FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Automatizované získávání dat z elektronických loterií

Dominik Vladař

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Automatizované získávání dat z elektronických loterií

Dominik Vladař

Vedoucí práce
doc. Ing. Dalibor Fiala, Ph.D.

PLZEŇ 2024

Citace v seznamu literatury:
VLADAŘ, Dominik. *Automatizované získávání dat z elektronických loterií*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce doc. Ing. Dalibor Fiala, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Dominik VLADAŘ**
Osobní číslo:	**A21B0317P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Automatizované získávání dat z elektronických loterií**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s technikami získávání dat z webových stránek (tzv. web scrapingu) formou automatického harvestování prováděného pomocí tzv. web crawlerů.
2. Prostudujte možnosti existujících web crawlerů a analyzujte vhodnost jejich použití na automatizované získávání dat z elektronických loterií a to jak historických dat, tak i dat získávaných v reálném čase.
3. Navrhněte a vypracujte postupy přípravy (konfigurace) zvolených web crawlerů pro různé konkrétní elektronické loterie a vhodně navrhněte datový formát pro stahovaná data (např. CSV), včetně popisu aktualizace při opakovaném stahování.
4. Realizujte opakovaný web scraping všech dostupných historických dat z vybraných elektronických loterií i web scraping v reálném čase pro loterie generující data s vysokou frekvencí.
5. Připravte uživatelskou dokumentaci v podobě metodické příručky popisující, jak si připravit vlastní web crawler pro úplně nový typ elektronické loterie.

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. Dalibor Fiala, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

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

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 30. dubna 2024

. . . . . .
Dominik Vladař

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Tato práce se zabývá automatizovaným sběrem dat z elektronických loterií. Cílem je vytvořit program, který bude na základě konfigurace provádět pro jednotlivé loterie tzv. web scraping, tedy automatizované získávání dat. V teoretické části práce jsou podrobně popsány možnosti implementace automatizovaného sběru dat. Dále je uveden popis, jakým způsobem jsou dostupná data u konkrétních vybraných elektronických loterií. Pro srovnání je kromě web crawleru implementován i program, který získává data pomocí API, přičemž práce oba přístupy srovnává. Praktická část obsahuje zejména popis implementace práce, včetně jednotlivých realizačních detailů obou programů.

Klíčová slova

API • elektronické loterie • web crawler • web scraping • získávání dat

Abstract

This thesis focuses on automatic data retrieval from electronic lotteries. The goal is to create a program that, based on the configuration, will perform the so-called web scraping, i.e. automated data acquisition for individual lotteries. In the theoretical part of the thesis, the possibilities of implementing automatic data retrieval are described in detail. Furthermore, a description of how data is available for specific selected electronic lotteries is provided. For comparison, in addition to the web crawler, a program that retrieves data using an API has been implemented, and the thesis compares both approaches. The practical part mainly contains a description of the job implementation, including individual implementation details of both programs.

Keywords

API • electronic lotteries • web crawler • web scraping • data retrieval

Poděkování

Chtěl bych tímto poděkovat panu doc. Ing. Daliboru Fialovi, Ph.D. za odborné vedení práce, panu doc. Ing. Janu Pospíšilovi, Ph.D. z katedry matematiky za věnovaný čas a cenné rady a panu Ing. Michalu Sejákovi za možnost konzultace problematiky automatizovaného sběru dat.

Dále bych chtěl poděkovat všem, kteří přispěli ke zdárnému dokončení této práce, zejména svým rodičům, kteří mě během celého studia podporovali.

Obsah

Úvod 1

V době, kdy digitalizace ovlivňuje každý aspekt našeho života, není překvapivé, že elektronické loterie zaujímají významné místo ve spektru moderního hazardu. Zatímco v minulosti museli lidé, kteří chtěli zkusit své štěstí, shánět papírové tikety a sledovat noviny, ve kterých bylo zveřejněno číslo vítězného losu, dnes stačí přístup k internetu. Vsadit je možné on-line, stejně jako zobrazit si čísla nedávno vyhrávajících losů.

Sázení z pohodlí domova přináší do elektronických loterií zajímavou možnost losování s vysokou frekvencí, což v minulosti nebylo myslitelné. Dnes se ale setkáváme s loteriemi, kde nové losování probíhá každých 5 minut, čímž se nabízí hráčům téměř okamžitá a opakující se šance na výhru.

Rozvoj elektronických loterií a snadný přístup k dříve losovaným číslům může nutit k zamyšlení, do jaké míry jsou vylosovaná čísla náhodná. Pro zaručení férovosti těchto online loterií je zásadní kvalita generátorů náhodných čísel. Cílem této práce je získat sadu historických dat o proběhlých losováních z různých elektronických loterií. Tyto výsledky budou následně použity v předmětu Základy náhodných procesů na katedře matematiky na Západočeské univerzitě v Plzni. Zde budou testovány aspekty náhodnosti jednotlivých losovaných čísel, které tato práce shromáždí.

Seznam legálních provozovatelů hazardních her v České republice zveřejňuje pravidelně Ministerstvo financí.[1] Ze seznamu vyplývá, že internetové číselné loterie provozují společnosti Fortuna Game a. s., Chance a. s., Loterie Korunka s. r. o., Sazka a. s. a Tipsport.net a. s. (údaj ke dni 1. 4. 2024).

Tato práce má následující strukturu. V kapitole 2 si představíme základní metody získávání dat z internetu. V kapitole 3 budou představeny vybrané loterie, pro které bude probíhat sběr dat včetně možností, pomocí kterých je možné pro tyto loterie data získat. V kapitole 4 je podrobně popsáno programové řešení a v závěrečné kapitole 5 jsou shrnuty dosažené výsledky. V přílohách práce jsou pak dostupné uživatelské příručky pro oba programy, které k bakalářské práci vznikly.

[1] Dostupné z URL `https://www.mfcr.cz/cs/kontrola-a-regulace/hazardni-hry/prehledy-a-statistiky/prehledy-legalnich-provozovatelu-whiteli/2024`.

Metody získávání dat z internetu — 2

Data uložená na serveru je možné sbírat více způsoby - prvním a nejjednodušším je přístup pomocí API (Application Programming Interface), které programátoři webové aplikace vytvářejí za účelem snadného předávání dat mezi serverem a klientem. Takto je možné získat dobře zpracovatelná data většinou ve formátu JSON (JavaScript Object Notation) [1]. Navíc je přístup k API rychlý, není totiž nutné hledat informace přímo ve zdrojovém kódu, případně čekat na načtení požadované stránky. Pokud webová stránka umožňuje přístup k API, měla by tato metoda být pro sběr dat zvolena, jelikož je typicky implementačně mnohem jednodušší a program bývá mnohem efektivnější a rychlejší [1].

Alternativou, kterou je vhodné použít v případě, že webová aplikace neposkytuje API, je vytvoření takzvaného web crawleru. Web crawler je program, který prochází jednotlivé stránky, přičemž provádí různé operace (může například se stránkou manipulovat) a získává data přímo na základě zdrojového kódu [2]. Proces sběru dat ze zdrojového kódu se nazývá web scraping. Web scraping by bylo možné provádět i bez použití web crawleru s ručním přepínáním mezi stránkami [3]. Pokud je však web scraping automatizovaný pomocí web crawleru, jedná se o takzvané automatické harvestování.

2.1 Možnosti přístupu k API

Jelikož je přístup k API velmi používaný způsob získávání dat, vzniklo mnoho knihoven pro různé programovací jazyky, které volání API umožňují, případně ulehčují. Například součástí jazyka Java je od verze 11 balíček `java.net.http` [4]. S využitím tříd `HTTPRequest`, `HTTPResponse` a `HTTPClient` z tohoto balíku je možné vytvořit požadované volání API.

Pro jazyk Python existuje například knihovna `requests`, která poskytuje pokročilejší možnosti práce s API, kromě samotného volání například také získání dat z odpovědi ve formátu JSON, který se často používá jako formát odpovědi serveru [5]. Tuto knihovnu je však nutné instalovat, jelikož není součástí Pythonu.

Jazyk C# poskytuje například jmenný prostor `System.Net` a přístup k API se v mnohém podobá principu, jakým funguje zmíněný balíček `java.net.http` v Javě [6].

2.1.1 HTTP metody GET a POST

HTTP je protokol umožňující komunikaci mezi klientem a serverem. Během komunikace pomocí HTTP je nutné určit tzv. HTTP metodu, pomocí které budou data přenášena. Existují například HTTP metody PUT, HEAD, DELETE, ale nejpoužívanějšími jsou metody GET a POST [1]. Každá z těchto metod má své specifické vlastnosti a komunikace může probíhat jiným způsobem. Například metoda GET předává data přímo jako součást URL adresy. Z toho důvodu by pomocí GET neměla být předávána citlivá data, jako jsou hesla, která by pak byla viditelná v historii prohlížeče. Pro bezpečnější přenos dat existuje metoda POST, kde jsou data předávána přímo v těle požadavku. Pro přístup k API jsou velmi často využívány zmíněné metody GET a POST, pomocí kterých předáme serveru data specifikující náš požadavek, a server poskytne odpověď na náš dotaz.

2.2 Způsoby implementace web crawleru

Web crawler je možné naprogramovat bez využití specializovaných knihoven. V takovém případě by program posílal na server HTTP požadavky a ze získané odpovědi ve formátu HTML by vyextrahoval všechny nezbytné poznatky. Mohl by vytvořit seznam všech odkazů na stránce a podle jasně specifikovaných pravidel se rozhodnout, na jakou z těchto URL adres se ptát v příštím HTTP požadavku.

Toto řešení je ale při implementaci složitějšího web crawleru často neúnosně implementačně náročné, z toho důvodu vznikly knihovny, které mají poskytnout jednoduché rozhraní, díky kterému bude web scraping možné realizovat mnohem snadněji. V této kapitole tedy zmíníme existující software, který je často používaný k web crawlingu.

2.2.1 Selenium

Selenium[1] umožňuje pomocí zvoleného prohlížeče vykreslovat stránky a automatizuje interakce na těchto stránkách - je možné například po načtení stránky najít prvek s určeným CSS selektorem (viz dále Využití CSS selektorů, podkapitola 2.4), na ten kliknout a získat data, která se objeví v prvku s jiným CSS selektorem. Ačkoliv tato knihovna vznikla za účelem testování webových stránek, je možné ji využít právě i na automatizované harvestování [7]. Selenium je dostupné v 5 variantách

[1] Dostupné z URL `https://www.selenium.dev/`.

pro různé programovací jazyky - C#, Ruby, Java, Python a JavaScript. Aby Selenium dokázalo ovládat různé verze prohlížeče, musí však mít k dispozici aktuální ovladač, tzv. WebDriver. S každou verzí prohlížeče se může měnit způsob, jak s ním WebDriver musí zacházet. To způsobuje značné nepříjemnosti, naštěstí však existuje knihovna WebDriverManager pro programovací jazyk Java, která se o ovladače postará automaticky, sama tedy stáhne novou verzi ovladače, pokud je to nezbytné. Na druhou stranu knihovna WebDriverManager ne vždy funguje podle očekávání a uživatel musí občas podnikat kroky navíc, aby vše fungovalo správně (některé z těchto problémů popisuje v [8] tvůrce knihovny WebDriverManager, Boni García).

Selenium poskytuje mnoho pokročilejších funkcí, jako je například vykonávání JavaScriptového kódu na stránce, přesun kurzoru nad nějaký element, simulace vstupu na klávesnici a mnoho dalších, díky čemuž umožňuje vytvoření i komplexnějších web crawlerů.

Kromě výše popsané knihovny (Selenium WebDriver) existují další 2 varianty Selenia. Jedná se o Selenium IDE a Selenium Grid, ovšem Selenium WebDriver je nejrobustnějším řešením. Zatímco Selenium Grid slouží ke spouštění testů na více počítačích najednou, Selenium IDE je pouze funkcionálně osekanější verze knihovny Selenium WebDriver v grafickém uživatelském rozhraní. Vzhledem k tomu, že Selenium WebDriver je ze všech těchto verzí nejpoužívanější, často se setkáme s pouhým označením Selenium, které má poukazovat právě na Selenium WebDriver.

Selenium WebDriver umožňuje kromě použití zvoleného prohlížeče vykreslování stránek i v takzvaném *headless browser*u, který webovou stránku nezobrazuje, pouze nad ní provádí operace bez použití konkrétního prohlížeče. Díky použití tohoto univerzálního prohlížeče sice odpadne starost o ovladače a samotný web scraping bude probíhat rychleji, ale sběr dat na stránkách s JavaScriptovým kódem často způsobuje problémy.

Při každém spuštění programu se Seleniem trvá poměrně dlouhou dobu (maximálně však jednotky sekund) inicializace ovladače a spuštění prohlížeče, což trochu znepříjemňuje použití pro sběr dat v reálném čase. Pokud ale toto malé zdržení tolerujeme, je možné jednak sbírat historická data, i periodicky jednou za čas harvestovat nově dostupná data.

2.2.2 Beautiful Soup

Další knihovnou, vytvořenou pro programovací jazyk Python, je Beautiful Soup.[2] Na rozdíl od Selenia, Beautiful Soup nemanipuluje s prohlížečem, díky čemuž odpadá starost o ovladače. Jelikož používá pro web crawling pouze informace získané ze zdrojového kódu, neumožňuje tato knihovna web scraping z dynamicky generovaných stránek (stránky využívající skripty na straně klienta za účelem vykreslení

[2] Dostupné z URL `https://pypi.org/project/beautifulsoup4/`.

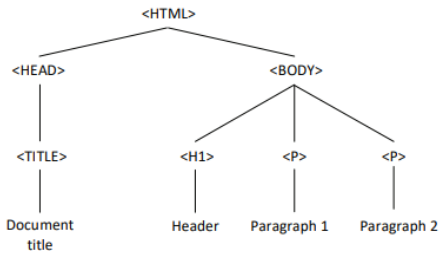
obsahu), což je zásadní nevýhoda vůči Seleniu. Beautiful Soup může být vhodné použít pro sbírání dat z jednoduchých webových stránek, pro složitější web scraping je vhodnější využít Selenium. Implementace s knihovnou Beautiful Soup však bývá jednodušší a sběr dat rychlejší než v případě Selenia, jelikož není nutné vykreslovat obsah stránky a manipulovat s prohlížečem. Beautiful Soup vytvoří pro načtenou stránku strom HTML prvků, které je pak možné procházet, nebo například vypsat text nějakého z prvků [9]. Kromě přímého procházení stromu je pak například možné získat veškeré elementy s určitým názvem HTML tagu (HTML tag určuje, o jaký element se jedná - například element s tagem *<a>* je odkaz, tag *<b>* zase značí tučně psaný text) ve stromě, nebo třeba najít prvky jen s určitým HTML atributem (specifikuje nějakou podrobnost k HTML prvku, například pro element *<a>* je často používán atribut *href* specifikující URL adresu, na kterou při kliknutí prohlížeč přejde).

2.2.3 Octoparse

Octoparse[3] je software umožňující web scraping bez nutnosti psaní kódu, uživatel nastaví parametry sběru rovnou v grafickém uživatelském prostředí. Navíc jsou stránky během sběru dat doopravdy načítány, takže stejně jako v Seleniu je vykonáván veškerý JavaScriptový kód, což umožňuje web scraping i na stránkách s dynamicky generovaným obsahem. Ačkoliv může být Octoparse vhodným řešením pro menší projekty, pro robustnější řešení je vhodnější spíše použití nějaké knihovny, které typicky nabízejí větší kontrolu nad celým web scrapingem.

2.3 Document Object Model

Document Object Model (DOM) reprezentuje prvky webové stránky pomocí stromu. Každý prvek stránky je znázorněný jako uzel stromu, a pokud je do prvku na stránce vnořený jiný element, je tento vnořený element v DOM hierarchii potomkem tohoto prvku.



