis deals with the visualization of hierarchically structured test results. A test management tool is used in the KIV/OKS course to display the continuity of the results. However, with large amounts of data, the display of continuity becomes opaque. Therefore, the goal of this work was to create an application that would allow the display of continuity.

In the first part, a Java application was created for use in the KIV/OKS course. This application is fully functional and works with the same file hierarchy.

The second part of the work was to create a visualization library also for arbitrary hierarchically organized input files. To use the library, a representative dataset was created and real data from the TbUIS project was also used.

Abstrakt

Tato bakalářská práce se zabývá vizualizací výsledků hierarchicky strukturovaných testů. V předmětu KIV/OKS je používán nástroj pro řízení testů, který zobrazuje návaznost výsledků. Při větším množství dat se však zobrazení návazností stává nepřehledným. Cílem této práce proto bylo vytvořit aplikaci, která by zobrazování návazností umožňovala.

V první části byla vytvořena aplikace v jazyce Java pro využití v předmětu KIV/OKS. Tato aplikace je plně funkční a pracuje se stejnou hierarchií souborů.

Druhou částí práce bylo vytvoření knihovny vizualizace i pro libovolné hierarchicky organizované vstupní soubory. Pro využití knihovny byla vytvořena reprezentativní sada dat a převzata i reálná data z projektu TbUIS.

Obsah

1 Úvod

Pokrytí aplikace testy je jedna z nejdůležitějších metrik v oblasti testování. Definice tohoto pokrytí má mnoho forem v závislosti na použitých testech a samozřejmě i na ně navazujícího množství různých zobrazení tohoto pokrytí.

V předmětu KIV/OKS je používán nástroj pro řízení testů SquashTM, který umožňuje provázat požadavky (dále RQM) na testovací případy (dále TC). Pokud se do popisu požadavků dají i odkazy na příslušné případy užití (dále UC), je možné provést dopředné trasování, tedy UC -> RQM -> TC i trasování zpětné, tj. TC -> RQM -> UC.

Pokud je ale UC, RQM a TC větší množství (nižší desítky UC, vyšší desítky RQM a stovky TC) stává se zobrazení návazností nepřehledné a je třeba uvažovat o dalším způsobu zobrazení.

Z tohoto důvodu byl vytvořen prototyp, který využívá jazyka HTML 5, kaskádových stylů CSS a jazyka JavaScript. Kombinace těchto nástrojů umožňuje zobrazit závislosti dynamicky, což znamená, že je využito možnosti rozbalování a sbalování hierarchicky závislých částí. Tento přístup dává uživateli možnost agregovaného pohledu a současně i po rozbalení možnost pohledu v libovolném detailu.

Zmíněný existující prototyp má však tři zásadní nedostatky. Za prvé byl připraven přesně na míru pouze jedné z úloh v předmětu KIV/OKS, tj. postrádá obecnost. Za druhé nebyl naprogramován dostatečně robustně, přenositelně a elegantně. Za třetí umožňuje pouze rozbalování řádek, nikoliv sloupců.

První dva zmíněné nedostatky budou řešeny v první části této bakalářské práci.

Dále bylo při analýze a širších diskuzích o celé problematice zjištěno, že uvedený koncept hierarchického sbalování a rozbalování (agregace) zobrazených informací by mohl být dobře využitelný i pro jiné domény, než je testování. Proto se ve druhé části práce řeší právě problematika vizualizace obecných souborů s možností agregace jak řádek, tak i sloupců. Cílem je umožnit co nejjednodušší postup pro zobrazení závislosti jedněch dat na druhých.

2 Analýza

2.1 Analýza knihoven pro práci s formátem JSON

V současnosti existují čtyři nejznámějšími a nejpoužívanější knihovny pro práci s JSON soubory s názvy:

- JSON.simple
- Gson
- Jackson
- JsonP

Přestože všechny knihovny pracují s JSON soubory, nemají všechny stejnou funkčnost, proto je důležité vybrat knihovnu podle práce, kterou s ní chceme vykonávat.

Daniel Bechtel udělal spolu s týmem aktuální letošní výzkum [2], kde testoval různé sady testů a velikosti souborů pro zjištění rychlosti a kompatibility jednotlivých knihoven. Výzkum se opírá o výsledky z roku 2017 a porovnává je.

Testovací sada sestávala z dvou hlavních částí, testování malých souborů (do jednoho KB) a velkých souborů (200 MB)
Podle výsledků k roku 2017:
Nejrychlejší knihovnou, která zpracovala výsledky pro velké soubory je s přehledem Jackson, který je na prvním místě. Těsně za ním je JSON.simple. Nejhůře je na tom právě Gson, jehož zpracování velkých souborů je oproti Jackson knihovně dvojnásobně dlouhé.

	Parsovací rychlost dle knihovny Jackson	**Nárůst času oproti knihovně Jackson**
Jackson	100%	0%
Gson	47%	111%
JSON.simple	98%	1.3%
JSONP	68%	46.9%

Tabulka 2.1: Porovnání JSON knihoven pro velké soubory (200 MB) v roce 2017 [2]

Na druhou stranu u malých souborů měl Gson nejlepší výsledky.

Jackson je velmi konzistentní aplikace pro všechny velikosti souborů, pokud by uživatel měl rozmanitou velikost vstupních souborů s převahou velkých souborů, je tato knihovna nejvhodnějším řešením.
Podle výsledků k roku 2021:
Není velmi překvapivé, že skoro všechny vlastnosti daných knihoven zůstaly stejné. Jediná zásadní změna je u knihovny Gson. Gson se zaměřil na kompatibilitu převážně s jazykem Java. Jak daný jazyk, tak i tato knihovna zvýšily svou rychlost. Výsledky testů z roku 2021 ukazují, že Gson je jasný výherce jak pro soubory velké, tak i malé.

	Parsovací rychlost dle knihovny Gson	**Narůst času oproti knihovně Gson**
Gson	100%	0%
Jackson	72%	39.1%
JSON.simple	95%	5.4%
JSONP	85%	17.1%

Tabulka 2.2: Porovnání JSON knihoven pro velké soubory (200 MB) v roce 2021 [2]

Jelikož práce se soubory v aplikaci pro vizualizaci je v řádu desítek kB pro velikost všech souborů, se kterými knihovna pracuje, rychlost zpracování by nedělala problém u žádné z knihoven. Vzhledem k tomu, že aplikace je převážně psaná v jazyku Java, pro který je Gson nejkompatibilnější, byla pro práci tato knihovna vybrána. Navíc má Gson významné vlastnosti: rozhraní Gson API poskytuje vysokoúrovňovou fasádu, která zjednodušuje běžně používané případy použití, není třeba vytvářet mapování – rozhraní Gson API poskytuje výchozí mapování pro většinu objektů, které mají být serializovány, Gson vytváří čistý a kompaktní výsledek JSON, který je snadno čitelný a také knihovna Gson nevyžaduje kromě JDK žádnou další knihovnu třetích stran.

2.2 Principy rozbalování sloupců

Existuje mnoho různých řešení pro rozšíření tabulky ve smyslu skládání více sloupců, či řádek do jedné. Pro aplikaci sloužící k získání výsledků z aplikace Squash by stačila pouze možnost rozbalování řádek, ovšem pro zobecněnou

verzi je potřeba využít i rozbalovatelnost sloupců. Vzhledem k tomu, že je použita pouze jedna verze aplikace, bude v obou případech využito celkové rozbalování. Pro HTML tabulku je možné využít tři základní principy rozbalování. Čistý HTML kód, HTML s využitím JavaScriptu nebo implementací některé již vytvořené knihovny.

2.2.1 Čistý HTML kód

Možnosti využití čistého HTML kódu nejsou příliš velké, proto se tato varianta používá velmi zřídka. Jedná se o využití pouze jazyka HTML a stylizovacího jazyka CSS. Výhodou je, že není potřeba žádného dalšího jazyku, či knihovny. Oproti výhodám má však velké nedostatky. Hlavním nedostatkem je nestabilita tabulky, rozbalování funguje pouze při podržení myši na části tabulky, kterou chceme rozbalit, pokud kurzor odstraníme, či s tabulkou nijak neinteragujeme, tabulka se vrátí do původního stavu. Využitelnost tohoto rozšíření je prakticky nulová. Příklad využití je například u CSS3 prohlížečů, kdy rozbalení tabulky funguje pouze při podržení tlačítka myši na dané řádce.

```
<table>
    <tr>
        <td>1<div>More info on 1</div></td>
        <td>1<div>More info on 1</div></td>
    </tr>
     <tr>
        <td>1<div>More info on 1</div></td>
        <td>1<div>More info on 1</div></td>
    </tr>
</table>
```

Zdrojový kód 2.1: Použití čistého HTML kódu s využitím CSS

Aby rozbalování fungovalo je nutné doplnit HTML kód o stylizovací CSS jazyk.

```css
td {
   vertical-align:top;
}
td > div {
   height: 0;
   overflow: hidden;
}
td: active > div {
   height: auto;
}
```

Zdrojový kód 2.2: CSS kód

Použití tohoto kódu není kompatibilní s rozbalováním sloupců a celkově má výše zmíněné zásadní nedostatky, proto se tento způsob moc často nevyužívá.

2.2.2 HTML s využitím JavaScriptu

HTML s využitím JavaScriptu plní podobnou funkcionalitu jako využívání některé z možných knihoven. Není nutná znalost nových knihoven, kód si programátor napíše sám a může si jej libovolně přizpůsobit. JavaScript je jazyk přímo určený pro dynamiku a interaktivnost webové stránky, proto je vhodný pro rozšiřitelnost tabulky, například pro rozbalování sloupců. Řešeních existuje více, v podstatě záleží pouze na autorovi, jakým způsobem se HTML s JavaScriptem zkombinuje.

Jednou z možností je pro každou třídu HTML reprezentující sloupec, či třídu zavolat funkci `onclick()` pro kterou je vytvořena funkce v JavaScriptu, která rozbalení sloupců umožní. [12]

```javascript
//Hide all elements with class="containerTab", except for the
//one that matches the clickable grid column
function openTab(tabName)  {
   var i, x;
   x = document.getElementByClassName("containerTab");
   for (i = 0; i < x.length; i++) {
       x[i].style.display = "none";
   }
   document.getElementById(tabName).style.display = "block";
}
```

Zdrojový kód 2.3: Možná funkce JavaScriptu pro rozbalení sloupce [12]

2.2.3 JQuery

Mezistavem mezi využitím JavaScriptu nebo knihovny je JavaScript s využitím JQuery. JQuery je speciální knihovna pracující s JavaScriptem a HTML, která ještě více přizpůsobuje webovou stránku animacím. Je to v podstatě zjednodušení JavaScriptu při práci s dynamickými částmi HTML kódu. Výhodou je větší přehlednost a čitelnost kódu, nevýhodou je nutnost přidání knihovny. JQuery je hojně využívaná, ve vyhledávání různých řešení jsem nalezla JQuery v osmdesáti procentech případů. Jde ovšem spíše již o zmíněnou úpravu pro čitelnost a nových funkcionalit pro tabulku příliš nenabízí. Tato knihovna využívá ukládání řádek jako potomků řádky, která bude rozbalována, další možností je využít ukládání sourozenců. Například pro dobře strukturovaný HTML kód, kde chceme rozbalit pouze jeden sloupec stačí pouze jedna řádka kódu.

```
$ ('td:nth-child(2)').hide();$
```

Zdrojový kód 2.4: Funkce JQuery pro sbalení sloupce

2.2.4 Knihovní funkce

Existuje množství vytvořených speciálních knihovních funkcí, které jsou určeny přímo pro práci s tabulkou, nebo funkcionality pro tabulky obsahují. Mezi nejznámější knihovny patří HandsonTable, která využívá React, Angular a Vue. Obsahuje několik funkcí pro práci s tabulkou, například řazení více sloupců, export do souboru, slučování buněk, změna velikosti řádků/sloupců [5]. Uživateli pak stačí pouze napsat data určená pro tabulku a určit co s nimi chce udělat.

```
const data =[
['','Tesla','Volvo','Toyota','Ford'],
['2019',10,11,12,13]
['2020',20,11,14,13]
['2021',30,15,12,13]
];
const container = document.getElementById('example');
const hot = new HandsonTable(container, {
    data: data, rowHeaders:true, colHeaders:true
});
```

Zdrojový kód 2.5: Ukázka využití knihovny HandsonTable [5]

Druhou nejpoužívanější knihovnou je Bootstrap table, která na rozdíl od HandsonTable využívá pouze react-bootstrap, stylesheets a JavaScript [3]. Pracuje podobně jako knihovna HandsonTable, navíc umí vytvářet vnoření více tabulek do sebe a dokáže pracovat s plochými JSON soubory. Na rozdíl od HandsonTable je zde potřeba více implementace k dosažení požadovaného výsledku [4]. Podobných knihoven existuje spousta, ku příkladu gridjs, slickgrid, ag-grid a jiné [8], všechny jsou ale funkcionalitou podmnožinami těchto dvou nejvyužívanějších. Z mého hlediska žádná z knihoven pro potřebu této aplikace neobsahuje funkcionalitu, která by byla bez knihovny implementačně složitá, proto je využit k rozbalování řádek a sloupců pouze čistý mnou implementovaný JavaScript.

2.3 Analýza projektů využívající aplikaci

2.3.1 SquashTM

Squash Test Managment je aplikace společnosti Squash, která slouží obecně ke správě a monitorování manuálních testů v agilním a tradičním režimu [11]. Squash nabízí hned několik funkcionalit.

- Řízení pokrytí testů a návaznost na požadavky – materializuje závislosti nebo vztahy mezi požadavky pro snadnou údržbu úložiště, synchronizuje objekty, přiřazuje požadavky testovacím objektům a měří jejich pokrytí.
- Formalizace testů a správa sady dat – umožňuje formalizovat testy, automaticky vytvářet strom testovacích případů, vytvořit jeden test a obměňovat jej pomocí datových sad.
- Plánování a správa manuální testovací kampaně – možnost stanovení priorit k optimalizaci úsilí při provádění testů.
- Řízení procesu testování – kontroluje vynechané testy, konzistenci testovacích případů, identifikuje nikdy neprovedené testy s vysokou důležitostí.
- Tvorba a struktura testovacího aktiva – správa uživatelů a přístupů, umožňuje vytvářet vlastní seznamy.

Aplikace je využívána v předmětu KIV/OKS studenty. Squash Test Managment a Squash API aplikace pro export a import, které slouží k výuce a seznámení se

s vytvářením testovacích scénářů, přípravou specifikace sady testovacích požadavků, provedení funkčních testů podle již existujících scénářů a práci se selhanými testy. Ze získaných souborů z aplikace mohou studenti vytvářet tabulku závislostí pro přehlednost a získání povědomí o rozsahu testů a jejich úspěšnosti.

Rozdělení SquashTM

Hlavní jednotkou je testovací projekt, tedy celá oblast, kterou budeme testovat. Testovací projekt můžeme rozdělit na tři části: přípravnou, realizační a manažerskou. Manažerská část je volitelná a pro tento projekt nevyužitelná, pomocí dalších dvou vytváříme zásadní formáty, které jsou využitelné v dalších krocích.

Pro tuto práci jsou zásadními formáty požadavky (RQM) a testovací případy (TC).

