VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

FRONTEND PRO GENERÁTOR TESTOVACÍCH DAT

FRONT-END FOR GENERATOR OF TEST DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR FREYBURG**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Freyburg Petr**
Program: Informační technologie
Název: **Frontend pro generátor testovacích dat**
Front-End for Generator of Test Data
Kategorie: Web

Zadání:

1. Nastudujte technologie pro tvorbu webových aplikací.
2. Navrhněte infrastrukturu klient-server pro tvorbu testovacích dat využívající projekt Gestr v platformě Testos. Navrhněte rozhraní REST pro tuto infrastrukturu.
3. Implementujte webové rozhraní pro projekt Gestr.
4. Správnost funkcionality podpořte automatizovanými testy.

Literatura:

- Domovská stránka projektu Gestr. https://pajda.fit.vutbr.cz/testos/gestr
- Žára, O.: JavaScript - Programátorské techniky a webové technologie. Computer Press, 2015.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Cílem této bakalářské práce je webová aplikace pro generování strukturovaných testovacích dat pro účely testování software. Projekt integruje existující generátor testovacích dat Gestr. Součástí tohoto projektu je aplikační rozhraní REST a webový frontend. Nástroj je implementován v jazyce Python a ve frameworku Flask, uživatelské rozhraní je realizováno pomocí HTML, CSS a JavaScriptu. Aplikace umožňuje připravit vstupy pro generátor a spustit generování.

Abstract

The goal of this bachelor thesis is to create web application for generating structured test data for software testing. The project integrates existing test data generator Gestr. The project includes creating REST application interface and web frontend. The tool is implemented using Python and Flask framework, user interface is realized by HTML, CSS and JavaScript. Application can prepare inputs for generator and start generating.

Klíčová slova

generátor testovacích dat, testování založené na vstupních doménách, REST API, frontend, backend, webová aplikace, JavaScript, Flask, Docker

Keywords

test data generator, input domain testing, REST API, frontend, backend, web application, JavaScript, Flask, Docker

Citace

FREYBURG, Petr. *Frontend pro generátor testovacích dat*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Frontend pro generátor testovacích dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Freyburg
27. května 2020

Poděkování

Rád bych poděkoval vedoucímu práce, panu Ing. Aleši Smrčkovi, Ph.D. za čas a cenné rady, které mi poskytl při vypracování této bakalářské práce.

Obsah

Kapitola 1

Úvod

Tato bakalářská práce se zabývá vytvořením uživatelského rozhraní pro existující generátor testovacích dat, kterých vznikl v rámci bakalářské práce Ondřeje Olšáka [16]. Tento generátor generuje strukturovaná testovací data ve formátu JSON nebo XML. Výstupy generátoru lze použít například pro testování informačních systémů.

Generátor testovacích dat *Gestr* je konzolová aplikace implementovaná v jazyce `Python`, jejímž vstupem je abstraktní strom, který popisuje požadovanou strukturu výstupních souborů. Tento abstraktní strom je ve formátu JSON a práce s ním nemusí být vždy zcela přívětivá. Tento nástroj si klade za cíl umožnit snadnou práci se vstupy nástroje a nástrojem obecně.

Vytvořený nástroj lze rozdělit do dvou částí. První část nástroje je aplikační rozhraní `REST`, které umožňuje zadat generování a poté si stáhnout jeho výsledky. Tato část je jako samotný generátor implementována v jazyce Python. Druhou částí je webová aplikace, která umožňuje snadno upravit vstupy generátoru a nastavit parametry generování, ta je implementována pomocí Javascriptu s využitím HTML a CSS. Tyto dvě části mezi sebou komunikují prostřednictvím aplikačního rozhraní.

Stejně jako zmíněný generátor práce vznikla jako součást platformy Testos [5], která vzniká na Fakultě informačních technologií VUT v Brně. Cílem tohoto projektu je vytvořit sadu automatických testovacích nástrojů. Kombinuje v sobě nástroje pro různé úrovně a druhy testování. Nástroj vytvářený v této práci spadá do kategorie testování založené na datech.

Kapitola 2 popisuje testování založené na vstupních doménách, existující nástroje pro generování testovacích dat, použité technologie a návrh nástroje na základě analýzy požadavků s návrhem komunikace a architekturou aplikace, včetně návrhu webového frontendu. Kapitola 3 popisuje implementační obou částí aplikace – nejdříve serverové části a poté webového frontendu. Poslední kapitola 4 seznamuje čtenáře s ověřením funkčnosti nástroje.

Kapitola 2

Návrh nástroje pro generování testovacích dat

Cílem této kapitoly je seznámit čtenáře s testováním založeném na vstupních doménách, se strukturovanými daty, s nástrojem Gestr, s existujícími řešeními generování testovacích dat, s použitými technologiemi, analyzovat požadavky a na základě zmíněných informací představit návrh našeho nástroje.

2.1 Testování založené na vstupních doménách

Nástroj Gestr umožňuje generovat se třemi typy pokrytí, které si v následujícím textu popíšeme. Jedná se o typy pokrytí Each-Choice Coverage, Base Choice Coverage a Pair-Wise Coverage.

Informace v této podkapitole byly převzaty z knihy *Introduction to software testing* [11]. Vstupní doména je množina všech vstupů testovaného systému, což mohou být například vstupní parametry metod, vstupy od uživatele nebo třeba globální proměnné. Vstupní domény jsou rozděleny do oddílů, které obsahují stejně užitečné hodnoty z hlediska testování. Oddíly jsou si vzájemně disjunktivní.

Nyní budou vysvětleny na příkladu jednotlivá kritéria pokrytí. Mějme tyto tři oddíly, obsahující bloky:

- $A = [a, b]$,
- $B = [1, 2, 3]$,
- $C = [x, y]$.

Each-Choice Coverage

Pokrytí Each-Choice Coverage znamená, že hodnota z každého bloku musí být použita pro každou charakteristiku alespoň jedenkrát. Toto pokrytí splňuje například testovací sada $T = \{t_1, t_2, t_3\}$.

- $t_1 : (a, 1, x)$,
- $t_2 : (b, 2, x)$,
- $t_3 : (c, 3, y)$

Base-Choice Coverage

Při pokrytí Base-Choice Coverage je vybrán bázový blok pro každou charakteristiku a bázový test je vytvořen použitím každé z vybraných charakteristik. Ostatní testy jsou vytvořeny vždy kombinací jednoho nebázového bloku s bázovými bloky. Z našeho příkladu zvolme bázový blok $(a, 1, x)$. Ostatní testy potom budou:

- $t_1 : (b, 1, x)$,
- $t_2 : (a, 2, x)$,
- $t_3 : (a, 3, x)$,
- $t_4 : (a, 1, y)$

Pair-Wise Coverage

Při pokrytí Pair-Wise Coverage je vyžadováno, aby každý blok každé charakteristiky byl zkombinován s párem bloků jiných charakteristik. PWC musí pokrýt následujících 16 kombinací:

$$\begin{array}{cccc} (A,1) & (B,1) & (1,x) & (1,y) \\ (A,2) & (B,2) & (2,x) & (2,y) \\ (A,3) & (B,3) & (3,x) & (3,y) \\ (A,x) & (A,y) & (B,x) & (B,y) \end{array}$$

Výsledná testovací sada bude nakonec vypadat, jak je uvedeno níže. Na místech, kde je znak $-$ se zvolí blok libovolně.

$$\begin{array}{ccc} (A,1,x) & (B,1,y) & (A,-,y) \\ (A,2,x) & (B,2,y) & (B,-,x) \\ (A,3,x) & (B,3,y) & \end{array}$$

Další kritéria pokrytí

Dalšími možnostmi, které však již Gestr nepoužívá je All Combination Coverage, což znamená použití kombinací všech bloků ze všech oddílů a T-Wise Coverage, od kterého je odvozeno Pair-Wise Coverage, kde místo pokrytí všech dvojic, je požadováno pokrytí všech T-tic.

Příklad pokrytí All Combination Coverage:

$$\begin{array}{cccc} (A,1,x) & (A,1,y) & (B,1,x) & (B,1,y) \\ (A,2,x) & (A,2,y) & (B,2,x) & (B,2,y) \\ (A,3,x) & (A,3,y) & (B,3,x) & (B,3,y) \end{array}$$

2.2 Strukturovaná data

Nyní se seznámíme se strukturovanými daty. Strukturovaná (nebo někdy též *serializovaná*) data je formát dat sloužící k výměně dat. Jedná se o reprezentaci dat v jednom textovém řetězci. Jako jeho zástupce můžeme uvést XML, JSON nebo YAML. My se zaměříme na první dva, neboť právě s těmi generátor strukturovaných dat Gestr pracuje, respektive je produkuje.

Formát XML

XML (*Extensible Markup Language*) je formát serializace dat, který byl vyvinut v roce 1996 jako podmožina jazyka SGML. XML dokument je tvořen jedním nebo více elementy. Elementy je možné do sebe zanořovat. Syntakticky je velmi podobný s dokumenty v jazyce HTML.

```
<?xml version="1.0"?>
<greeting>Hello, world!</greeting>
```

Výpis 2.1: Příklad jednoduchého dokumentu XML.

Ve výpisu 2.1 je ukázka jednoduchého dokumentu v jazyce XML. Na první řádku je deklarace, že jedná o jazyk XML a o jakou jeho verzi. Na dalším řádku je již jednoduchý element `greeting` s obsahem „Hello, world!“. Každý takový element může obsahovat i atributy, přičemž u jednoho elementu se nesmí opakovat atributy stejného označení.

Speciálním typem elementu je tzv. prázdný element. Takový prázdný element je znázorněn ve výpisu 2.2, tento element v sobě neobsahuje žádný text, ale obsahuje atribut `text`, který obsahuje „Hello, world!“. Prázdný element však na konci musí obsahovat znak `/`, tak aby se nezaměnil s běžným (neprázdným) element.

```
<greeting text="Hello, world!" />
```

Výpis 2.2: Příklad prázdného elementu v dokumentu XML.

Převzato z *dokumentace jazyka XML* [3].

Formát JSON

JSON (*JavaScript Object Notation*) je novější formát pro výměnu dat, je založen na podmnožině programovacího jazyka JavaScript. Oproti jazyku XML má úspornější syntaxi.

JSON je založen na dvou strukturách: kolekce párů název/hodnota a seřazený seznam hodnot. V tomto jazyce jsou tyto dvě struktury realizovány s využitím následujících konstrukcí:

- `Objekt` – neuspořádaná množina párů název/hodnota, samotný objekt začíná znakem `{` a končí znakem `}`, dvojice název/hodnota se zapisují ve tvaru `název:hodnota`,
- `Pole` – seřazená kolekce hodnot, začínající znakem `[` a končící znakem `]`, hodnoty jsou v poli odděleny znakem čárka,
- `Hodnota` – řetezec uzavřený do dvojitých uvozovek, číslo, true, false, null, objekt nebo pole,
- `Řetězec` – textový řetězec o délce 0 nebo více znaků ohraničený dvojitými uvozovkami (znak `"`).

```
{
   "greeting":"Hello, world!",
   "array":["a", "b", "c"]
}
```

Výpis 2.3: Příklad jednoduchého JSON dokumentu.

Ve výpisu 2.3 je ukázka jednoduchého JSON dokumentu s jedním objektem, co obsahuje dvě dvojice název/hodnota. První tento pár obsahuje jako hodnotu textový řetězec a druhý pár obsahuje jako hodnotu pole.

Převzato z *oficiální stránek jazyka JSON* [7].

2.3 Generátor testovacích dat

Tato podkapitola se zabývá popisem nástroje Gestr [4] a zejména skutečnostmi, které je potřeba v návrhu samotného frontendu zohlednit. Je zde popsán abstraktní strom, možné způsoby tvorby hodnot a v neposlední řadě také konfigurační soubor.

Implementační detaily nástroje Gestr

Nástroj Gestr je implementovaný v jazyce `Python3`, integruje několik modulů, které vznikly v rámci platformy Testos [5]. Jedná se o nástroje Combine[1], Dbgenx[2] a Combine-BCC[3]. Nástroj Dbgenx slouží k samotné tvorbě hodnot, Combine se v programu využívá k tvorbě pokrytí PWC a Combine-BCC k tvorbě pokrytí BCC.

Popis abstraktního stromu

Abstraktní strom je vstupem nástroje Gestr a definuje strukturu výsledných testovacích sad souborů. Tento popis je ve formátu JSON a obsahuje několik typů uzlů.

```
{
   "type": "O",
   "children": []
}
```

Výpis 2.4: Příklad uzlu typu Objekt.