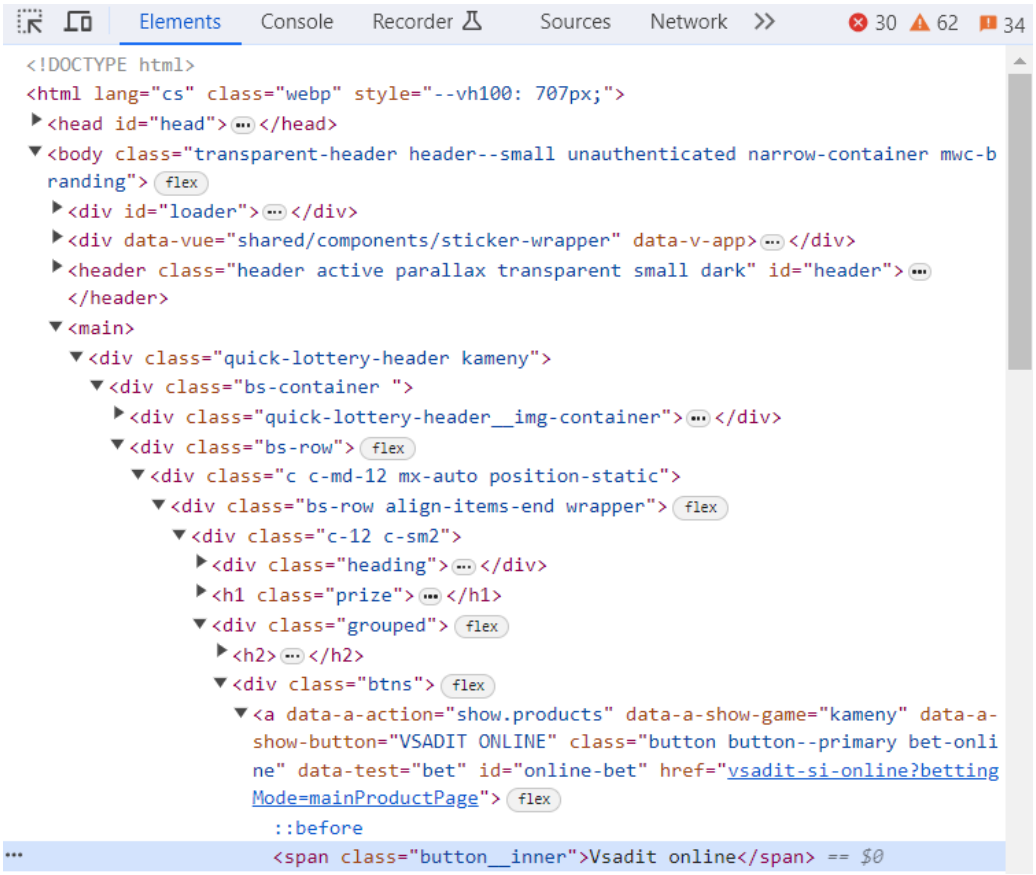
Obrázek 2.1: Příklad DOM stromu, převzato z [10]

[3] Dostupné z URL `https://www.octoparse.com/`.

Například pokud *<body>* element na stránce obsahuje prvek *<p>*, bude v DOM hrana mezi *<body>* a *<p>* znázorňující, že *<body>* je předkem elementu *<p>*. S DOM reprezentací pak dokáže pracovat například JavaScript na webové stránce, nebo právě knihovna Selenium.

2.4 Využití CSS selektorů

S využitím CSS selektoru je možné identifikovat prvek nebo skupinu prvků na stránce. Často je například používán CSS selektor identifikující prvek pomocí ID, které může být přiřazeno maximálně jednomu prvku, nebo pomocí třídy, která může být přidělena více prvkům najednou. Existuje ale mnoho dalších způsobů, jak identifikovat prvek, navíc je možné selektory kombinovat. Zkusíme se nyní podívat, jak najít CSS selektor určitého elementu. Otevřeme ve webovém prohlížeči stránku `https://www.sazka.cz/loterie/kameny/sazky-a-vysledky` a na prvek, který nás zajímá (např. tlačítko *VSADIT ONLINE*), klikneme pravým tlačítkem myši, poté zvolíme *Prozkoumat*.



Obrázek 2.2: Procházení zdrojového kódu loterie Kameny

Otevřou se Nástroje pro vývojáře a vidíme zdrojový kód stránky s vyznačeným elementem, na který jsme klikli pravým tlačítkem. Vidíme, že prvek je reprezentovaný jako element `span` a má nastavenou třídu (*class*) na *button__inner*, ale nemá nastavené žádné ID, které by ho jednoznačně definovalo. Přepneme se nyní v Nástrojích pro vývojáře na kartu *Console* a pokusíme se zjistit, zda prvek s touto třídou je na stránce jen jeden, nebo je nutné CSS selektor více specifikovat. Karta *Console* umožňuje zápis JavaScriptového kódu, který se následně vykoná. Pokud chceme najít všechny prvky s určitým CSS selektorem, je možné využít funkci `document.querySelectorAll(cssSelector)`. Pro prvek s ID *tlacitko* je pak CSS selektor definovaný jako *#tlacitko*, s třídou *specialni* například *.specialni*, všechny prvky `span` mají CSS selektor jednoduše *span*. CSS selektory je možné kombinovat, pokud se zajímáme o všechny prvky s tagem `span` a třídou *specialni*, CSS selektorem bude *span.specialni*, pokud bychom chtěli vybrat všechny prvky s třídou *specialni* které jsou potomkem prvku s ID *tlacitko*, selektorem bude *#tlacitko .specialni*. CSS selektory umožňují mnoho dalších způsobů identifikace prvků, ale pro účel demonstrace stačí výše zmíněné možnosti. My tedy vykonáme následující kód:

```
document.querySelectorAll(".button__inner")
▼ NodeList(2) [span.button__inner, span.button__inner] i
  ▶ 0: span.button__inner
  ▶ 1: span.button__inner
    length: 2
  ▶ [[Prototype]]: NodeList
```

Obrázek 2.3: Získání všech prvků s určitým CSS selektorem

Jak je vidět, na stránce existují dva odpovídající prvky. Nicméně je možné využít složitějších CSS selektorů, aby byl náš výběr jednoznačný. Z obrázku 2.2 je vidět, že prvek, o který se zajímáme, je vnořený do prvku s atributem id="online-bet". Jelikož ID je jedinečné na celé stránce, je možné náš prvek definovat pomocí selektoru *#online-bet .button__inner*, vybíráme tedy pouze takové prvky s třídou *button__inner*, které jsou potomky prvku s ID *#online-bet*. Po této modifikaci dostaneme pomocí funkce `document.querySelectorAll` jen prvek, o který jsme se zajímali.

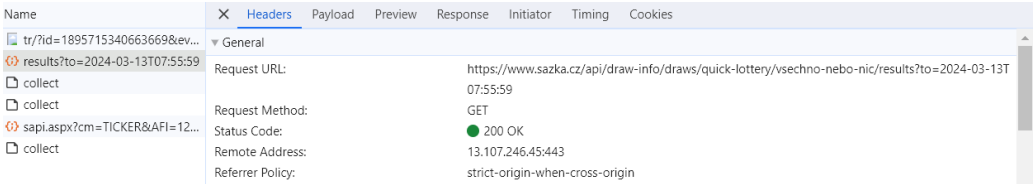
Metodologie 3

3.1 Dostupnost dat u vybraných loterií

Tato bakalářská práce se zaměřuje na sběr dat z následujících loterií, provozovaných společností Sazka a. s.: Kameny, Kasička, Keno a Rychlé kačky. Ačkoliv jsou zmíněné loterie zřízeny stejnou společností, uživatelské prostředí, kde jsou přístupné výsledky, se mezi hrami liší. Obsah stránek je generován dynamicky, přepínání mezi dostupnými daty vyžaduje většinou složitější manipulace se stránkou. Dále jsou do výběru zařazeny následující loterie společnosti Fortuna Game: Lucky Six Online, Lucky X, Next 6, Lucky Six Pobočka. Pro tyto loterie je však uživatelské prostředí téměř totožné a sběr probíhá pro všechny hry stejným způsobem.

Na stránkách loterií jsou dostupná historická data, ovšem nikoliv veškerá. Například loterie Keno umožňuje zobrazení výsledků losování za posledních 365 dní. Kameny a Keno jsou losovány každých 5 minut, Kasička dvakrát za den a Rychlé kačky každou minutu.

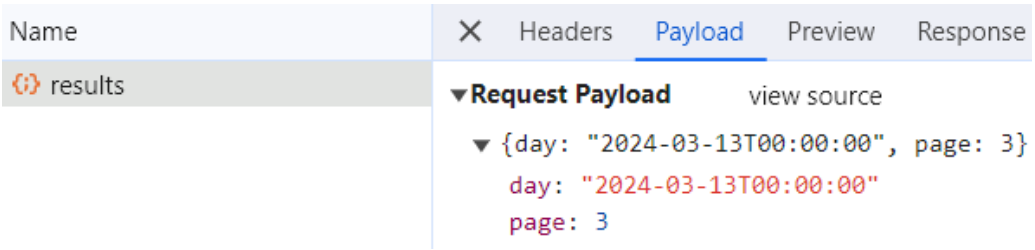
Nicméně, jak nyní dokážeme, pro všechny tyto loterie existuje API. Společnost Sazka ani společnost Fortuna Game jeho přesnou implementaci nezveřejnila, ale je možné ji alespoň částečně odvodit, pokud ve zvoleném prohlížeči otevřeme Nástroje pro vývojáře a zvolíme záložku *Network*. Pak stačí načíst stránku s loterií a pozorovat, jaké požadavky jsou posílány na server při procházení výsledků loterie. Například na stránce `https://www.sazka.cz/loterie/vsechno-nebo-nic/vysledky` si můžeme všimnout, že se odešle požadavek v následujícím tvaru:



Obrázek 3.1: Ukázka GET požadavku v loterii Všechno nebo nic

Díky tomuto postupu jsme získali URL adresu API a zjistili jsme, že je umožněn přístup pomocí metody GET s parametrem *to*, který zřejmě značí, jakým dnem a časem mají být výsledky omezeny (např. výsledky starší než 13. 3. 2024, 7:55:59).

Pokud přepneme na záložku *Payload*, zobrazí se přehledně parametry dotazu. Tato funkcionalita je užitečná zejména pro loterie, které získávají data metodou POST, jelikož posílaná data nejsou přímo součástí URL adresy. Pro loterii Kameny se zobrazí něco podobného:



Obrázek 3.2: Ukázka POST požadavku v loterii Kameny

Vidíme, že data z jednoho dne jsou rozdělena po stránkách, které můžeme postupně navyšovat a sebrat tak všechna dostupná data za určitý den.

Obdobným způsobem můžeme postupovat u všech loterií a zkoumat, jak vypadá požadavek na API. Pokud zkusíme měnit parametry volání a vytvářet vlastní dotazy, zjistíme, že pomocí API je navíc u některých loterií možné získat starší data než přes webové rozhraní, které je omezeno například na 365 minulých dní.

Grafické uživatelské rozhraní (GUI) stránky se může časem měnit, což by při použití web crawleru znamenalo nutnost změnit konfiguraci, pomocí které se budou data stahovat. Na druhou stranu, změna API je mnohem méně pravděpodobná.

3.1.1 Loterie Rychlé kačky

Výsledky této loterie jsou dostupné na stránce `https://www.sazka.cz/loterie/rychle-kacky/vysledky`. Data jsou zobrazena v tabulce, při kliknutí na tlačítko *NAČÍST DALŠÍ* přibudou do tabulky starší data, původní tam ale zůstanou také. API používá HTTP metodu GET s adresou ve tvaru https://www.sazka.cz/api/draw-info/draws/quick-lottery/rychle-kacky/results?to=<rok>-<měsíc>-<den>T<hodina>:<minuta>:<sekunda>. Server jako odpověď na požadavek vrací JSON pole s jednotlivými výsledky losování v tomto tvaru (příklad):

```
[
    {
        "drawId":3014269,
        "drawDate":"13. 3. 2024",
        "drawTime":"15:36",
```

```
        "utcDrawDate":null,
        "localDrawDate":"2024-03-13T15:36:00",
        "results":[49,61,50,8,47,46,51,30,42,52,21,29]
    },
    ...
]
```

Ačkoliv je API nedokumentovaná, je možné ze získaných výsledků odvodit význam jednotlivých JSON vlastností:

JSON vlastnost	význam
`drawId`	Jedinečný identifikátor slosování.
`drawDate`	Datum losování.
`drawTime`	Čas losování s přesností na minuty.
`utcDrawDate`	Pro libovolné datum a čas je vždy poskytnuto pouze *null*, zřejmě tedy nebylo nutné tuto funkcionalitu implementovat.
`localDrawDate`	Datum a čas losování, tentokrát je ale poskytnut čas s přesností na sekundy.
`results`	Pole čísel, která byla vylosována.

Tabulka 3.1: Význam dat získaných z API loterie Rychlé kačky

3.1.2 Loterie Kameny a Keno

Výsledky loterií Kameny a Keno jsou zobrazeny s téměř identickým grafickým uživatelským rozhraním, přičemž zobrazení výsledků vyžaduje totožné kroky. Data jsou dostupná na adrese `https://www.sazka.cz/loterie/kameny/sazky-a-vysledky`, respektive `https://www.sazka.cz/loterie/keno/sazky-a-vysledky`.

Pro zobrazení tabulky s posledně losovanými čísly je nutné kliknout na tlačítko *MINULÉ VÝSLEDKY*. Pro zobrazení starších dat během určitého dne (ve výchozím nastavení se jedná o den, ve kterém si výsledky zobrazujete) je možné kliknout na tlačítko šipky doprava, zobrazené pod tabulkou. Pokud již starší záznamy z určeného dne neexistují, tlačítko šipky zmizí. Pro přepnutí mezi jednotlivými dny je možné použít nabídku dnů vpravo nad tabulkou. Jelikož je stránka generována dynamicky a reaguje na pohyby kurzoru nad určitými prvky, implementace přepnutí na jiný den pomocí web crawleru by nebyla triviální, vyžadovala by simulaci pohybu kurzoru.

Jak již bylo uvedeno výše, loterie Kameny používá k přístupu k API metodu POST. Stejně tak loterie Keno, obě tyto API vyžadují stejné parametry a vrací stejně formátovaná data. URL adresy API jsou následující: `https://www.sazka.cz/api/draw-info/draws/kameny/results` pro Kameny a pro Keno se používá URL adresa `https://www.sazka.cz/api/draw-info/draws/keno/results`. Jak je uvedeno v podkapitole 3.1, při volání API se předávají 2 parametry, *day* a *page*. Vrácená data mají pak tento formát:

```
{
    'results':
    [
        {
            'combinations': '---',
            'drawDate': '15. 12. 2023',
            'drawId': 1384988,
            'drawTime': '23:56',
            'localDrawDate': null,
            'number': 24,
            'results': [5, 1, 3, 3, 6, 6],
            'utcDrawDate': null
        },
        ...
    ],
    'totalPages': 29
}
```

Vidíme, že data jsou nyní vrácena ve vlastnosti *results* JSON objektu, který získáme jako odpověď na volání API. Kromě výsledků je vrácena ještě vlastnost *totalPages*, jejíž hodnota určuje počet stránek s daty dostupných pro určitý den (viz podkapitola 3.1). Výsledky losování mají stejné vlastnosti jako u loterie Rychlé kačky, kromě těchto změn:

JSON vlastnost	význam
`combinations`	Pojmenování kombinace vylosovaných čísel. Například pro výsledek [5, 1, 1, 6, 6, 5] bude kombinace "Tři dvojice", pro [6, 6, 6, 5, 6, 5] "Čtveřice"apod. Pokud není vylosována žádná speciální kombinace, bude hodnota vlastnosti *combinations* "—".
`localDrawDate`	Pro obě loterie je pokaždé vráceno pouze *null*.