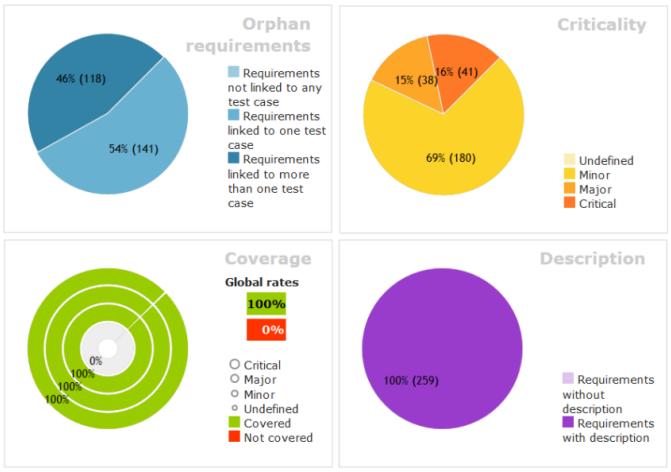
`Požadavek`: Vzniká ze specifikace a rozděluje návrh na co nejmenší přesně specifikované, kontrolovatelné části. Může být svázán s testovacími případy. Z požadavků se vytvářejí skupiny požadavků (RQS), což je seskupení logicky vzájemně souvisejících požadavků, tato seskupení je dále možné zobrazit v tabulce případů užití.

`Testovací případ`: Základní jednotka aplikace SquashTM, je souborem podmínek, za kterých tester určí, zda aplikace, softwarový systém nebo jedna z jeho funkcí funguje tak, jak bylo původně zamýšleno. Vytváření seskupení (TS), které využíváme obdobně jako seskupení požadavků. TC obsahuje několik typů informací: důležitost, stav (rozpracovanost), existence provázání TC na RQM a existence jednotlivých kroků TC [14].

Každý požadavek by měl být propojen jedním či více testovacími případy, což nám aplikace umožňuje, poté je možná zobrazit návaznost těchto jednotek ve vizualizaci pokrytí.

Výstupy SquashTM

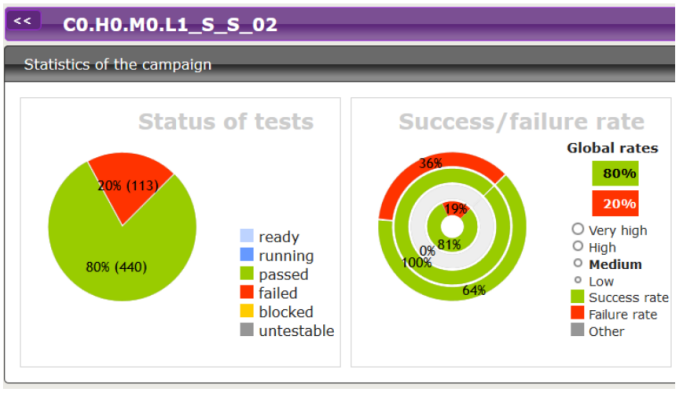
Pracovní prostory ve SquashTM: Requirements a Test case mají možnost importovat a exportovat soubory. Importovat požadavky, či testovací případy můžeme pomocí API aplikace, což je plugin pro SquashTM sloužící k importu/exportu do aplikace. Po importu je možné pomocí aplikace zprostředkovat vizualizaci přehledů pomocí kruhových grafů, zobrazených souhrně v Dashboard. Existují možná nastavení pro zobrazení více druhů dat, defaultně nalezneme statistiku důležitostí jednotlivých požadavků, potomků, či obsazení popisků, nebo pokrytí testů.



Obrázek 2.1: Různé agregované údaje ve SquashTM [13]

Pro testovací případy můžeme získat informaci o asociaci s požadavky, postupu práce, důležitost jednotlivých případů a rozdělení kroků postupu.

Hlavním výstupním formátem, který získáme z aplikace SquashTM a dále použijeme ve vizualizaci dat je CSV soubor, který obsahuje data potřebná k vizualizaci. Squash umožňuje export jednotlivých CSV souborů pouze po kampaních, nikoliv jako celistvý projekt. Kampaň je zásadní částí pro provedení testovacích případů, z čehož jsou poté získány výsledky. Pro každou kampaň existuje také výstupní Dashboard, který graficky zobrazuje její vlastnosti. Není



Obrázek 2.2: Zobrazení výsledků kampaně z aplikace SquashTM [13]

nutné do exekučního plánu zahrnout vždy všechny existující TC – dle stavu projektu jsou vybírány ty, které jsou momentálně vhodné. Je možné získat výsledný kompletní soubor s obsahem všech testovacích případů spojením jednotlivých exportovaných kampaní.

2.3.2 TbUIS

TbUIS, tedy Testbed University Information System, je projekt tvořený několika softwarovými produkty, sloužící jako řešení, které lze využít pro řízené experimenty hodnotící nově vyvinuté testovací metody a přístupy. Projekt vznikl za předpokladu, že nově vytvořené testovací metody jsou obvykle vyhodnocovány na triviálních a velmi jednoduchých aplikacích, s kterými se v reálném světě příliš nesetkají. Hlavní myšlenka tohoto projektu je tedy že již existuje jedna bezchybná aplikace, která může sloužit jako etalon, tedy jakýsi standard sloužící k realizaci a uchovávání. Paralelně s touto aplikací je vytvořeno několik klonů s rozdílnými injektovanými defekty, které by mohly s nějakou pravděpodobností nastat. Projekt se dá obecně rozdělit na čtyři hlavní části:

- UIS – aplikace, z univerzitního prostředí, připravena na funkční testování
- Testovací sada – rozdělena na testy jednotkové, nezávislé funkční a akceptační testy
- ErrorSeeder – sofistikovaný nástroj pro vkládání defektů do zdrojového kódu
- Defekt-klony-UIS – klony s injektovanými defekty

Nejdůležitější částí pro vizualizaci výsledků hierarchicky strukturovaných testů jsou výsledky testů. Ty aplikace přebírá a zobrazuje analýzu struktury pokrytí aplikace testy. Současně také zobrazuje výsledky proběhlých testů, což umožní různé pohledy na testovaný systém.

Způsoby zobrazování výsledků testů

Pro projekt TbUIS můžeme rozdělit testy do tří hlavních kategorií.

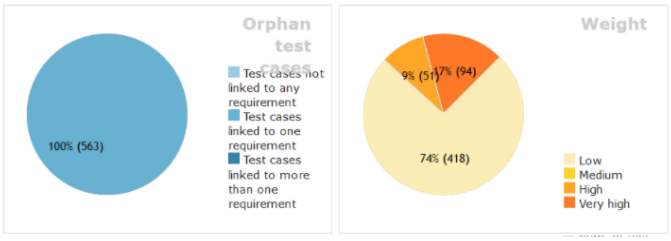
- Jednotkové testy (JUnit) – Jednotkové testy jsou implementovány ve frameworku JUnit. Testy byly vytvořeny s vývojem aplikace UIS a jsou aktualizovány s vývojem nových verzí UIS.
- Funkční testy – funkční testy, které simulují testy uživatelů přistupujících k uživatelskému rozhraní systému. Tyto testy jsou napsány v jazyce Java pomocí rozhraní Selenium Web Driver API.
- Akceptační testy – rozsáhlé scénářové testy pomocí Robot Framework

Funkční testy je možné rozdělit na dvě hlavní části, na samostatně použitelnou první část s názvem Support. Jednu z jejích důležitých dílčích částí tvoří modul Oracle, který představuje paralelní byznys logiku a který využit při testech pro stanovení očekávaných výsledků. Support využívá pro organizaci testů návrhový vzor PageObject a JUnit5 Framework. Při vývoji této části byly současně vytvořeny testy JUnit pro ověření její funkčnosti. Vznikla tak robustní, dobře otestovaná knihovna, která se pak používá nejen v popsaných funkčních testech, ale i ve zcela jiném typu akceptačních testů.

Knihovna Support je využívána v současných experimentech s UIS a může ji využít i kdokoliv další, kdo bude aplikaci UIS používat pro svůj vlastní výzkum.

Druhou část tvoří samotné funkční testy. Psaní zdrojových kódů testů předcházela důkladná systematická analýza, která byla završena přípravou samostatných sad testů a testovacích případů. Základ analýzy tvořily podrobné popisy případů užití. Případy užití byly popsány sadami požadavků. Lze prokázat, že požadavky pokrývají celou aplikaci UIS. [13] Prokázání pokrytí je obsahem aplikace vizualizace pokrytí, kdy aplikace vytváří závislost případů užití na jednotlivých požadavcích.

Projekt TbUIS využívá dále pro zobrazování výsledků výše zmíněný SquashTM. Konkrétně využívá přehledy a agregované grafy pro organizaci požadavků a testovacích případů, nebo také zobrazení testů všech klonů defektů.



Obrázek 2.3: Přehledy testů pro projekt TbUIS [13]

2.4 Analýza anotací

Anotace slouží k poskytování doplňujících informací o programu. Byly zavedeny, respektive byly k dispozici ve verzi 1.5 Java Development Kit.

Anotace v Javě poskytují více informací o datech přítomných ve struktuře kódu, jsou to data o datech známá také jako metadata [7]. Mají své základní charakteristiky. Anotace začínají znakem "@", nemění činnost zkompilovaného

programu, anotace pomáhají přiřadit metadata k prvkům programu, tj. proměnných instancí, konstruktorům, metodám, třídám a tak dále [1]. Anotace nejsou čistě komentáře, protože mohou změnit způsob, jak kompilátor s programem zachází.

2.4.1 Účely anotací

Anotace v jazyce Java se obvykle používají k následujícím účelům:

- Instrukce překladače
- Pokyny při sestavování
- Pokyny pro běh

Java má tři vestavěné anotace, které můžete použít k zadání instrukcí kompilátoru Javy. Tyto anotace jsou podrobněji vysvětleny dále v tomto textu.

Anotace jazyka Java lze použít v době sestavení, když je sestavován softwarový projekt. Proces sestavování zahrnuje generování zdrojového kódu, kompilaci zdrojového kódu, generování souborů XML (např. deskriptorů nasazení), zabalení zkompilovaného kódu a souborů do souboru JAR atd. Sestavení softwaru se obvykle provádí pomocí automatického nástroje pro sestavení, jako je Apache Ant nebo Apache Maven. Nástroje pro sestavení mohou skenovat kód jazyka Java a hledat v něm specifické anotace a na základě těchto anotací generovat zdrojový kód nebo jiné soubory.

Pro jazyk Java existuje několik vestavěných typů anotací. Vestavěné anotace Javy používané v kódu Javy

- `@Override`
- `@SuppressWarnings`
- `@Deprecated`

Vestavěné meta anotace Java použité v jiných anotacích

- `@Target`
- `@Retention`
- `@Inherited`
- `@Documented`

Pro pochopení principu a využití anotací máme například anotaci `@Override`, která zajišťuje, že metoda podtřídy přebírá metodu nadřazené třídy. Pokud tomu tak není, dojde k chybě při kompilaci. Při přepisování názvu metody lze jednoduše udělat chybu proto je lepší označit anotaci `@Override`, která poskytuje jistotu, že je metoda přepsána.

Principiálně lze anotace použít stejným způsobem, bez omezení na anotace vestavěné.

3 Implementace

3.1 Původní aplikace

Již dříve byl vytvořen prototyp aplikace pro vizualizaci dat JSON souborů získaných ze SquashTM. Tato aplikace nebyla plně zpracována, pouze splňovala základní funkčnosti pro použití v předmětu OKS.

3.1.1 Analýza aplikace

První myšlenkou bylo využít prototyp této aplikace, upravit a doplnit jej do finální podoby, která by splňovala požadavky jak pro předmět OKS, tak také pro projekt TbUIS a byla by obohacena o řešení pro obecnou hierarchickou strukturu. Aplikace je rozdělena do tří hlavní částí. Zpracování a úprava dat před zahájením parsování objektů pro HTML kód. V druhé části jsou zpracována data, hodnoty jsou ukládány do seznamů, v poslední části je generován samotný HTML kód.

V první části jsou vytvářeny nové soubory nutné pro zobrazení všech dat předmětu OKS. Tyto soubory jsou tvořeny pomocí dat z předaných parametrů. Jedná se o dvě hlavní třídy, obě slouží k vytváření výsledků. První třída slouží k vytváření souboru s výsledky jednotlivých testů, jde o závislost testovacích případech na požadavcích, druhá třída se stará o vyváření souboru obsahující výsledky jednotlivých testů dle předaného logovacího souboru. Pro parsování hodnot ze souboru JSON jsou vytvořeny třídy reprezentující jednotlivé objekty v předaném souboru.

Pro parsování je využita knihovna Gson. Po vytvoření entit jednotlivých tříd reprezentující JSON objekty jsou postupně načítány a ukládány data jednotlivých řádek, sloupců a jejich číselných hodnot. Pro parsování jednotlivých hodnot kód využívá definitních hodnot specifických pro JSON soubory z předmětu OKS, z toho důvodu by nešla zařídit žádná obecnost a aplikace by měla pouze jedno využití.

3.1.2 Využité technologie

Aplikace využívá jazyky Java, HTML, CSS a JavaScript. HTML a CSS slouží pouze pro zobrazení tabulky a její stylizaci. CSS styl je upraven do konkrétních sekcí pojmenovaných dle OKS definicí a pevně spjat se hodnotami tabulky pro výsledky SquashTM aplikace. Celý algoritmus běží v jazyce Java s pomocí Gson

knihovny.

Javascript slouží k správnému fungování tabulky (rozbalování sloupců, filtrace...). Využívá metody `toggleClass()`, tato metoda přepíná mezi přidáním a odebráním jednoho nebo více názvů tříd z vybraných prvků. Tato metoda kontroluje každý prvek na přítomnost zadaných názvů tříd. Názvy tříd jsou přidány, pokud chybí, a odebrány, pokud jsou již nastaveny.

3.1.3 Využité části

Nakonec jsem tuto aplikaci nevyužila a nepokračovala v úpravách a rozšířeních, jak bylo prvotní myšlenkou. Aplikace pro 2D vizualizaci hierarchicky strukturovaných testů byla psaná zcela od začátku. Z původní aplikace pouze byly využity myšlenky, či princip fungování některých částí pro práci se soubory. Konkrétně byl převzat princip třídy `ResultTc2Rqm`, která slouží k parsování souboru požadavků v závislosti na testech užití. Z třídy `Result2TC` byla využita celá funkcionalita, nasazena na mou aplikaci.

3.2 Rozvržení aplikace

3.2.1 Rozdělení částí

Dle zadání je určeno udělat dvě funkcionálně podobné aplikace. Proto jsem při rozvržení zvolila formu připravit aplikaci jednu a případně ji rozdělit na dvě samostatné. Aplikaci, která umožní ve 2D vizualizovat jak závislosti požadavků získaných exportem ze SquashTM na případech užití, tak i výsledky proběhlých testů na případech užití a aplikaci která umožní vizualizaci výsledků datových zdrojů popisujících závislosti jiných (principiálně obecných) hierarchicky strukturovaných entit.

V první aplikaci jsou zobrazována data, která jsou využita pouze pro účely předmětu OKS a projektu TbUIS, tedy data s pevně danou strukturou. Jak pro TbUIS, tak pro výsledky z aplikace Squash je tedy tabulka naprosto totožná, až na konkrétní hodnoty jednotlivých testů. Struktura i styl, tedy barvy a způsob zobrazení, zůstávají zachovány.

Pro druhou aplikaci mohou být data, která chce uživatel zobrazit, v jistém smyslu značně rozdílná. Zůstává zachována JSON struktura, tedy typ uložení jednotlivých dat, ale pro konkrétní případy můžou být využity různé hierarchické struktury a vnoření, navíc má aplikace pouze jednoduchý CSS styl a je možnost přidání osobního vzhledu tabulky. Výsledné HTML soubory se pak mohou podstatně lišit.