Prvním typem uzlu je Object, značí se O. Představuje objekt a vždy jej následuje uzel typu klíč. Tento uzel je znázorněn ve výpisu 2.4. Jedná se vždy o první uzel stromu. Další uzly jsou klíč (K), ten popisuje klíč ve výsledném souboru, pole (A)a hodnota (V). Poslední typem uzlu je variantní uzel (R), který značí, že daný uzel může nabývat více variant, které jsou seskupeny v poli `children`.

Každý uzel obsahuje atribut `type`, který specifikuje typ uzlu a pole `children`, které obsahuje potomky daného uzlu. Uzel typu klíč obsahuje navíc atribut `keyId`, který specifikuje název klíče. Hodnotový uzel navíc obsahuje pole `tags`, které obsahuje datové typy, které se využívají při tvorbě hodnot typu `tags`.

Dále je třeba u uzlů počítat s omezeními, které se týkají potomků jednotlivých uzlů, které jsou uvedeny v tabulce 2.1.

Konfigurační soubor Dbgenx

Dalším vstupem nástroje Gestr je konfigurační soubor pro nástroj Dbgenx. Jedná se o vstup pro modul, který se stará o samotné vytváření hodnot. Tento vstup je nepovinný. V případě,

[1] https://pajda.fit.vutbr.cz/testos/combine
[2] https://pajda.fit.vutbr.cz/testos/dbgenx
[3] https://pajda.fit.vutbr.cz/testos/combine-bcc

	O	K	A	V	R
O		×			
K	×		×	×	×
A	×		×	×	×
V					
R	×		×	×	

Tabulka 2.1: Uzel v prvním sloupci a označení, které uzly mohou být jeho následníky [16].

že není zadán, používá se výchozí konfigurační soubor. Soubor je ve formátu JSON, specifikuje datové typy a omezení nad nimi, dále soubor může obsahovat schéma databáze, textové řetězce či možnost vložit do konfiguračního souboru `dataset`[4] [14]. Při použití v nástroji Gestr jsou povinné pouze definice datových typů a omezení nad nimi.

Definice datových typů se v konfiguračním souboru značí jako **types**, jedná se o asociativní pole, kde klíčem je název datového typu a hodnotou je samotná definice typu. Ukázka takové definice datových typů s jedinou definicí je ve výpisu 2.5.

```
"types": {
   "int_neg": {
     "int_neg": "int"
   }
}
```

Výpis 2.5: Ukázka definice datového typu záporného čísla.

Omezení se značí **constraints**, což je pole, které obsahuje výčet jednotlivých omezujících pravidel. Příklad pole omezení s jedním omezením je ve výpisu 2.6.

```
"constraints": [
   "int_neg.int_neg < 0"
]
```

Výpis 2.6: Ukázka definice omezení datového typu záporného čísla.

Způsob tvorby hodnot

Posledním parametrem, který přijímá nástroj Gestr, je způsob tvorby hodnot. Jsou to tyto dva způsoby: `domains` a `tags`.

Při způsobu tvorby hodnot `tags` jsou podle značek v poli `tags`, které obsahuje hodnotový uzel abstraktního stromu, vygenerovány hodnoty.

U `domains` jsou vygenerovány hodnoty podle určujícího datového typu, který je opět určený v hodnotovém uzlu, v poli značek. Hodnoty jsou vygenerovány tak, aby byly pokryty všechny domény daného datového typu.

Výstupní formát

Výstupní formát nástroje Gestr je buď JSON a nebo XML, přičemž JSON je výchozí. V případě volby formátu XML je třeba, aby abstraktní strom obsahoval klíč `data`.

[4] **Dataset** – předpřipravená data pro generování

Shrnutí vstupních parametrů nástroje Gestr

V tabulce 2.2 jsou shrnuty vstupní argumenty programu Gestr. Lze si všimnout, že zde vystupuje ještě kritérium *single file*. Toto kritérium je použito ve chvíli, kdy není vybráno žádné jiné kritérium a znamená, že bude vygenerován pouze jeden soubor.

	Možnosti			
Kritérium pokrytí	ECC	BCC	PWC	single file
Způsob tvorby hodnot	tags	domains		
Výstupní formát	JSON	XML		

Tabulka 2.2: Vstupní parametry nástroje Gestr.

2.4 Existující nástroje pro tvorbu testovacích dat

V této podkapitole budou popsány některé existující nástroje, které slouží k tvorbě strukturovaných testovacích dat. Všechny tyto nástroje jsou implementovány jako webové aplikace.

Cílem této podkapitoly je najít nedostatky existujících nástrojů a těm se ve výsledném produktu vyhnout a naopak odhalit i silné stránky, které by bylo možné dále rozvíjet.

json-generator.com

Služba json-generator.com[5] slouží ke generování testovacích dat ve formátu JSON. Editor je pouze textový, nelze provádět interaktivní úpravy, což může práci s nástrojem zkomplikovat, neboť složitější JSON není příliš lidsky čitelný. Ovšem na druhou stranu občas může být práce přímo v textové reprezentaci jednodušší a rychlejší. Oba tyto směry by se měly ve výsledném nástroji dát uživateli k dispozici.

Při práci v tomto nástroji se zapíše do textového editoru výsledný JSON, ale na místa hodnot se nezapisují hodnoty přímo, ale jejich definice, což může například být číselné rozmezí, datový typ nebo i definice na vyšší úrovni jako například jméno, přijmení, telefonní číslo nebo odstavec o stanoveném počtu slov.

generatedata.com

Služba generatedata.com[6] je aplikace, která umožňuje generovat data ve tvaru tabulky. Tato aplikace umí generovat stejně jako Gestr ve formátech JSON a XML, ovšem k tomu, ale také umí i SQL, HTML tabulky a data v různých programovacích jazycích (JavaScript, PHP, ...). Je zde možnost vybrat si ze širokého spektra datových typů. Jedná se ovšem spíše o nástroj určený k tvorbě ukázkových dat do SQL databází.

Oproti nástroji json-generator.com z podkapitoly 2.4 tato služba poskytuje pouze interaktivní editor, takže je jeho použití pohodlné a může tak otevřít dveře k použití i méně technicky zdatným uživatelům.

[5] https://www.json-generator.com
[6] https://generatedata.com

2.5 Použité technologie

Tato podkapitola popisuje technologie, které budou použity k tvorbě výsledného nástroje. V rámci implementace je třeba nejdříve vytvořit aplikační rozhraní pro práci s generátorem a až poté v další fázi vytvořit samotnou webovou aplikaci – k tomu poslouží níže zmíněné technologie.

Hlavní aplikace bude implementována v jazyku Python s použitím frameworku Flask. Samotná webová aplikace bude napsána v jazyku HTML, styl bude definován za pomoci jazyka CSS. Oba editory a ovládání generování, v části webového frontendu, budou ovládány jazykem JavaScript s použitím knihovny JQuery. Celá aplikace poběží v Docker kontejneru.

Python

Programovací jazyk Python[7] bude použit na implementaci hlavního programu. Důvodů je několik. Existující nástroje, jako Gestr je a jeho komponenty, jsou již v jazyku Python implementovány. Dalším důvodem pro volbu tohoto jazyka je zejména to, že se jedná o v dnešní době velmi populární jazyk [10].

Důvodem proč je Python, tak populární a dalším důvodem pro použití právě Pythonu v tomto projektu, je jeho jednoduchost [20]. Většinu požadavků lze v Pythonu zpracovat mnohem jednodušeji než v jiných moderních programovacích jazycích jako C++ nebo Java.

Flask

Framework Flask[8], který se používá k tvorbě webových služeb, umožňuje velmi jednoduše definovat adresy URI a jejich obsah (viz 2.7).

```
@app.route("/test")
def test():
   return "Hello"
```

Výpis 2.7: Příklad definice jednoduché URL ve Flasku.

Tato knihovna bude v projektu využita na tvorbu aplikačního rozhraní a též k webovému rozhraní, které však bude od zbytku oddělitelné. Tato knihovna v sobě zahrnuje nástroje na správu sezení a také na použití HTML šablon a statického obsahu.

HTML

HTML (*Hypertext Markup Language*) je značkovací jazyk sloužící k tvorbě webových stránek. V rámci tohoto projektu bude využit k tvorbě uživatelské části. Jazyk HTML vznikl na počátku 90. let minulého století, nejnovější verze jazyka je HTML5 [12]. HTML slouží k definici obsahu stránek.

Dokument v jazyce HTML se běžně skládá skládá z těchto komponent [12]:

- definice typu dokumentu – `DOCTYPE`,
- elementu `html` s volitelným atributem `lang`,
- elementu `head`, v němž jsou různé `meta` informace, které se přímo na webové stránce nezobrazují,

[7] https://www.python.org
[8] https://flask.palletsprojects.com/en/1.1.x/

- elementu `title`, jenž obsahuje titulek stránky,
- elementu `body`, který obsahuje jednotlivé zobrazované prvky webové stránky.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title></title>
</head>
   <body>
   </body>
</html>
```

Výpis 2.8: Základní HTML dokument.

Ve výpisu 2.8 je znázorněna základní struktura HTML dokumentu.
Pomocí jazyka HTML jsou sestaveny jednotlivé stránky webového frontendu.

CSS

CSS je jazyk, který slouží k definici vzhledu webových stránek. Nejnovější verze jazyka je CSS3 [12]. Původně definice vzhledu byly obsaženy v jazyku HTML. V dnešní době se již ke stylování webu používá pouze CSS, HTML5 plní pouze sémantickou funkci [12].

Pomocí CSS se definuje vzhled a pozice jednotlivých HTML elementů. Lze plošně určit pro daný typ elementů nebo pomocí selektorů pro určitou podskupinu elementů.