(tabulka pokračuje na další stránce)

Tabulka 3.2 *(pokračování z předchozí stránky)*

JSON vlastnost	význam
`number`	Pouze pro loterii Kameny, pro loterii Keno není vlastnost *number* vrácena. Určuje součet vylosovaných čísel z vlastnosti *results*, na který je možné vsadit.

Tabulka 3.2: Význam dat získaných z API loterií Kameny a Keno

3.1.3 Loterie Kasička

Loterie Kasička je dostupná na stránce `https://www.sazka.cz/loterie/kasicka/sazky-a-vysledky`. Po kliknutí na odkaz *STÁHNOUT VÝHERNÍ LISTINU* se zobrazí výsledky za poslední týden. Data za celý týden jsou zobrazena v jedné tabulce na stránce. Kasička nepodporuje přímý přechod mezi týdny, nicméně je možné zobrazit dřívější data úpravou URL parametrů. URL má tvar https://www.sazka.cz/system/vyherka?year=<rok>&week=<týden>&game=kasicka a je tedy možné procházet všechny týdny ve všech dostupných letech.

Pokud otevřeme v prohlížeči Nástroje pro vývojáře a pokusíme se pro Kasičku najít URL adresu API, zjistíme, že žádný síťový požadavek není posílán, ale data jsou ze serveru vrácena rovnou jako součást stránky, na které jsou poté vykreslena. Experimentálně jsem však ověřil, že pokud v URL adrese API pro loterii Rychlé kačky změníme řetězec *rychle-kacky* za *kasicka*, použijeme tedy pro API URL adresu ve tvaru https://www.sazka.cz/api/draw-info/draws/quick-lottery/kasicka/results?to=<rok>-<měsíc>-<den>T<hodina>:<minuta>:<sekunda>, dostaneme výsledky z loterie Kasička, které mají navíc stejný formát jako výsledky Rychlých kaček.

3.1.4 Loterie společnosti Fortuna Game

Výsledky rychlých loterií provozovaných společností Fortuna Game jsou dostupné na stránce `https://www.ifortuna.cz/cms/loterie/vysledky-rychlych-loterii`. Pokud však otevřeme Nástroje pro vývojáře a procházíme zdrojový kód stránky, všimneme si, že samotné výsledky jsou zobrazeny na stránce pomocí prvku *<iframe>*, který slouží pro vykreslování obsahu z jiné URL adresy přímo na stránku. URL adresa, ze které jsou výsledky vykreslovány a kterou bude výhodnější použít pro web scraping, je následující: `https://feg-virt-webshellroc.ifortuna.cz/`.

V horním menu je možné vybrat jednu ze zmíněných loterií, tedy Lucky Six Online, Lucky X, Next 6 nebo Lucky Six Pobočka. Dále je v nabídce možné vybrat, z jakého dne mají být zobrazeny výsledky a z jakého časového rozmezí (ve čtyřhodinových intervalech). Pokud se podíváme opět do záložky *Network* v Nástrojích pro vývojáře, všimneme si, že je opět používáno API pro všechny tyto loterie. Dotaz na server je poslán pomocí metody GET a URL adresa má například tento tvar: `https://webshelloa.ifortuna.cz/api/results?gameType=1000&startDateTime=2024-04-21T10:00:00.000Z&endDateTime=2024-04-21T14:00:00.000Z`. Jednotlivé hry jsou identifikovány pomocí parametru *gameType* následujícím způsobem.

loterie	**hodnota parametru *gameType***
`Lucky Six Online`	1000
`Lucky X`	1001
`Next 6`	1002
`Lucky Six Pobočka`	1010

Tabulka 3.3: Hodnota parametru *gameType* pro jednotlivé loterie

Parametry *startDateTime* a *endDateTime* omezují, z jakého data a času mají být vráceny výsledky losování. Server vrátí jako odpověď JSON pole v následujícím tvaru.

```
[
    {
        "eventId":579787,
        "eventIdToday":102,
        "startDateTime":"2024-04-21T10:01:03Z",
        "result":
        [
            41,5,4,35,2,15,30,47,31,33,19,40,
            20,39,17,9,45,36,32,37,21,14,11,
            29,8,34,16,22,27,44,42,38,24,13,6
        ]
    },
    ...
]
```

Nicméně pro nedávné slosování vrací API ve vlastnosti *result* hodnotu *null*.

3.2 Požadavky na web crawler pro vybrané loterie

Z předchozí sekce vyplývá, že pokud bychom chtěli vytvořit web crawler, který by byl schopný sbírat data z určených loterií, musí kromě procházení webu a klikání na tlačítka nebo odkazy umět také simulovat pohyb kurzoru, nebo umožnit načítat stránku s modifikovanými URL parametry, abychom byli schopni procházet jednotlivé sázkové týdny u loterie Kasička.

Web crawler by měl umožnit jednoduše měnit způsob, jakým jsou data z jednotlivých loterií sbírána pro případ, že by se změnila forma reprezentace dat na stránce. Také by měl umožnit v případě potřeby přidání dalších loterií. Bylo by nepraktické, kdyby způsob, jakým se mají sbírat data, byl popsán přímo v kódu programu a uživatel musel v případě jakékoliv změny měnit zdrojový kód. Z toho důvodu by se hodilo dát uživateli možnost řídit získávání například pomocí konfiguračního souboru, který bude moci modifikovat.

Je důležité, aby web crawler nějakým způsobem umožnil čekání na načtení stránky a čekání, než se na stránce načtou data. Výstup by měl být zapsán ve strojově dobře čitelném formátu do výstupních souborů, například ve formátu CSV.

Musí být umožněný web scraping pouze nových dat, který se zastaví, jakmile by program měl začít sbírat stará data, díky čemuž bude možné implementovat sběr dat v reálném čase. Také, jelikož loterie Rychlé kačky přidává nová data na konec tabulky s již načtenými daty, musí web crawler umožnit načítat jen data nově přidaná na stránku, ne vždy načítat všechna data na stránce.

3.3 Problémy s přístupem k API

Přístup k API může být omezený z více důvodů. Jednak by se mohlo stát, že bude API neustálými požadavky zahlceno a nedokáže obsloužit regulérní uživatele, ale vytvoření určitých limitů na straně serveru může být také opodstatněno tím, aby bylo obtížné získat všechna potenciálně dostupná interní data. Některá API vyžadují pro přístup vygenerování speciálního klíče, který je pak předáván jako parametr volání a mohou omezit počet přístupů s tímto klíčem za časovou jednotku, nebo například dovolit neomezený přístup k API jen po zaplacení poplatku. Další možností limitace je omezení přístupu na základě IP adresy, jak zmiňuje například [11].

API, které jsou používány v námi definovaných loteriích nevyžadují generování klíče. Na druhou stranu, jelikož funkcionalita těchto API není zdokumentována, není jasné, jaký způsob limitování IP adresy (pokud vůbec) je používán, lze ovšem předpokládat, že nějaký ano. Pokud předpokládáme, že je omezený přístup k API na základě IP adresy, není možné posílat sekvenčně jeden požadavek za druhým, ale je nezbytné program mezi jednotlivými sběry dat uspat. Aby nedokázal pokročilejší

software na straně serveru odhalit ani uspání na vždy stejnou dobu, je ideální uspávat program nebo vlákno na náhodně zvolenou dobu z definovaného intervalu, čímž se eliminuje riziko odhalení. Tato doba však nesmí být příliš krátká, aby server používanou API adresu nezakázal.

Pokud by přesto došlo k zařazení IP adresy používané pro sběr dat na černou listinu, je možné využít proxy IP adresu podporující HTTP respektive HTTPS požadavky a vydávat se za jiný stroj. Komunikace se serverem pak ale bývá zdlouhavější.

Řešení 4

Práce nejdříve uvažovala pouze implementaci web crawleru, pomocí kterého budou sesbírána všechna dostupná data. Během poznávací části však bylo nalezeno nedokumentované API společnosti Sazka i společnosti Fortuna Game a jelikož má tento postup mnoho výhod (viz kapitola 2) a naopak web scraping přinášel nové problémy (viz dále), bylo po konzultacích domluveno vytvoření programu, který sbírá data pomocí API. Nejen z důvodu splnění zadání práce, které počítá s vytvořením web crawleru, ale také z důvodu porovnání obou metod jsou nakonec součástí bakalářské práce dva programy a každý z nich používá jinou z těchto metod.

4.1 Web crawler

Z podkapitoly 3.2 vyplývá, že je nezbytné umožnit složitější sběr dat, např. i se simulovaným pohybem kurzoru, navíc umožňující manipulaci s dynamicky generovaným obsahem stránek, jak je uvedeno výše. Z toho důvodu se jeví jako ideální řešení použít knihovnu Selenium, která všemi těmito funkcemi disponuje. Aby odpadla starost o stahování nového WebDriveru pro každou novou verzi prohlížeče, je využívána knihovna WebDriverManager, což implikuje použití Javy jako programovacího jazyka, jelikož právě pro tento jazyk je knihovna napsána. Aby bylo navíc jednoduché změnit verzi použitých knihoven, jako nástroj pro sestavení projektu byl zvolený Gradle.

Ačkoliv knihovna WebDriverManager zjednodušuje práci automatickým stahováním ovladačů prohlížeče, není příliš spolehlivá. Občas je možné vyřešit problém jednoduše tím, že se knihovna aktualizuje na nejnovější dostupnou verzi. Z toho důvodu je v souboru *build.gradle*, který řídí sestavení projektu, uvedeno, že má být při sestavení připojena nejnovější verze WebDriverManageru. To však na druhou stranu může způsobit případné komplikace, pokud by nějaká funkcionalita využívaná programem přestala být v novějších verzích podporovaná, nicméně tento problém je možné jednoduše vyřešit změnou verze knihovny na starší, ve které vše funguje správně. I přes aktualizaci knihovny se mohou objevit potíže, v tom případě se pravděpodobně objeví jejich možné řešení na Github stránce projektu v záložce

Issues (`https://github.com/bonigarcia/webdrivermanager`). Během práce na bakalářské práci se například přihodilo, že bylo nutné smazat dočasné soubory vygenerované WebDriverManagerem, aby vše fungovalo správně.

Naopak Selenium je stabilní a není nutné měnit jeho verzi, takže používaná verze 4.1.2 by měla fungovat bez problémů. Případná změna verze by opět mohla způsobit komplikace, ale je možná.

Aby bylo možné sběr dat přizpůsobit potřebám uživatele, bude se program řídit konfiguračním souborem *config.json*, ze kterého si přečte potřebné informace. V souboru *config.json* jsou navíc uvedeny konfigurační soubory pro jednotlivé loterie, ve kterých může uživatel změnit výchozí konfiguraci a nastavit nějaké detaily, které mohou být pro jednotlivé loterie odlišné. Tímto způsobem je možné si chod programu přizpůsobit podle vlastních požadavků.

Uživatelská příručka, která popisuje mimo jiné možná nastavení konfigurace, je dostupná v příloze A.

4.1.1 Načítání konfigurace

O načítání konfigurace se stará třída `ConfigManager`. Každá vlastnost konfigurace je reprezentovaná jako objekt třídy `Property`. Třída `Property` navíc poskytuje statické metody, které `ConfigManager` používá pro jednodušší načítání konfigurace. Třída tedy například umožňuje získat ze seznamu objektů typu `Property`, které obsahují jednotlivé hodnoty vlastností, pouze jednu třídu, kde jsou již všechny hodnoty vlastností ověřeny a mají správný datový typ (`Property` definuje hodnotu vždy jednoduše jako řetězec). Tato třída, uchovávající informace o globální konfiguraci (konfiguračním souboru *config.json*) se jmenuje `GlobalProperties`, zatímco třída, která uchovává informace o jednotlivých loteriích se nazývá `SiteProperties`.

4.1.2 Web crawling

Samotný web scraping je spuštěný pomocí třídy `HTMLManager`, která postupně pro všechny definované loterie pomocí objektu třídy `HTMLManipulator` spustí sběr dat. K tomu slouží metoda `start`, která pro každou loterii nejdříve načte stránku, poté vykoná akce definované v *beforeActions* a následně cyklicky sbírá data, pokud ještě nějaká existují. Aby bylo možné načíst správně další stránku, je nutné ještě zavolat metodu `htmlManipulator.endExecuting()`. Po dokončení sběru ze všech loterií je třeba ještě tento sběr korektně ukončit.

`HTMLManipulator` se řídí konfigurací konkrétní loterie, přičemž tato konfigurace je mu předána jako parametr metody `prepareSite`. Třída umožňuje například provádět uživatelem definované operace před sběrem dat a mezi jednotlivými sběry. `HTMLManipulator` zároveň přímo manipuluje s ovladačem prohlížeče, který ve svém konstruktoru inicializuje (jedná se buď o ovladač pro Chrome, nebo Edge).

Během této inicializace je na chybový výstup vypisováno velké množství nepotřebných debugovacích informací, kterých se program zbaví přesměrováním chybového výstupu do dočasného souboru, přičemž po inicializaci ovladače je chybový výstup opět obnoven a dočasný soubor vymazán. Některé další výpisy vytvářené Seleniem jsou vypnuty v třídě `Main`. Jelikož jsou v tomto případě použity logovací knihovny Javy, je možné výpisy vypnout tímto jednoduchým způsobem:

```
1 java.util.logging.Logger.getLogger("org.openqa.selenium")
2     .setLevel(Level.OFF);
```

4.1.3 StaleElementReferenceException

Výjimka `StaleElementReferenceException` nastane, pokud máme uloženou referenci na prvek na stránce, pak bude stránka znovu načtena nebo bude změněný DOM stránky (například bude přidán nový prvek) a poté se pokusíme přistoupit k našemu prvku (vykonat s ním nějakou operaci). Ačkoliv program neuchovává reference na žádné prvky dlouhodobě, může se přesto teoreticky stát, že DOM stránky bude dynamicky změněn zrovna ve chvíli, kdy budou procházeny jednotivé záznamy a výjimka `StaleElementReferenceException` bude Seleniem vyhozena. Program si v takovém případě dokáže poradit tím, že začne celý sběr pro určitou loterii úplně od začátku, maximálně však třikrát. Pokud bude i po tomto třetím pokusu znovu vyhozena výjimka `StaleElementReferenceException`, rozhodne se radši sběr ukončit, jelikož zřejmě uživatel zadal špatnou konfiguraci, která tyto problémy způsobuje, a program by tak mohl neustále začínat od úplného začátku.

4.1.4 Výpis získaných dat

O výpis dat se stará třída `Logger`. Kromě výpisu do souboru je možné data zapisovat také na konzoli. Navíc konfigurace dovoluje definovat, kam bude vypsán chybový výstup. O samotný výpis dat na výstup se stará metoda `logOut`. Jelikož může být sběr dat přerušen výjimkou `StaleElementReferenceException` (viz podkapitola 4.1.3), v takovém případě by probíhal sběr znovu, je nutné odfiltrovat data, která již jednou byla zapsána. Z toho důvodu je jedním z atributů třídy `Logger` atribut `outRows` typu `HashSet<String>`, do kterého jsou ukládány všechny zaznamenané hodnoty. Pokud je právě zapisovaný záznam v tomto `HashSetu`, zapisovat se již nebude. `HashSet` je použit z toho důvodu, že operace nad ním je možné provádět v čase *O(1)*. Mezi další důležité metody patří `logErr` pro výpis na chybový výstup a `closeLogger`, která zavře případné soubory pro výpis dat a chybový výpis, pokud jsou používány. `Logger` dále například počítá, kolik záznamů bylo zapsáno, což je vhodné, pokud je v konfiguraci nastavena vlastnost *maxData*, která omezuje počet získaných dat.