Obecná aplikace má tedy zásadní stejné charakteristiky, hlavní částí je vytvoření tabulky, s tím souvisí konkrétně vytvoření sloupců a řádek. Jelikož jde o hierarchistickou strukturu, tak musíme vzít v potaz i jednotlivé skupiny řádek a sloupců a jejich vlastností. Můžeme tedy rozdělit charakteristiky, které jsou potřeba pro vytvoření tabulky. Tuto část lze rozdělit na zásadní charakteristiky a vedlejší charakteristiky, kosmetické úpravy, či funkcionality navíc.

Typické zásadní charakteristiky vytváří základní kostru tabulky. Jedná se o HTML elementy, například `<tbody>`, `<th>` a další. Další části jsou: `title`, `meta`, `value`, `label`, `depth`, `type` a `text` v kolonkách. Pomocí těchto částí jsme schopni naplnit tabulku daty.

Jako poslední souhrn částí charakterizuje nějaké dodatečné funkcionality, na které je potřeba mít definované základní prvky: funkce po zmáčknutí tlačítka (například myši), link pro hypertextový odkaz hodnot, filter pro filtraci hodnot, s kterým souvisí menu a option. Tedy možnosti filtrace, sumaci řádků, sloupců a například tooltip, tedy informační vyskakovací okno, s kterým se pojí třída tohoto okna a text, který se zde nachází.

3.2.2 Využité technologie

V aplikaci jsou využity čtyři základní programovací jazyky: Java ve verzi 11, JavaScript, HTML a CSS stylizovací jazyk. Dle výše zmíněných okrajově popsaných charakteristik je zřejmé, že aplikace bude využívat jazyk HTML, konkrétně k vytvoření a zobrazení tabulky. Data pro tabulku jsou tvořena pomocí jazyka Java, JavaScript slouží pro ovládání a dynamičnost tabulky. Jazyk CSS slouží k zobrazení tabulky v nějaké stylizované podobě. Jazyk Java je implementačně nejvíce využitý, využívá se k zpracování vstupních souborů na objekty, se kterými je dále možno pracovat, vytváří data připravená pro tvorbu HTML kódu, plní tabulku daty a generuje výstupy.

Jako poslední je využita nástavba Javy a to JavaFX, pomocí které je vytvořená GUI verze aplikace pro uživatele, sloužící pro snazší získání požadovaného výstupu.

3.3 Anotace

K vytvoření obecnosti aplikace slouží Java anotace. Pomocí nich lze využít libovolný hierarchický JSON a vytvořit z něj požadovanou tabulku závislostí.

3.3.1 Princip fungování

Jedná se o způsob získávání jednotlivých hodnot objektů. Je nutné předem označit všechny hodnoty, které budeme chtít zpracovat. Je třeba vytvořit třídu, reprezentující obecný objekt, který chceme získat (například `Factory` třída reprezentuje objekt řádky tabulky). Pokud je třída vytvořená, dalším krokem je potřeba zapsat všechny atributy, které daná třída má a těm korektně přiřadit anotace. Po vytvoření všech potřebných atributů a anotací je vytvořen JSON Parser, kterému jsou předány skutečné parametry: název JSON souboru a výsledná třída, do které se budou ukládat hodnoty z JSON, tedy třída, kterou chceme vygenerovat. Konkrétně pro jeden atribut s danou anotací jsou nalezeny všechny výsledky odpovídající požadované anotaci a ty jsou postupně přetvářeny na generované objekty v souboru třídy.

Chceme vytvořit tabulku, reprezentující továrny (hodnoty řádků) a jejich výrobu v závislosti na dnech (hodnoty sloupců) dle daného JSON souboru.

```
"factory": {
     "name": "Factory 1",
     "productionLineList": [
       {
         "line": 1,
         "name": "linka A",
         "furnitureList": [
           {
             "name": "Coffee Table",
             "amount": 9,
             "day": 2,
             "month": 2,
             "year": 2021
           },
```

Zdrojový kód 3.1: JSON soubor reprezentující továrny

```
public class Factory {

    @Name
    private String name;

    @SerializedName(value = "productionLineList")
    @TableElement(TableType.CHILDREN_LIST)
```

```
    private List<Line> lineList;
}
```

Zdrojový kód 3.2: Třída `Factory` pro parsování JSON souboru 3.1

Třída `Factory` má dva privátní atributy, `name` a `List<Line> lineList`. Atribut `name` je povinný údaj (je třeba využít anotaci `@Name`) slouží jako identifikátor jednotlivých hodnot. Anotace zařizuje zmíněnou identifikaci a přebírá hodnotu z JSON souboru 3.1 `"name"`, která se zobrazuje v řádcích tabulky. Druhý atirbut využívá anotaci `@SerializedName`, která udává programu vyhledávání dle jiného zvoleného řetězce (`lineList`). Dle parametru anotace je v JSON souboru vyhledána hodnota `"productionLineList"` a mapuje se do zmíněného `lineList`. Pokud by se atribut jmenoval identicky vůči JSON souboru (`private List<Line> productionLineList`), tuto anotaci by nebylo třeba využít.

Druhá anotace tohoto atributu značí relaci mezi řádky, konkrétně se přiřazuje seznam linek `List<Line>` jako seznam dětí pro každou hodnotu `Factory`. Třídu line je třeba také implementovat.

```
public class Line {
    @Name
    private String name;

    @TableElement(TableType.TOOLTIP)
    private int line;

    @TableElement(TableType.CHILDREN_LIST)
    private List<Furniture> furnitureList;
```

Zdrojový kód 3.3: Třída `Line` reprezentující děti třídy `Factory`

Tato třída využívá anotaci `@Name` identicky jako třída `Factory`, stejně tak je vytvořen seznam dětí, tentokrát jsou děti reprezentovány třídou `Furniture`. Oproti třídě `Factory` je zde využita `@TableElement(TableType.TOOLTIP)` anotace, která zajišťuje zobrazení textu nápovědy.

```
public class Furniture {
    @Name
    private String name;

    @TableElement(TableType.VALUE)
```

```
    private int amount;

    @Relational(equals = true, targetClass = Day.class)
    private Integer day;

    @Relational(equals = true, targetClass = Month.class)
    private Integer month;

    @Relational(equals = true, targetClass = Year.class)
    private Integer year;
```

Zdrojový kód 3.4: Třída `Furniture` reprezentující děti třídy `Line`

Tato třída reprezentuje také hodnoty řádků, navíc reprezentuje relaci na sloupce dle výsledků. Obsahuje anotaci `@TableElement(TableType.VALUE)`, značící hodnoty, které se budou reprezentovat ve vnitřní části tabulky. Tyto hodnoty musí mít nějakým způsobem udělanou návaznost na sloupce (nyní nevíme, do kterého sloupce by se hodnota měla zařadit), k tomu slouží druhá anotace `@Relational`. Název atributu značí opět hodnotu v JSON souboru. Víme tedy, že pro relaci `month` se ze souboru 3.1 bude ukládat hodnota `"month"`. Parametr relace `targetClass = Month.class` reprezentuje, dle které třídy se budou relace vytvářet. Tato třída opět využívá anotace k vytváření hodnot pro sloupce, principiálně podobné jako třídě `Factory` a `Line`. Anotace je využita třikrát, jelikož má JSON tři hodnoty pro vytvoření relace, stejně tak jsou sloupce reprezentovány třídou `Year`, která má seznam dětí třídy `Month`, která má seznam dětí třídy `Day`. Tímto zajistíme mapování do správného sloupce a dále správné sčítání hodnot dle jednotlivých skupin, pokud tabulka tuto funkci využívá.

3.3.2 Použití anotací

V řešení byly použity anotace, jelikož jsou v celkovém pohledu nejlepším řešením dané problematiky, tedy vytvoření specifických vymezení struktury tabulky, respektive jejího formátu. Alternativním řešením bylo vytvořit popisný JSON soubor, který by obsahoval klíč, značící název dané charakteristiky v tabulce, dle klíče by pak bylo možné mapovat jednotlivé hodnoty na dané charakteristiky (například hodnota řádku v tabulce). Oproti tomuto řešení výsledné řešení anotací zajišťuje jistou kontrolu správnosti, tedy využití korektních typů anotací pro dané hodnoty. Znamená to tedy, že pro princip s popisným JSON souborem je mnohem snaží vytvořit chybná data. Zároveň je využití anotací

následně rychlejší při kompilaci výsledných dat a také rozšířitelnější o nové typy charakteristik v tabulce. Pokud by byla potřeba přidat novou vlastnost tabulky, nebo samotného elementu, stačí vytvořit pouze novou anotaci nebo její vlastnost (metodu).

3.3.3 Příklady anotací

Vytvořené anotace v aplikaci, které slouží ke zmapování JSON souboru do konkrétních objektů souboru třídy. Všechny anotace jsou využity při běhu programu (hodnota `RunTime metaanotate @Target`)

`@Name`

Anotace `Name` je povinnou a jednou z nejdůležitějších anotací. Je použita jako identifikační klíč k rozpoznání jednotlivých hodnot pro uživatele. Hodnoty, které využívají tuto anotaci by měly být unikátní pro celou tabulku, tedy pro každý řádek i sloupec. Anotace `Name` je omezená pouze na využití pro anotování entit jako atributů, znamená to tedy, že ji nelze využít například pro celou třídu, ale pouze pro její atributy.

`@TableElement`

Obsahuje pouze jednu metodu `TableType`, která obsahuje, o jaký typ prvků tabulky se jedná, konkrétně: `Link`, `tooltip`, `value` a `childrenList`. Anotace je opět využitelná pouze pro atributy dané třídy.

`@TableElement(TableType.LINK)`

Anotace `Link` slouží k vytvoření externího linku pro přechod na jiné webové stránky. Nejtypičtější využití je pro odkaz na jednotlivé sloupce (například Případy užití pro Squash Test Management). Pro tabulku závislostí testovacích dat na webové stránce `kiv.oks/prevodnik` z předmětu OKS jsou pro jednotlivé hodnoty sloupců vytvořeny pomocí anotací odkazy na jednotlivé záložky dané internetové stránky, které s daným případem užití souvisí. Anotace `Link` se vždy vytvoří v závislosti na anotaci `Name`, tedy pro uživatele se zobrazí odkaz na jméno dané entity, tedy na části, které byla přiřazena anotace `Name`.

`@TableElement(TableType.TOOLTIP)`

Anotace slouží pouze k vyobrazení textu po podržení kurzoru na určité oblasti. Funkcionalitou této anotace je zobrazit dodatečné informace pro lepší pochopení jednotlivých částí tabulky. Anotaci `Tooltip` je možné využít na jakoukoliv

oblast z JSON souboru, která bude zprostředkována do vyskakovacího textu. V tabulce je opět vázána na hodnoty s anotací `Name`, například pro názvy řádků, nebo jiných hodnot, kterým byla přiřazena tato anotace.

`@TableElement(TableType.VALUE)`

Poskytuje možnost navázat hodnoty do tabulky. Hodnota může být reprezentována jako objekt. Pomocí této anotace je vytvářená část tabulky, ve které jsou uložené data. Hodnoty se rozmisťují podle anotace `Relational`, která by měla být vždy společně s touto anotací obsahem jedné entity. Hodnota může být typu `Object`, tedy instance jakékoliv třídy. Hodnoty stejného typu lze načítat do hromadné sumy, pokud je tato vlastnost využita, typ atributu je omezen na instanci dědící `Number` nebo instanci implementující rozhraní `Summable`.

Rozhraní `Summable` má pouze jednu metodu.

```
public interface Summable<T> {
    T sumUp(T another);
}
```

Zdrojový kód 3.5: Rozhraní Summable

Generický typ `T` značí typ objektu, který bude sčítán, metoda `sumUp()` sečte hodnotu s kterou pracujeme (`this`) s objektem předaným v parametru. Obě sčítané hodnoty musí být stejného typu, tedy celá část tabulky s hodnotami je tvořena objekty pouze jednoho typu.

`@TableElement(TableType.CHILDREN_LIST)`

Vyskytuje se u atributů, kteří jsou potomci daného elementu. Tato anotace může být využívána vícekrát pro danou entitu, tedy označí více kolekcí jejich potomků. Pokud je označeno více kolekcí, všechny jsou uloženy do pole potomků dané entity. `Children_list` je tvořen jako potomek `Collection` rozhraní jazyka Java.

`@Cover, @CoverElement`

Anotace `Cover` je využita, pokud je naše entita v JSON souboru zabalena do obalovacího elementu.

```
[
    {                           // cover
        "element": {            // coverElement
```

```
            ...
        }
    },
    {                                   // cover
        "element": {                    // coverElement
            ...
        }
    }
]
```

Zdrojový kód 3.6: Využití anotací `@Cover` a `@CoverElement` v JSON struktuře

`Cover` anotací jdou anotovat pouze třídy, a ne její atributy.

Naopak `CoverElement` anotací jsou možné anotovat pouze atributy dané třídy. `CoverElement` nemusí být nutně využit, za podmínky kdy třída obsahuje pouze jeden atribut, v tomto případě stačí pro pokrytí anotace `Cover`.

`@Filter`, `@FilterCriteria`

Tato anotace je využívána, pokud je v tabulce potřeba filtrovat hodnoty. Pomocí anotace je označen atribut entity, podle kterého budou vyhledávány všechny hodnoty v tabulce odpovídající danému atributu. Anotace má možnost volného filtrování, pokud není uvedena metoda `filterCriteria()`. Výběr lze omezit definováním této metody. K tomu `FilterCriteria` využívá své dvě metody, `value()` a `template()`.

```
public @interface FilterCriteria {
    String value();
    String template() default "";
}
```

Zdrojový kód 3.7: Anotace `FilterCriteria`, sloužící k mapování hodnot filtru

Metoda `value()` je předdefinovaná hodnota, podle které se bude provádět filtrace. Oproti tomu metoda `template()` definuje formát zobrazení hodnoty, dle které se bude filtrovat.

`@Relational`

Společně s anotací `Value` tvoří nejdůležitější část anotací. Tato anotace zajišťuje správné rozložení hodnot v tabulce a jejich zobrazení. Lze používat pouze na atributy třídy.

```java
public @interface Relational {
    //Regex for separating value from string
    String regex() default "";
    boolean equals() default false;
    Class<?> targetClass();
    boolean tableUnique() default false;
}
```

Zdrojový kód 3.8: Anotace `Relational` sloužící k vytváření závislostí

Může se zdát, že je metoda `equals()` zbytečná, jelikož není zatím plně využívána, ale pro budoucí rozšíření by anotace mohla být doplněna o další metody, například `contains()`, `equalsIgnoreCase()` a jiné metody pro srovnávání hodnot do tabulky. Defaultně je metoda nastavená na hodnotu `false`, po použití anotace by se vždy měla nastavit na hodnotu `true`. Při použití anotace uživatel nastaví hodnotu na `true`, pokud by tak neučinil, vytvoří se prázdná tabulka.