```
p {
   color: red;
}
```

Výpis 2.9: Příklad definice stylu v jazyce CSS.

Vždy je nejdříve určen prvek, ke kterému má být styl přiřazen a poté jsou ve složených závorkách určeny jednotlivá pravidla zobrazení prvků. Ve výpisu 2.9 je příklad definice stylu v jazyce CSS. Kód v tomto příkladu znamená, že všechny odstavce (html značka `p`) budou mít červenou barvu textu. Kdyby místo `p`, bylo napsáno například `.red`, znamenalo by to, že všechny prvky, které mají v HTML dokumentu tuto třídu, budou mít červenou barvu.

V projektu je použit na definici vzhledu jednotlivých prvků ve webovém frontendu.

JavaScript

JavaScript je programovací jazyk, který se používá při tvorbě webů. Narozdíl například od jazyka PHP běží na straně klienta, díky čemuž je možné vytvářet webové stránky s dynamickým obsahem a není nutné pokaždé znovu přenačítat webovou stránku. První verze JavaScriptu byla vytvořena v roce 1995, původním posláním jazyka byla validace údajů zadaných do HTML formulářů na straně klienta, avšak dnes se jedná o plnohodnotný programovací jazyk [21].

V tomto projektu je jazyk JavaScript použit k obsluze editoru, k interakci uživatele se stránkou formou různých hlášení a k voláním aplikačního rozhraní.

Samotná volání aplikačního rozhraní zajišťuje `Ajax` (*Asynchronous JavaScript and XML*). Hlavní výhody Ajaxu jsou [1]:

- odesílání požadavků na server bez nutnosti znovu načítat stránku,
- příjem dat ze serveru a následná práce s nimi.

Jquery

JQuery[9] je knihovna nad jazykem JavaScript. Jedná se o knihovnu podporovanou napříč moderními webovými prohlížeči, která přináší řadu funkcí, animací a zejména často zjednodušení oproti čistému jazyku JavaScript [2].

Na knihovně JQuery je postavena knihovna JQueryUI[10], která obsahuje prvky uživatelského rozhraní, efekty a dokonce i šablony. Z této knihovny jsou v tomto projektu použity například modální dialogová okna.

Bootstrap

Bootstrap[11] je knihovna pro tvorbu webů a webových aplikací. Tato knihovna obsahuje definice vzhledu v jazyce CSS pro různé elemenenty webových stránek. Také obsahuje různé pluginy v jazyku JavaScript.

Práce s touto knihovnou je snadná, zejména v případě použití v tomto projektu. Je třeba pouze přiřadit elementům odpovídající třídy.

V tomto projektu bude Bootstrap použit pro styl některých prvků jako jsou tlačítka, tabulky nebo hlášení pro uživatele.

Docker kontejnerizace

Docker[12] kontejnerizace je v dnešní době velmi rozšířená možnost distribuce a provozu aplikací. Kontejner v sobě obsahuje samotnou aplikaci a také její závislosti. Mnoho kontejnerů může běžet na jednom počítači a sdílet jádro operačního systému, jak je znázorněno na obrázku 2.1. Jedná se však o izolované procesy.

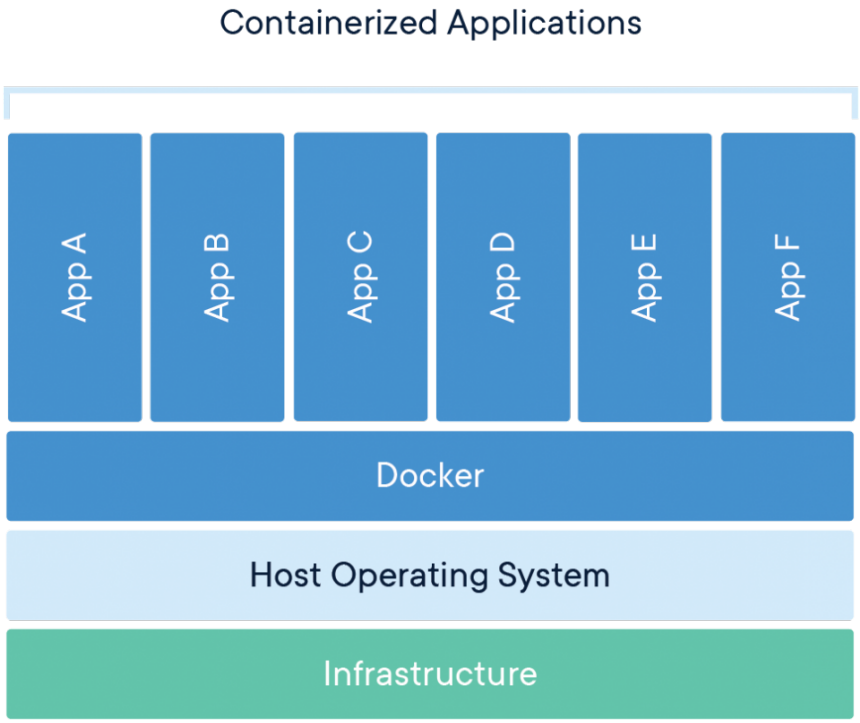
Kontejnerizace není virtualizace. Virtuální stroje pro svůj provoz potřebují mít v sobě uložený kompletní hostitelský operační systém, se všemi knihovnami a binárními soubory. Z tohoto je patrné, že kontejnerizace je úspornější na místo na disku na rozdíl od virtualizace. Převzato z *What is a Container?* [9].

Obrazem, ze kterého bude kontejner sestaven, je `Python`[13]. Po sestavení se doinstalují všechny závislosti, pak může být kontejner spuštěn.

2.6 Analýza požadavků

Tato podkapitola se zabývá analýzou zadání a také analýzou požadavků. Na základě těchto informací byl navrhnut výsledný nástroj.

[9] https://jquery.com
[10] https://jqueryui.com
[11] https://getbootstrap.com
[12] https://www.docker.com
[13] https://hub.docker.com/_/python



Obrázek 2.1: Docker kontejnerizace [9].

Vyhodnocení požadavků ze zadání

Cílem tohoto projektu je vytvořit nástavbu nad programem Gestr, která umožní uživatelsky přívětivé ovládání nástroje. Na druhou stranu se jedná o aplikaci, kterou budou používat testeři softwaru, a proto není cílem vytvářet žádnou velkou abstrakci nad tímto nástrojem.

Projekt lze rozdělit do dvou podprojektů:

- **serverová část** – obsluhuje nástroj Gestr, spravuje generování, sezení a též provádí automatickou údržbu diskového prostoru, poskytuje **aplikační rozhraní**,
- **frontend** – poskytuje uživatelské rozhraní, komunikuje se serverovou částí

Požadavky

Jednotlivé požadavky projektu jsou:

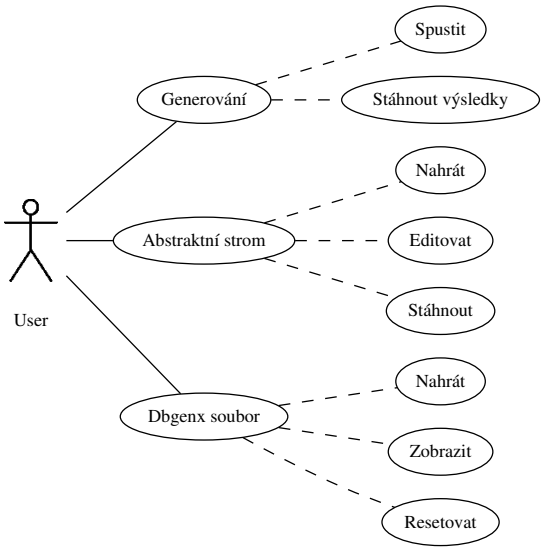
REQ1 Vstupem aplikace je abstraktní strom ve formátu json.

REQ2 Nepovinným vstupem aplikace je konfigurační soubor Dbgenx.

REQ3 Aplikace umožní uživateli vytvořit zcela nový abstraktní strom nebo upravovat stávající.

REQ4 Aplikace předá vstup uživatele programu gestr, který běží na serveru.

REQ5 V případě úspěšného generování pomocí nástroje Gestr předá aplikace uživateli výstup formou odkazu na archív zip s výsledky generování.



Obrázek 2.2: Diagram případů užití.

REQ6 Aplikace umí tyto kritéria pokrytí: PWC, ECC, BCC

REQ7 Aplikace umí tyto způsoby vytváření hodnot: tags, domains.

REQ8 Výstupy aplikace, které jsou součástí archívu zip, jsou ve formátu json nebo xml.

REQ9 Aplikace umožní uživateli generovat opakovaně s různými kritérii pokrytí, způsoby vytváření hodnot a různými konfiguračními soubory dbgenx.

REQ10 Aplikace umožní uživateli stáhnout si upravený abstraktní strom.

REQ11 Aplikace je neperzistentní.

REQ12 Aplikace informuje uživatele o přerušeném generování.

Je zřejmé, že požadavky **REQ1**, **REQ2** a **REQ6** – **REQ8** budou splněny, pokud implementace neomezí nástroj Gestr a nabídne jeho kompletní funkčnost.

Požadavek **REQ3** bude splněn tím, že frontendová část bude obsahovat editor abstraktního stromu. Požadavek **REQ4** bude vyřešen vytvořením aplikačního rozhraní typu REST.

Případy užití

Případy užití lze zobecnit do tří podskupin. První podskupina se týká generování, uživatel může spustit generování a poté si stáhnout vygenerované výsledky. Další podskupinou je práce s abstraktním stromem, který uživatel může nahrát, může upravovat abstraktní strom a také si jej může stáhnout. Poslední podskupinou je práce s konfiguračním souborem, kdy uživatel může nahrát vlastní, může si aktuální konfigurační soubor zobrazit a nakonec může nechat obnovit výchozí. Případy užití včetně jednotlivých podskupin jsou znázorněny na obrázku 2.2.

2.7 Komunikace

V této podkapitole bude popsána komunikace výsledného nástroje a role jednotlivých subsystémů. Jak již uvadí zadání tohoto projektu, aplikace bude komunikovat v módu *Klient-Server* a pomocí infrastruktury *REST*.

Klient-server komunikace

Tento typ komunikace je složen ze dvou na sobě nezávislých a autonomních procesů, klienta a serveru, kteří většinou komunikují po síti. Server poskytuje požadované služby pro klienta [19]. Ke komunikaci poslouží protokol HTTP.

Směrování (anglicky *Routing*) je mechanismus, jehož prostřednictvím jsou požadavky propojeny s určitým zdrojovým kódem [19].

Existují dva typy směrování – *server-side* a *client-side*. V případě server-side směrování je načtena celá jedna URL, jedním požadavkem, pokud uživatel potřebuje další data, musí přenačíst celou stránku. Hlavní nevýhodou tohoto přístupu tedy je, že každý požadavek znamená přenačíst celou webovou stránku. Pokud se jedná o client-side, požadavky jsou obsluhovány interně pomocí JavaScriptu. Nemění se celá stránka, ale pouze některé elementy. Nevýhodou může být náročnější implementace nebo to, že kompletní zdrojové kódy klienta a jeho knihoven musí být na začátku načteny ke klientovi, což může trvat déle. Výhodou však je rychlejší práce, neboť není nutné vždy znovu načítat kompletní webovou stránku. Převzato z *Server-side vs Client-side Routing* [17].

V rámci tohoto projektu je třeba zadefinovat role klienta a serveru:

- **Server** – zahrnuje v sobě generátor testovacích dat a přijímá požadavky na generování, stažení výsledných generovaných dat a také na změny konfiguračního souboru.
- **Klient** – přímo komunikuje s uživatelem. Mimo odesílání zmíněných požadavků pro Server, zahrnuje v sobě editor abstraktního stromu.

Server, díky svojí roli a návaznosti na projekt Gestr, bude implementován v jazyce Python s využitím knihovny Flask. Klient bude implementován jako webová stránka, která kromě jednoduché obsluhy odesílání požadavků na server bude obsahovat i zmíněný editor pro abstraktní strom.

Udržení kontextu

Protokoly HTTP a HTTPS jsou bezstavové. Je však nutné nějakým způsobem udržet kontext a identifikovat spolu související požadavky přicházející z jednoho místa, jako je například nejdříve nastavení konfiguračního souboru a poté generování. Z tohoto důvodu je třeba vždy při začátku komunikace ustanovit *sezení*. K tomu poslouží třída `session`, která je součástí knihovny Flask. Tato třída používá k udržení kontextu posílání podepsaných *cookies* [8].

Cookie je malý objem dat, který server posílá klientovi a jsou uloženy do klientského webového prohlížeče. Klient, pak tento malý objem dat, zahrnuje do všech následujících požadavků komunikace [6]. Tímto způsobem dochází k udržení kontextu.

V případě udržení sezení je pomocí Cookies vyměňováno id sezení (anglicky *session id*). Samotný managment sezení probíhá na straně serveru. Prakticky to znamená, že klient pošle první požadavek, server na tento požadavek odpoví a v této odpovědi zasílá klientovi

identifikátor sezení. Klient v dalších požadavcích posílá přiřazené id a tímto způsobem je udržen kontext.

Aplikační rozhraní REST

REST (*Representational state transfer*) je architektonický styl webových rozhraní. K přístupu ke zdrojům se používají URI (*Uniform Resource Identifiers*). Základním pravidlem je, že URI je přístup k jednomu a ne k více zdrojům. Jednotlivé URI adresy mohou představovat hierarchii jednotlivých zdrojů. Cílem je poskytnout standardizované a poměrně i lidsky čitelné rozhraní.

Pomocí klíčových slov protokolu HTTP `GET`, `POST`, `PUT` a `DELETE` je možné nad zdroji aplikace pomocí aplikačního rozhraní provádět úpravy, zobrazovat si je, přidávat a mazat. Informace v této podkapitole výše byly převzaty z knihy Marka Massého [15].

Mějme jednoduchý příklad na České republice a jejich krajích a městech. Mějme adresu na obrázku 2.10, která umožňuje pouze zobrazení. V případě otevření pouze části `/cr` se vypíší jednotlivé kraje, při otevření `/cr/jihomoravsky/` se zobrazí jednotlivá města v Jihomoravském kraji a při celé adrese se zobrazí například informace o městě, v tomto případě o Brně.