4.1.5 Automatické zastavení sběru dat

Třída `ExternalDataManager` ukládá do souboru *.program_data.txt* informace o posledním záznamu pro jednotlivé loterie, díky čemuž je možné sběr dat automaticky ukončit v momentě, kdy web crawler narazí na jednou již získaná data. To je vhodné pro opakovaný web scraping v reálném čase. Do souboru *.program_data.txt* je dále ukládána hodnota proměnné *i* pro jednotlivé loterie (proměnné jsou popsány dále v textu). Mezi důležité metody, které třída `ExternalDataManager` poskytuje, patří zejména metoda `read` pro čtení těchto metadat a `write` pro zápis. Metoda `incrementAndGetI` pro určitou loterii zvýší proměnnou *i* a vrátí její novou hodnotu. Metoda `addLastRecord` pak například uloží informaci o tom, jakým záznamem má příští sběr skončit, `lineIsBreakPoint` dokáže určit, zdali záznam, který je právě zpracovávaný, není právě záznamem, který již byl minule zaznamenán jako první (program má v takovém případě skončit). Pro snazší manipulaci s daty definuje třída `ExternalDataManager` dvě statické podtřídy `SiteIndexInfo` a `LastRecordInfo`, které však slouží pouze jako přepravky (přepravka je datová struktura sloužící k seskupení více hodnot do jednoho objektu).

4.1.6 Proměnné ve vlastnostech *sysout* a *syserr*

Vlastnost *sysout* konfiguračního souboru určuje, kam budou zapsána data získaná web scrapingem, zatímco *syserr*, jaký bude použit chybový výstup. V obou těchto vlastnostech je možné použít proměnné, které umožní generování přehlednějších názvů výstupních programu. Například proměnná *i* je při každém spuštění sběru dat pro konkrétní loterii zvýšena o 1, což umožňuje pojmenovávat výstupní soubory například postupně *1.csv*, *2.csv*, ... Nebo je díky proměnným možné pojmenovávat výstupní soubory podle dne, kdy byl sběr dat spuštěný. Více informací o proměnných je dostupných v příloze práce. O práci s proměnnými se stará zejména třída `Variables`. Její statická metoda `initialize` je využívána během spuštění programu a nastaví datum a čas, kdy byl program spuštěný (může být opožděno v řádu zlomků sekundy). Další statická metoda, `replaceWithVariables` pak umožňuje v řetězci zadaném jako hodnotu jedné z těchto vlastností nahradit všechny proměnné jejich hodnotou. Jelikož znak *$* má v používané metodě `replace` speciální význam, je to totiž speciální znak značící konec řetězce, je nutné převést každý *$* na nějaký jiný speciální znak. Ve vytvořeném programu je k tomuto účelu využíván znak €.

4.1.7 Výjimky a ukončení programu

Představme si loterii, ve které se objeví nová data po kliknutí na tlačítko *další* a stará data zmizí. Zároveň si představme, že tímto způsobem je možné získat informace

o všech losováních, není tedy nutné například přepínat mezi různými dny, vždy stačí pouze kliknout na dané tlačítko. V tomto případě bude konfigurace celkem jednoduchá - uživatel jen určí, že mezi jednotlivými sběry dat je nutné kliknout na tlačítko *další*. Nicméně po nějaké době program buď klikne na tlačítko *další*, zjistí, že již nejsou zobrazena žádná data, a sám ukončí sběr dat, nebo se tlačítko *další* vůbec nezobrazí. V takovém případě bude vyhozena výjimka, kterou program odchytí, a i v této alternativě určí, že by měl skončit. To tedy znamená, že program sám automaticky skončí, pokud již nejsou zobrazena žádná data. V určitých případech by ale bylo užitečné na vyhození výjimky reagovat a nějakým způsobem tuto událost obsloužit. Například u loterie Keno, kde je možné se v rámci jednoho dne posouvat pomocí tlačítka s šipkou doprava, ale pak šipka zmizí a je nutné přepnout na další den. Právě zde by se přepnutí na jiný den mohlo odehrávat jako obsluha výjimky, která je vyhozena, pokud se program pokusí kliknout na neexistující tlačítko. Program umožňuje reakci na tyto výjimky tím, že akce, která může vyhodit výjimku je uzavřena mezi akce *try start* a *try end* a reakce na vyhození výjimky následuje mezi akcemi *catch start* a *catch end* (viz příloha A).

4.1.8 Čekání

Jedním ze zásadních problémů při sběru dat je určit, jak dlouho má program čekat, než budou data na stránce zobrazena. Čekání na načítání stránky je celkem jednoduché, Selenium umožňuje následující zápis.

```
1 driver.get(href);
2 new WebDriverWait(driver, Const.PAGE_LOAD_TIMEOUT)
3     .until(ExpectedConditions.urlToBe(href));
```

Tímto způsobem program vždy automaticky čeká, než bude stránka načtena. Konstanta `Const.PAGE_LOAD_TIMEOUT` specifikuje, po jakém timeoutu má Selenium načítání vzdát (například pro případ nedostupnosti síťového spojení). Při jednotlivých přechodech mezi daty je již situace komplikovanější. Selenium sice nabízí možnost čekat na zobrazení určitého prvku, jenže by mohl nastat problém s nalezením CSS selektoru, který bude tento prvek jednoznačně identifikovat. Čekáme na načtení všech dat, ale přitom nemusíme nutně vědět, kolik jich bude a jaký CSS selektor bude možné použít pro identifikaci posledního načteného prvku. Nejlepším řešením se tedy zdá použít čekání po stanovenou dobu, kterou určí uživatel. Ačkoliv i tuto funkcionalitu poskytuje Selenium, jako nejstabilnější a nejlépe použitelné řešení v programu bylo zvoleno jednoduché uspání celého vlákna.

4.1.9 JavaScriptový kód

Některé funkce, které poskytuje Selenium, se nechovají úplně ideálně. Například pokud použijeme knihovní funkci Selenia pro kliknutí na element, v případě, že

sice element na stránce je, ale není momentálně vidět na obrazovce, bude vyhozena výjimka. To by bylo velmi nepříjemné, vzhledem k tomu, že program může otevřít prohlížeč v okně s různou velikostí - vlastně by se mohl chovat pokaždé jinak. Právě k vyřešení tohoto problému můžeme využít pokročilejší funkcionalitu, kterou Selenium nabízí, a to vykonávání JavaScriptového kódu na zobrazené stránce. Kód, který simuluje kliknutí na element na stránce, ať již je právě zobrazený na obrazovce nebo mimo obrazovku, pak vypadá takto:

```
((JavascriptExecutor) driver).executeScript(String.format(
   document.querySelector('%s').click();", cssSelector));
```

V kódu vidíme funkci `document.querySelector`, která se chová podobně jako dříve zmíněná funkce `document.querySelectorAll`, jen s tím rozdílem, že vrací právě jeden prvek odpovídající danému CSS selektoru, případně *null*, pokud takový prvek neexistuje. Pokud by funkce `document.querySelector` vrátila pro uživatelem specifikovaný CSS selektor *null*, následné přistoupení k metodě `click` vyvolá v JavaScriptu chybu `TypeError` a Selenium si s tím poradí vyhozením výjimky `JavascriptException`, na kterou může uživatel právě například reagovat, pokud je vykonávaný kód v bloku `try`. Pokud tomu tak není, sběr dat skončí.

4.1.10 Sběr dat v reálném čase

Aby program sbíral data v reálném čase, je třeba při spuštění zadat parametr *realtime*. Program pak vždy vykoná jeden sběr pro všechny loterie a na uživatelem definovanou dobu se uspí. Po probuzení bude znovu spuštěn sběr postupně u všech loterií a takto program postupuje cyklicky donekonečna, dokud není ukončen. Je tedy možné například nastavit, aby se každých 6 hodin stáhla veškerá nově dostupná data, pokud uživateli nevadí, že program bude celou dobu spuštěný. O sběr dat v reálném čase se stará třída `RealTimeMain` a její metoda `main`, která je zavolaná z hlavní třídy `Main` při spuštění s odpovídajícím parametrem. V metodě `main` třídy `RealTimeMain` je načten vstup od uživatele odpovídající počtu milisekund, které má program spát mezi jednotlivými sběry, následně pouze v cyklu volá `Main.main(new String[]{});` a spí.

4.1.11 Inkrementální sběr dat

Pokud jsou při načítání nových dat na stránce stále ponechána i starší, již načtená data (například loterie Rychlé kačky), program by sice mohl postupovat jako v jiných případech, přičemž by již zapsané výsledky byly odfiltrovány pomocí `HashSetu` popsaném v podkapitole 4.1.4, nicméně během testování jsem zjistil, že takový přístup způsobuje zásadní zpomalení programu, zvláště když již je na stránce načteno větší množství dat. Z toho důvodu je možné sběr dat spouštět v režimu *incremental*, který stará data ignoruje. Data z tabulky jsou získávána následujícím způsobem.

```
toAdd = getElements(String.format("%s tr:nth-of-type(%d) %s",
        properties.getTargetTable(), i++, selector));
```

Zde nabývá `selector` hodnoty buďto *td* nebo *th* v závislosti na tom, zda jsou právě sbírána data z hlavičky nebo těla tabulky. Použitím *:nth-of-type* CSS pseudotřídy můžeme vybrat pouze *n*-tý prvek se zmíněným identifikátorem, tedy *n*-tý řádek v uživatelem definované tabulce (ta je získána voláním `properties.getTargetTable()`). Právě proměnná *i*, která určuje řádek s daty, se skládá mimo jiné i z postupně se zvyšujícího inkrementu, pokud je spuštěn sběr dat v inkrementálním režimu. Díky tomu při další iteraci, kdy budou opět procházena nová data, již nebude *i* začínat na hodnotě 1 a v důsledku budou některé záznamy v tabulce vynechány.

4.1.12 Zápis dalších informací na výstup

Data o losování, která jsou dostupná v tabulkách, nemusí pokaždé poskytovat kompletní informace. Například pro loterii Kameny by bylo vhodné ke každému záznamu uložit i datum, ze kterého pochází. Ačkoliv toto datum není součástí řádky s daty v tabulce, je na stránce s loterií zobrazeno hned na dvou místech. Program tedy umožňuje přidávat do jednotlivých záznamů i obsahy jiných elementů na stránce, které nejsou přímo součástí cílené tabulky. Toho je možné dosáhnout pomocí vlastnosti *rowData*. V konfiguraci je možné například zapsat:

```
"rowData": [
    "content #datum",
    "tableContent"
  ]
```

Díky tomu bude ještě před samotnými daty z tabulky (*tableContent*) vypsán na výstup obsah elementu s ID *datum*.

4.1.13 Problém s knihovnou WebDriverManager

Zřejmě nejzávažnějším problémem, který však nejde úplně odfiltrovat, je relativně častá nefunkčnost knihovny WebDriverManager. Pokud pokus o inicializaci driveru selže (buď se může jednat o dříve zmíněnou chybu v knihovně WebDriverManager, nebo třeba nemáte nainstalovaný zmíněný prohlížeč, případně je nainstalovaný do nestandardního adresáře), program na to sice upozorní chybovou zprávou a ukončí svoji činnost, ale nic jiného s tím udělat nedokáže. Aby byl tento problém alespoň částečně eliminován, automaticky se ke kódu v době sestavení přibaluje nejnovější dostupná verze knihovny WebDriverManager, přičemž se očekává, že problémy předchozí verze byly s velkou pravděpodobností opraveny. Pokud problém přetrvává, je nutné navštívit Github stránku projektu `https://github.com/bonigarcia/webdrivermanager` a podívat se do záložky Issues, jak již bylo

zmíněno dříve. S velkou pravděpodobností stejnému problému s knihovnou čelí více uživatelů a problém již bude nahlášený i s případným řešením.

4.1.14 Chyba na webu společnosti Fortuna Game

Během procházení výsledků loterií provozovaných společností Fortuna Game se občas přihodí, že jsou sice zobrazena data od stanoveného okamžiku, ale dále také nějaká novější, která již program zapsal. Tento problém jsem zaznamenal pouze při používání webového rozhraní, API zřejmě funguje spolehlivě. Nicméně, pomocí odfiltrování již jednou zaznamenaných dat, které je popsané v podkapitole 4.1.4, si s touto chybou dokáže program poradit.

4.1.15 Některé další problémy

Během implementace bylo nezbytné rozlišit dva typy kliknutí: kliknutí na klasický prvek a kliknutí na odkaz. Vzhledem k tomu, že kliknutí na odkaz vede ke změně URL adresy a změně DOM reprezentace stránky, s čímž si následně Selenium nedokáže samo poradit, je nutné při každém kliknutí detekovat, zda se nejedná o element *<a>*, který v HTML reprezentuje odkaz, a pokud ano, najít jeho atribut *href*, který tuto URL adresu obsahuje a odkaz načíst. Poté se však stávalo, že ačkoliv to nebývá dobrým zvykem, na stránce se objevovaly *<a>* elementy, jejichž cílem nebyl přechod na jinou URL adresu. Aby bylo možné takovým případům předejít, můžeme do hodnoty vlastnosti, která určuje element, na který bude kliknuto, přidat řetězec *localURL*. Tím bude zaručeno, že ačkoliv se jedná o element *<a>*, nebude načítána jiná stránka, ale má být vykonáno jen klasické kliknutí. Kód pak vypadá následovně.

```
if (getElements(cssSelector).get(0).getTagName().equals("a")
        && !action.split(" ")[1].equals("localURL")) {
    href = getElements(cssSelector).get(0)
        .getAttribute("href");
    driver.get(href);
    new WebDriverWait(driver, Const.PAGE_LOAD_TIMEOUT)
        .until(ExpectedConditions.urlToBe(href));
    break; //cely kod je uvnitr switch(actionName)
}
((JavascriptExecutor) driver).executeScript(String.format(
        "document.querySelector('%s').click();",
        cssSelector));
```

Na zmíněném kódu je vidět, že se `getElements(cssSelector).get(0)` volá zdánlivě redundantně. Nicméně, právě tento způsob je ideální pro zamezení výjimky `StaleElementReferenceException`, neboť pokud bychom si ukládali prvek a poté k němu přistupovali, mohlo by to způsobovat popsané komplikace.

Program dále umožňuje například simulaci pohybu kurzoru, kliknutí na místě, kde se nachází kurzor nebo například přesun na určitý prvek, který se momentálně nemusí nacházet na obrazovce. Tyto funkcionality bylo nezbytné implementovat, aby crawler dokázal fungovat na některých dynamicky generovaných stránkách.

Dále bylo nutné řešit automatické snižování URL parametrů. Například u loterie Kasička potřebujeme manipulovat se dvěma parametry, *year* a *week*. Můžeme tedy obecně z n parametrů vytvořit n-tici, v našem případě (*week*, *year*), kde pořadí v n-tici určuje, v jakém pořadí mají být parametry snižovány (v našem případě nejdřív budeme postupně snižovat parametr *week* dokud se nedostaneme na minimální hodnotu, tedy na první týden v roce a pak snížíme rok a týden nastavíme na maximum). Každý prvek této n-tice pak může mít minimální hodnotu, maximální hodnotu a hodnotu, při které má program skončit. Program ukončí sběr dat pro určitou loterii, pokud všechny prvky n-tice nabývají právě této hodnoty. Implementace popsané funkcionality, která umožňuje práci s URL parametry, je také součástí třídy `HTMLManipulator`.