Atribut s touto anotací může být jakéhokoliv typu, hodnoty jsou porovnávány pomocí metody `java.lang.Object.equals()`. Můžou nastat dva základní způsoby porovnání. Porovnání všech hodnot vůči sobě, pokud jsou oba atributy pole, nebo instance `java.util.Collection`, nebo porovnání jedné hodnoty vůči všem prvkům v poli (kolekci). Metoda `targetClass()` představuje třídu, na kterou jsou hodnoty mapovány. Metoda `regex()` slouží k separování stringové hodnoty dle zvoleného regulárního výrazu. Vyfiltrovaná hodnota je dále předávána metodě `equals()`.

```
//Text in JSON
"description": " Day that Coffee Chair was made is 2.9.2021."

//regex to separate the desired value
regex = "\d\d?\\.\d\d?.\d{4}"

result = "2.9.2021"
```

Zdrojový kód 3.9: Využití regulárního výrazu k parsování hodnot z JSON souborů

Metoda `tableUnique()` slouží k identifikaci porovnávaného prvku. Kontroluje, zda je prvek jedinečný skrze celou tabulku, pokud ne, kontroluje hierarchické zanoření a jedinečnost v dané skupině.

3.4 Data Generator

Tento program je hlavní částí práce, obsahuje kompletní algoritmus pro tvorbu tabulky z předaných parametrů. Aplikace přijme upravená data, tedy vytvořené zparsované objekty a vytvoří z nich HTML kód.

3.4.1 Implementace

Aplikace je rozdělena do tří hlavních částí, parsování JSON souborů do objektů, vytvoření dat pro tabulku pomocí anotací a generování HTML souboru.

Parsování JSON

Pro parsování JSON souboru na objekty je využívána knihovna Gson. Základní třídou, kterou lze použít, je `Gson`, kterou stačí vytvořit voláním `new Gson()`. Instance `Gson` neudržuje žádný stav při volání operací JSON. Lze tedy libovolně používat stejný objekt pro více operací serializace a deserializace JSON souboru. Hlavní metodou, která je použita je `fromJson()`.

```
public <T> T fromJson(JsonElement json,
                      java.lang.Class<T> classOfT)
               throws JsonSyntaxException
```

Zdrojový kód 3.10: Obecná Gson funkce pro deserializaci JSON souboru

Tato metoda deserializuje JSON načtený ze zadaného parsovacího stromu do objektu zadaného typu. Typ parametru `T` – typ požadovaného objektu.

Parametry: `json` – kořen parsovacího stromu `JsonElements`, ze kterého má být objekt deserializován, `classOfT` – třída vrací objekt typu `T` z `json`, vrací `null`, pokud je `json` `null`

```
  @Override
    public T parseFile
         (String filename, Class<T> targetType) throws IOException {
    return new Gson().fromJson(Files.newBufferedReader(Paths.get(filename)),
    TypeToken.get(targetType).getType());
    }
}
```

Zdrojový kód 3.11: Metoda využívající Gson funkci pro parsování JSON souboru

Před použitím této metody je nutné mít definované objekty tříd, na které chceme JSON parsovat. Hlavní třídu, na kterou budou hodnoty deserializovány předáváme v druhém parametru metody. V prvním parametru metody je potřeba definovat nějakou metodu pro čtení souboru a cestu k danému JSON. Pro náš kód je volána metoda `parseFile()` třídy `JsonParser<T>`.

```
"factory": {
    "furniture_list": [
      {
        "chair": {
          "id": "ID.01.01",
          "name": "Jídelní židle",
        }
      }
    ],
    "id": "Fac.01",
    "name": "Truhlářství",
    "description": "Výrobna stolů a židlí"
  }
```

Zdrojový kód 3.12: Ukázka parsovaného JSON

Abychom mohli deserializovat JSON v ukázce, je potřeba vytvořit třídu `Factory` s příslušnými hodnotami, které se v JSON souboru vyskytují. Poté můžeme volat metodu `parseFile()`, která vytvoří daný objekt, jako instanci třídy `Factory`. Pokud bychom potřebovali hodnoty při serializaci/deserializaci jednotlivých objektů změnit, můžeme využít anotaci knihovny Gson `@SerializedName`. Anotaci `@SerializedName` lze použít pro serializaci pole s jiným názvem místo skutečného názvu pole. Očekávaný serializovaný název můžeme uvést jako atribut anotace, Gson pak může zajistit čtení nebo zápis pole s uvedeným názvem. Pokud bychom chtěli pro daný JSON soubor uložit seznam nábytku do třídy `Factory.java` můžeme použít anotaci `@SerializedName` ve tvaru:

```
public class Factory {
  private String id;
  private String name;
  private String description;
  @SerializedName(value = "furniture_list", alternate = "list_of_products")
  private List<Furniture> list_of_products;
```

Zdrojový kód 3.13: Využití anotace `@SerializedName`

Generování objektů dle anotací

V této části aplikace je využíván princip anotací. Tedy vytvoření anotací, pomocí kterých získáme hodnoty, ty jsou dále předávány danému algoritmu, který z nich vytváří objekty pro následné generování tabulky. Tyto objekty mají určené vlastnosti definující později tvořený HTML kód. Pro využití výše zmíněného algoritmu je využita třída `AnnotationRelation` implementující rozhraní `IRelational`.

```
public interface IRelational
        <T extends Collection<?>, S extends Collection<?>> {
    Table makeRelation(T rowItemList, S columnElementList) throws
    AppRelationException;
}
```

Zdrojový kód 3.14: Rozhraní `IRelational`

Rozhraní `IRelational` má dva generické parametry `T` a `S`. Oba musí být potomci rozhraní `java.util.Collection`. Oba parametry představují objekty, z kterých budou vytvářeny závislosti a data tabulky. Elementy jednotlivých kolekcí mohou být libovolného typu, pouze musí být správně nadefinovány. Rozhraní obsahuje pouze jednu metodu `makeRelation(<T>, <S>)`. Na začátku této metody neprobíhá žádná validace. Výsledkem metody je třída `Table` představující objekt jako kostru pro následné generování HTML kódu.

Tato třída je typu přepravka, tedy pouze obaluje výsledná data: listy sloupců a řádek. Ty jsou postupně naplňovány v metodě `processTable()`. Pro každý typ anotace jsou získány všechny objekty, které byly dle dané anotace zparsovány. K tomu jsou využity třídy `AnnotationUtils` a `ClassUtils`. Tyto třídy slouží k usnadnění práce s `java.lang.reflect` a napomáhají k vytvoření objektů z anotovaných objektů. Obě třídy jsou finální, nelze z nich tedy dědit, fungují na principu knihovních tříd, používají privátní konstruktor, není možné je využít jako instance, pouze používat jejich statické metody. `Java reflection` umožňuje kontrolovat a/nebo měnit atributy tříd, rozhraní, polí a metod za běhu [9]. To se hodí zejména tehdy, když v době kompilace neznáme jejich názvy.

```
public class Person {
    private String name;
    private int age;
}

@Test
public void parseFieldsReflection() {
```

```
    Person person = new Person();
    //here is used reflection to parse Person to fields
    Field[] fields = person.getClass().getDeclaredFields();

    List<String> actualFieldNames = getFieldNames(fields);

    assertTrue(Arrays.asList("name", "age")
      .containsAll(actualFieldNames));
}

private static List<String> getFieldNames(Field[] fields) {
    List<String> fieldNames = new ArrayList<>();
    for (Field field : fields)
      fieldNames.add(field.getName());
    return fieldNames;
}
```

Zdrojový kód 3.15: Využití reflexe v Javě [9]

Metoda `parseFieldsReflection` zajišťuje pomocí reflexe předávání deklarace atributů, to znamená jak privátní, tak veřejné atributy třídy `Person`. Pomocí objektu `Field` můžeme získat informace jako jsou typ, název, hodnota a další.

`ClassUtils`

Třída `ClassUtils` slouží k přijetí daného atributu na kterém je volána metoda `get(Object)` získávající hodnotu atributu pro daný objekt. Pokud tedy chceme získat hodnotu atributu využívající danou anotaci, použijeme objekt v zmíněné metodě `Field::get(Object)`, která vrací naší chtěnou hodnotu atributu.

`AnnotationUtils`

Třída `AnnotationUtils` slouží k získání atributů, které jsou dále využívány ve třídě `ClassUtils`. V třídě `AnnotationUtils` jsou poskytovány metody, sloužící pro kontrolu výskytu anotace pomocí metod `isAnnotationPresent()`. Tyto metody kontrolují, zda je předaná anotace využívána. Další metody slouží k získávání atributů, to probíhá pomocí několika přístupů.

- Hledání atributu pomocí jeho anotace. Můžeme hledat první nalezený, nebo všechny atributy s danou anotací.

- Hledání atributu pomocí hodnot `TableType`. Pokud je podmínka využití anotace `TableElement` splněna, kontroluje se zdali jsou hodnoty shodné, tedy hledaný atribut vůči hodnotě `TableElement::value()`.

Metody vrací atributy nebo pole atributů vytvořených dle zadaných anotací, s těmito atributy dále pracuje třída `ClassUtils`, která je využívá pro získání konkrétních hodnot.

Poté co jsou získány konkrétní hodnoty se tyto hodnoty využívají pro mapování hodnot do sloupců a řádků tabulky. Jelikož tabulka může obsahovat data zanořující se do sebe, je každá řádka/sloupec při zpracování označen unikátním identifikátorem v rámci všech řádek/sloupců a zároveň neunikátním identifikátorem hloubky zanoření.

Dále se data převádí do jednorozměrné struktury, kdy jejich zanoření a pořadí je definováno pouze přiřazenými identifikátory. Rozdíl mezi metodami pro řádky a sloupce je v tvorbě relací pomocí třídy `Relation`. Jelikož je pole hodnot tabulky ukládané do instancí řádků, metoda obsahuje navíc mapování těchto hodnot a vytváření závislosti na sloupcích. Metoda pro sloupce využívá třídu `Relational` pouze pro ukládání hodnot, které tuto anotaci využívají.

V metodě mapování řádků je vždy hodnota využívající anotaci `Relation` spjata s konkrétní hodnotou řádku, k vytvoření závislosti na sloupci je využita metoda `getColumnIndex()`. Tato metoda vrací index daného sloupce, na který má být hodnota mapována, pokud hodnota neexistuje, metoda vrací `-1`. Pokud relace existuje, hodnota využívající `Relational` je přiřazena do pole hodnot řádek v tabulce dle přiřazeného indexu sloupce.

Metoda `getColumnIndex()` je vytvoření tabulky zásadní. Vždy se volá pro hodnotu jedné řádky tabulky. První průchod se provádí přes všechny relace dané řádky, kontroluje se, zda je využit `regex`, který je nepovinný a je povolen pouze pro hodnoty typu `String`.

Dále je kontrolována unikátnost tabulky. Unikátnost tabulky je vlastnost charakterizující jedinečnost hodnot pro danou skupinu řádek či sloupců. Pokud jsou řádky charakterizovány například linkami v továrně, které vyrábí produkty se stejnými názvy, pak pro každou skupinu řádek, tedy jednu linku, se opakují hodnoty vyráběných produktů, tím pádem tabulka není unikátní. Pokud se jedná o tabulku s unikátními hodnotami, jsou použity všechny sloupce, jako hodnoty relace, které budou kontrolovány. Pro každé zavolání metody je tedy vždy procházen celý seznam sloupců při hledání závislosti daného řádku.

Pokud tabulka neobsahuje hodnoty, které jsou unikátní, procházení sloupců je hierarchické od indexu zanoření nula a dále přes všechny jeho potomky. Při procházení je vytvořen seznam sloupců, v kterém bude následně kontrolována existence relace mezi sloupcem a hodnotou řádku. Relace sloupců je vytvářena

v metodě pro tvorbu sloupců, zde je pouze převzata a zkontrolována její správnost. Pokud relace odpovídá pro indexy řádku i sloupce vrací metoda index sloupce pro hodnotu daného řádku, čímž vytváří relaci. Správnost hodnot není nijak validována, za běhu programu pouze probíhá kontrola existence anotace `@Name` pro každou entitu.

Generování HTML kódu

Po vytvoření všech hodnot tabulky a vytvoření jejich relací jsou tyto hodnoty mapovány do tvaru pro generování HTML kódu. Hlavní metodou pro vytvoření závislostí HTML kódu je metoda `generateTable()`

```
public void generateTable
(Table table, String resultName, boolean sumUp) throws IOException
```

Zdrojový kód 3.16: Hlavička metody `generateTable()` pro vytváření HTML kódu

Parametr `table` je získán pomocí metody `makeRelation()`, která pochází z rozhraní `IRelational`. Tabulka udává objekty jako jednotlivé části tabulky v HTML kódu, jejich atributy a hodnoty. Druhý parametr udává cestu a název výsledného HTML souboru. Poslední parametr definuje, zda bude HTML tabulka využívat funkcionalitu sumace hodnot, dle toho je odvíjen kód.

Využití funkcionality sumace v tabulce

Pokud se funkcionalita nevyužívá, hodnota zobrazující se v tabulce může být typu `java.lang.Object`. Aby bylo hodnoty možné sumovat, musí být typu `Number` nebo implementovat rozhraní `Summable`.

```
    public interface Summable<T> {
    T sumUp(T another);
}
```

Zdrojový kód 3.17: Rozhraní `Summable` pro funkcionalitu součtu hodnot v tabulce

Rozhraní `Summable` je generické s parametrem `T`, který by měl být stejného typu jako typ hodnot, které budou sumovány. Pokud by tedy rozhraní implementovala třída definující úspěšnost validace výrobků na lince, všechny hodnoty, které budou načítány musí být stejného typu jako atributy této třídy. Metoda `sumUp()` pouze sčítá hodnoty předaných objektů a vrátí instanci jako součet hodnot.

Pro vytvoření HTML souboru je využita knihovna Jsoup. Jsoup je open-source knihovna jazyka Java, která se používá především k extrakci dat z HTML [6]. Umožňuje také manipulaci a výstup HTML. Jsoup lze také použít k analýze a sestavování XML.

Knihovna obsahuje několik funkcionalit: načtení a rozbor HTML do dokumentu, výběr požadovaných dat do prvků a jejich procházení, získání atributů, textu a HTML uzlů a úpravu uzlů a jejich atributů.

Jsoup načte stránku HTML a vytvoří odpovídající strom DOM. Tento strom funguje stejně jako DOM v prohlížeči a nabízí metody podobné JQuery a vanilla JavaScriptu pro výběr, procházení a manipulaci s HTML.

Na začátku metody `generateTable()` Jsoup nastaví hlavičku nového HTML souboru, vytvoří závislosti tohoto souboru na skriptech, CSS souboru pro stylizaci a předá způsob kódování. Pokud neexistuje cesta k adresáři, ve kterém se má HTML kód vytvořit, je tento adresář vytvořen a společně s HTML souborem jsou zkopírovány závislosti.

Kromě definovaných závislostí obsahuje třída i metody pro vytvoření závislosti na osobním CSS a JavaScript souboru. Samotné generování HTML kódu probíhá ve statické metodě `generateTable()` třídy `HtmlRenderer`.

```
public static Element generateTable(RowElementList rowElementList,
List<ColumnElement> columnElementList, boolean sumUp) {
```

Zdrojový kód 3.18: Metoda pro generování HTML kódu

Parametry `RowElementList` a `ColumnElementList` značí seznamy řádků a sloupců. Tyto seznamy obsahují data, uložená jako abstraktní reprezentace hodnot pro každou řádku/sloupec tabulky. Třetí parametr `boolean sumUp` metody `generateTable()` 3.18 opět indikuje, zda bude mít tabulka možnost sumace hodnot. Data `RowElementList` a `ColumnElementList` jsou definovaná pomocí svých privátních atributů, které obsáhnou uložení všech dat potřebných pro vytvoření tabulky. Tato data s již vytvořenými závislostmi byla vytvořena pomocí předchozích algoritmů.

```
public class RowElement {
  private int depth;
  private int index;
  private String name;
  private boolean filter;
  private String filterValue;
  private Relation[] relationalArray;
  private FilterCriteria[] filterCriteria;
  private String tooltip;
```

```
private Object[] valueList;
private String link;
```

Zdrojový kód 3.19: Atributy abstraktní třídy představující řádku tabulky

Atributy `index` a `depth` identifikují na kterém místě se bude řádka vyskytovat. Dále jsou zde atributy nutné pro funkci filtrování a pole se závislostmi na sloupcích.