```
/cr/jihomoravsky/brno
```

Výpis 2.10: Příklad jednoduchého aplikačního rozhraní REST.

Vytvořené rozhraní musí umožnit zejména ovládání nástroje Gestr a přístup k výsledkům generování. V rámci tohoto projektu je třeba vytvořit aplikační rozhraní, které umožní:

- vytvořit generování,
- nahrát konfigurační soubor,
- obnovit výchozí konfigurační soubor,
- zobrazit aktuální konfigurační soubor,
- zobrazit stav generování,
- stáhnout výsledky úspěšného generování.

2.8 Architektura aplikace

Tato podkapitola se zabývá architekturou aplikace. Nejdříve budou shrnuty teoretické poznatky o architekturách webových aplikací a poté bude představena výsledná architektura.

Architektura webových aplikací

V dnešní době existují dva typy architektur webových aplikací – *monolytická architektura* a *mikroslužby*. Informace byly převzaty z *Monolithic vs. Microservices Architecture* [13].

Monolytická architektura je architektura, kdy celá aplikace funguje jako jeden celek. Přestože aplikace má uvnitř modulární strukturu, aplikace je zabalena do jednoho celku a je taky jako jeden celek nasazena. Výhodou tohoto přístupu je jednoduchost implementace, testování a nasazení. Naopak to ale přináší také jisté neduhy: aplikace je příliš velká

a komplexní na pochopení, což znamená, že provedení změn v implementaci nebude jednoduché a bude naopak časově náročné. Velikost aplikace může způsobit, že její vlastní start bude velmi pomalý. Problém může být i se spolehlivostí aplikace, chyba v některém modulu může způsobit selhání celé aplikace.

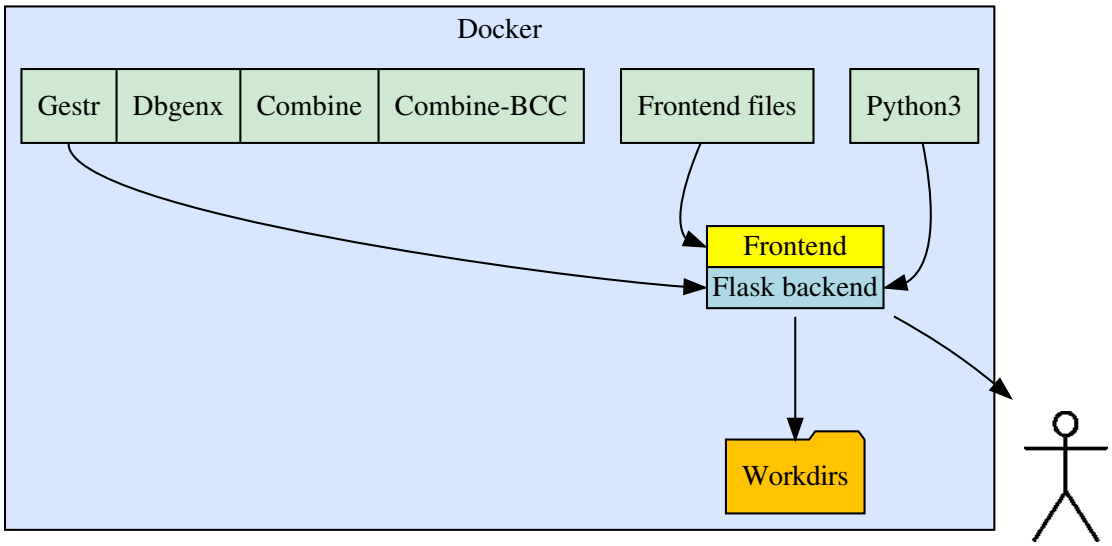
Mikroslužby se vydávají značně protichůdným směrem. Jedná se o rozdělení aplikace do několika služeb. Některé služby mohou například implementovat aplikační rozhraní REST, jiné třeba webové uživatelské rozhraní. Je tím vyřešen problém komplexnosti, který mohl nastat u monolytické architektury. Porozumět jedné službě a rozšířit ji, je takto mnohem snadnější. Problém s rozšířením implementace nastává, když je třeba udělat změnu, která prochází více službami. Nasazení může být složitější, neboť řešíme závislosti každé služby samostatně.

Na jednu stranu může vývoj mikroslužeb vypadat komplikovaněji a vyvolávat dojem, že výhodnějším přístupem je monolytická architektura. Na jednu stranu sice mikroslužby do jisté míry zesložití vývoj a testování produktu, ovšem zvýší jeho udržitelnost a rozšiřitelnost. Přínosem mikroslužeb též může být rozdělení vývoje mezi více skupin vývojářů, kdy každá tato skupina se zabývá vývojem nějaké služby pro daný nástroj.

Jak bylo zmíněno je třeba, v rámci tohoto projektu, implementovat dvě části – serverovou část a webový frontend. Výsledný nástroj bude tvořen z právě těchto dvou mikroslužeb. Ty budou moci pracovat samostatně, nezávisle na sobě nebo také dohromady jako celek a jeden nástroj. Jejich vývoj však bude do značné míry na sobě nezávislý.

Výsledná architektura

Na obrázku 2.3 je znázorněna architektura aplikace, která právě respektuje architekturu mikroslužeb. Hlavní částí je *Flask backend*, do kterého je naimportován i frontend. Díky šablonovému systému, který Flask má je možné definovat i jednotlivé URL pro frontend. *Frontend files* na obrázku jsou všechny HTML, CSS a Javascriptové soubory, které Flask vrací. Serverová část i webový frontend mohou běžet samostatně.



Obrázek 2.3: Architektura aplikace, běžící v Docker kontejneru, obsahující serverou část a webový frontend a jejich moduly.

Do backendu je importován Gestr se všemi jeho moduly (Combine, Combine-BCC a Dbgenx), backend na základě zaslání požadavku na příslušnou URL spouští generování a s tím spojené události si poté již Gestr řídí sám.

Pracovní adresáře (na obrázku Workdirs) jsou úložiště pro dané sezení. Při úvodním spojení je uživateli vytvořen pracovní adresář a do něj je mu nakopírován výchozí konfigurační soubor Dbgenx, ten si uživatel poté může změnit – splnění požadavku **REQ2** z podkapitoly 2.6. V tomto pracovní adresáři jsou poté uloženy všechny výsledky generování a též dočasné soubory, které při generování vznikají.

Celá aplikace běží v Docker kontejneru, port kontejneru je připojen port serveru 443[14].

2.9 Návrh webového frontendu

Při návrhu aplikace bylo jedním z hlavních cílů, vytvořit jednoduchou, uživatelsky přívětivou a minimalistickou aplikaci, která však splní všechny požadavky a zejména umožní uživateli používat nástroj Gestr bez omezení jeho funkčnosti, ale také mu umožní vytvářet a upravovat své abstraktní stromy. Opět takovým způsobem, aby pokud možno uživatel nebyl omezen a nástroj se mu při tom dobře a také pohodlně ovládal.

Na obrázku 2.4 je návrh webového frontendu. V levé části obrazovky se nachází interaktivní editor abstraktního stromu, který umožní měnit typy jednotlivých uzlů, v případě uzlu typu klíč změnit jeho identifikátor a také měnit datové typy pro daný atribut.

V pravé části se nachází textový editor, do kterého bude možné přenést některý uzel s jeho potomky a nad nimi provádět textové úpravy, které se poté opět přenesou do interaktivní reprezentace.

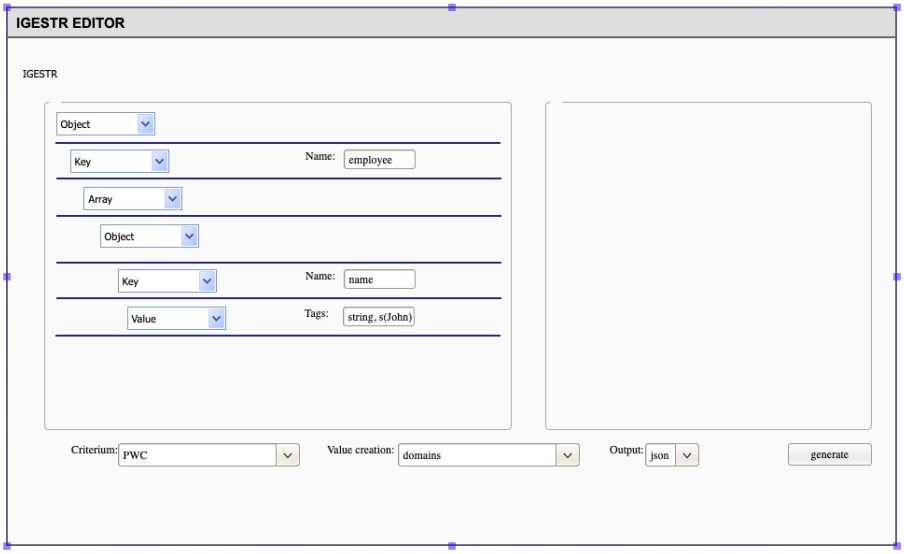
V části obrazovky pod editory je možné nastavit parametry pro dané generování:

1. zvolit kritérium pokrytí,
2. zvolit způsob tvorby hodnot,
3. zvolit výstupní formát.

Po zvolení těchto parametrů uživatel stiskne tlačítko `generate`, čímž zahájí generování. V případě úspěšného generování se v horní části obrazovky nad editory zobrazí hlášení s odkazem na zip archív s výsledky generování. Pokud generování skončí neúspěšně, ve stejné části obrazovky se bude zobrazovat chybové hlášení.

Obrázek ukazuje hlavní stránku webové aplikace. Aplikace bude ještě v sobě zahrnovat několik statických stránek s informacemi o nástroji a také manuálové stránky.

[14] výchozí port pro protokol HTTPS



Obrázek 2.4: Mockup frontendu.

Kapitola 3

Implementační detaily

Tato kapitola se zabývá implementačními detaily nástroje pro generátor testovacích dat. Nejdříve je v podkapitole 3.1 popsána implementace serverové části a poté v podkapitole 3.2 je popsán webový fronted aplikace.

Nejdříve si však přiblížíme jednotlivé soubory nástroje a také se podíváme na to, jak lze nástroj nainstalovat a spustit

Soubory aplikace

Na obrázku 3.1 jsou vypsány soubory aplikace. V této části budou soubory stručně představeny. Jejich detailnější popis je v následujících podkapitolách o implementačních detailech tohoto projektu. Obrázek nezobrazuje všechny soubory, obsah adresářů je popsán v tomto textu.

V adresáři `config/` je uložen výchozí konfigurační soubor pro Dbgenx. V případě dalšího rozšíření nástroje bude tento adresář v sobě zahrnovat další konfigurační soubory.

Adresář `static/` slouží pro uložení statických částí webového frontendu. Obsahuje podadresáře `css`, `js`, `lib` a `media`. Jak již názvy napovídají, první dvě složky obsahují CSS soubory a JavaScript soubory. Složka `media` obsahuje obrázky a adresář `lib` obsahuje soubory knihovny Jquery.

V adresáři `templates` jsou uloženy veškeré soubory HTML pro webový frontend.

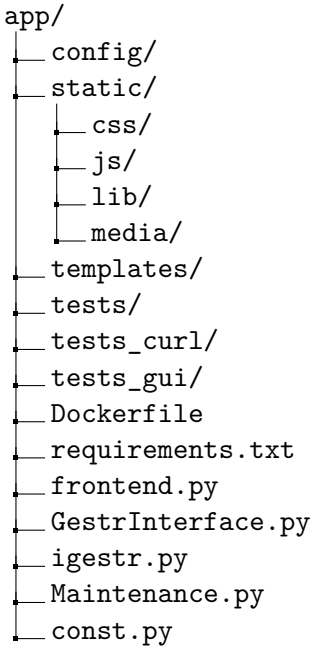
Adresáře `tests`, `tests_curl` a `tests_gui` obsahují automatizované testy aplikace. Tyto testy jsou blíže popsány v kapitole 4. Složka `tests` zahrnuje jednotkové testy serverové části, další adresář `tests_curl` obsahuje testy aplikačního rozhranní serverové části, které volají toto rozhraní prostřednictvím programu *Curl*.

`Dockerfile` slouží pro konfiguraci Docker kontejneru. Obsahuje instalace všech závislostí i samotného nástroje.

Velmi důležitým souborem je soubor `requirements.txt`. Tento soubor obsahuje všechny závilosti projektu, bez nichž by nástroj nebylo možné spustit.

Soubor `frontend.py` je modul, který slouží k přímému zahrnutí webového frontendu do serverové části.

Posledních čtyři soubory s příponou `py` obsahují impelementaci serverové části, `igestr.py` je hlavní soubor, obsahuje implementaci aplikačního rozhraní a volají se v něm příslušné další akce. V souboru `GestrInterface.py` se nachází stejnojmenná třída, která slouží jako rozhraní nad nástrojem Gestr. Soubor `Maintenance.py` také obsahuje stejnojmennou třídu, která slouží k automatické údržbě. Poslední `const.py` obsahuje definice některých konstant.



Obrázek 3.1: Soubory aplikace.

Instalace a spuštění nástroje

Existují dvě možnosti spuštění nástroje:

- pomocí Python virtuálního prostředí (anglická zkratka *venv*),
- pomocí Docker kontejnerizace.

```
> python3 -m venv --system-site-packages env
> source env/bin/activate
> pip install -r requirements.txt
> python3 igestr.py [-m] [-n]
```

Výpis 3.1: Spuštění pomocí virtuálního prostředí.

Ve výpisu 3.1 je znázorněno spuštění pomocí virtuálního prostředí. Nejdříve je třeba vytvořit virtuální prostředí, pak jej aktivovat. Dále je třeba nainstalovat potřebné závislosti pomocí zmíněného souboru `requirements.txt`. Nakonec se spustí samotný nástroj. Ten poté poběží na `localhostu`, na portu 5000.