Ačkoliv jsou výsledky losování pro definované loterie dostupné v tabulkách, aby byl program co nejobecnější, je dobré definovat alternativní způsob, pomocí kterého by bylo možné sbírat i data, která jsou zaznamenána jiným způsobem. Zatím byl popisován pouze případ, kdy je jako jeden záznam brána jedna řádka tabulky, přičemž jednotlivé buňky tabulky (<*td*> elementy) reprezentují vždy jednu informaci. Program nicméně nabízí možnost, jak definovat sběr dat podobným způsobem i pro jiné prvky než tabulky pomocí vlastnosti *targetRowItems*. Následující dva řádky jsou pak ekvivalentní:

```
"targetTable": "#table"
"targetRowItems": "#table tr[] td"
```

Hranaté závorky za elementem *tr* naznačují, že procházíme jednotlivé řádky. Data jsou dostupná v jednotlivých *td* elementech. Vzhledem k tomu, že mohou být použity libovolné CSS selektory, nikoliv jen *tr* a *td*, je možné sbírat data z téměř libovolně strukturované stránky. Implementace této funkcionality je poměrně jednoduchá. Zásadní je provedení následující operace, pak již může sběr dat probíhat obdobným způsobem jako sběr dat z tabulky.

```
String regex = properties.getTargetRowItems()
    .replace("[]", ":nth-child(%d)");
```

4.1.16 Obecnost řešení

Ačkoliv hlavním cílem byl sběr dat z určených loterií, snažil jsem se o co nejobecnější implementaci, aby bylo možné případně sběr rozšířit o další loterie. Program je možné využít pro sběr dat z libovolných stránek. Omezením je pouze možnost zápisu konfigurace, která pro některé stránky nemusí být dostačující (je nicméně

vymyšlena tak, aby umožňovala sběr na co možná největším počtu stránek). Navíc je možné vlastní potřebě přizpůsobit verze knihoven Selenium i WebDriverManager, které bude program používat.

4.2 Program přistupující k API

Jako programovací jazyk, ve kterém byl tento program napsán, byl zvolen Python, zejména kvůli jednoduché manipulaci s daty v JSON formátu, která jsou vrácena jako odpověď serveru na volání API. Navíc má Python jednoduchou syntaxi a vzhledem k tomu, že je to interpretovaný jazyk, je výsledný kód snadno spustitelný na různých zařízeních.

Stejně jako u web crawleru, i tento program se bude řídit konfigurací uživatele uvedenou v souboru *config.json* s možností dodefinovat konkrétní detaily jednotlivých loterií v samostatném konfiguračním souboru.

Vzhledem k vlastnostem jazyka Python navíc není použit žádný nástroj pro sestavení projektu, jelikož není třeba kompilovat žádné skripty a jediná použitá závislost, tedy knihovna *requests* je definována v souboru *requirements.txt*, pomocí kterého je možné tuto závislost jednoduše nainstalovat.

4.2.1 Načítání konfigurace

O celé načtení konfiguračních souborů se stará výhradně skript *config_manager.py*. Pomocí importovaného balíčku *json* je načtení dat ve formátu JSON mnohem jednodušší než u web crawleru psaném v Javě. Po načtení konfigurace probíhá kontrola načtených hodnot. Jednou z nejdůležitějších funkcí z tohoto skriptu je `load_global_config`, která načte informace z globální konfigurace a následně se postará o načtení všech lokálních konfiguračních souborů, které byly v souboru *config.json* definovány. Vlastnosti globální konfigurace jsou načítány do slovníku `global_property` a většinu vlastností je možné definovat v globální konfiguraci s tím, že lokální konfigurace může hodnotu vlastnosti použít pro sběr dat, případně může hodnotu přepsat. K načtení konfigurace jedné loterie pak slouží funkce `load_local_config`. Mezi důležité funkce patří například ještě `check_local_config`, která kontroluje hodnoty nastavené v lokální konfiguraci. Třída `Params` definovaná ve skriptu *config_manager.py* slouží pouze jako přepravka, do které jsou ukládány parametry volání API načtené z konfigurace.

4.2.2 Přístup k API

Přístup k API řeší skript *api_accessor.py*. Přístup je možný realizovat pomocí HTTP metody GET nebo POST. Pro každou loterii je v hlavním skriptu *main.py* vytvořen objekt třídy `APIAccessor` a je nad ním zavolána metoda `run`, která spustí sběr dat.

Skript *main.py* může na základě konfigurace spustit více vláken najednou, které si rozdělí práci a budou k API přistupovat ve stejný čas. Aby byl synchronizován výpis informací o sběru na obrazovku (pokud jsou data zapisována do souborů), každé vlákno, které chce vypisovat na standardní výstup, musí zamknout společný zámek pro vlákna (v proměnné `lock`). Pro výpis aktuálního stavu všech vláken je nutné přistoupit k proměnné `thread_jobs`, kam každé vlákno ukládá počet požadavků, které úspěšně odeslalo. Během přístupu k proměnné `thread_jobs` je samozřejmě také nezbytné, aby byl uzamčený sdílený zámek. Každé vlákno se pak stará o to, aby při svém ukončení z proměnné `thread_jobs` informace o svém stavu odstranilo. Informace o stavu sběru jednotlivých vláken jsou v případě zápisu dat do souboru a zároveň vícevláknového sběru vypisovány tímto způsobem:

```
lock.acquire()
s = "\r\t"
q = 0
thread_jobs[config['name']] = accessed
for key in thread_jobs.keys():
    s += f"{key}: {thread_jobs[key]}{', ' \
        if q < len(thread_jobs.keys()) - 1 else ''}"
    q += 1
print(s, end="")
lock.release()
```

\r je speciální znak (carriage return) a značí, že se má začít zapisovat znovu od začátku řádky. V důsledku tedy bude starý výstup ukazující již nerelevantní data přemazán novým. Proměnná *accessed* uchovává počet přístupů k API, které konkrétní vlákno uskutečnilo.

Samotná metoda `run` pak v cyklu volá metodu `access_api`, která se stará pouze o přístup k API, ale `run` kromě volání `access_api` zařizuje například také změnu hodnot proměnných nebo různé výpisy na obrazovku. Metoda `access_api` pošle na server požadavek s uživatelem definovanými parametry a zavolá nad objektem třídy `OutputManager` metodu `write`, pomocí které budou vrácená data zapsána. Další metoda třídy `APIAccessor`, `replace_variables`, je pomocná metoda sloužící k nahrazení proměnných v řetězci za jejich hodnotu.

4.2.3 Výpis získaných dat

O výpis získaných dat do souboru nebo na konzoli se stará skript *output_manager.py*. Ten definuje třídu `OutputManager`, a každá loterie tak řeší pomocí objektu této třídy zapisování dat zvlášť. Jelikož se může název souboru, do kterého chceme zapisovat, měnit (např. pokud má název souboru odpovídat datu, ve kterém proběhlo losování), program dynamicky v případě potřeby vytvoří nový soubor pro zapisování a pokračuje v zápisu. O to, aby byl případně změněn soubor pro zápis dat, se stará me-

toda `check_changes`. Nejdříve je nezbytné v konstruktoru třídy `OutputManager` nastavit hodnotu atributu `flush_on_day_change` na *True*, pokud název souboru závisí na datu, v opačném případě na *False*. Pak část kódu, která řeší vytvoření nového zapisovacího souboru, vypadá následovně.

```
if date_change and self.flush_on_day_change:
    self.file.close()
    self.construct(api_accessor.APIAccessor() \
      .replace_variables(self.abstract_name, i, date, time) \
      .replace("${name}", config["name"]), self.append_mode)
```

Hodnotu proměnné *date_change* zjistí metoda porovnáním používaného data a data, které vrátilo API. Metoda `construct` pak slouží k zapisování do nového souboru bez nutnosti znovu vytvářet nový objekt třídy `OutputManager`. O samotný zápis sesbíraných dat se stará metoda `write`. Data jsou zapisována ve formátu CSV, jednotlivé záznamy jsou tedy odděleny čárkami a každý záznam je na vlastní řádce. To, jak vypadá jeden záznam a jaká data budou vypsána, nicméně stanovuje uživatel v konfiguraci a tento záznam z dat sestavuje funkce `get_data`.

4.2.4 Automatické zastavení programu

Pokud program zjistí, že odpověď od serveru sice přišla, ale nebyla vrácena žádná data, tento stav typicky indikuje, že již sesbíral všechna dostupná data a další nejsou k dispozici, takže program se sběrem pro určitou loterii skončí. Kromě toho je možné v konfiguraci nastavit, kolik maximálně má proběhnout přístupů k API. Pokud program pošle tento počet požadavků pro jednu loterii, také automaticky skončí se sběrem dat. Dále je možné nastavit inkrementální sběr dat, tedy aby se sběr zastavil, pokud jsme se dostali se sběrem dat na den losování, který již byl zaznamenán. Tato funkcionalita je možná, jelikož program zaznamenává datum začátku sběru dat pro každou loterii do souboru *.program_data.txt*. O zaznamenání těchto metadat se stará funkce `store_metadata` ze skriptu *output_manager.py* a o čtení funkce `read_metadata`. Ta vrací slovník, kde klíčem jsou názvy loterií a hodnotou je vždy datum, kterým začínal minulý sběr dat.

V konfiguraci je nicméně možné tento způsob zastavení programu zakázat, případně sbírat data ještě o jeden den déle (viz uživatelská příručka v příloze B).

4.2.5 Proměnné a parametry

Aby bylo možné automaticky sbírat data například v loterii Kameny, kde je pro každé datum dostupných několik stránek záznamů, je nutné nějakým způsobem navázat parametry volání API na nějakou proměnlivou hodnotu. Program definuje 3 proměnné, na které je možné volání navázat: celočíselná proměnná *i*, která začíná

na hodnotě 1 a je zvyšována, proměnná pro čas (*time*), kterou je možné snižovat automaticky o 1 sekundu a proměnná pro datum (*date*), kterou je možné automaticky snižovat o 1 den. Do konfigurace je možné přidat více parametrů navázaných na tyto proměnné, při každém přístupu k API pak bude automaticky snížena/zvýšena právě jedna proměnná odpovídající určitému parametru. Postupně se program snaží najít sekvenčně právě tento jeden parametr, který by mohl snížit. Iteruje tedy od prvního definovaného parametru, pokud již ale dosáhl své maximální/minimální hodnoty, nemůže jej zvýšit/snížit, takže se musí posunout na další parametr. Při změně parametru pak může uživatel nastavit, že má být nějaký jiný parametr resetován. Pro loterii Kameny bychom tedy například zvolili konfiguraci se dvěma parametry: jeden bude navázán na proměnnou *i* a bude reprezentovat *page* parametr API, druhý bude navázaný na datum, bude reprezentovat parametr *day* a při jeho snížení bude resetován parametr *page*. Tímto způsobem bude možné projít veškeré záznamy, je jen nutné specifikovat maximální hodnotu parametru *page*. Tu zjistíme během volání API, které vrací mimo jiné JSON vlastnost *totalPages* (viz podkapitola 3.1.2). Konfigurace pak uživateli umožňuje jako maximální hodnotu zvolit i takto získaná data z volání API.

Kromě parametrů navázaných na proměnnou je možné využít parametry navázané na data, typicky získaná z minulého volání API. To můžeme využít například u loterie Rychlé kačky, kde je jako parametr volání API předáváno datum a čas, ke kterému chceme získat data. Díky navázání tohoto parametru na data získaná z API je možné získat čas posledního vráceného slosování (je vrácen v datech), odečíst od něj jednu sekundu a takto získaný nový čas využít k novému volání API.

O korektní práci s parametry se stará přímo metoda `run` ve třídě `APIAccessor`, která řídí celý sběr dat.

4.2.6 Sběr dat v reálném čase

Program umožňuje sbírat data i v reálném čase, přičemž mezi jednotlivými spuštěními sběru dat pokaždé spí po specifikovanou dobu. O sběr dat v reálném čase se stará skript *realtime_main.py*. Pokud tedy například existuje loterie, ve které probíhá nové losování každou minutu a každé volání API vrátí 10 posledních záznamů, je možné spustit sběr v reálném čase s intervalem 10 minut, díky čemuž se každých 10 minut stáhnou veškerá nová data. Pro spuštění programu stačí namísto spouštění *main.py* jednoduše spustit skript *realtime_main.py*.

4.2.7 Některé další funkce

Sběr dat je možné provádět také na více vláknech najednou, konfigurace pro jednotlivé loterie se v takovém případě mezi tato vlákna rozdělí a každé vlákno obsluhuje

jinou loterii. Dále je například možné stanovit, jaký datum má být při spuštění programu v proměnné *date*, díky čemuž může sběr u některých loterií začít u dřívějších dat, nikoliv těch nejnovějších. Program obsahuje další celou řadu možností konfigurace, které jsou popsány v uživatelské příručce.

4.2.8 Experimenty s proxy

Při přístupu k API je často používaný přístup pomocí proxy IP adres (IP adres třetí strany, které máme propůjčené pro komunikaci). Díky tomu, že se server domnívá, že naše IP adresa je adresa proxy serveru, v případě blokace není zablokována naše reálná IP adresa, ale adresa proxy serveru, kterou můžeme v takovém případě změnit. Zkoušel jsem tedy přistupovat k API pomocí proxy IP adres, které podporují protokol HTTPS, používaný zmíněnými API. Nicméně, komunikace přes proxy IP adresu každý přístup k API opozdila průměrně asi o 1 sekundu (zkoušel jsem různé proxy IP adresy) a navíc byla komunikace po chvíli ukončena, zřejmě z důvodu přetížení proxy serveru. Z toho důvodu sběr dat nakonec neběží přes proxy, ale má celkem vysokou dobu spánku mezi jednotlivými sběry dat (1 až 2 sekundy), aby nebyla IP adresa, ze které sběr dat probíhá, zablokována.

4.2.9 Problém na straně serveru

API společnosti Sazka, pro kterou je primárně spolu s API společnosti Fortuna Game program vytvářen, není stoprocentně spolehlivé. Jednou za čas se stane, že API reaguje na požadavek odpovědí v následujícím tvaru, která značí, že na serveru došlo k potížím (nastane tzv. *Internal server error*).

```
{
'type': 'https://tools.ietf.org/html/rfc7231#section-6.6.1',
'title': 'An error occurred while processing your request.',
'status': 500,
'detail': 'The request was canceled due to the configured
           HttpClient.Timeout of 5 seconds elapsing.',
'traceId': '00-...-00'
}
```

Program detekuje, že nebyly vrácené požadované vlastnosti, ale to může znamenat buďto právě zmíněný problém, nebo mohou již být všechna data sesbírána. Z toho důvodu vypíše na konzoli také odpověď vrácenou serverem. Nechá tedy na uživateli, zda spustí program znovu a pokusí se získat další data, nebo rozhodne, že již jsou všechna data sesbírána.

4.2.10 Obecnost řešení

Stejně jako web crawler, i tento program byl vytvářen tak, aby bylo případně možné sběr dat rozšířit pro další API. S pomocí programu je tedy možné přistoupit k libovolnému API přes metodu POST nebo GET, pokud jsou k tomu dostačující možnosti konfigurace a pokud API vrací data ve formátu JSON, což je ale dnes zvykem.

4.3 Srovnání obou řešení

Vývoj web crawleru přinášel spoustu výzev. Zatímco pomocí API dostaneme jako odpověď ze serveru strukturovaná data, s využitím web scrapingu, který bude dostatečně obecný pro všelijaké další loterie, které je možné přidat, bylo nutné pomocí konfigurace definovat, jak si má crawler počínat. Často se přitom jedná o komplikovanější akce, například bylo nutné zařídit, aby byla další data ze stejného dne načítána do té doby, dokud bude na stránce přítomné tlačítko načítající další data, pokud tlačítko není přítomné, musíme z rozbalovacího menu zvolit další den. Kromě toho se během vývoje stalo, že společnost Sazka změnila zdrojový kód stránek a bylo nutné celou konfiguraci přepisovat. Web crawler navíc používá knihovnu WebDriverManager, která může způsobovat potíže (viz podkapitola 2.2.1).