Ve třídě `HtmlRenderer` je vytvořen HTML element značící kostru tabulky. Tato kostra tabulky je postupně dotvářena a obohacována o data tvořící výsledný HTML kód. Jedna z metod zjistí, zda tabulka obsahuje funkci filtrování, pokud ano vytváří buňku tabulky s textovým polem pro filtraci, dále kontroluje, zda je využívána funkce součtu hodnot a podle využití vytváří nadpis pro nadcházející buňku tabulky uvádějící součet hodnot řádek. Dále vytváří jednotlivé hlavičky každého sloupce obsahující nadpis daného sloupce s hypertextovým odkazem a textovou nápovědou. Po získáni všech potřebných informací o hlavičce tabulky jsou tato data předávána na konec kostry HTML kódu. Podobně jsou HTML elementu předávána data pro tělo kódu `<tbody>`.

V těle tabulky jsou vytvářena data pro filtraci, tedy hodnoty, dle kterých bude filtrováno, ty hledá průchodem `RowEelementList` a zjišťuje, zda nějaké hodnotě nebyla přiřazena pomocí anotace `Filter` vlastnost zařazení do hodnot, dle kterých se bude filtrovat.

Dále jsou zde uceleny hodnoty všech řádků. Dle indexu a hloubky zanoření vybere metoda všechny hodnoty dané řádky, vytvoří jejich textovou nápovědu a pokud je využívána funkce součtu, spočítá celkovou sumu řádky a uloží jí, takto hotová data uloží do HTML elementu `<tr>` a přidá je do seznamu celé tabulky.

Pokud tabulka využívá jak funkci zanoření, tak funkci sumace hodnot, je navíc v metodě krok pro vytváření mezisoučtů v buňkách se stejnými předky hodnotami zanoření.

Třída `HtmlRenderer` vytváří pouze to, co bude zobrazeno, tedy části HTML kódu jako je `tbody`, `tr` a podobné. Funkce vytvářející data tabulky jsou uváděny pomocí tří JavaScript souborů, které jsou na dané HTML elementy navázány.

`table-processor.js`

Slouží k usnadnění manipulace s tabulkou. Využívá k tomu HTML elementy reprezentující řádky a sloupce, které si ukládá do seznamu celé tabulky. K řazení využívá atributy `index` a `depth`, které zařazení prvků definují. Dále obsahuje metody pro získání všech potomků daného prvku a získání rodičů.

table-renderer.js

JavaScript soubor obsahuje samostatnou třídu `TableRenderer`, která poskytuje metody pro usnadnění dynamické vizualizace tabulky. Samotný script nejprve nastaví tabulku tak, aby všechny řádky a sloupce byly zabaleny (defaultně je tabulka vždy automaticky plně rozbalená, aby šlo pracovat s daty). Dále JavaScript obsahuje několik metod pro zobrazení řádek. Tedy pokud uživatel klikne na vybranou řádku, která není rodičem, zabalí se pod svého rodiče. Dále jsou zde metody pro rozbalování a zpětné zabalování řádek, které fungují na obdobném principu, pouze vyhledávají své potomky. Uživateli se tedy po kliknutí na vybranou řádku rozbalí vždy potomci s hodnotou zanoření o jedna větší. A při opětovném stisknutí se opět všichni potomci schovají. Poslední metody zde vytvářejí stylizaci, pouze nastavují odstíny barev pro zanoření řádků.

table-filter.js

Při využití této metody jsou nejprve všechny řádky zabaleny zpět, aby se mohly poté zobrazit pouze hodnoty odpovídající danému filtru. Poté metoda prohledává element `TableRow` a vyhledává hodnoty, které byly anotovány pomocí anotace `filter`. Tyto filtry jsou vždy vázány na `<tr>` element v HTML kódu, který obsahuje atributy `filter` a `filterValue`.

```
function filter(filterValue) {
tableRowRenderer.collapseAll();
for (let element of tableRowRenderer.tableProcessor.getElementList()) {
    if (element.hasAttribute(Attrs.FILTER)) {
        if (element.getAttribute(Attrs.FILTER_VALUE) === filterValue) {
            tableRowRenderer.showElement(element);
        }
    }
}
}
```

Zdrojový kód 3.20: Využití funkce filtrování v JavaScript souboru

Atribut `filter` značí zda je řádka smysluplná pro využití filtrování, atribut `filterValue` je samotná hodnota využitá pro filtrování. Poté co uživatel vybere nějakou hodnotu filtru, metoda rozbalí všechny řádky, které této hodnotě odpovídají.

Testy aplikace `TableGenerator`

V aplikaci je testováno správné fungování pro parsování JSON souborů a práce s anotacemi. Nejprve je otestována funkcionalita správného parsování JSON z

testovacích souborů a následně je kontrolována konzistence dat při jeho využití. Dále je testována práce s anotacemi. Je kontrolováno správně přiřazení dle jedné a více anotací. Dále jsou kontrolovány hodnoty vytvořené v tabulce.

3.4.2 Využití programu

Program je vygenerován jako přenositelný JAR soubor. Uživatel jej přidá do svého nového programu a nastaví závislosti na knihovně (v `oks` i `general` aplikaci je JAR soubor přidán do `lib` adresáře). Poté lze využívat všechny funkcionality JAR souboru `TableGenerator`.

4 Použití programu

Aplikaci z hlediska využití uživatelem můžeme rozdělit na dvě hlavní části. Aplikace vytvořená pro předmět OKS a projekt TbUIS, tvořící závislost požadavků a případů užití získaných exportem ze SquashTM. Tato aplikace je neměnná a spouští se pomocí CLI či GUI režimu.

Druhá aplikace je vytvořena pro práci s obecnými daty. Stejně jako OKS aplikace využívá mou vytvořenou knihovnu `TableGenerator`, pro parsování je potřeba výslednou aplikaci doplnit o potřebné třídy - viz dále.

4.1 Aplikace pro OKS

Aplikace je kompletní, to znamená, že pro fungování není nic potřeba dotvářet nebo přidávat, stačí mít vstupní parametry, z kterých chceme tabulku závislostí vytvořit. Tato aplikace je rozdělena do čtyř hlavních kategorií, jakých typů dat v tabulkách lze dosáhnout.

Pro předmět OKS lze vytvořit závislost případů užití na požadavcích, který využívá výsledků aplikace SquashTM, nebo závislost případů užití na testovacích případech.

Pro obě sekce dat je tvořena závislost kdy sloupcům tabulky jsou předávány hodnoty případů užití.

Druhé dvě kategorie jsou obdobné, pouze vytváří data pro projekt TbUIS. Pro tento projekt je opět sloupcovými hodnoty případy užití a závislost na hodnotách řádků může značit buďto závislost na testovacích případech nebo požadavcích.

Ačkoliv se hodnoty ve výsledné tabulce zdají identické pro oba projekty, zdrojové soubory mají jinou strukturu, a proto je tvorba tabulky jednotlivých projektů oddělena. Spuštění aplikace je možné provést jak GUI, tak řádkovou CLI formou.

4.1.1 Fungování aplikace

K vytvoření požadovaného výsledku jsou předávány parametry dle typů dat tabulky. Pokud jsou data předávány pro projekt OKS, je třeba mít soubory vytvořené z aplikace SquashTM, tedy JSON testovací případy a požadavky, textový soubor případů užití a výsledky ve formátu CSV exportované ze SquashTM.

Implementace

Aplikace OKS přebírá tvorbu tabulky z vytvořené knihovny `TableGenerator`. Pro její funkčnost byla potřeba naimplementovat třídy, které využívají anotace a podle kterých jsou hodnoty z předaných parametrů parsovány do HTML kódu.

Tyto třídy jsou v programu rozděleny na dvě hlavní části. První balík `rqm.table` slouží k parsování požadavků na případech užití, druhý balík `tc.oks` slouží k parsování testovacích případů. Oba balíky mají velmi podobné rozvržení tříd, jelikož vychází ze struktury JSON souborů vytvářených aplikací SquashTM. Pro pochopení tvorby anotací je zde ukázka JSON souboru vygenerovaného aplikací SquashTM obsahující data jednotlivých požadavků (RQM).

```
[                                          // every data is in array
  {                                        // cover for rqs
    "rqs": {                               // rqs
      "rqmList": [                         // list of rqm
        {                                  // rqm cover
          "rqm": {                         // rqm
            "prefix": "RQM.01.01",         // annotated as name
            "name": "Happy day scenario",  // annotated as tooltip
            "criticality": "critical",     // annotated as filter
            "description": "description",  // anotated as relational
            "category": "undefined"        // not used in this table
          }
        },
    ...
```

Zdrojový kód 4.1: Struktura JSON souboru požadavků v aplikaci OKS

Pro výše zmíněnou strukturu je potřeba vytvořit třídy takové, aby obsáhly všechna data pro tvorbu tabulky a zároveň zachovaly strukturu a smysl dat. Konkrétně je tento JSON využíván pro tvorbu řádků. V balíku jsou vytvořeny třídy: `Rqs`, `RqsList`, `RqsCover`, dále `Rqm`, `RqmCover` a `RqmValue`. Třída `Cover` slouží k zanoření dat v závorkách, třída `RqmValue` slouží k reprezentací vnitřních hodnot tabulky. Stejným způsobem jsou vytvořeny třídy pro reprezentaci testovacích případů. Každá třída využívá anotací, jednotlivé anotace jsou vysvětleny v předchozích kapitolách. Zde je uveden pouze zjednodušený příklad třídy `Rqm`, tyto hodnoty můžeme vidět v JSON souboru 4.1.

```
@Name
private String prefix;
```

```
@TableElement(TableType.VALUE)
private RqmValue rqmValue;

@TableElement(TableType.TOOLTIP)
private String name;

@Relational
private String description;

@Filter
private String criticality;
```

Zdrojový kód 4.2: Využití anotací pro OKS tabulku požadavků

Stejně tak jsou vytvořeny třídy pro hodnoty sloupců, tedy případů užití.

Kromě tříd sloužících k anotování hodnot je zde navíc několik tříd, které doplňují funkčnost generování.

RqsParser

Implementující rozhraní `IParser`. Tato třída slouží k parsování řádek do seskupení odpovídající hierarchii JSON souboru. Vyhledává jednotlivé řádky pomocí hodnoty `prefix` z JSON souboru, který je anotován jako název daného elementu pomocí anotace `@Name`. Po nalezení dané hodnoty je hodnota přidávána do korektního `rqsList` seznamu.

UcParser

Slouží k rozdělení textového souboru na jednotlivé objekty, které budou využity na vytváření hodnot sloupců. Pro aplikaci OKS je soubor s případy užití vždy ve stejném formátu.

```
UC.01=http://oks.kiv.zcu.cz/Prevodnik/Uvod;Happy Day
UC.02=http://oks.kiv.zcu.cz/Prevodnik/Prevodnik;Jednotlivé položky
UC.03=http://oks.kiv.zcu.cz/Prevodnik/Napoveda;Uživatel se spletl
UC.04=http://oks.kiv.zcu.cz/Prevodnik/Napoveda;Ostatní
```

Zdrojový kód 4.3: Formát textového souboru s případy užití

Každá řádka daného souboru obsahuje informace pro jeden sloupec. Třída `UcParser` načte soubor v tomto formátu a rozděluje ho na jednotlivé objekty, tedy rozdělí textový řetězec pomocí znaménka = a poté středníkem. Dostane

tak tři informace pro daný sloupec. První informací je jméno, které uloží třídě `Uc` pomocí metod pro nastavení atributů. Druhým parametrem je externí odkaz a poslední slouží jako textová nápověda. Po nastavení všech hodnot daného `Uc` je celá entita uložena do seznamu `UcList`

```
String name = line.substring(0, line.indexOf('='));
String link = line.substring(line.indexOf('=') + 1,
line.lastIndexOf(';'));
String tooltip = line.substring(line.lastIndexOf(';') + 1);

uc.setName(name);
uc.setTooltip(tooltip);
uc.setLink(link);
ucList.add(uc);
```

Zdrojový kód 4.4: Parsování textového souboru případů užití na entitu `Uc`

Z původní aplikace jsou zde převzaty dvě myšlenky, dle kterých je implementováno několik tříd.

`TcResultsToRqm`

Slouží k předávání hodnot mezi testovacími případy a požadavky pro předmět OKS. Aplikace SquashTM umožňuje exportovat pouze výsledky testovacích případů ve formátu CSV. Jelikož existuje závislost mezi testovacími případy a požadavky, tyto výsledky zároveň značí i výsledky požadavků.

`TcResultsToRqm` slouží k přenesení hodnot z CSV souboru výsledků na výsledky pro hodnoty požadavků. Tyto hodnoty jsou vytvářeny pomocí zmíněného CSV souboru a obou JSON souborů pro požadavky a testovací případy.

Nejprve jsou nalezeny výsledky v CSV souboru a uloženy pro atributy třídy `TsResult`. Dále jsou dle prefixů vyhledávány hodnoty jak požadavků, tak testovacích případů, pokud je nalezena shoda, jsou uloženy hodnoty z `TsResult` seznamu výsledků do `rqmValue` hodnoty požadavku.

Tyto metody fungují pouze za podmínky, že identifikace testovacích případů a požadavků je identická. To znamená, že pokud existuje požadavek, který má prefix `RQM.01.01`, abychom mohli hodnoty parsovat, musí existovat i testovací případ `TC.01.01.xy`.