```

> docker build --tag igestr .
> docker run -p 5000:5000 igestr:latest
```

Výpis 3.2: Spuštění pomocí Docker kontejnerizace.

Ve výpisu 3.2 je znázorněno spuštění pomocí Dockeru. Základním předpokladem pro toto spuštění je mít nainstalovaný a spuštěný Docker. Dále je nutné být ve stejném adresáři jako `Dockerfile`. Pomocí tohoto souboru a prvního příkazu se vytvoří Docker kontejner, který bude obsahovat všechny závislosti. Pomocí druhého příkazu se kontejner spustí. Za

přepínačem `-p` se definují porty aplikace. První port je port Docker kontejneru a druhý port je portem počítače, na kterém je Docker spuštěn. Tyto porty jsou tímto způsobem provázány. Náš nástroj nativně běží na portu 5000.

Při spuštění aplikace bez přepínače se spustí server. Když je nástroj spuštěn s přepínačem `-m`, je spuštěna pouze automatická údržba a tím také takové spuštění končí. Pomocí přepínače `-n` je spuštěna pouze serverová část bez části webového frontendu. Frontend pak lze, třeba i na jiném zařízení, spustit obdobně jako serverou část, akorát spouštěným souborem bude soubor `frontend.py` bez přepínačů.

3.1 Serverová část

Tato kapitola se zabývá popisem implementace serverové části tohoto projektu. Součástí této části je aplikační rozhraní, správa sezení, asynchronní zpracování požadavků a automatická údržba.

3.1.1 Integrace nástroje Gestr

Nástroj Gestr je do serverové části integrován v třídě `GestrInterface`, která tvoří rozhraní mezi nástrojem Gestr s jeho moduly a naším nástrojem. Tato třída v sobě zahrnuje následující metody:

- konstruktor,
- metodu pro generování,
- metodu pro vytvoření zip archívu s výsledky úspěšného generování,
- metodu pro vytvoření unikátního názvu složky, ve které se budou nálezat výsledky generování

Úlohou konstruktoru je inicializace objektu a nastavení domovského adresáře, do kterého budou umístěny výsledky generování.

Metoda pro generování provádí samostatné generování a volá metody nástroje Gestr. Vstupem metody jsou parametry nástroje Gestr z tabulky 2.2. V případě neúspěšného generování metoda vyvolá vyjímku, která nakonec vede k vrácení chyby `400` s popisem, kde nastal problém. V průběhu generování je volána metoda pro vytvoření unikátního názvu výstupní složky.

```
gestr_interface = GestrInterface(workdir)
gestr_interface.generate(abstract_tree, values, criterium, config_file,
    output_format)
gestr_interface.create_zip()
```

Výpis 3.3: Pseudokód práce s nástrojem Gestr prostřednictvím rozhraní.

Metoda pro vytvoření názvu výstupní složky vrací takový název, který se nevyskytuje v příslušném pracovním adresáři. Výchozí název je `gestr_output`, v případě opakovaného generování, kdy je název již obsazen, je vrácen název ve tvaru `gestr_outputX`, kde pro X platí $X \in \langle 100; 999 \rangle$ a zároveň X je celé číslo, opět tak aby název byl unikátní.

Vstupní podmínkou pro zavolání metody vytvoření zip archívu je, že generování proběhlo úspěšně. Toho je docíleno tím, že neúspěšné generování končí vyjímkou. Při zavolání metody,

metoda vytvoří zip archív, jehož název vrací ve tvaru `workdir/Y.zip`, kde `workdir` je pracovní adresář a `Y` je název vygenerovaný podle pravidel výše.

Pseudokód generování je shrnut ve výpisu 3.3. Nejdříve je vytvořen objekt rozhraní, poté se generuje a nakonec je vytvořen zip archív.

3.1.2 Zpracování více požadavků

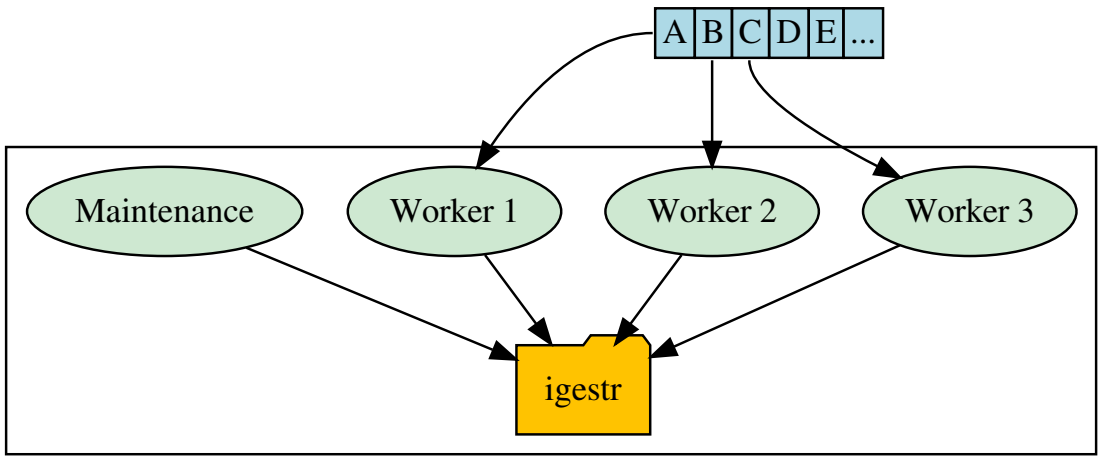
Protože se jedná o aplikaci, která má běžet veřejně na webu je zapotřebí zajistit obsluhu více požadavků zároveň. Toho bude docíleno pomocí modulu `multiprocessing`[1]. Za běhu aplikace jsou vytvořeny tři procesy, tzv. *Workers*, které jsou připraveny asynchronně zpracovat příchozí požadavky. Je zde i čtvrtý proces, který se stará o automatickou údržbu (viz 3.1.5). V případě potřeby lze počet procesů přizpůsobit situaci.

Základní obsluha požadavků probíhá v procesu hlavní aplikace. Asynchronní zpracování přichází na řadu až pro samotné generování a vytváření zip archívu, které již má na starosti některý Worker, protože tato část je nejnáročnější.

Součástí modulu `multiprocessing` je i nastavení časového intervalu (anglicky *timeout*, ve kterém musí být daná úloha vyřešena. V rámci tohoto projektu je to 90 sekund. Důvodem pro stanovení takového intervalu je, aby nedocházelo k přetěžování serveru ze strany uživatelů.

Pokud jsou všechny procesy obsazené, ostatní požadavky čekají ve frontě na obsloužení po uvolnění některého procesu.

Na obrázku 3.2 je znázorněna obsluha požadavků. Požadavky A, B, C jsou obsluhovány, ostatní požadavky (D, E, ...) čekají ve frontě na obsloužení, až na ně přijde řada. Na obrázku je znázorněn i zvláštní proces `Maintenance`, který přistupuje do diskového prostoru tím, že vykonává údržbu.



Obrázek 3.2: Znázornění asynchronního zpracování.

3.1.3 Stavy aplikace při generování

Aplikace při generování prochází několika stavy, které jsou popsány níže. Stav generování je pro každé generování uložen v globálním slovníku `session_status`. Tento slovník pro

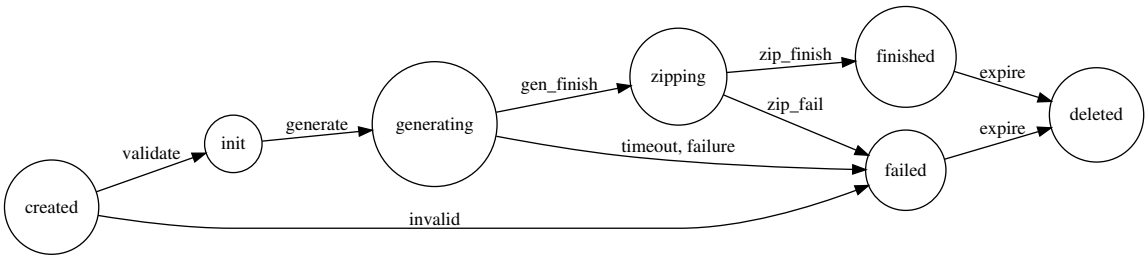
[1] https://docs.python.org/2/library/multiprocessing.html

Adresa	Metoda
`/generate`	`POST`
`/modify_config`	`POST`
`/status`	`GET`
`/download`	`GET`
`/show_config`	`GET`

Tabulka 3.1: Jednotlivé URL adresy aplikačního rozhraní.

každé sezení a jeho běžící generování kromě samotného stavu zahrnuje i čas poslední modifikace. Jednotlivá sezení jsou ve slovníku identifikovány identifikátorem daného pracovního adresáře.

Na obrázku 3.3 je znázorněna posloupnost stavů aplikace při generování. Každé generování začíná ve stavu `created`, v němž je zvalidován abstraktní strom a zadané parametry. Pokud je validace úspěšná, přechází se do stavu `init`, pokud není úspěšná, aplikace jde do stavu `failed`. Do stavu `init` se přechází navíc až ve chvíli, kdy je k dispozici některý Worker, který generování bude moct obsloužit.



Obrázek 3.3: Stavy aplikace.

Ze stavu `init` se jde přechází do dalšího stavu a rovnou se začíná generovat. Po celou dobu generování je aplikace ve stavu `generating`, ze kterého se v případě selhání generování nebo v případě vypršení času, kdy je nutné generování stihnout přechází do stavu `failed`. Pokud ani jedna z těchto událostí nenastane a generování je nakonec úspěšné přechází se do stavu `zipping`, kdy dochází k vytvoření výsledného zip archívu s výsledky generování.

Pokud selže vytvoření zip archívu, přechází do neúspěšného stavu. Je-li vytvoření archívu úspěšné přechází se do stavu `finished`, protože generování bylo úspěšné. Nakonec jsou úspěšná i neúspěšná generování po expiraci smazána ve stavu `deleted`.

3.1.4 Aplikační rozhraní

V této podkapitole si popíšeme jednotlivé URL adresy, které používá aplikace ke komunikaci s klientem. Všechny adresy jsou vypsány v tabulce 3.1.

Generování

Ke generování slouží adresa `generate`, která spouští generování a v případě úspěšného generování vrací odkaz na stažení zip archívu s výslednými soubory. Adresa přijímá několik parametrů, které jsou vstupny nástroje Gestr popsaného v podkapitole 2.3. Parametry jsou znázorněny ve výpisu 3.4. Všechny parametry jsou nepovinné, ale vždy musí být povinně

v těle požadavku zaslán abstraktní strom. Pokud při komunikaci s touto URI ještě není vytvořeno sezení, vytvoří se.

```
/generate?criterium=(pwc|ecc|bcc)&values=(tags|domains)&format=(xml|json)
```

Výpis 3.4: Parametry generování.

Změna konfiguračního souboru

Při zaslání nového konfiguračního souboru v těle požadavku na adresu `modify_config` je změněn konfigurační soubor. Tato adresa má nepovinný parametr `default`, který slouží k obnovení výchozího konfiguračního souboru. Parametr musí mít hodnotu 1, jinak je ignorován. Co se týče sezení, tak je to obdobně jako u generování, pokud sezení ještě není vytvořeno, vytvoří se.

Status

Účelem adresy `status` je zjištění stavu generování. Při odeslání dotazu v rámci platného sezení je vrácen některý ze stavů aplikace z podkapitoly 3.1.3. Pokud je tato adresa zavolána bez vytvořeného sezení, vrací se chyba.

Stažení výsledků generování

Stažení výsledků generování probíhá podle výpisu 3.5, taková adresa je navrácena po úspěšném generování. Parametr `key` znamená identifikátor daného sezení a druhý parametr `zip_archive_name` slouží jako identifikace požadovaného archívu, který se má stáhnout.

Tato adresa je specifická tím, že vůbec nepracuje se sezeními. Je to proto, aby si vygenerované výsledky mohl stáhnout kdokoliv, kdo má správný odkaz. Jediné omezení je z hlediska automatické údržby, aby výsledky pořád ještě byly k dispozici.