Name	Status	Type	Initiator	Size	Time
sapi.aspx?cm=GGL&CSI=12...	200	xhr	556.js?v=202...	56.1 kB	305 ms
tVb0MQESLQFQmWHMxuX...	(failed)	script	VM1454:2	0 B	303 ms
async	(failed)	script	VM1564:2	0 B	266 ms
rc.js	(failed)	script	VM1682:4	0 B	221 ms
uwt.js	(failed)	script	VM1734:1	0 B	211 ms
0?ti=343067183&Ver=2&m...	204	text/pl...	bat.js:1	122 B	157 ms
landing?gcs=G111&gcd=1...	200	ping	landing	65 B	154 ms
kameny	200	xhr	api.ts:14	1.2 kB	152 ms
sazky-a-vysledky	200	docum...	Other	15.2 kB	151 ms
countdown?gameNames=v...	200	xhr	lottery-count...	473 B	148 ms
3760.0c31b574d067ead0.js	200	script	load script:41	(disk c...	142 ms
6974.3db7222d9a8179a3.js	200	script	load script:41	(disk c...	137 ms
618.0f2ab144b520c6e7.js	200	script	load script:41	(disk c...	137 ms
4022.bbc0dd22cb52ebaf.js	200	script	load script:41	(disk c...	137 ms
1020.e7fbd923e0b07b41.js	200	script	load script:41	(disk c...	136 ms

116 requests | 96.2 kB transferred | 7.1 MB resources | Finish: 2.43 s | DOMContentLoa

Obrázek 4.1: Načítání obsahu loterie Kameny

Program pracující s API tyto nedostatky vyřešil. Navíc sběr pomocí API probíhá mnohem rychleji než s použitím Selenia. Není totiž nutné načítat celý kód stránky, všechny skripty a další soubory a poté stránku vykreslovat. Navíc, jak bylo uvedeno v podkapitole 3.1, všechny loterie, ze kterých sbíráme data, kromě Kasičky, stejně nakonec volají API. To znamená, že vykreslování stránky a vykonávání spoustu dalšího kódu, načítání obrázků apod. je vlastně úplně zbytečná zátěž navíc, jelikož nám jednoduše stačí zavolat API. Na obrázku 4.1 je pro ukázku vidět, jaký čas zabírá načítání jakého obsahu na stránce s výsledky loterie Kameny (seřazeno sestupně podle nejdelší doby načítání).

Jedná se o přenosy těsně po načtení stránky, a jak je vidět, v tomto případě bylo načtení dokončeno za 2,43 sekundy a bylo přeneseno 7,1 MB zdrojů pomocí 116 požadavků. Samotné API zatím ani nebylo voláno, jelikož to se děje po kliknutí na tlačítko *MINULÉ VÝSLEDKY*. Je tedy vidět, že načítání všech těchto zdrojů je zbytečnou zátěží, která web crawler ve výsledku zpomaluje. Program přistupující k API navíc umožňuje sběr dat na více vláknech najednou, zatímco program v Javě sbírá data pouze sekvenčně.

Dále bylo experimentálně zjištěno, že s napojením na API dokážeme sesbírat i starší dostupná historická data, zatímco s využitím webového rozhraní se dostaneme například u loterií Kameny a Keno pouze 365 dní zpět. Je tedy možné konstatovat, že toto řešení je mnohem vhodnější.

Aby bylo možné lépe porovnat efektivitu obou programů, spustil jsem je postupně oba na dobu 2 minut, cílem bylo sesbírat co nejvíce dat z loterie Keno. Veškeré časy strávené čekáním byly úplně stejné, jako jsou nastavené v konfiguračních souborech v příloze bakalářské práce. Web crawler během této doby zapsal 296 záznamů, program psaný v Pythonu 574, tedy skoro dvojnásobek. Navíc nebyla použita možnost vícevláknového sběru, která by pro 8 loterií znamenala přibližně až šestnáctinásobné zefektivnění využitého času.

Veškeré měření probíhalo na notebooku HP 250 G8 s 8 GB paměti RAM a procesorem Intel i3-1115G4 @ 3.00GHz. Rychlost stahování i nahrávání dat přes síť se pohybovala během připojení k Wi-Fi mezi 50 a 60 Mb/s.

Web crawler byl implementačně mnohem náročnější a jak již bylo zmíněno, je náchylný na celou řadu problémů. Navíc konfigurace je mnohem složitější a uživatel je nucen zkoumat zdrojový kód stránky a měnit konfiguraci pokaždé, když je změněn způsob reprezentace dat na stránce.

4.4 Testování

Vzhledem k tomu, že oba programy manipulují s daty získanými z internetu a jejich výstup je jednoduše možné zkontrolovat vůči očekávaným datům, bylo pro oba programy zvoleno ruční testování. Díky testování bylo odhaleno větší množství chyb,

které bylo nutné opravit. Například web crawler byl nejdříve napsán rekurzivně, ale při ručním testování byl tento problém odhalen, když program spadl kvůli přetečení zásobníku. Dále bylo zjištěno během vývoje právě pomocí ručního testování více chyb ve výstupu, které byly následně odstraněny.

Bylo zvažováno další testování web crawleru jednotkovými testy, jelikož web crawler je implementačně mnohem náročnější a náchylnější na případné chyby než program přistupující k API, ale vzhledem k tomu, že by bylo nutné simulovat napojení na ovladač prohlížeče, načítání stránky a mnoho dalších operací pracující s webovou stránkou, nakonec k tomuto kroku nedošlo.

Jak bylo zmíněno, během testování bylo objeveno větší množství chyb a oba programy by se nyní měly chovat dle očekávání.

Závěr 5

Bakalářská práce seznámila čtenáře s některými technikami získávání dat z webových stránek. Byly zmíněny možnosti existujících řešení pro web crawlery (například s využitím Selenia, knihovny Beautiful Soup nebo softwaru Octoparse). Bakalářská práce dále popsala vytvořený web crawler, který plně splňuje požadavky zadání. Byla sesbírána všechna dostupná historická data k okamžiku sběru, navíc je umožněn web scraping v reálném čase.

V teoretické části byly srovnány dvě metody přístupu k datům na webové stránce: pomocí web scrapingu a pomocí přístupu k API. Z tohoto srovnání jednoznačně vyplynulo, že přístup k API by měl být vždy upřednostněn před web scrapingem, pokud je možný. Přestože API existuje, ne vždy je k němu však možné získat popis, což byl i tento případ. Dále bylo v teoretické části ukázáno, že pro konkrétní loterie doopravdy API existuje, navíc poskytuje ještě víc dat, než webové rozhraní. Z toho důvodu je součástí bakalářské práce ještě jeden program, který získává data právě pomocí API. Vzhledem k tomu, že pomocí API je navíc možné sesbírat i historická data, která nejsou dostupná pomocí webového rozhraní, data sesbíraná Seleniem jsou podmnožinou dat získaných pomocí API (API poskytuje nějaké informace navíc a nějaká data navíc) a pro účely předmětu Základy náhodných procesů je po konzultaci nejlepším řešením pojmenovávat soubory podle data, ze kterého záznamy pocházejí, což umožňuje právě program napsaný v Pythonu, jako hlavní způsob sběru dat se jeví právě použití tohoto programu. Sesbíraná historická data (až do 16. 4. 2024) jsou k dispozici ve složce *Vysledky* v příloze bakalářské práce. Aby bylo názorně demonstrováno, že je možné rozšířit sběr dat pro další loterie, jsou navíc ještě sbírána data z loterií Rychlá 6 a Všechno nebo nic. To navíc nezpůsobí zásadní časovou zátěž, jelikož sběr pro každou loterii v nastavení konfigurace, která je součástí bakalářské práce, běží na vlastním vlákně.

Přes veškeré problémy se podařilo vytvořit oba programy, přičemž oba poskytují co nejobecnější funkcionalitu pro případné rozšíření sběru dat o další loterie.

Bakalářská práce následně oba přístupy srovnala (viz podkapitola 4.3) s tím výsledkem, že přístup k API je doopravdy mnohem lepším řešením, než web scraping, jak bylo zmíněno již na začátku textu.

Uživatelská příručka web crawleru

A

Cílem této bakalářské práce je vytvořit program, který bude z webu stahovat informace o výsledcích loterií. K tomu je používána Java a knihovna Selenium. Pro sestavení spustitelného souboru je používán Gradle. Je možné si volitelně přizpůsobit konfiguraci programu, která nabízí poměrně dost možností.

A.1 Struktura projektu

- src - zdrojové soubory
  - config - třídy pracující s uživatelovou konfigurací
  - logger - logování
  - settings - konstanty
  - web - práce se Seleniem a webovým prohlížečem
- build.gradle - obsahuje zejména seznam všech knihoven, které budou k programu přibaleny a jejich verze
- config.json - globální konfigurační soubor

A.2 Příprava

Je nutné mít nainstalovanou Javu, ideálně verze 19 nebo novější (verze 19 je ke stažení zde: `https://www.oracle.com/java/technologies/javase/jdk19-archive-downloads.html`). Dále zajistěte, že máte nastavenou proměnnou JAVA_HOME tak, aby odkazovala na Váš adresář s Javou, např. C:\Program Files\Java\jdk-19. Dále je nutné přidat do proměnné prostředí Path informaci o Javě třeba tímto způsobem: %JAVA_HOME%\bin. Jelikož je používán WebDriverManager, není nutné stahovat žádný driver prohlížeče, ani ho neustále aktualizovat, tato knihovna se o to stará automaticky. O stažení Selenia verze 4.2.1. a nejnovější verze WebDriverManageru se zase stará Gradle. V případě problémů je možné změnit verze těchto

knihoven, např. pokud by s příchodem nové verze knihovny WebDriverManager přestala být nějaká využívaná funkcionalita podporována, můžete nastavit stabilnější verzi. Aby bylo možné používat Gradle, je nutné ho nejdříve stáhnout ze stránek `https://gradle.org/` a provést potřebné kroky, které jsou uvedeny na adrese `https://gradle.org/install/`.

Zadáním příkazu `gradle build` se vygeneruje spustitelný JAR soubor v adresáři *build/libs*. Přesuňte tento JAR soubor do hlavního adresáře projektu, alternativně do libovolné složky, v tom případě ale nakopírujte do této složky také všechny konfigurační soubory. V terminálu zadejte příkaz `java -jar <název programu>.jar`, čímž se program spustí.

A.3 Konfigurace

A.3.1 Soubor *config.json*

V souboru *config.json* je uvedena konfigurace celé aplikace, ale obsahuje také názvy souborů, ve kterých jsou konfigurace jednotlivých loterií. Následující konfigurace je pouze ilustrativní, jelikož použití některých následujících vlastností vylučuje použití jiných (viz dále).

```
{
  "sysout": "System.out",
  "syserr": "System.err",
  "sites":
    ["kameny.json",
    "kasicka.json"],
  "continueEval": true,
  "skipHeaderRows": false,
  "skipAdditionalRows": "1",
  "appendOut": false,
  "appendErr": true,
  "pattern": "{a, b}",
  "browser": "chrome",
  "maxData": "500"
}
```

sysout a *syserr* - Nastavují, kam bude vypsán výstup. Obě vlastnosti mohou nabývat tyto hodnoty: *System.out* (standardní výstup), *System.err* (standardní chybový výstup), *<název souboru>.<přípona>* (pro zápis do souboru), nebo *none* (pokud chcete výstup ignorovat). Důležité je zmínit, že pokud nastane chyba před nebo během čtení konfiguračních souborů, nebude toto nastavení uplatněno a bude použito výchozí nastavení, což je *"sysout": "System.out"* a *"syserr": "System.err"*. Tyto dvě vlastnosti **není nutné** uvádět, může být použito výchozí nastavení. V hodnotě těchto vlastností je možné používat proměnné (viz dále). Na *sysout* se vypisují získaná data.

sites - Pole s konfiguračními soubory pro jednotlivé stránky s loteriemi. Všechny tyto soubory musí být ve formátu JSON. Vlastnost *sites* **je nutné** definovat v souboru *config.json*, jinak nebude program fungovat.

continueEval - Pokud je nastaveno na *true* a nějaké stažení dat se nepodaří vykonat, bude se pokračovat další stránkou. Ve výchozím nastavení je však nastaveno na *false*, a proto **tuto vlastnost nemusíte specifikovat**.

skipHeaderRows - Ve výchozím nastavení je nastaveno na *true*, což znamená, že jsou přeskočeny všechny *<th>* elementy specifikované tabulky. Ačkoliv je možné získávat data i jiným způsobem než z tabulky, v takovém případě by *skipHeaderRows* nemělo žádný efekt. Pokud se rozhodnete do dat přidávat i obsah *<th>* elementů, stačí tedy tuto vlastnost nastavit na *false*. Tato vlastnost **nemá povinný výskyt**. Pokud se však rozhodnete nastavit *skipHeaderRows* na *false*, není možné zároveň používat *skipAdditionalRows*.

skipAdditionalRows - Specifikuje počet řádek, které budou od začátku vynechány. To může být užitečné například, pokud data nejsou sbírána z tabulky, ale z jiné struktury, nebo pokud hlavička tabulky nepoužívá tag *<th>*, ale *<td>*. Tato vlastnost **není povinná**, nicméně pokud zvolíte nenulový počet, nesmíte nastavit vlastnost *skipHeaderRows* na *false*. Hodnota musí být určena ve formě řetězce.

appendOut a *appendErr* - Určuje, zda se při standardním/chybovém zápisu do souboru má použít režim *append* (připsání na konec souboru). Smysl má pouze tehdy, pokud je v odpovídající vlastnosti *sysout* nebo *syserr* nastaven název výstupního souboru, nikoliv konzolový výstup nebo *none*. Definice těchto vlastností **je volitelná**, ve výchozím nastavení je použito *true*.

pattern - Udává, jakým způsobem bude zapsán výstup do souboru. V hodnotě vlastnosti se musí právě jednou vyskytnout symbol *a*, stejně jako symbol *b*. Výpis se bude konstruovat takto: Každý záznam bude mít svoji vlastní řádku. Na začátku této řádky budou všechny znaky, které jsou ve vlastnosti *pattern* před symbolem *a*. Poté bude následovat první člen záznamu, a další členy záznamu budou odděleny znaky mezi symbolem *a* a symbolem *b*. Po posledním znaku budou vypsány všechny symboly, které jsou ve vlastnosti *pattern* za *b*. Tuto vlastnost **je nutné specifikovat buď v globální nebo lokální konfiguraci**.

browser - Určuje, jaký prohlížeč se bude používat ke stahování dat. Umožňuje hodnoty *chrome* a *edge*. Může se stát, že Vámi zmíněný prohlížeč nebude správně fungovat, WebDriverManager se snaží najít cestu k Vámi definovanému prohlížeči, ovšem v případě nestandardní instalace by se mu to nemuselo podařit [8]. V tom případě můžete zkusit buď druhý prohlížeč, nebo prohlížeč reinstalovat. Tato vlastnost **musí být zmíněna**.

maxData - Maximální počet získaných řádek. Tuto vlastnost *není nutné uvádět*, v takovém případě je maximální počet získaných řádek 2 147 483 647, což odpovídá konstantě *Integer.MAX_VALUE*. Hodnotu vlastnosti je nutné specifikovat ve tvaru

řetězce.