`Result2TcSquash`

Je vytvořen pro projekt TbUIS. V projektu TbUIS se výsledky nezískávají z aplikace Squash, ale pomocí vygenerovaných logů v projektu. Dle souboru logů

se nastaví kampaň a iterace. Při generování JSON souboru s výsledky z těchto logů, se nejprve zkontroluje nastavení kampaně a iterace a procházením výsledků z předaného parametru se ukládají výsledky. Tyto výsledky jsou reprezentovány třídou `ResultT`, která obsahuje pouze kampaň, iteraci a list výsledků reprezentovaný třídou `Reuslt`.

```
class Result {
      private String id;
      private String result;

      public Result(String id, String result) {
          this.id = id;
          this.result = result;
      }
```

Zdrojový kód 4.5: Reprezentace výsledků po získání dat z LOG souborů

`Result` je konkrétní reprezentace hodnot z logů, která má atributy `Id`, reprezentující identifikaci hodnoty řádky a `result`, která z souboru získává hodnoty `succes` a `failure`. Po načtení všech hodnot jsou výsledné hodnoty reprezentované třídou `Result` uloženy do JSON souboru.

```
 {
  "id": "TC.A.05.01.01",
  "result": "failure"
},
{
  "id": "TC.A.05.02.01",
  "result": "success"
},
```

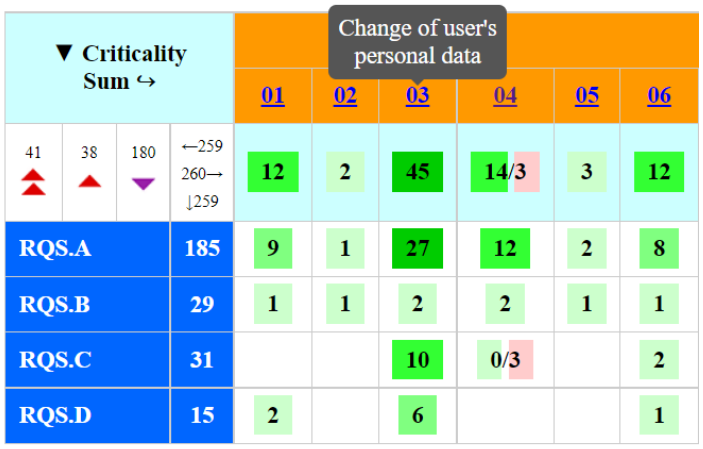
Zdrojový kód 4.6: Ukázka výsledného JSON souboru třídy `Result2TcSquash`

CSS styl

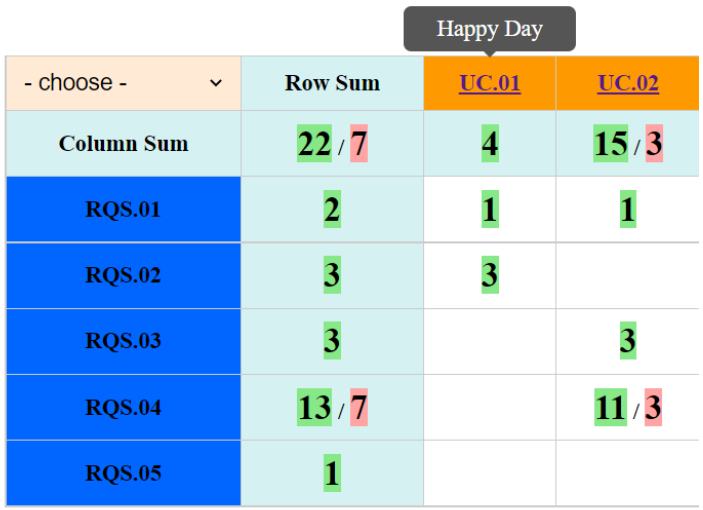
Styl tabulky je tvořen pomocí CSS souboru, který je přidán ke každému nově vytvořenému HTML kódu. Je pouze jeden pro všechny typy vytvořených tabulek. Znamená to tedy, že uživatel nemusí žádný soubor dodávat, tabulce se vytvoří stylizace sama.

Hlavní myšlenkou této stylizace bylo co nejvíce se přiblížit vzhledu tabulky v původní aplikaci, dle zadání vedoucího projektu. Byl pro to vytvořen nový

CSS soubor, jelikož se struktura jednotlivých částí s vlastnostmi od původního CSS značně lišila. Zůstaly zachovány barvy pro hlavičku tabulky a pozadí řádku a sloupce pro součty hodnot. Barva při rozbalování řádek byla pozměněna, jelikož se barevná odlišnost dělá automaticky, v původní aplikaci byly definovány barvy pro každé zanoření zvlášť. Byl převzat styl textové nápovědy a barevné pozadí hodnot při rozlišování dat na dvě možnosti `failed, passed`.



Obrázek 4.1: Stylizace původní aplikace



Obrázek 4.2: Stylizace OKS aplikace

Původní aplikace měla navíc oproti OKS aplikaci možnost grafického znázornění filtru, pomocí šipek na obrázku 4.1. V OKS aplikaci je filtrace vytvořena pomocí `selectOneMenu`. Tag `selectOneMenu` slouží k vykreslení jediného

vstupního prvku HTML typu `select` [10]. Je tak možné vytvořit seznam hodnot, z kterých je vybíráno, dle čeho se hodnoty tabulky vyfiltrují. Možnosti tohoto tagu nepodporují vložení obrázků, proto nebyl způsob zobrazení filtrace pomocí šipek využit.

Možným rozšířením by bylo vyhledávat v tabulce místo `selectOneMenu`, pomocí `div` elementů. Pro aktuální případ je problém však řešen přepínání mezi `selectOneMenu` a jednoduchým `Textfield` elementem pro vyhledávání hodnot pomocí textu.

Testy aplikace OKS

Je otestováno smazání a vytvoření testovacího adresáře, a poté na stejné místo vygenerování HTML kódu. Dále je zkontrolováno, zda odpovídá počet souborů, které se vytváří společně s HTML souborem jako jeho závislosti. Jsou prováděny testy jak pro projekt TbUIS tak pro OKS, kdy jsou porovnávány výsledné celkové hodnoty HTML souboru s očekávanými výsledky.

4.1.2 Spuštění a výsledek aplikace

Aplikace má dvě základní formy, CLI a GUI formu.

CLI forma

První formou je CLI, tedy spuštění pomocí JAR souboru. Předávání parametrů v příkazovém řádku je ve formátu s přepínači na jednotlivé parametry.

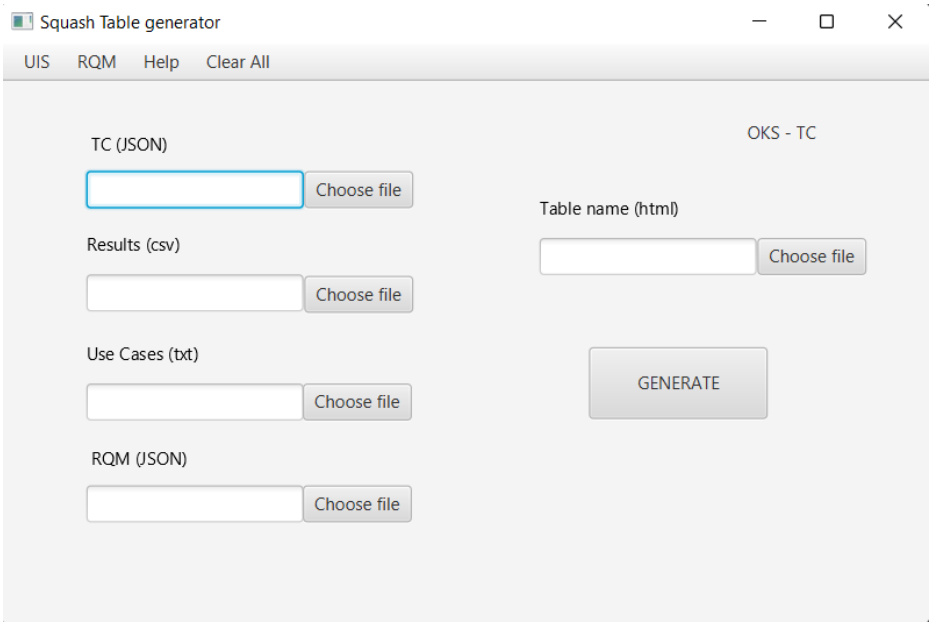
```
java -jar app.jar -cli -appMode oks -inputMode rqm TC_FILE
-useCase USE_CASE_FILE -rqm RQM_FILE -target TARGET_FOLDER
-results RESULT_FILES
```

Zdrojový kód 4.7: Spuštění pomocí příkazové řádky

Rozdíl mezi spuštěním CLI a GUI formy je v prvním parametru předaném příkazové řádce. Pokud je předán parametr `-gui` další parametry nejsou potřeba a spustí se GUI aplikace. Pokud uživatel zadá parametr `-cli` musí zadat další parametry, potřebné k vytvoření HTML kódu. Druhý a třetí parametr udává o jaké vstupní parametry se bude jednat. Uživatel může v druhém parametru volat `uis` nebo `oks` a v třetím parametru udává, zda se jedná o testovací případy nebo požadavky (`tc`, `rqm`).

GUI forma

Aplikace je vytvořená pomocí jazyka JavaFX. Spouští se pomocí JAR souboru. Uživateli je zobrazeno vždy pouze jedno okno, v kterém je možné zadat vstupní parametry, dle kterých je tabulka vytvářena.



Obrázek 4.3: GUI aplikace pro závislost testovacích případů na případech užití pro aplikaci OKS

Hlavním rozcestníkem je menu nahoře v aplikaci. První dvě položky tohoto menu udávají, jaký typ tabulky bude vyvářen. Kombinací těchto položek můžeme získat čtyři druhy GUI okénka.

Změnu okna provedeme pouze kliknutím na danou položku, kterou chceme měnit. Pro lepší orientaci je v pravém horním rohu nadpis, která tabulka právě bude vytvářena.

V obrázku 4.3 je zobrazeno okno pro generování tabulky testovacích případů pro předmět OKS. Kliknutím na první položku v menu pod názvem UIS se nám zobrazí okno pro generování tabulky testovacích případů pro projekt TbUIS. Pokud by uživatel zvolil druhou položku, otevřelo by se okno pro generování požadavků (RQM) pro předmět OKS.

Třetí položka v menu slouží k zobrazení nápovědy, aby uživatel věděl, jak s aplikací zacházet. Poslední položka `Clear all` slouží k smazání již předaných parametrů, znamená to, že vymaže všechna data z textových polí v daném okně. U každého výběru souboru je poznámka, který parametr se má zde předávat a jakého typu má soubor být. Pro výběr `Results (csv)` je možné

vybrat pomocí tlačítka více souborů, které jsou odděleny v textovém okně pomocí středníků. Pro výběr tabulky se místo souboru vybírá cesta, kam se výsledný HTML soubor uloží. Pro vygenerování HTML kódu je potřeba vyplnit vždy všechny parametry, jinak kód není možné vygenerovat. Pro každý typ generování dat se vstupní parametry minoritně liší.

4.2 Obecná aplikace

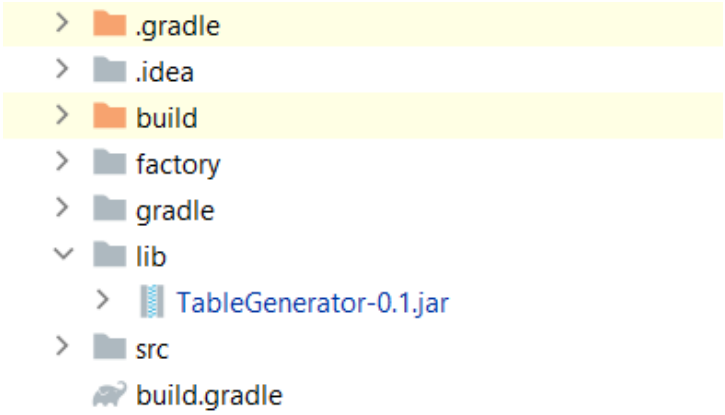
Dle zadání má existovat druhá aplikace, která je schopná generovat vizualizaci pro výsledky z datových zdrojů popisujících závislosti jiných (principiálně obecných) hierarchicky strukturovaných entit. Jelikož se může struktura dramaticky lišit, není možné vytvořit konečnou aplikaci, která by byla schopna řešit úplnou obecnost. Proto je koncept této obecné aplikace tvořen odlišně oproti aplikaci OKS. Pro aplikaci OKS je vytvořen JAR soubor s kompletním algoritmem pro generování, jelikož mají všechna data stejnou strukturu, která je pomocí tříd a anotací naimplementována. V obecné aplikaci je využita pouze stejná knihovna `TableGenerator`, která je schopna vytvořit HTML kód dle použitých anotací.

4.2.1 Fungování aplikace

Pokud chce uživatel vytvořit svou vizualizaci, musí si nejprve naimplementovat třídy popisující vstupní JSON a jejich atributy řádně anotovat. Poté co si vytvoří sadu tříd s anotacemi, může využívat množství typově stejných dat ke generování HTML kódu závislostí. Nejprve je třeba založit nový projekt a přidat `TableGenerator` do projektu.

Přidání knihovny do projektu

`TableGenerator` je předávaná jako JAR soubor `TableGenerator-0.1.jar`. Pro práci s touto knihovnou je doporučené využít Gradle, nebo nějaký jiný nástroj pro automatizaci sestavování programu, který umožní navázání knihovny. JAR soubor předáme projektu do adresáře `lib`.



Obrázek 4.4: Výsledek vložení knihovny do adresáře lib

Po přidání je třeba vytvořit závislosti na předané knihovně, toho lze dosáhnout úpravou souboru `build.gradle`

```
dependencies {
    implementation fileTree(dir: 'lib', include: ['*.jar'])
    }
```

Zdrojový kód 4.8: Přidání závislostí na knihovně `TableGenerator`

Dalším krokem je vytvořit třídy s využitím anotací dle dat, které chceme reprezentovat v tabulce. Je doporučeno vytvořit si samostatný balík pro třídy řádek a sloupců, aby bylo jednoduše rozeznatelné k čemu třídy slouží. Vzhledem k faktu, že si uživatel vytváří nový projekt, nebo aspoň nové třídy pro možnost anotovat objekty a tím získat výsledný HTML kód, šlo by o neustálé generování JAR souboru, nebo aktualizace GUI formy. Z toho důvodu není vytvořena CLI ani GUI forma. Druhým důvodem je odlišnost dat. Pro sloupce mohou být data předána jako JSON, nebo textový soubor, každý uživatel by GUI potřeboval trochu jiné, aby byla zajištěna určitá přehlednost a intuitivnost aplikace.

4.2.2 Vytvoření sady reprezentativních dat

Pro reprezentaci obecné aplikace byla jedna testovací sada vytvořena a jedna převzata z aplikace OKS. První testovací sada představuje seznam továren, které obsahují všechny informace o výrobě svých produktů. Druhá testovací data ukazuje závislost požadavků aplikace Squash na testovacích případech. Tato závislost nebyla v počátku úmyslu, použila jsem ji až v závislosti na demonstraci obecnosti, jelikož data nejsou generována s žádnou závislostí kromě stejných prefixů.

Factory

Datový set `Factory` reprezentuje počet vyrobených kusů nábytku dle jednotlivých výrobních linek. JSON soubor obsahuje více továren, každá je reprezentována různým počtem výrobních linek, kdy každá linka má určený počet daného kusu nábytku a den v roce, kdy byl tento počet výrobků vytvořen.

```
"factory": {
      "name": "Factory 1",
      "productionLineList": [
        {
          "line": 1,
          "name": "linka A",
          "furnitureList": [
            {
              "name": "Coffee Table",
              "amount": 9,
              "day": 2,
              "month": 2,
              "year": 2021
            },
```

Zdrojový kód 4.9: Struktura JSON souboru `Factory`

Závislost výrobních dat je vytvářena na dnech v roce, sloupce tedy představují roky, dělící se na měsíce a dále na jednotlivé dny. Data pro sloupce jsou reprezentovány v souboru `Days.json`.

```
    "name": 2021,
    "months": [
      {
        "name": " February",
        "id": 2,
        "days": [
          {
            "name": "1.2.2021",
            "id": 1
          }
        ]
      },
      {
        "name": "January",
```

```
 "id": 1,
 "days": [
   {
     "name": "1.1.2021",
     "id": 1
     }
}
```

Zdrojový kód 4.10: Struktura JSON souboru `Days`

RqmTc

Pro tyto data bude vytvořena tabulka závislostí, kdy hodnoty budou pouze jednotkové. JSON `TC` obsahuje závislosti na jednotlivých požadavcích (RQM) udávaných v poli `verified_requirements`.

```
{
    "tc": {
        "nature": "functional",
        "prefix": "TC.04.02.01.09",
        "importance": "medium",

        "verified_requirements": [
        {
          "id": "04.02.01"
        }
        ]
    }
},
```

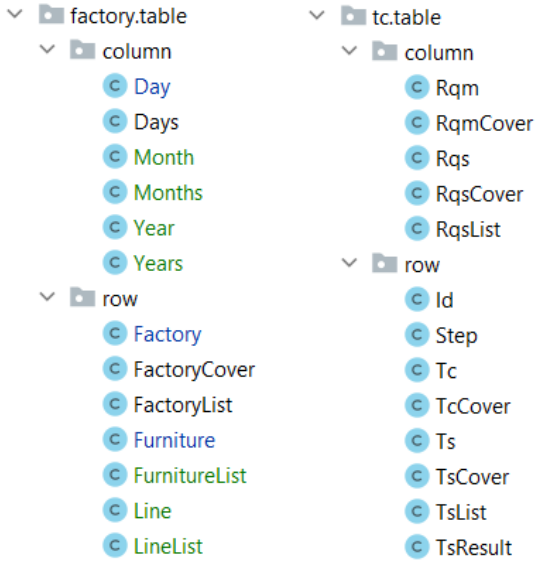
Zdrojový kód 4.11: Ukázka návaznosti testovacích případů na požadavcích

V ukázce 4.11 je v poli požadavků hodnota `"id": "04.02.01"`, znamená to tedy, že testovací případ TC.04.02.01.09 se bude mapovat na požadavek, který má id RQM.04.02.01. Testovacích případů je větší množství než požadavků, vždy má ale skupina testovacích případů jednoho rodiče požadavků, žádný testovací případ nemá návaznost na více požadavků.