```
/download/<key>/<zip_archive_name>
```

Výpis 3.5: Parametry stažení výsledků generování.

Zobrazení konfiguračního souboru

Pro zobrazení aktuálně nastaveného konfiguračního souboru je třeba zavolat adresu `show_config`. V případě volání v rámci platného sezení je navrácen obsah daného konfiguračního souboru, pokud je voláno bez sezení je vrácen výchozí konfigurační soubor.

3.1.5 Automatická údržba

Každá serverová aplikace, která vytváří nějaká data pro uživatele, by měla mít svou vlastní automatickou údržbu, tak aby se nezahlcoval datový prostor serveru. Automatická údržba této aplikace se za běhu aplikace spouští periodicky každých 15 minut. Je možné ji také spustit jednorázově. Tuto údržbu má na starosti třída `Maintenance`.

V první fázi údržby jsou smazány všechna sezení ze stavů sezení, jejichž čas poslední modifikace je starší než 24 hodin. Soubory zatím nejsou ovlivněny.

Ve druhé fázi jsou smazány všechny archívy s výsledky generování napříč pracovními adresáři, jejichž poslední změna je starší než 72 hodin.

V poslední fázi jsou smazány všechny pracovní adresáře, pro které platí, že obsahují pouze konfigurační soubor a nevyskytují se ve slovníku stavů generování.

Životnost jednotlivých souborů

V tabulce 3.2 jsou shrnuty životnosti jednotlivých souborů. Je zřejmé, že životnost pracovního adresáře je buď alespoň 72 hodin, pokud se v něm nachází alespoň jeden úspěšně vygenerovaný soubor, nebo pouze 24 hodin, pokud pro dané sezení nebylo provedeno ani jedno úspěšné generování.

Soubor	Životnost
Stav sezení	$\geq$ 24 hodin
Pracovní adresář	$\geq$ 24 hodin $\bigvee \geq$ 72 hodin
Archív s výsledky	72 hodin

Tabulka 3.2: Životnost jednotlivých souborů

3.2 Webový frontend aplikace

Tato podkapitola popisuje uživatelské rozhraní webového rozhraní implementovaného nástroje, jeho ovládání a také implementaci jednotlivých částí. Snímek hlavní obrazovky aplikace je na obrázku 3.4.

Webový frontend obsahuje tři stránky, na domovské stránce se nacházejí oba editory a též ovládání generování. Zbylé dvě stránky jsou statické. První z nich obsahuje manuál, který popisuje, jak nástroj používat. Druhá z nich obsahuje abstrakt a odkazy na repozitáře a domovskou stránku platformy Testos.

3.2.1 Ovládání nástroje

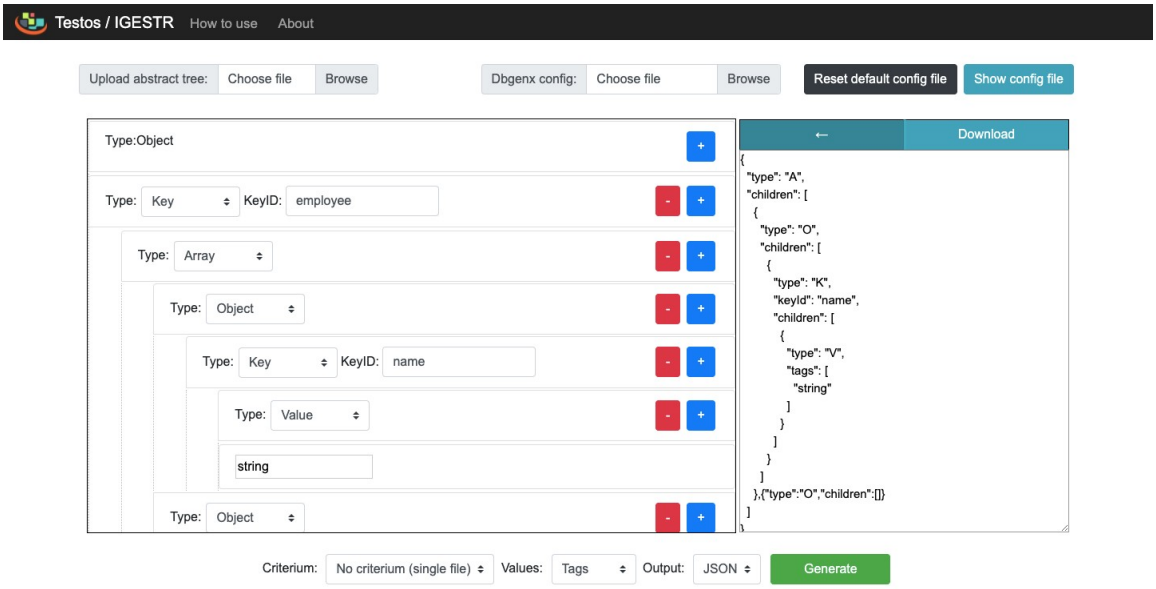
Tato podkapitola popisuje ovládání webového rozhraní implementovaného nástroje, zejména editaci abstraktního stromu, ale také aktivaci generování a změny konfiguračního souboru.

Nejdříve je třeba ale uvést několik důležitých informací o editaci abstraktního stromu. Při načtení stránky s editorem editor obsahuje pouze uzel typu Objekt, který je povinně prvním uzlem abstraktního stromu, a proto jej nelze odstranit. Operacemi popsanými níže lze vytvořit nový abstraktní strom nebo případně lze nahrát existující strom, a ten upravit, nebo s ním rovnou začít generovat.

Editor kontroluje jednotlivá omezení potomků, která jsou popsána v tabulce 2.1.

Přidání nového uzlu

Pro přidání nového uzlu je třeba kliknout na tlačítko „+“ u budoucího rodiče přidávaného uzlu. Po kliknutí je třeba vybrat typ uzlu a případně doplnit informace, které jsou pro daný typ vyžadovány. Výběr uzlu je omezen, tak aby nebyla porušena výše zmíněná omezení. Nový uzel se pak stává jeho následníkem.



Obrázek 3.4: Hlavní stránka aplikace.

Odstranění uzlu

Pro odebrání uzlu je potřeba kliknout na tlačítko „-“ u daného uzlu. Uživatel dostává na výběr, zda chce odstranit uzel a všechny jeho potomky nebo pouze samotný uzel.

Pokud zvolí odstranit uzel se všemi jeho potomky, uzel je odstraněn a místo po něm zůstane prázdné. V případě zvolení druhé možnosti, je odstraněn pouze onen uzel a na jeho místo jsou umístěni všechni jeho potomci. V tomto případě jsou opět vyžadována splnění zmíněných omezení, jinak uzel není odstraněn.

Použití textového editoru

Aktivace textového editoru probíhá dvojklikem na vybraný uzel v interaktivním editoru. V textovém editoru je daný uzel a všichni jeho potomci zobrazen jako JSON kód. Po provedení změn se kliknutím na tlačítko šipky změny zkopírují do interaktivního editoru na pozici vybraného uzlu.

Stažení abstraktního stromu

Pro stažení celého abstraktního stromu nebo jeho vybrané části je třeba dvojklikem na počáteční uzel daného podstromu aktivovat textový editor. Poté kliknutím na tlačítko „download“ je daný podstrom stažen jako soubor ve formátu JSON.

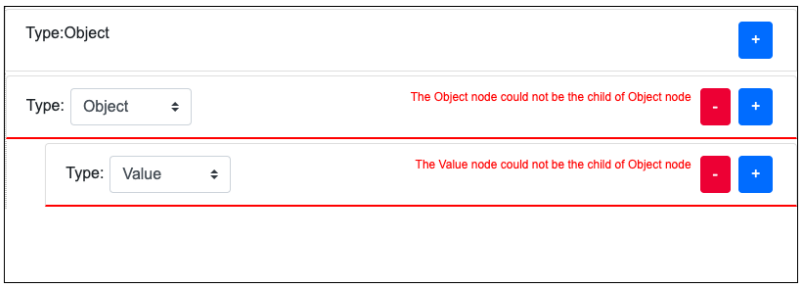
Konfigurační soubor

Konfigurační soubor je možné změnit, zobrazit nebo obnovit výchozí. Ke změně konfiguračního souboru je třeba jej nejdříve nahrát, k tomu slouží element „Dbgenx config“. Konfigurační soubor je poté změněn při aktivaci generování.

Ke změně konfiguračního souboru slouží tlačítko „Show config file“ a pro obnovu výchozí konfiguračního souboru slouží tlačítko „Reset default config file“.

Použití vlastního abstraktního stromu

Pro použití vlastního abstraktního stromu je třeba jej nahrát. To lze učinit pomocí elementu „Upload abstract tree“. Ihned po nahrání se v případě, že se jedná o validní JSON, jež obsahuje uzly a první uzel je typu Objekt, zobrazí v interaktivním editoru. Pokud soubor nesplňuje zmiňovaná omezení, tak tyto uzly jsou v interaktivním editoru zvýrazněny (viz obrázek 3.5).



Obrázek 3.5: Příklad upozornění při nespravném uspořádání uzlů.

Generování

Generování se spouští stisknutím tlačítka „Generate“. Před stisknutím tlačítka si lze vybrat kritérium pokrytí, způsob tvorby hodnot a výstupní formát generování.

Po zahájení generování se periodicky kontroluje stav generování (pomocí adresy `/status`) a ten se zobrazuje uživateli. Pokud je generování rychlé (v řádu jednotek sekund) stav se nezobrazí.

V případě úspěšného generování se zobrazí hlašení, ve kterém je odkaz, kde si lze stáhnout výsledky. Pokud generování naopak není úspěšné, objeví se chybové hlášení, ve kterém je popsána chyba. Tento popis chyby vrací server.

3.2.2 Komunikace se serverem

Ke komunikaci se serverem se využívá implementované aplikační rozhraní REST. Komunikaci zajišťuje Ajax. K volání aplikačního rozhraní slouží objekt `IGESTR`, ve kterém je pro každou adresu aplikačního rozhraní implementována metoda, která volání dané adresy provede a poté zpracuje její data.

Objekt `IGESTR` obsahuje vlastnost `SERVER`, která obsahuje adresu serveru. V každé metodě, která volá aplikační rozhraní, je tato vlastnost použita ve volání. Při změně adresy serveru je potřeba změnit tuto konstantu a nic víc. Pokud webový frontend funguje na stejné adrese jako serverová část, nechá se tato vlastnost prázdná.

Ve výpisu 3.6 je kód volání Ajaxu. Pomocí `url` se určí, se kterou adresou má být navázáno spojení. Zde se právě k požadované adresy připojí výše zmíněná vlastnost `SERVER`. Dále se určí která data mají být odeslána v datovém poli (anglicky *payload*) požadavku, typ požadavku, typ dat. Pomocí `xhrFields` a vlastnosti `withCredentials` určujeme, že chceme použít Cookies atp. a udržet tak sezení, aby bylo možné udržet komunikaci.