A.3.2 Konfigurační soubory jednotlivých stránek

Tyto soubory musí být zmíněny v poli *sites* v souboru *config.json*. Tady je přehled všech možných vlastností, které je možné v těchto souborech definovat (opět pouze ilustrativní):

```
{
  "name": "sazka_1",
  "url": "https://www.sazka.cz/loterie/kameny/sazky-
  a-vysledky",
  "continueEval": true,
  "beforeActions": [
    "click #button-history-results a",
    "wait 3000"
  ],
  "targetTable": ".prize-table__table.kameny",
  "targetRowItems": ".prize-table__table.kameny tr[] td",
  "skipHeaderRows": false,
  "skipAdditionalRows": "1",
  "sysout": "output/${name}/${iii}.csv",
  "syserr": "System.err",
  "appendOut": false,
  "appendErr": true,
  "pattern": "a b",
  "nextDataActions": ["click .ds-icon__thin", "wait 1000"],
  "maxData": "2000",
  "incremental": false,
  "dataOrder": "DESC",
  "rowData": [
    "content #lotteryHistoryResultsDropdownSelectBoxItText",
    "tableContent"
  ]
}
```

name - Název přidělený dané loterii. Ten může být součástí chybových výpisů, ale je také dostupný v proměnné *${name}* (viz dále). Tuto vlastnost **je nutné uvést**.

url - URL adresa loterie, kterou **musíte v konfiguračním souboru uvést**.

continueEval - Obdobně jako vlastnost v *config.json* definuje, zda při nepodařeném stažení dat z konkrétní loterie program spadne (*false*), nebo bude pokračovat (*true*). Tuto vlastnost **není nutné specifikovat**, přejímá se hodnota z globální konfigurace.

beforeActions - Popisuje akce, které je nutné vykonat před samotným sběrem dat. Všechny akce jsou popsány níže. Tato vlastnost **není povinná**, pokud ji nespecifikujete, bude se předpokládat, že nejsou nezbytné žádné akce.

nextDataActions - Pole akcí, které je nutné vykonat, když už v tabulce nejsou další data, aby se objevila. K dalším datům bude přistoupeno pomocí stejného selektoru jako k původním. Umožňuje použití všech akcí z *beforeActions* (kompletní seznam povolených akcí je níže). Tuto vlastnost **není nutné zapsat**, v tom případě se nebudou vykonávat žádné akce.

targetTable - Specifikuje tabulku, ze které budou sbírána data pomocí CSS selektoru. Použití *targetTable* zároveň znemožňuje současné použití vlastnosti *targetRowsItems*, ale **jedna z těchto vlastností musí být definována**.

targetRowItems - Uvádí jiný způsob získání dat než z tabulky. Hodnota je reprezentována CSS selektorem, ve kterém jsou přidány hranaté závorky za element odpovídající řádku tabulky. Pokud je použita konfigurace *targetRowItems* stejná jako na ukázce výše, je ekvivalentní s použitou *targetTable*. Použití této vlastnosti znemožňuje použít *targetTable*. V souboru **musí být definována buď tato vlastnost, nebo *targetTable***. Zde je příklad použití vlastnosti *targetRowItems* s více hranatými závorkami:

```
"targetRowItems":".row[] .time, .row[] .date, .row[] .record"
```

V příkladě výše bude nahrazena každá závorka za *:nth-of-type(cislo_radku)*, pokud tedy sbíráme data ze třetího řádku, budou zaznamenány obsahy všech prvků, které odpovídají následujícímu CSS selektoru:

```
.row:nth-of-type(3) .time, .row:nth-of-type(3) .date,
    .row:nth-of-type(3) .record
```

skipHeaderRows - Ve výchozím nastavení je nastaveno na *true*, což znamená, že v tabulce jsou přeskočeny všechny *<th>* elementy, a jsou vybrány pouze *<td>*. Pokud chcete zahrnout i *<th>* elementy, nastavte tuto vlastnost na *false*, nicméně **nemá povinný výskyt** (případně se přebírá z globální konfigurace). Pokud je tato vlastnost nastavena na *true*, není možné zároveň nastavit *skipAdditionalRows* a pokud se rozhodnete nastavit *targetRowItems*, bude *skipHeaderRows* ignorováno.

skipAdditionalRows - Určuje počet řádek, který se stejně jako hlavička tabulky bude igorovat, data z těchto řádek tedy nebudou sesbírána. Tuto vlastnost **není nutné deklarovat**, v tom případě je převzata z globální konfigurace. Pokud však zvolíte nenulové číslo, nesmí být nastaveno *skipHeaderRows* na *false*.

sysout a *syserr* - Pomocí těchto vlastností je možné přepsat globální konfiguraci standardního a chybového výstupu pro konkrétní loterii. **Není nutné je nastavovat**, v takovém případě bude převzata jejich hodnota z globální konfigurace. Stejně jako v globální konfiguraci je možné používat proměnné.

appendOut a *appendErr* - Umožňuje přepsat pro konkrétní loterii globální konfiguraci pro režim *append* při zapisování do souboru. **Není nutné uvádět**.

pattern - Je možné přepsat globální konfiguraci a pro konkrétní loterii určit, jakým způsobem se budou získaná data zapisovat. **Je nutné uvést, pokud není**

nastaveno v globální konfiguraci.

maxData - Umožňuje přepsat globální konfiguraci a pro určitý soubor určit, jaké nejvyšší množství dat bude sbíráno. **Není nutné uvést**, pokud není zmíněno, jsou stažena všechna data.

incremental - Ve výchozím nastavení je nastaveno na *false*, což znamená, že po provedení *nextDataActions* se očekává, že stará data zmizela a objevila se nová. Pokud je však nastaveno na *true*, předpokládá se, že stará data zůstala na stránce a bude se zpracovávat jen nově se objevivší inkrement. **Není nutné uvést**.

dataOrder - Určuje pořadí, ve kterém budou zapisována právě získávaná data. Jednotlivé skupiny dat jsou odděleny pomocí *nextDataActions*, a každá tato skupina může být seřazena. Celkové řazení napříč skupinami není možné. Data mohou být zpracována buď sestupně (*DESC*, výchozí), nebo vzestupně (*ASC*). Tuto vlastnost **není nutné zmínit**.

rowData - Umožňuje připsat do každého řádku výstupního souboru další informace. Hodnota vlastnosti je reprezentována polem, ve kterém pořadí položek určuje, v jakém pořadí budou vypsány v řádku na výstup. Toto pole musí obsahovat položku *tableContent*, která bude ve výstupu nahrazena získanými údaji z tabulky. Dále je možné použít *content <CSS selektor>*, díky čemuž bude vypsán do výstupu obsah HTML prvku, který je určen zmíněným CSS selektorem. Tuto vlastnost tedy můžete využít například pro vypsání data. Kromě toho je možné zadat jakoukoliv jinou textovou hodnotu, a ta bude vypsána. Tuto vlastnost **není nutné použít**, ve výchozím nastavení se používá *"rowData": ["tableContent"]*, takže jsou vypsána pouze získaná data.

A.3.3 Proměnné v konfiguračních souborech

Následující proměnné jsou dostupné pouze v hodnotách těchto vlastností: *sysout* a *syserr*. Tady je kompletní seznam proměnných:

${name} - Obsahuje hodnotu vlastnosti *name* pro danou loterii. Proměnná může být použita i v globální konfiguraci, a získá hodnotu až během čtení konfigurací loterií.

${date[d]}* a *${date[dd]} - Aktuální den v měsíci právě v okamžiku spuštění programu. Pokud bude program spuštěn těsně před půlnocí 1. března a k proměnné se bude chtít dostat až 2. března, bude obsah *${date[d]}* 1 a *${date[dd]}* bude obsahovat 01 (má dvě místa). Nicméně, 10. března bude *${date[d]}* i *${date[dd]}* obsahovat 10.

${date[m]}* a *${date[mm]} - Podobně jako *${date[d(d)]}* obsahuje informaci o měsíci v době spuštění programu v jedno- nebo dvouciferném formátu.

${date[yy]}* a *${date[yyyy]} - Obsahuje rok v době spuštění. V roce 2024 by *${date[yy]}* nabývalo hodnoty 24 (zde se číslo osekává) a *${date[yyyy]}* hodnoty 2024.

${time[h]}* a *${time[hh]} - Hodina dne v okamžiku spuštění programu. Ve 3:41 bude *${time[h]}* obsahovat 3, zatímco *${time[hh]}* 03.

${time[m]}* a *${time[mm]} - Stejně jako *${time[h(h)]}* obsahuje informaci o minutě spuštění v jedno- nebo dvouciferném formátu.

${time[s]}* a *${time[ss]} - Sekunda, ve které byl program spuštěn.

${i}* - Pokud je poprvé spuštěna určitá loterie, bude pro ni proměnná *${i} nejdříve nastavena na hodnotu 1, a při každém dalším spuštění se zvýší o 1. Díky tomu je možné získávat logovací soubory s jménem jako *loterie_1*, *loterie_2*, *loterie_3*,... Každá loterie má své nezávislé *${i}*, takže pokud bude přidána nová loterie, začíná automaticky od 1.

${i...i}* - je možno použít až 9 znaků *i* v proměnné. Tento zápis značí, na kolik míst zarovnávat výsledné číslo, pokud bude tedy použito *${iiii}, výsledkem bude postupně 0000, 0001, ..., 9999, 10000, ...

V hodnotách vlastností, ve kterých je možné používat proměnné, je znemožněno použití symbolu €, který se programem interně používá pro zpracování proměnných.

A.3.4 Akce v *beforeActions* a *nextDataActions*

V *beforeActions* a *nextDataActions* je možné uvést pole akcí, které se mají vykonat před sběrem dat, respektive během načítání dalších dat. Tyto akce mají následující syntaxi: *<název akce> <parametry>*. Následuje seznam povolených akcí:

click <css selektor> - Kliknutí na element s CSS selektorem specifikovaným v *<css selektor>*. Např. *click #button-history-results a*. Pokud je prvek, na který ukazuje CSS selektor odkazem, provede přechod na stránku a počká, než bude načtena. Pokud ale smyslem odkazu není přejít na jinou stránku (např. posuvník na stránce `https://www.sazka.cz/loterie/kameny/sazky-a-vysledky`), je nutné použít klíčové slovo *localURL*, např. takto: *click localURL <css selektor>*.

click - Můžete také provést jen obyčejné kliknutí na místě, kde se zrovna nachází kurzor Selenia bez toho, abyste upřesňovali element, na který chcete kliknout.

select <css selektor> - Tato vlastnost vznikla za účelem vybrání *<option>* elementu z nabídky, jelikož na tento element není možné kliknout výše zmíněnými způsoby - nastane `JavascriptException`, neboť podle prohlížeče má prvek *<option>* nulovou délku a nulovou výšku, tedy není možné na něj kliknout nebo nad něj přesunout kurzor. Nicméně, akce *select* provádí simulované kliknutí pomocí Selenia, které dokáže prvek vybrat. Tuto akci je tedy možné použít i pro kliknutí na jiný libovolný element, pouze s nevýhodou popsanou v kapitole 4.1.9, tj. prvek, na který klikáme, musí být na obrazovce.

move to <css selektor> - Přesune kurzor na element specifikovaný pomocí CSS selektoru. Tento element musí právě nyní být zobrazen na obrazovce (toho je možné

dosáhnout pomocí akce *scroll to*).

scroll to <css selektor> - provede přesunutí obrazovky tak, aby byl zobrazený element specifikovaný pomocí CSS selektoru.

wait <ms> - Počká *<ms>* milisekund, než proběhne další akce. Např. *wait 1000*. Není nutné uvádět *wait* jako první akci v *beforeActions*, čekání na načtení stránky probíhá automaticky. Toto čekání je neúspěšné, pokud se stránka nezobrazí ani po 7 sekundách.

Akce pro ošetření výjimek - V *beforeActions* nebo *nextDataActions* je možné přidat akce, které mohou potenciálně vyhodit výjimku a následně akce, které na výjimku reagují. Akce *try start* znamená začátek bloku `try`, ve kterém se vyskytují nebezpečné akce, *try end* tento blok ukončuje. Za každým blokem `try` musí být blok `catch`, který začíná použitím *catch start* a je ukončen pomocí *catch end*. `Catch` blok zachytí veškeré výjimky děděné od `java.lang.Exception`, veškeré `Throwable` tedy zachyceny nejsou. Bloky `try` nesmí být zanořeny. Pokud je v bloku `catch` vyhozena výjimka, způsobí ukončení programu. Použití bloku `try` slouží k odchycení výjimek způsobených manipulací s DOM stránky, ale pokud budou akce v `try` bloku nesprávně zapsány, přesto bude program ukončen. Problematika ošetřování výjimek je více vysvětlena dále.

decreaseUrlParam <parametry> - Sníží číselnou hodnotu v URL o 1. Zároveň zajistí, že tato hodnota neklesne pod dolní mez, která je zadána. Umožňuje sofistikovanější změny URL tím, že je přidáno více URL parametrů a způsoby, jak je měnit. Parametry jsou děleny po čtveřicích, kde první člen čtveřice je název parametru, druhý je minimum, kterého může nabývat, třetí maximum a čtvrtý určuje dolní mez. Princip je lépe ilustrován na příkladu - uvažujme následující *nextDataActions* jako součást konfigurace.

```
{
  "nextDataActions": [
    "decreaseUrlParam week 1 52 12 year 0 5000 2012"
  ]
}
```

V tomto případě se nejdříve bude program pokoušet snížit parametr *week* v URL o 1, pokud ale již má hodnotu 1, která je určena jako minimum, nastaví se opět na své maximum a pokračuje se vyhodnocováním dále, kde se sníží parametr *year* o 1. Pokud ale již *year* nabývá hodnoty 0, další parametry nejsou zmíněny, takže program končí. Také, pokud *week* právě nabývá hodnoty 12 a *year* hodnoty 2012, program skončí. Při použití *decreaseUrlParam* není nutno uvádět příkaz *wait*, program bude pokračovat teprve, jakmile je nová stránka načtena.

A.4 Poznámky k získávání dat

Data jsou harvestována do té doby, dokud se neobjeví již jednou získaná data. Z toho důvodu není možné používat *skipHeaderRows* nastavené na *false* za účelem logování hlavičky tabulky, nebo jiným způsobem získávat tuto hlavičku, jelikož při následujících spuštěních programu se hlavička vyhodnotí jako již zapsaná a program skončí s prázdným výstupním souborem. Vlastnost *skipHeaderRows* je tedy určena spíše pro případ, kdy samotná hlavička také obsahuje data. Ukládá se vždy nejnovější získaný řádek tabulky, a považuje se za shodný s právě procházeným řádkem, pokud všechny získané informace v tomto řádku jsou stejné s informacemi v řádku právě získaném web scrapingem.

Data programu jsou přechovávána v souboru *.program_data.txt* (vytvoří se při prvním spuštění). Pokud chcete tato data odstranit, stačí soubor vymazat. Díky tomu budou resetovány všechny proměnné *${i}* pro všechny stránky, zároveň budou u všech loterií odstraněny informace o posledním záznamu. Nebo můžete soubor otevřít a editovat, je zapsán v čitelné podobě.

Mezi jednotlivé manipulace s DOM stránky je vhodné přidávat příkazy uspání, aby se stihla stránka přegenerovat. Pokud se bude stránka přegenerovávat během doby, kdy s ní program manipuluje (což může být ale také způsobeno dynamickým generováním stránky), způsobí to `StaleElementReferenceException`, se kterou si nicméně program dokáže poradit tím způsobem, že celý sběr dat pro tuto loterii začne znovu. Pokud bude však tato výjimka pro stejnou stránku vyhozena potřetí, program se rozhodne, že sběr dat ukončí chybou, radši než aby způsobil nekonečné zacyklení. Z toho důvodu je dobré přidávat příkazy *sleep* s pečlivě zváženou dobou.