4.2.3 Aplikace s využitím reprezentativních dat

Byl vytvořen projekt `General_app` pro generování vizualizace dat továren a požadavků na testovacích případech. Byla přidána knihovna `TableGnerator` 4.4.

Projekt obsahuje dva balíky, každý představuje jednu testovací sadu.



Obrázek 4.5: Balíky reprezentující testovací sady

Každý balík je rozdělen dále na třídy reprezentující sloupce a řádky. Anotace jsou tvořené podobně jako v aplikaci OKS. Pro hodnoty je povinná anotace `@Name`. Pomocí `JsonParser` jsou poté vytvořeny hlavní data `factories` a `years`, obsahující všechny informace o sloupcích a řádcích.

```
FactoryList factories = new
JsonParser<FactoryList>().parseFile("factory.json", FactoryList.class);

Years years = new JsonParser<Years>().parseFile("Days.json", Years.class);

IRelational<FactoryList, Years> relational = new AnnotationRelation<>();
Table table = relational.makeRelation(factories, years);
TableGenerator tableGenerator = new TableGenerator();
tableGenerator.generateTable(table, "factory/f.html", true);
tableGenerator.addCustomCss("");
```

Zdrojový kód 4.12: Metody k vygenerování HTML kódu Factory

V kódu 4.12 můžeme vidět volání metod k vytvoření HTML kódu po vytvoření všech tříd s anotacemi potřebnými k parsování entit. Po vytvoření instancí `factories` a `years` je volána metoda k vytvoření jejich relace. Po generování HTML souboru je možné přidat svou stylizaci tabulky v CSS formátu – zde nepoužito.

Obdobným způsobem je vytvářen kód pro generování testovacích případů a požadavků. Původní myšlenkou bylo parsovat hodnoty testovacích případů na sloupce a požadavky jako hodnoty řádku. Toto řešení bylo jednodušší, jelikož požadavky využívaly anotaci `@Value`, které je třeba v řádcích pro vytváření dat tabulky. Jelikož ale bylo hodnot testovacích případů více, od první myšlenky se opustilo a parsují se testovací případy do řádku, k tomu byla potřeba vytvořit anotaci `@Value` v testovacích případech s hodnotou jedna.

```
@TableElement(TableType.VALUE)
private int tcValue = 1;
```

Zdrojový kód 4.13: Využití obecnosti anotace @Value

5 Závěr

V rámci bakalářské práce bylo důkladně analyzováno několik způsobů zajištění 2D vizualizace s možnostmi prostupného odkrývání či skrývání detailů. Následně byla vybrána možnost realizace pomocí HTML 5 a JavaScriptu. Tím byl položen základ pro další práce.

V další fázi bylo třeba zvolit vhodný programovací jazyk a způsob zpracování vstupních JSON souborů. Po zhodnocení možností a i s ohledem na předpokládané praktické využití aplikace byla jako programovací jazyk zvolena Java, která díky propracovanému systému anotací významně zjednoduší zpracování zmíněných JSON souborů.

Jako první část byla na těchto základech vytvořena plnohodnotná aplikace pro využití v předmětu KIV/OKS. Aplikace je požadavkům tohoto předmětu "ušita na míru"nicméně i ona má určité prvky obecnosti. To je dokázáno tím, že umí zpracovávat vstupní data i z projektu TbUIS. Nutno ale říci, že ten se svých charakterem velmi blíží požadavkům z KIV/OKS. Aplikace je plně funkční a od uživatele nevyžaduje žádné speciální znalosti.

Druhou částí práce bylo vytvoření knihovny, která umožňuje výše zmíněný způsob rozbalovací 2D vizualizace i pro libovolné hierarchicky organizované vstupní soubory. Funkčnost této knihovny byla ověřena na jednom demo příkladu a posléze i na rozsáhlých reálných datech. Ta pocházela opět z projektu TbUIS, ale jejich zobrazení je principiálně odlišné od předchozího způsobu. V případě požadavku na vizualizaci vlastních dat jsou samozřejmě nutné programátorské schopnosti, ale lze očekávat, že s pomocí dvou uvedených kompletně zpracovaných příkladů, tento úkol zvládne běžný uživatel Javy.

Všechny body zadání byly splněny a nasazení první části aplikace se předpokládá v letním semestru 2022/23 v předmětu KIV/OKS.

Seznam zkratek

CLI (Command Line Interface) – rozhraní příkazového řádku připojuje uživatele k počítačovému programu nebo operačnímu systému. Prostřednictvím rozhraní CLI uživatelé komunikují se systémem nebo aplikací zadáváním textu (příkazů).

CSV (comma–separated values) – je textový soubor, který má specifický formát umožňující ukládání dat ve strukturované tabulce.

DOM (Document object model) – je datová reprezentace objektů, které tvoří strukturu a obsah dokumentu na webu.

GUI (Graphical User Interface) – rozhraní operačního systému založené na grafice, které k řízení interakce se systémem používá ikony, nabídky a myš (ke kliknutí na ikonu nebo rozbalení nabídky).

JavaFX (special effects in the Java language) – JavaFX je sada grafických a mediálních balíčků, která vývojářům umožňuje navrhovat, vytvářet, testovat, ladit a nasazovat graficky orientované klientské aplikace.

JAR (Java ARchive) – jedná se o formát založený na oblíbeném formátu ZIP, který se používá pro sdružování mnoha souborů do jednoho.

JSON (JavaScript Object Notation) – je standardní textový formát pro reprezentaci strukturovaných dat založený na objektové syntaxi JavaScriptu.

LOG – soubor, který zaznamenává události, k nimž dochází v operačním systému nebo v jiném spuštěném softwaru.

OKS (Ověřování kvality software) – předmět zabývající se ověřováním kvality software. Pro jehož výuku bude využita první část aplikace.

regex (regular expression) – je posloupnost znaků, která určuje vyhledávací vzor v textu.

RQM (Requirement) – rozděluje návrh na co nejmenší specifikované, kontrolovatelné části v aplikaci SquashTM

selectOneMenu – tag `selectOneMenu` slouží k vykreslení jediného vstupního prvku HTML. Je tak možné vytvořit seznam hodnot, z kterých je možné vybrat dle čeho se hodnoty tabulky vyfiltrují.

SquashTM (Squash Test Manegement) – aplikace společnosti Squash, která slouží obecně ke správě a monitorování manuálních testů v agilním a tradičním režimu

TableGenerator – program implementující funkcionalitu celé aplikace. Výsledkem je hotový HTML soubor reprezentující tabulku závislostí.

TbUIS (Testbed University Information System) – projekt tvořený několika softwarovými produkty, sloužící jako řešení, které lze využít pro řízené experimenty hodnotící nově vyvinuté testovací metody a přístupy

TC (Test Case) – základní jednotka aplikace SquashTM. Jedná se o soubor podmínek, na kterých tester určí, zda aplikace funguje tak, jak bylo původně zamýšleno.

UC (Use Case) – popisuje chování systému, který reaguje na požadavek. Každý případ užití je reprezentován jako posloupnost jednoduchých kroků, které začínají cílem uživatele a končí, když je tento cíl splněn.

XML (extensible markup language) – jednoduchý textový formát pro reprezentaci strukturovaných informací.

Literatura

[1] *Annotations in Java* [online]. Open base, 2022. [cit. 2022/03/06]. Annotations in Java. Dostupné z: https://www.geeksforgeeks.org/annotations-in-java/.

[2] BECHTEL, D. *The Ultimate JSON Library: JSON.simple vs GSON vs Jackson vs JSONP* [online]. Open base, 2021. [cit. 2022/03/06]. Tbuis testing. Dostupné z: https://www.overops.com/blog/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/.

[3] *Bootstrap datatable without jQuery* [online]. Open base, 2022. [cit. 2022/03/06]. BootsTrapTable info. Dostupné z: https://bestofreactjs.com/repo/Imballinst-react-bs-datatable.

[4] *BootsTrapTable* [online]. Open base, 2022. [cit. 2022/03/06]. BootsTrapTable info. Dostupné z: https://bootstrap-table.com/.

[5] *HandsonTable* [online]. Open base, 2021. [cit. 2022/03/06]. HandsonTable info. Dostupné z: https://openbase.com/js/handsontable.

[6] HEDLEY, J. *Jsoup* [online]. Open base, 2021. [cit. 2022/03/06]. Jsoup. Dostupné z: https://jsoup.org/.

[7] JENKOV, J. *Java Annotations* [online]. Open base, 2021. [cit. 2022/03/06]. Annotation 2. Dostupné z: http://tutorials.jenkov.com/java/annotations.html.

[8] *10 Best Vanilla JavaScript Table Libraries* [online]. Open base, 2021. [cit. 2022/03/06]. Table libraries. Dostupné z: https://openbase.com/categories/js/best-vanilla-javascript-table-libraries.

[9] *Guide to Java Reflection* [online]. Baeldung, 2021. [cit. 2022/04/06]. JavaReflection. Dostupné z: https://www.baeldung.com/java-reflection.

[10] *Tag selectOneMenu* [online]. Oracle, 2015. [cit. 2022/04/06]. SelectOneMenu. Dostupné z: https://docs.oracle.com/javaee/7/javaserver-faces-2-2/vdldocs-facelets/h/selectOneMenu.html.

[11] *SquashTM - Software Testing Tools Guide* [online]. Software Testing Tools Guide, 2021. [cit. 2022/03/06]. Squash1. Dostupné z: http://www.testingtoolsguide.net/tools/squash-tm/#.

[12] *How TO - Expanding Grid* [online]. W3Schools, 2016. [cit. 2022/03/06]. W3Schools article. Dostupné z: https://www.w3schools.com/howto/howto_js_expanding_grid.asp.

[13] *Tbuis - testing* [online]. Open base, 2021. [cit. 2022/03/06]. Tbuis testing. Dostupné z: https://projects.kiv.zcu.cz/tbuis/web/page/testing.

[14] *Test Case* [online]. Open base, 2021. [cit. 2022/03/06]. Test Case. Dostupné z: http://testovanisoftwaru.cz/dokumentace-v-testovani/test-case/.

Množství informací bylo získáno z internetu, ovšem z neověřených zdrojů například stack overflow, proto nejsou citovány.

Přílohy

A Uživatelská příručka

A.1 Rozdělení aplikace

Adresář Aplikace_a_knihovny obsahuje tři další adresáře se zdrojovými soubory. oks_App obsahuje aplikaci generující HTML kód pro data z předmětu OKS a projektu TbUIS. Ke spuštění slouží JAR soubor `oks_app.jar`, jehož příkazy ke spuštění a chodu aplikace jsou navíc zaznamenány v textovém souboru `howToRun.txt`

Druhý adresář general_app obsahuje obecnou aplikaci, která umožňuje vizualizovat hierarchické JSON soubory. V tomto případě ukazuje obecnost na dvou testovacích datech, `Factory` a `RqsTc`. Spuštění této aplikace je vytvořeno pomocí JAR souboru `general_app.jar`

Třetí adresář TableGenerator obsahuje algoritmus využívaný v obou výše zmíněných programech, přebírají jeho funkcionalitu jako JAR soubor přidaný v lib adresáři obou projektů.

A.2 Podmínky vstupních dat

Pro OKS aplikaci je potřeba aby vstupní data splňovala několik podmínek. Pracuje se zde s požadavky (RQM) a testovacími případy (TC) v závislosti na případech užití (UC). Soubory reprezentující tato data mají pevně určená pravidla formující jejich strukturu.

Každý RQM JSON soubor musí mít návaznost v description na UC. To znamená kdekoliv v JSON souboru v hodnotě `"description"` identifikovat případ užití (například UC.01).

```
"rqm": {
        "prefix": "RQM.01.01",
        "name": "Happy day scenario",
        "criticality": "critical",
        "description": "Spouští scénář, závislost na UC.01"
        "category": "undefined"
        }
```

Zdrojový kód A.1: Ukázka návaznosti UC na RQM

Struktura JSON musí obsahovat prefixy, které jasně identifikují konkrétní data (například RQM.01.02, TC.04.03.01), což je možné vidět i v kódu A.1. Vzhle-

dem k předávání hodnot výsledků TC hodnotám RQM, musí mezi nimi existovat vazba, tedy stejný prefix (TC.01.01 = RQM.01.01), jinak nemohou být vytvořena data požadavků. To samé platí i naopak, jelikož je mapována návaznost případů užití (UC) na testovacích případech (TC), dle `"description":` požadavků (RQM).

Pokud nejsou splněné tyto podmínky, není možné tabulku pro OKS a TbUIS vytvořit. Rozdíl mezi aplikacemi OKS a TbUIS je pouze v získávání výsledků (LOG z projektu TbUIS, CSV soubory aplikace SquashTM)

A.2.1 general_app

General_app je vytvořený projekt pouze pro ukázku `TableGenerator` knihovny a její funkcionality. Funkcionalita je takřka podobná aplikaci oks_app, pouze je ukázána diverzita dat. Principem takové aplikace, respektive knihovny, je, že uživatel může parsovat jakékoliv data stejné hierarchické struktury, poté co si vytvoří základní sadu tříd a anotací, které určují závislosti dat. Je volně nastavitelný i stylizovací styl CSS, tabulka se tedy může změnit i vizuálně.