```
ajax({
    url: this.SERVER+"generate",
    data: "{}",
    type: "POST",
    contentType: "application/json",
    xhrFields: {
        withCredentials: true
    },
    success: function (data) {
        //
    },
    error: function () {
        //
    }
});
```

Výpis 3.6: Pseudokód volání aplikačního rozhraní pomocí Ajax.

3.2.3 Interaktivní editor

Interaktivní editor je hlavní část aplikace, ve které se odehrávájí veškerá zpracování abstraktního stromu a ten také abstraktní strom předává ke generování. Oba editory jsou obsluhovány objektem `Editor`. Významnou část tohoto objektu však tvoří právě metody a atributy určené k obsluze interaktivního editoru.

Abstraktní strom je uložen ve zmiňovaném objektu v objektové reprezentaci, zobrazována je její přesná kopie. Při některé změně se nejdříve provede změna v uložené objektové reprezentaci a až poté v zobrazeném stromě.

V této podkapitole jsou popsány jednotlivé operace nad abstraktním stromem, které se pro interaktivní editor provádí.

Průchod abstraktním stromem

Průchod abstraktním stromem je zajištěn rekurzivním průchodem *Preorder*. Průchod stromem obecně znamená navštívit každý uzel stromu právě jednou, přičemž binární strom je buď prázdný nebo obsahuje kořen (anglicky *root*), který obsahuje levý a pravý podstrom. V případě binárního stromu[2] rekurzivní průchod Preorder znamená:

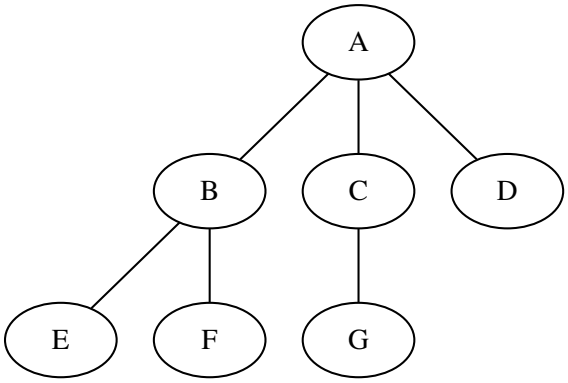
- pokud je strom prázdný, vyhledávání končíme,
- pokud strom není prázný, zpracujeme nejdříve jeho levý podstrom a poté jeho pravý podstrom.

Převzato z *The Analysis on Recursive Algorithm Implementation of Preorder-Traversing Binary Tree* [18].

Náš strom však není binární, počet podstromů může být libovolný. V takovém případě se, pokud je strom neprázdný, postupně zpracovávají podstromy zleva doprava. Mějme příklad stromu na obrázku 3.6.

Preorder průchod tímto stromem bude: `A`, `B`, `E`, `F`, `C`, `G`, `D`.

[2] Uzel stromu obsahuje maximálně dva potomky.



Obrázek 3.6: Příklad jednoduchého stromu.

Načtení abstraktního stromu

Při načtení abstraktního stromu je strom nejdříve procházen průchodem Preorder (viz 3.2.3). Pokud kořenový uzel abstraktního stromu není typu Objekt nebo tento uzel neobsahuje pole `children` (viz 2.3), je uživateli zobrazeno chybové hlášení informující o daném problému a načtení je zastaveno.

Při průchodu se také kontrolují posloupnosti jednotlvých uzlů (viz tabulka 2.1), je-li nalezen problém, je sice v průchodu pokračováno, ale je zobrazeno chybové hlášení u daného problematického uzlu.

Nejdůležitější událostí, která se ovšem děje při načítání je příprava HTML kódu pro zobrazení reprezentace abstraktního stromu v interaktivním editoru. Tato reprezentace je blíže popsána dále, v 3.2.3.

Během průchodu se postupně označuje pořadí uzlů daného typu, tak jak byly při průchodu navštíveny. Důvodem tohoto je pozdější případná identifikace uzlu při některých operacích – vytváření, odstraňování a editování uzlů (viz 3.2.3). Na obrázku 3.7 je toto označení znázorněno.

Toto očíslování uzlů je uloženo přímo do HTML kódu pro pozdější užití při uživatelských požadavcích o některou výše zmíněnou operaci. Elemenet, který obsahuje daný uzel, obsahuje dva datové atributy – `data-type` a `data-type-order`. Jak již názvy těchto atributů napovídají, první z nich obsahuje typ uzlu a druhý obsahuje pořadí.

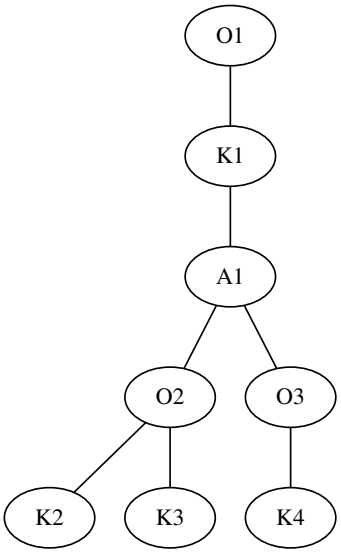
Zobrazení abstraktního stromu

Reprezentace abstraktního stromu v interaktivním editoru je zobrazena seznam. Každý potomek uzlu je zobrazen v zanořeném seznamu. O každém uzlu je zobrazen jeho typ, v případě uzlu typu klíč i jeho identifikátor (`keyId`).

Celý editor zabírá přibližně 70% výšky stránky. Pokud je daná reprezentace delší, v okně lze skrolovat.

Vytváření, editování a odstraňování uzlů

K vytváření, editování a mazání uzlů se využívá zmiňovaného průchodu Preorder. Každá z těchto operací začínají výběrem uzlu, nad kterým operaci chceme provést. Dle datového typu a jeho pořadí (viz 3.2.3) je uzel vybrán v abstraktním stromu a jsou nad ním provedny



Obrázek 3.7: Příklad abstraktního stromu s očíslovanými uzly.

požadované akce. Poté je abstraktní strom opět načten dle 3.2.3. Důvodem opětovného načtení je, že jsou opětovně provedeny kontroly a uživatel je o případných nedostatcích informován.

Při vytváření nového uzlu uživatel volí otce budoucího uzlu a nový uzel je poté vložen mezi jeho následníky. Při editování jsou upraveny změněné vlastnosti uzlu.

Pro odstraňování uzlu existují dva typy mazání – odstranit uzel a všechny jeho potomky nebo pouze samotný uzel. V případě zvolení odstranění uzlu i s potomky je pouze uzel nahrazen prázdným uzlem a ten je vymazán kompletně. Při druhé možnosti je na místo uzlu umístěn jeho následník (eventuelně následníci).

3.2.4 Textový editor

Textový editor je mnohem jednodušší než interaktivní editor, protože změny v abstraktním stromu provádí uživatel. Objekt `Editor` pro textový editor obsahuje pouze dvě metody – pro načtení vybrané části stromu do textového editoru a poté pro načtení editované části zpět do interaktivní editoru.

Načtení z interaktivního editoru

Pro načtení podstromu do interaktivního editoru je vybrán některý uzel uživatelem, a ten se načte do textového editoru i se všemi jeho následníky. Opět se využívá číslování typů uzlu, popsané dříve v 3.2.3.

Daná objektová reprezentace se převede do textové reprezentace, a ta je zobrazena v textovém editoru. Nyní uživatel může provádět libovolné úpravy daného podstromu.

Načtení do interaktivního editoru

Jakmile je uživatel hotov se svými úpravami a klikne na příslušné tlačítko, je daný podstrom nejdříve převeden do textové reprezentace. Pokud tento převod není úspěšný, uživatel vy-

tvořil JSON, který není validní. V tomto případě není načteno nic a uživateli je zobrazeno chybové hlášení.

V případě úspěšného převodu je soubor umístěn do uložené objektové reprezentace v objektu `Editor` na místo původního vybraného uzlu a poté je znovu načten abstraktní strom do interaktivního editoru.

3.2.5 Obsluha událostí

Obsluha událostí na stránce (např. kliknutí, nahrání souboru, změna textu atp.) se provadí v souboru `controller.js` pomocí jazyka Jquery. Obsluhují se takto události pro editory, ale také pro ostatní prvky stránky. Je zde také vyvolána inicializace interaktivního editoru při načtení stránky.

Při události jsou nejdříve získána data (jsou-li nějaká k dané následující akci třeba), např. je získán datový typ a pořadí datového typu daného uzlu, na který bylo kliknuto. Poté je zavolána příslušná metoda buď objektu `Editor` nebo objektu `IGESTR` a daná data jsou jí předána jako parametry.

Při obsluze události jsou před zavoláním příslušné metody zkontrolovány zadaná data a uživatel je případně o problémech informován vyskakovacím oknem a akce není prováděna vůbec. Vyžaduje-li to daná akce, uživatel je požádán o doplnění některých informací potřebných pro akci. Například jsou přidány příslušné vstupní textová políčka nebo jemu zobrazovacím oknem dáno na výběr z více možností.

```
('.generate-btn').click(function () {
   // get criterium, values, ...
   ...
   igestr.generate(JSON.stringify(editor.tree), criterium, values, format,
        '#result', '#error');

});
```

Výpis 3.7: Pseudokód obsluhy kliknutí.

Výpis 3.7 obsahuje pseudokód obsluhy kliknutí na tlačítko „Generate“. Když dojde na kliknutí na prvek se selektorem (třídou) `generate-btn`, je zavolána funkce, která je parametrem funkce `click`. Nejdříve se získají zvolené hodnoty kritéria pokrytí, způsobu tvorby hodnot a výstupního formátu. Dále se načte abstraktní strom z editoru (`editor.tree`) a ten se převede do textové reprezentace JSON. Všechny tyto získané hodnoty se předávájí metodě `generate` objektu `igestr`. Poslední dva parametry této metody specifikují elementy pro úspěšné generování a pro chybový výstup. Který z nich se použije, se rozhodne podle toho zda generování bude úspěšné.

3.2.6 Integrace do serverové části

Webový frontend je integrován do serverové části a lze tedy jednoduše obě části provozovat na jednom serveru. Samotná integrace do Flasku přináší také šablonovací systém.

Flask

Samotný webový frontend a přístup k němu je implementován v souboru `frontend.py`. Je zde implementována URI `/`, která vrací soubor `index.html`, obsahující editory. Další URI

hledá podle zadání HTML soubor, odpovídající zadané URI. Pokud soubor neexistuje vrací chybu 404 – nenalezeno.

```
@FRONTEND.route('/<page>')
def show(page):
   if page == 'layout':
      abort(404)
   try:
      return render_template(page + '.html')
   except Exception:
      abort(404)
```

Výpis 3.8: Obsluha jednotlivých URI.

Ve výpisu 3.8 je kód volání jednotlivých URI. Uživatel načte stránku, například `/hello`. Do proměnné `page` je vložena hodnota „hello“. Nejdříve se zjistí, jestli uživatel nezkouší načíst stránku `layout`, která nemá být veřejně dostupná. Dále se zkusí načíst soubor `hello.html`, pomocí funce `render_template`, ze složky `templates`, pokud existuje, uživateli se načte ona stránka. Pokud metoda však vyhodí vyjímku, například proto, že soubor neexistuje, je vrácen kód 404 – stránka nenalezena.

Veškeré soubory stránek jsou uloženy v adresáři `templates/`, v adresáři `static/` jsou jednotlivé soubory CSS, JavaScript a soubory knihoven Jquery. Tento adresář také obsahuje veškerá média (obrázky).

Soubor `frontend.py` je vložen do hlavního souboru serverové části. Lze jej spustit i samostatně, pak poběží pouze webové rozhraní bez serverové části.

Šablony

Zmiňovaný adresář `templates` obsahuje i soubor `layout.html`, ke kterému nelze přistoupit přímo. Tento soubor obsahuje prvky , které se opakují na všech stránkách – menu a patička. Soubor s požadovanou stránkou se poté vloží do vymezeného místa v šabloně. Přínosem použití šablon je zejména do budoucnosti možnost snadného redesignu, přidání dalších stránek nebo i jejich odebrání.

```
<!DOCTYPE html>
<html lang="en">
   <head></head>
   <body>
   <header> ... </header>
   <main>
      {% block content %}
      {% endblock %}
   <main>
   <footer> ... </footer>
   </body>
</html>
```

Výpis 3.9: Fragment zdrojového kódu souboru layout.html.

Ve výpisu 3.9 je fragment zdrojového kódu souboru `layout.html`. Soubor obsahuje HTML kód, který se opakuje na všech stránkách aplikace. V bloku `main` je umístěn blok

pojmenovaný `content`. Do tohoto místa bude umístěn kód, který v ostatních souborech, které se do tohoto souboru vkladájí, je umístěn ve stejně pojmenovaném bloku.