Akce *move to <css selektor>* simuluje pohyb kurzoru, a je tedy vhodné, aby byl během provádění této akce kurzor mimo spuštěné okno prohlížeče a nehýbal se, jinak program může zacílit na obrazovce nesprávný prvek.

A.5 Složitější sbírání dat po dnech

Pokud jsou data členěna po dnech a každý den vyžaduje ještě své *nextDataActions*, je možné použít konstrukci typu:

```
{
  "nextDataActions": [
    "try start",
    "click li:last-child .button--sm.btn--interaction",
    "wait 3000",
    "try end",
    "catch start",
    "click #lotteryHistoryResultsDropdownSelectBoxItText",
```

```
    "wait 2000",
    "click #lotteryDropdownSelectBox li:nth-child(50)",
    "wait 1000",
    "catch end"
  ]
}
```

kde se v `try` větvi vykonává přechod mezi záznamy v určitý den, pokud se to nepodaří, v `catch` větvi se přejde na další den. Nicméně pokud nechcete přecházet na další den, nemusíte výjimku ošetřovat, program si poradí sám a určí, že je čas skončit.

A.6 Sběr dat v reálném čase

Pro sběr dat v reálném čase je možné použít parametr *realTime* při spouštění JAR souboru, například tedy

```
java -jar .\bakalarska_prace-1.0-SNAPSHOT.jar realTime
```

Program se Vás zeptá, kolik milisekund má spát mezi jednotlivými sběry dat. Běží pak neustále, dokud ho neukončíte např. klávesovou zkratkou Ctrl+C, přičemž pokaždé zaznamená jen nově dostupná data. Pokud si přejete pro každou konfiguraci zvolit jinou dobu spánku programu, je možné jednoduše vytvořit více duplikací JAR souboru v různých adresářích a ke každému přiřadit vlastní soubor *config.json*, ve kterém budou uvedeny jen ty loterie, které má určitý program obsluhovat.

Uživatelská příručka programu pracujícím s API

B

Program se napojí na API, ze kterého stahuje výsledky losování. Způsob, jakým sběr dat probíhá, je určen konfigurací, kterou je možné přizpůsobit dle potřeb.

B.1 Struktura projektu

- main.py - vstupní bod programu
- realtime_main.py - vstupní bod programu pro sběr dat v reálném čase
- api_accessor.py, config_manager.py, output_manager.py - ostatní skripty
- config.json - globální konfigurační soubor
- site_configs - obsahuje konfigurace pro jednotlivé loterie
- requirements.txt - obsahuje balíčky, které je nutné doinstalovat

B.2 Příprava

Je nutné mít stažený Python, ideálně verze 3.11.2 nebo novější. Program používá knihovnu requests, kterou je nutné doinstalovat, aby vše fungovalo správně. K tomu můžete použít příkaz `pip install -r requirements.txt`. Program je poté možné spustit pomocí příkazu `python main.py`.

B.3 Konfigurace

B.3.1 Soubor *config.json*

Soubor *config.json* obsahuje konfiguraci společnou pro všechny loterie. Většinu z těchto vlastností může každá loterie přepsat, nebo je může převzít. Zde je výpis vlastností,

které může soubor *config.json* obsahovat:

sites - Pole obsahující názvy souborů, ve kterých jsou uloženy konfigurace pro jednotlivé stránky (loterie). Tato vlastnost **musí být v globální konfiguraci uvedena**.

output - Určuje, kam bude vypsán výstup. Je možné uvést název souboru nebo *console*, pokud chcete výstup zobrazit na konzoli. Tuto vlastnost **není nutné uvádět**, je možné použít výchozí hodnotu, tj. *console*. V hodnotě této vlastnosti je možné použít proměnné (viz dále).

params - Parametry pro volání API. Jedná se o GET parametry nebo data posílaná v POST požadavku. Tuto vlastnost **není nutné uvést**, pokud se neočekává odeslání jakýchkoliv parametrů dotazu. Parametry se dělí na 2 skupiny, *bound* a *changing*. Obě tyto skupiny parametrů jsou JSON objekty, obsahující objekty parametrů - klíčem je vždy název parametru (viz příklad globální konfigurace níže). Každý takový objekt reprezentující parametr má vlastnost *value*, která určuje, jaké hodnoty nabývá v závislosti např. na proměnných nebo dat získaných z API. Dělení parametrů je tedy následující:

- *bound* - Tyto parametry jsou navázány na nějakou hodnotu získanou voláním API. Například pokud volání API vrátí čas slosování posledního záznamu, můžeme při dalším volání API tento čas využít a vyžádat si pouze starší záznamy. Všechny *bound* parametry musejí definovat vlastnost *bindValueWith*, ve které je určeno, jak navázat hodnotu parametru na data získaná z API (např. *"bindValueWith": "${api[record.localDrawDate]}"*).

- *changing* - Parametry, které automaticky zvyšují nebo snižují hodnotu nějaké proměnné při každém přístupu k API (viz dále). Je možné ke každému parametru nastavit vlastnost *maxValue*, při které se parametr dále nezvyšuje. Je také možné použít více *changing* parametrů, nejdříve se při každém přistoupení k API bude zvyšovat/snižovat první uvedený dokud nedosáhne hodnoty *maxValue*, pak se začne zvyšovat/snižovat následující parametr. Pokud navíc použijeme další vlastnost, *resetOnChange* (určuje pole parametrů, které budou při zvýšení/snížení hodnoty proměnné resetovány), je možné například procházet data po dnech, ale "vnitřním cyklem"navíc ještě po stránkách.

sleepTime - Udává čas, jaký bude program/vlákno spát po obdržení odpovědi ze serveru, než opět přistoupí k API. Může mít buď číselnou hodnotu (kolik milisekund bude vlákno spát), nebo může být ve tvaru *"random <min> <max>"*, kde *<min>* je minimální doba spánku v milisekundách a *<max>* maximální. Program pak zvolí náhodnou dobu spánku z intervalu. Vlastnost *sleepTime* **není nutné uvést**, pak bude použita výchozí hodnota, tj. 0 (okamžitě po zpracování odpovědi serveru bude poslán další požadavek).

dataTarget - Určuje, jak v JSON odpovědi najít data o losování. Pokud celá odpověď obsahuje pouze data, *dataTarget* by měla být nastavena na prázdný řetězec (""). V opačném případě obsahuje název JSON vlastnosti, pod kterou jsou dostupná data z losování (např. "results"). Tuto vlastnost **není nutné specifikovat**, pak bude použita výchozí hodnota, tj. "".

maxRequests - Maximální počet požadavků, po kterých bude sběr dat pro určitou loterii ukončený. Pokud není specifikováno, sběr dat není nijak omezený.

startDate - Nastaví při spuštění sběru dat datum losování na určitou hodnotu (je požadována hodnota ve tvaru *"YYYY-MM-DD"* nebo *"today"*, pokud má začít sběr dat dnešním dnem). Výchozí hodnota je *"today"*, tudíž **není nutné tento parametr specifikovat**.

threads - Počet vláken, mezi které se rozdělí jednotlivé loterie a na kterých bude běžet sběr dat. Počet vláken nesmí být menší než 1 nebo větší než počet loterií. **Není nutné tuto vlastnost uvádět**, ve výchozím nastavení bude nastaven počet vláken na 1.

autoStop - Pokud je hodnota této vlastnosti *true*, sběr dat se zastaví, jakmile bude proměnná *date* obsahovat datum, se kterým začínal minulý sběr. Vlastnost **není nutné uvádět**, ve výchozím nastavení je *false*, takže se program automaticky nezastaví.

rewriteLastDay - Pokud je nastaveno na *true* a je využíván *autoStop*, nezastaví se sběr při přechodu na den, ve kterém začal minulý sběr, ale ještě o jeden den déle. To je užitečné, pokud po sběru dat mohou být ještě další slosování, takže minulý sběr nemusel být kompletní. Ve výchozím nastavení je vlastnost nastavena na *false* a **není tedy nutné ji specifikovat**.

appendMode - Určuje, zda má zápis do výstupního souboru probíhat v režimu *append*. Vlastnost **není nutné uvádět**, ve výchozím nastavení je použito *false*.

record - Určuje, jaké záznamy z loterie mají být vypsány na výstup a v jakém tvaru. Ačkoliv je tuto vlastnost lepší definovat v lokální konfiguraci, je možné ji zapsat i do souboru *config.json*. Ve výchozím nastavení nabývá hodnoty *""* (prázdný řetězec), což značí, že mají být vypsána úplně všechna data z *dataTarget*. Vlastnost tedy **není nutné specifikovat**. Je nicméně možné navázat hodnotu na nějaká data právě z *dataTarget*, například takto: *"record": "${data[drawId]}, ${data[drawDate]}, ${data[drawTime]}, ${data[results]}"* (jednotlivá data budou oddělena čárkou, mezi jednotlivé záznamy (řádky souboru) program doplní čárku automaticky).

Příklad globální konfigurace:

```
{
    "sites": [
      "site_configs/kameny.json",
      "site_configs/keno.json",
      "site_configs/rychla-6.json",
```

```
      "site_configs/rychle-kacky.json",
      "site_configs/vsechno-nebo-nic.json"
    ],
    "output": "output/${name}/${date[yyyy]}-
    ${date[mm]}-${date[dd]}.csv",
    "params":
    {
      "changing": {
          "page": {
              "value": "${i}",
              "maxValue": "${api[totalPages]}"
          },
          "day": {
              "value": "${date[yyyy]}-${date[mm]}-
              ${date[dd]}T00:00:00",
              "resetOnChange": ["page"]
          }
       }
    },
    "sleepTime": "random 1000 2000",
    "dataTarget": "results",
    "maxRequests": 45,
    "startDate": "today",
    "threads": 5,
    "rewriteLastDay": true,
    "autoStop": true,
    "appendMode": false
}
```

B.3.2 Konfigurace jednotlivých loterií

Konfigurace pro jednotlivé loterie jsou uloženy v souborech uvedených ve vlastnosti *sites* souboru *config.json*. Nepřepsané vlastnosti globálni konfigurace budou zděděny. Je možné přepsat všechny hodnoty vlastností z globální konfigurace, kromě vlastností *sites* a *threads*. Dále je nutné uvést 2 vlastnosti speciální pro každou loterii:

url - URL adresa API bez parametrů dotazu.

method - Metoda přístupu k datům, nabývá hodnoty *GET* nebo *POST*.

Příklad lokální konfigurace, která přepisuje vlastnosti *dataTarget* a *params*:

```
{
    "url": "https://www.sazka.cz/api/draw-info/draws/
    quick-lottery/vsechno-nebo-nic/results",
    "method": "GET",
    "dataTarget": "",
    "params":
    {
      "bound": {
```

```
            "to": {
                "value": "${datetime}",
                "bindValueWith": "${api[record.localDrawDate]}"
            }
        }
    },
  "record": "${data[drawId]}, ${data[drawDate]},
  ${data[drawTime]}, ${data[results]}"
}
```

B.4 Proměnné a přístup k datům z API

Program umožňuje vytvořit parametry navázané na datum, čas a na proměnnou *i*. Jelikož sběr dat typicky probíhá od nějakého okamžiku zpět v čase, datum a čas je automaticky při přistoupení k API snižován (pokud je na proměnnou navázán *changing* parametr), zatímco *i* je zvyšováno. Pokud chcete navázat parametr na nějakou proměnnou, např. *i*, stačí zadat *"value": "${i}"*. Datum a čas je přístupný přímo pod proměnnou *${date}* a *${time}*, ale také je možné specifikovat, jakou složku data a času chcete získat - např. měsíc/minutu. Pro datum tedy můžete využít *${date[yyyy]}*, *${date[mm]}* a *${date[dd]}*, pro čas pak *${time[hh]}*, *${time[mm]}* nebo *${time[ss]}*. Čas je vždy nastaven na *00:00:00*, datum závisí na konfiguraci (vlastnost *startDate*) a *i* je nastaveno na *1*.

V hodnotě *bindValueWith* vlastnosti *bound* parametrů je možné zadat, jakým způsobem se propojí hodnota parametru s API pomocí zápisu *${api[identifikatorObsahu]}*, kde *identifikatorObsahu* definuje obsah, který se má na parametr navázat. Identifikátor může být například *"date"*, pokud ve vráceném JSON objektu je vlastnost *date*, nebo například *record.date*, pokud vlastnost *date* je uvnitř každého ze záznamů, které jsou vypisovány na výstup.

V hodnotě vlastnosti *record* je pak možné pomocí *${data[identikator]}* specifikovat, jaká konkrétní data mají být zapsána (viz ukázka výše).

B.5 Sběr dat v reálném čase

Pro sběr dat v reálném čase spusťte příkaz `python realtime_main.py`. Program se Vás zeptá, jak dlouhou dobu má vyčkávat mezi jednotlivými sběry dat. Sběr dat v reálném čase je možné vykonávat pouze na jednom vlákně. Nejspíše bude užitečné nastavit *appendMode* na *true*, aby se nová data připisovala na konec již existujícího souboru.

Pokud uvažujeme hypotetickou loterii, ve které probíhá slosování každé 2 sekundy a každé volání API poskytne posledních 100 slosování, budete zřejmě chtít nastavit *maxRequests* na 1 a mezi sběry dat vždy uspat program na 200 sekund. Tímto

způsobem se každých 200 sekund, během kterých se vygeneruje kompletně nová odpověď API, program připojí k API a zapíše nová data.

Bibliografie

1. CAVALCANTE, Y. APIs and Data Collection. *ResearchGate*. 2024. Dostupné z DOI: 10.13140/RG.2.2.32543.53920.
2. KAUSAR, M. A.; DHAKA, V. S.; SINGH, S. K. Web Crawler: A Review. *International Journal of Computer Applications*. 2013. Dostupné z DOI: 10.5120/10440-5125.
3. SCHINTLER, L. A.; MCNEELY, C. L. *Encyclopedia of Big Data*. Web Scraping. Ed. ZHAO, B. School of Policy, Government a International Affairs, George Mason University, Fairfax, USA, 2017. ISBN 978-3-319-32001-4.
4. *java.net.http (Java SE 11 & JDK 11)* [online]. Oracle. [cit. 2024-04-21]. Dostupné z: https://docs.oracle.com/en%2Fjava%2Fjavase%2F11%2Fdocs%2Fapi%2F%2F/java.net.http/java/net/http/package-summary.html.
5. *requests 2.31.0* [online]. The Python Package Index. [cit. 2024-04-21]. Dostupné z: https://pypi.org/project/requests/.
6. *Call a Web API From a .NET Client (C#)* [online]. Microsoft. [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client.
7. TANASĂ, A. M.; OPREA, S.; ADELA, B. Web Scraping and Review Analytics. Extracting Insights from Commercial Data. In: *"Ovidius" University Annals, Economic Sciences Series Volume XXIII, Issue 2 /2023*. 2023.
8. GARCÍA, B. *WebDriverManager* [online]. [cit. 2024-04-21]. Dostupné z: https://bonigarcia.dev/webdrivermanager/webdrivermanager.pdf.
9. THÉR, J. *Extrakce dat z webu pomocí web scrapingu*. 2022. Diplomová práce. Univerzita Hradec Králové, Katedra informačních technologií.
10. GALUSHKA, V.; MARSHAKOV, D.; FATHI, V. Dynamic document object model formation technique for corporate website protection against automatic coping of information. In: *MATEC Web of Conferences*. 2017. Dostupné z DOI: 10.1051/matecconf/201713205001.

11. SERBOUT, S.; MALKI, A. E.; PAUTASSO, C.; ZDUN, U. API Rate Limit Adoption – A pattern collection. In: *28th European Conference on Pattern Languages of Programs At Kloster Irsee, Germany*. 2023.

Seznam obrázků

Seznam tabulek