Založení projektu

Je třeba založit nový projekt, jemuž je předána vytvořená knihovna do adresáře `lib` (`TableGenerator-0.1.jar`). Dále je nutné na JAR soubor vytvořit závislost nějakým nástrojem pro automatizaci sestavování programu.

```
dependencies {
implementation fileTree(dir: 'lib', include: ['*.jar'])
}
```

Zdrojový kód A.2: Ukázka přidání závislostí pomocí `gradle.build` souboru

Vytvoření anotací

Po nastavení knihovny, je potřeba vytvořit třídy dle JSON souborů, které budeme chtít vizualizovat ve výsledném HTML kódu. Pro aplikaci `general_app` šlo konkrétně o mapování JSON souborů `Factory.json` na `Days.json`. Výsledná tabulka má reprezentovat závislost výroby výrobků továren na jednotlivých dnech v roce.

```
[
  {
    "factory": {
```

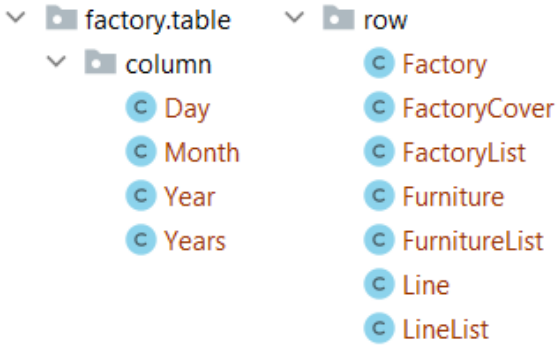
```
        "name": "Factory 1",
        "productionLineList": [
    {
        "line": 1,
        "name": "linka A",
        "furnitureList": [
          {
            "name": "Coffee Table",
        "amount": 9,
        "day": 2,
        "month": 2,
        "year": 2021
    },
```

Zdrojový kód A.3: Ukázka struktury JSON souboru `Factory`

```
"name": 2021,
"months": [
{
    "name": " February",
    "id": 2,
    "days": [
      {
        "name": "1.2.2021",
        "id": 1
      }
    ]
},
{
    "name": "January",
    "id": 1,
    "days": [
    {
        "name": "1.1.2021",
        "id": 1
    }
}
```

Zdrojový kód A.4: Ukázka struktury JSON souboru `Days`

Pro možnost parsování hodnot musíme vytvořit třídy reprezentující hodnoty řádků a sloupců ve výsledné tabulce.



Obrázek A.1: Vytvořené třídy pro parsování JSON souborů

```
public class Factory {
    @Name
    private String name;
    @SerializedName(value = "productionLineList")
    @TableElement(TableType.CHILDREN_LIST)
    private List<Line> lineList;
}
```

Zdrojový kód A.5: Třída `Factory` pro parsování JSON souboru A.3

Třída `Factory` má dva privátní atributy, `name` a `List<Line> lineList`. Atribut `name` je povinný údaj (je třeba využít anotaci `@Name`) slouží jako identifikátor jednotlivých hodnot. Anotace zařizuje zmíněnou identifikaci a přebírá hodnotu z JSON souboru A.3 `"name"`, která se zobrazuje v řádcích tabulky.

Druhý atribut využívá anotaci `@SerializedName`, která udává programu vyhledávání dle jiného zvoleného řetězce (`lineList`). Dle parametru anotace je v JSON souboru vyhledána hodnota `"productionLineList"` a mapuje se do zmíněného `lineList`. Pokud by se atribut jmenoval identicky vůči JSON souboru (`private List<Line> productionLineList`), tuto anotaci by nebylo třeba využít. Druhá anotace tohoto atributu značí relaci mezi řádky, konkrétně se přiřazuje seznam linek `List<Line>` jako seznam dětí pro každou hodnotu `Factory`. Třídu line je třeba také implementovat.

```
public class Line {
    @Name
```

```
    private String name;

    @TableElement(TableType.TOOLTIP)
    private int line;

    @TableElement(TableType.CHILDREN_LIST)
    private List<Furniture> furnitureList;
}
```

Zdrojový kód A.6: Třída `Line` reprezentující děti třídy `Factory`

Tato třída využívá anotaci `@Name` identicky jako třída `Factory`, stejně tak je vytvořen seznam dětí, tentokrát jsou děti reprezentovány třídou `Furniture`. Oproti třídě `Factory` je zde využita `@TableElement(TableType.TOOLTIP)` anotace, která zajišťuje zobrazení textu nápovědy.

```
public class Furniture {
    @Name
    private String name;

    @TableElement(TableType.VALUE)
    private int amount;

    @Relational(equals = true, targetClass = Day.class)
    private Integer day;

    @Relational(equals = true, targetClass = Month.class)
    private Integer month;

    @Relational(equals = true, targetClass = Year.class)
    private Integer year;
}
```

Zdrojový kód A.7: Třída `Furniture` reprezentující děti třídy `Line`

Tato třída reprezentuje také hodnoty řádků, navíc reprezentuje relaci na sloupce dle výsledků. Obsahuje anotaci `@TableElement(TableType.VALUE)`, značící hodnoty, které se budou reprezentovat ve vnitřní části tabulky. Tyto hodnoty musí mít nějakým způsobem udělanou návaznost na sloupce (nyní nevíme, do kterého sloupce by se hodnota měla zařadit), k tomu slouží

druhá anotace `@Relational`. Název atributu značí opět hodnotu v JSON souboru. Víme tedy, že pro relaci `month` se ze souboru A.3 bude ukládat hodnota `"month"`. Parametr relace `targetClass = Month.class` reprezentuje, dle které třídy se budou relace vytvářet. Než se dostaneme k struktuře tříd reprezentující sloupce, je třeba vysvětlit třídy `FactoryCover` a `FactoryList`. Jak je vidět v ukázce A.3 JSON data jsou obalována dvěma závorkami `[` značící pole a `{` značící výčet hodnot.

```
@Cover
public class FactoryCover {
    @CoverElement
    private Factory factory;
}
```

Zdrojový kód A.8: Třída `FactoryCover`

Proto třída `FactoryCover` slouží k získání dat extrahováním ze závorek v JSON souboru A.3. Slouží tedy pouze jako obalovací třída.

```
public class FactoryList extends ArrayList<FactoryCover>
```

Zdrojový kód A.9: Třída `FactoryList`

`FactoryList` slouží k ukládání hodnot jako seznamu. Konkrétně se zde ukládají všechny hodnoty `"factory"` JSON souboru A.3. Na stejný princip fungují i třídy `FurnitureList` a `LineList`.

Třídy pro reprezentaci sloupce jsou anotovány stejným způsobem jako hodnoty pro řádky. Všechny tři třídy ovšem obsahují anotaci `@Relational` aby bylo možné mapovat výsledné hodnoty do vnitřní části tabulky.

```
public class Year {

 @Name
 @Relational(targetClass = Furniture.class, equals = true)
 private int name;

 @TableElement(TableType.CHILDREN_LIST)
 private List<Month> months;
 }

 public class Day {
```

```
@TableElement(TableType.TOOLTIP)
@Name
private String name;

@Relational(targetClass = Furniture.class, equals = true)
private int id;
}
```

Zdrojový kód A.10: Třídy reprezentující hodnoty sloupců JSON souboru

Třída `Years` funguje na principu třídy `FactoryList`.

Generování HTML kódu

Za využití předané knihovny `TableGenerator` je dále vytvářen výsledný HTML kód. Nejprve jsou vytvořeny oba listy `years` a `factories` reprezentující sloupce a řádky. Následně je mezi nimi pomocí anotací vytvořena relace a vytvořena abstraktní reprezentace tabulky. Z té je nakonec generován výsledný HTML kód.

```
JsonParser<FactoryList> parser = new JsonParser<FactoryList>();
FactoryList factories = parser.parseFile(factoryPath, FactoryList.class);
Years years = new JsonParser<Years>().parseFile(daysPath, Years.class);

IRelational<FactoryList, Years> relational = new AnnotationRelation<>();
Table table = relational.makeRelation(factories, years);
TableGenerator tableGenerator = new TableGenerator();
tableGenerator.generateTable(table, tablePath + "/f.html", true);
```

Zdrojový kód A.11: Vytváření HTML výsledného souboru

Nastavení CSS stylu

Uživatel má možnost přidat vlastní stylizovací CSS jazyk. Kód zajistí generování CSS stylu v adresáři společně s výsledným HTML souborem.

```
tableGenerator.addCustomCss("factory.css");
InputStream cssSrc = TableGenerator.class.getResourceAsStream("/factory.css");
if (cssSrc != null) {
Files.copy(cssSrc, Paths.get(tablePath + "/factory.css"),
StandardCopyOption.REPLACE_EXISTING);
```

Zdrojový kód A.12: Kód pro vytvoření návaznosti na CSS stylu `factory`

A.3 Předání parametrů

Obě aplikace mají k dispozici JAR soubory pro spuštění aplikace. Pro OKS aplikaci je nejprve potřeba zadat parametr udávající zda chceme spustit aplikaci ve formě CLI nebo GUI

A.3.1 Parametry obecné aplikace

```
-inputMode factories -factory FACTORY_FILE -days DAYS_FILE
-target TARGET_FOLDER

-inputMode rqmTc -rqm RQM_FILE -tc TC_FILE -target TARGET_FOLDER
```

Zdrojový kód A.13: Dva způsoby zadávání vstupních dat v CLI formě obecné aplikace

Aplikace reprezentující obecnost má pouze CLI formu, data jsou předávána pomocí identifikátorů.
Parametry:

- factory (JSON) – reprezentuje hodnoty řádků pro závislost továren na dnech
- days (JSON) – reprezentuje hodnoty sloupců pro závislost továren na dnech
- rqm (JSON) – reprezentuje hodnoty požadavků předávané jako hodnoty sloupců při závislosti TC na RQM
- tc (JSON) – reprezentuje hodnoty testovacích případů jako hodnoty řádků při závislosti TC na RQM
- target (filePath) – značí cestu, kde se vygeneruje výsledný HTML soubor

A.3.2 Parametry OKS aplikace

```
java -jar oks_app.jar -cli
java -jar oks_app.jar -gui
```

Zdrojový kód A.14: Počáteční parametry aplikace OKS

Pokud uživatel zvolí GUI formu, není potřeba zadávat žádné další parametry a spustí se vizuální aplikace pro zadání parametrů. CLI forma vyžaduje zadání vstupních dat v jednom příkazu.

```
java -jar oks-app.jar -cli -appMode oks -inputMode -tc TC_FILE -useCase
UC_FILE -rqm RQM_FILE -target TARGET_FOLDER -results RESULT_FILES

java -jar oks-app.jar -cli -appMode uis -inputMode tc -logs LOG_FILE
-useCase USE_CASE_FILE -rqm RQM_FILE -target TARGET_FOLDER
```

Zdrojový kód A.15: Dva způsoby zadávání vstupních dat v CLI formě aplikace OKS

První parametry udávají o jaký typ vygenerované tabulky půjde – tabulka pro OKS či tabulka pro projekt TbUIS. Dle vybraného `-appMode` parametru (rqm, tc) jsou zadávány další parametry.
Parametry:

- rqm (JSON) – pokud jde o generování typu RQM, udává tento JSON hodnoty řádků, pokud jde o typ TC, předává datům TC hodnoty description
- tc (JSON) – pokud jde o generování typu TC, udává hodnoty řádků, pokud jde o RQM, předává datům RQM hodnoty (passed / failed)
- results (CSV) – složka v které je možné jako jediné vybrat více položek, soubory ve formátu csv, exportované z programu Squash, z kterých se parsují výsledky tabulky
- useCase (TXT) – značí pro všechny typy dat hodnoty sloupců
- target (filePath) – značí cestu, kde se vygeneruje výsledný HTML soubor
- logs (LOG) – značí výsledky pro aplikace TbUIS projektu

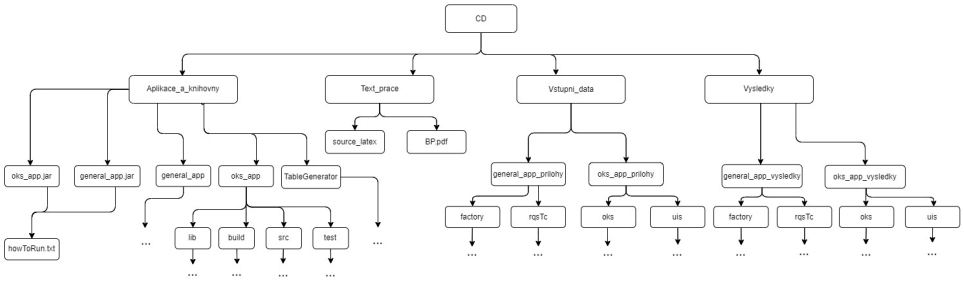
Pokud uživatel nepředá parametry, vypíše se metoda `help()`, která napovídá jak s aplikací zacházet (které parametry vložit), pořadí parametrů není důležité, je pouze potřeba využít správné názvy identifikátorů. A předat jako první parametr zda se jedná o CLI, nebo GUI formu. Společně s HTML kódem se generují ve stejném adresáři JavaScript a CSS soubory nutné ke správnému zobrazení tabulky, je proto dobré vytvořit si nový adresář.

Aby uživatel nebyl zahlcen hláškami v CLI formě, vytvoří se LOG soubor ve stejném adresáři jako byl spuštěn JAR soubor aplikace. V něm se nachází veškeré informace o generování výsledku. Pokud by někde nastala chyba, lze zjistit kde se informace o stavu generování přestaly vypisovat do daného souboru a případně je vypsána výjimka pokud nějaká nastala.

B Obsah CD

B.1 Struktura obsahu CD

Na obrázku B.1 je uvedena základní adresářová struktura obsahu CD. V této struktuře nejsou uvedeny všechny adresáře, struktura zobrazuje pouze hlavní adresáře.



Obrázek B.1: Základní adresářová struktura obsahu CD

B.2 Popis obsahu CD

B.2.1 Aplikace_a_knihovny

Adresář obsahuje tři další adresáře se zdrojovými soubory.

- oks_App:
  - obsahuje zdrojové soubory aplikace generující HMTL kód dat z předmětu OKS a projektu TbUIS,
  - zdrojové soubory, lib adresář, test adresář, pom.xml.
- general_App:
  - obsahuje zdrojové soubory aplikace generující HTML kód obecných hierarchických formátů,
  - zdrojové soubory, lib adresář, build.gradle.
- TableGenerator:

  - obsahuje zdrojové soubory programu, který zajišťuje generování HTML kódu a je využíván oběma aplikacemi,
  - zdrojové soubory, test adresář, build.gradle.
- spustitelné JAR soubory aplikací:
  - názvy: `general_app.jar`, `oks-app.jar`.
- textový soubor s příkazy pro spuštění JAR souborů:
  - název: howToRun.txt.

B.2.2 Text_prace

Adresář obsahuje podadresář se zdrojovými (`.tex`) soubory a výsledný PDF soubor.

- Zdrojové texty bakalářské práce:
  - adresář se zdrojovými texty a obrázky bakalářské práce potřebné k vygenerování výsledného dokumentu,
  - název: source_latex .
- Dokument s bakalářskou prací:
  - dokument s bakalářskou prací ve formátu `.pdf` včetně příloh,
  - název: BP.pdf.

B.2.3 Vstupni_data

Je rozdělen na adresáře dle aplikací: general_app_prilohy, oks_app_prilohy.

- general_app_prilohy:
  - obsahuje soubory vstupních dat aplikace pro obecné JSON soubory,
  - JSON soubory.
- oks_app_prilohy:
  - obsahuje soubory vstupních dat aplikace pro předmět OKS, projekt PSTSP a projekt TbUIS,
  - JSON soubory, CSV soubory, TXT a LOG soubory.

B.2.4 Vysledky

Adresář je rozdělen identicky vůči adresáři Vstupni_data. Konečné adresáře obsahují výsledné výstupy z aplikací.

- general_app_vysledky:
  - obsahuje soubory pro zobrazení výsledků aplikace pro obecné JSON soubory,
  - HTML soubor, CSS a JavaScript soubory.
- oks_app_vysledky:
  - obsahuje soubory pro zobrazení výsledků aplikace pro předmět OKS a projekt TbUIS,
  - HTML soubor, CSS a JavaScript soubory.