```
{% extends "layout.html" %}
{% block content %}
<div>
  <h1>How to use</h1>
  ...
</div>
{% endblock %}
```

Výpis 3.10: Fragment zdrojového kódu souboru howtouse.html.

Výpis 3.10 obsahuje fragment kódu soubor `howtouse.html`. Na prvním řádku je specifikováno, že soubor, který rozšiřujeme je soubor `layout.html`. Dále specifikujeme, že zdrojový kód má být vložen do bloku `content`.

Kapitola 4

Validace a ověření funkcionality

Tato kapitola se zabývá ověřením funkcionality našeho nástroje. Nejdříve je popsána validace serverové části, poté jsou popsány jednotlivé automatizované testy nad serverovou části, jejím aplikačním rozhraním a nad webovým frontendem.

4.1 Kontrola kvality kódu prostřednictvím Pylint

Nástroj Pylint[1] slouží ke statické analýze zdrojových kódů v jazyce Python. Kontroloje úhlednost kódu, různé standardy a také detekuje chyby v kódu. Kód hodnotí v rozmezí 0 – 10, kde 0 je nejhorší a 10 je nejlepší. Celkové hodnocení všech Python souborů nástrojem Pylint je `9,62`. Nyní si popíšeme, kde náš nástroj při hodnocení ztratil.

V souboru `GestrInterface.py` byly ztraceny body za název stejnojmenné třídy, který neodpovídá *hadí notaci* (anglicky *snake case*). Dále za to, že metoda `generate` má příliš mnoho argumentů. Maximum argumentů je podle validátoru pět, tato metoda však má šest. Vzhledem k tomu, že tato metoda přijímá argumenty, které jsou zapotřebí pro nástroj Gestr a hraniční počet byl překročen pouze o jeden argument, je tento prohřešek tolerovatelný. Navíc přemístění některého argumentu, například do konstruktoru, by vedlo ke snížení čitelnosti zdrojového kódu.

Dalším problémem je importování všech vyjímek nástroje Gestr pomocí hvězdičkové notace. Vzhledem k tomu, že jsou použity všechny a také proto, že jich není málo, jedná se opět o průchozí chybu.

Dalším souborem s chybami je soubor `Maintenance.py`. Opět je zde chyba, že název obsažené stejnojmenné třídy neodpovídá hadí notaci. Další chybou je, že metoda, na čištění pracovních adresářů (`clean_work_dirs`), neobsahuje žádné volání `self` a metoda by tedy mohla být funkcí. Nicméně tato metoda i přesto patří do této třídy, protože vykonává pouze jeden z několika kroků automatické údržby.

V rámci souboru `frontend.py` je hlášena pouze jedna chyba, touto chybou je použití příliš obecné odchycení vyjímky (odchytávání přímo `Exception`). K tomuto odchytávání dochází v rámci definice URI adresy, která hledá jestli existuje HTML šablona pro danou stránku. Hlavní důvod odchytávání vyjímky zde je situace, kdy uživatel požádá o načtení stránky, která neexistuje. V této situaci je sice vyhozena určitá vyjímka, ale obecné odchycení vyjímky je zde i pro situaci, že bude vyhozena jiná vyjímka, pak bude zobrazena chyba i v této situaci.

[1] https://www.pylint.org

Posledním souborem s chybami je soubor `igestr.py`, který je hlavním souborem aplikace. Zde je vyčítáno redefinování `TimeOutError` při odchytávání vyjímky, nicméně o redefinici se skutečně nejedná. Dalším problémem je použití globální proměnné, jedná se o globální slovník, ve kterém jsou uchovány jednotlivé stavy sezení. Proměnná je globální kvůli jednoduchému přístupu. Dále je dvakrát zmíněna chyba s tím, že globální proměnná z předchozí chyby je údajně konstantní a měla by tedy být napsána velkými písmeny.

4.2 Testy serverové části

Serverová část obsahuje dvě sady testů. Jednak jsou to jednotkové testy, které ověřují funkčnost některých součástí. Druhá sada testů se soustředí na aplikační rozhraní serverové části.

Jednotkové testy serverové části

Jednotkové testy zkoumají funkčnost rozhraní nad nástrojem Gestr a funkčnost automatické údržby.

Při testech rozhraní nástroje Gestr se průběžně testuje generování s různými parametry. Kromě správnosti implementace lze tímto zjistit zda jsou instalovány všechny potřebné závislosti a nástroj je tedy plně funkční. Dále se ověřuje funkčnost vytváření archívu zip.

Při testech automatické údržby se nejdříve testuje mazání adresářů, které nejsou uvedeny ve slovníku stavů sezení a zároveň jsou prázdné. Dále se testuje mazání vygenerovaných zip archívů v rámci jednoho pracovního adresáře. Vytvoří se adresář s několika prázdnými archívy a některým se přiřadí čas poslední modifikace souboru starší než je doba expirace, tyto archívy by pak měly být automatickou údržbou smazány. Poslední je testování promazávání slovníku se stavy sezení, které se testuje obdobně jako předchozí případ.

Jednotkové testy jsou napsány ve frameworku `unittest`[2] pro jazyk Python.

Testy aplikačního rozhraní serverové části

Testy aplikačního rozhraní serverové části jsou realizovány pomocí programu Curl, pomocí tohoto programu se volají jednotlivé URI aplikačního rozhraní a pro danou situaci je srovnán návratový kód s referenčním návratovým kódem. Obdobně i s výstupem operace nebo jsou stažena data a ta jsou zkontrolována. Tyto testy jsou napsány jako Shell skripty.

Nejdříve jsou před zahájením sezení testovány operace zobrazení konfiguračního souboru a zobrazení stavu generování. První zmíněná operace má správně vrátit výchozí konfigurační soubor, druhá správně selže. Dále se několikrát testuje generování, testují se také chybné argumenty. Průběžně se testuje stav sezení při různých stavech. Testuje se stahování vygenerovaného výstupu, ale i tak jeho chybové stavy.

Sada obsahuje 25 testů, odpovídajících specifikaci nástroje. Při editaci testů lze ve zdrojovém kódu jednoduše změnit adresu serveru a pak je možné zkontrolovat i funkčnost nástroje po nasazení.

4.3 Automatické testování GUI

Posledními důležitými testy jsou testy webového frontendu. Tyto testy byly realizovány pomocí nástroje Selenium IDE, což je rozšíření dostupné pro webové prohlížeče Google

[2] https://docs.python.org/3/library/unittest.html

Chrome a nebo Firefox. Toto rozšíření umožňuje uživateli jednoduše nahrát testy webové aplikace a poté si je znovu přehrát.

Nahráním testu se rozumí, že uživatelovi akce na webové stránce (kliknutí na prvek, přejetí kurzorem nad prvkem, ...) jsou, během spuštěného nahrávání, nahrávány a poté jsou uloženy. Potom si je lze opět spustit a tyto akce se znovu odehrávají.

V rámci těchto testů bylo nahráno několik testů. Tyto testy zkoumají práci s interaktivním editorem, spouštění generování, práci s textovým editorem, zobrazování a změny konfiguračního souboru.

Opět lze tuto sadu využít i po nasazení a jednoduše tak ověřit, že vše funguje korektně.

4.4 Spuštění jednotlivých testů

Tato podkapitola pojednává o spuštění jednotlivých testů popsaných v předchozích podkapitolách. Jsou také zmíněny jednotlivé závislosti, které je potřeba pro korektní běh testů splnit.

```
> python3 unittest
```

Výpis 4.1: Spuštění jednotkových testů.

Ve výpisu 4.1 je příkaz na spuštění jednotkových testů. Je potřeba mít zprovozněnou aplikaci podle kapitoly 3 a nainstalovaný nástroj `unittest`.

```
> chmod +x run_tests.sh
> ./run_tests.sh
```

Výpis 4.2: Spuštění testů aplikačního rozhraní.

Ve výpisu 4.2 jsou příkazy potřebné pro spuštění testů aplikačního rozhraní. Je třeba být v adresáři `tests_curl` (nebo použít příslušnou cestu). Dále třeba před prvním spuštěním nastavit skript jako spustitelný (viz první příkaz ve výpisu). Pro běh testů je zapotřebí mít někde spuštěné serverové rozhraní. V případě potřeby je třeba v testech změnit proměnnou `SERVER`.

Pro spuštění automatických testů webového frontedu je třeba mít ve webovém prohlížeči Google Chrome nebo Firefox nainstalované rozšíření Selenium. Po otevření tohoto rozšíření je třeba naimportovat soubor `test_gui/igestr_test0.side`. Testy byly spouštěny v prohlížeči Google Chrome.

Kapitola 5

Závěr

Práce si kladla za cíl vytvořit grafické uživatelské rozhraní a aplikační rozhraní pro generátor strukturovaných testovacích dat, Gestr. Zejména šlo o to, zpřístupnit nástroj testerům a umožnit snadné a uživatelsky přívětivé ovládání.

Výsledný nástroj umožňuje plnohodnotně ovládat nástroj Gestr, vytvářet nové abstraktní stromy nebo editovat stávající abstraktní stromy.

Vzniklá serverová část projektu poskytuje aplikační rozhraní, jehož prostřednictvím se serverovou části komunikuje webový frontend. Toto aplikační rozhraní je typu REST a je možné jeho prostřednictvím ovládat nástroj bez nutnosti využívat webový frontend. Serverová část dále řídí generování, umožňuje uživateli stáhnout si výsledky generování v archívu zip a poskytuje automatickou údržbu nad vygenerovanými výslednými výstupy.

Webový frontend obsahuje interaktivní a textový editor abstraktního stromu, díky nimž je možné nad tímto stromem provádět libovolné úpravy. Kromě výsledků generování je možné si také stáhnout upravený abstraktní strom a ten použít později nebo jej případně využívat v samotném nástroji Gestr. Lze měnit konfigurační soubor pro generování.

Vytvořený nástroj vznikl v rámci platformy Testos, kde bude využíván například pro vytváření datových vstupů ve formátu JSON a XML pro testování informačních systémů. Funkčnost byla ověřena jednotkovými testy, testy aplikačního rozhraní serverové části a automatickými testy uživatelského rozhraní webového frontendu. Tento nástroj je funkční a je k dispozici veřejnosti na adrese `igestr.testos.org`.

Literatura

[1] *Ajax – Getting started* [online]. [cit. 2020-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started.

[2] *Dokumentace knihovny JQuery* [online]. [cit. 2020-04-07]. Dostupné z: https://api.jquery.com.

[3] *Dokumentace XML* [online]. [cit. 2020-05-19]. Dostupné z: https://www.w3.org/TR/xml/#elemdecls.

[4] *Domovská stránka nástroje Gestr* [online]. [cit. 2020-03-26]. Dostupné z: https://pajda.fit.vutbr.cz/testos/gestr.

[5] *Domovská stránka platformy Testos* [online]. [cit. 2020-03-26]. Dostupné z: http://www.testos.org.

[6] *HTTP – State & Session Management* [online]. [cit. 2020-05-20]. Dostupné z: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_StateManagement.html.

[7] *Oficiální stránky jazyka JSON* [online]. [cit. 2020-05-19]. Dostupné z: https://www.json.org/json-cz.html.

[8] *Sessions in Flask* [online]. [cit. 2020-05-20]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/api/#flask.session.

[9] *What is a Container?* [online]. [cit. 2020-04-20]. Dostupné z: https://www.docker.com/resources/what-container.

[10] *Žebříček popularity programovacích jazyků PyPL* [online]. [cit. 2020-04-04]. Dostupné z: http://pypl.github.io/PYPL.html.

[11] Ammann, P. a Offutt, J. *Introduction to Software Testing*. Cambridge University Press, 2016. ISBN 9781107172012.

[12] Castro, E. a Hyslop, B. *HTML5 a CSS3 – názorný průvodce tvorbou WWW stránek*. Computer Press, 2012. ISBN 9788025137338.

[13] Kharenko, A. *Monolithic vs. Microservices Architecture* [online]. [cit. 2020-05-19]. Dostupné z: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59.

[14] Kotyz, J. *Nástroj pro tvorbu obsahu databáze pro účely testování software*. Brno, CZ, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/18066/.

[15] Massé, M. *REST API Design Rulebook*. O'Reilly Media, Inc., 2012. ISBN 9781449310509.

[16] Olšák, O. *Generování strukturovaných testovacích dat*. Brno, CZ, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21539/.

[17] Schepenaar, W. *Server-side vs Client-side Routing* [online]. [cit. 2020-05-19]. Dostupné z: https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f.

[18] Song, Y. C. a Jin, S. L. The Analysis on Recursive Algorithm Implementation of Preorder-Traversing Binary Tree. *Applied Mechanics and Materials*. Trans Tech Publications. 2014, 631-632, s. 99–102. ISSN 1660-9336.

[19] Subhash, C. *Introduction To Client Sever Computing*. New Age International (p) Limited, 2009. ISBN 9788122426892.

[20] Telles, M. *Python for Professionals: Learning Python as a Second Language*. BPB PUBN, 2019. ISBN 9789389423754.

[21] Zakas, N. *JavaScript pro webové vývojáře*. Computer Press, 2009. ISBN 9788025125090.

Příloha A

Obsah odevzdaného CD

Odevzdané CD obsahuje tyto adresáře a soubory:

- **text** – adresář obsahující zdrojové texty a obrázky technické zprávy,
- **igestr** – adresář obsahující soubory a dokumentaci implementovaného nástroje igestr,
  - **README.md** – soubor obsahující základní dokumentaci a pokyny k instalaci a spuštění nástroje,
  - **doc** – adresář obsahující další dokumentaci nástroje a log z poslední `pylint` analýzy,
  - **app** – adresář obsahující implementaci nástroje,
    - **README.md** – soubor obsahující popis adresářové struktury aplikace,
    - **config** – adresář obsahující konfigurační soubor pro Dbgenx,
    - **static** – adresář obsahující statický obsah pro webový frontend,
      - **css** – adresář obsahující soubory s kaskádovými styly,
      - **js** – adresář obsahující soubory JavaScript,
      - **lib** – adresář obsahující knihovny pro frontend,
      - **media** – adresář obsahující média (obrázky),
    - **templates** – adresář obsahující HTML šablony pro webový frontend,
      - **layout.html** – adresář s hlavní šablonou stránky,
      - ***.html** – ostatní html stránky,
    - **test_gui** – testy uživatelského rozhraní webového frontendu (pro Selenium IDE),
    - **tests** – jednotkové `Python` testy serverové části,
    - **tests_curl** – testy aplikačního rozhraní pomocí programu Curl,
  - **Dockerfile** – konfigurační soubor pro spuštění nástroje v Docker kontejneru,
  - **___init___.py**,
  - **const.py** – soubor definující konstanty,
  - **frontend.py** – soubor obsahující rozhraní pro webový frontend,
  - **GestrInterface.py** – soubor sloužící jako rozhraní mezi serverovou částí a nástrojem Gestr,

– **igestr.py** – hlavní soubor aplikace s definicemi aplikačního rozhraní,
– **Maintenance.py** – soubor sloužící k automatické údržbě,
– **requirements.txt** – soubor obsahující závislosti projektu.