Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

****

**Sběr dat z webu pomocí nástroje Selenium**

Do předmětu 4IZ470 DOBÝVÁNÍ ZNALOSTÍ Z WEBU

ve studijním programu Znalostní a webové technologie

Autor: Vítězslav Slavík

Praha, červen 2020

Obsah

[1 Selenium 1](#_Toc42206618)

[1.1 Webdriver 1](#_Toc42206619)

[1.2 Osobní zkušenost 1](#_Toc42206620)

[1.3 Alternativy 2](#_Toc42206621)

[1.3.1 Scrapy 2](#_Toc42206622)

[1.3.2 BeautifulSoup 2](#_Toc42206623)

[1.3.3 Srovnání 2](#_Toc42206624)

[2 Praktická část 3](#_Toc42206625)

[2.1 Cíl praktické části 3](#_Toc42206626)

[2.2 Zdroj dat 3](#_Toc42206627)

[2.3 Příprava prostředí 3](#_Toc42206628)

[2.3.1 Selenium Webdriver 3](#_Toc42206629)

[2.3.2 Python prostředí 4](#_Toc42206630)

[2.3.3 Build-In knihovny 4](#_Toc42206631)

[2.3.4 Externí knihovny 4](#_Toc42206632)

[2.4 Průběh vývoje 5](#_Toc42206633)

[2.5 Problémy 6](#_Toc42206634)

[2.5.1 Identifikátory elementů 6](#_Toc42206635)

[2.5.2 Dynamický počet elementů 6](#_Toc42206636)

[2.5.3 Zaokrouhlování 6](#_Toc42206637)

[2.5.4 Omezení pro počet requestů 6](#_Toc42206638)

[2.5.5 Načítání výsledků 7](#_Toc42206639)

[2.6 Výsledky 7](#_Toc42206640)

[Závěr 10](#_Toc42206641)

[Seznam Literatury 11](#_Toc42206642)

# Selenium

Selenium je populární sada nástrojů pro automatické testování webových aplikací. Skládá se z několika komponentů, které jsou zaměřené na různé role v automatickém testování a liší se také svým nárokem na znalosti uživatele.

Nejsnadnějším nástrojem ze sady je Selenium IDE. Jak už zkratka napovídá, jedná se o integrované vývojové prostředí pro testování. Je implementováno jako plugin do Firefoxu a Chrome. (wikipedia.com 2020) Hlavní limitací tohoto přístupu je limitovaný počet prohlížečů. Pomocí IDE není možné testovat alternativní prohlížeče jako Safari, Opera, Internet Explorer. Podle webu W3couter mají tyto chybějící prohlížeče skoro 20% podíl na trhu. (W3Counter.com 2020) Navíc ne všechny z nich využívají vykreslovací engine Chromium, je vhodné tyto prohlížeče řádně otestovat, protože chování aplikace může být velmi různé v každém z nich.

Druhým nástrojem v sadě je Selenium Grid. Ten umožňuje provádět testování ve více instancích naráz. Grid server slouží jako hub pro vzdálené počítače, které mají nainstalovány Webdriver. Pomocí gridu tak můžeme provádět korigované testování ve více instancích naráz.

Dalším nástrojem je Selenium Remote Control. Je předchůdcem pro námi využívaného Webdriveru. Jedná se o server, který je ovládán skrze http požadavky plynoucích z testovacích skriptů. Výhodou tohoto přístupu je vzdálené ovládání skrze různé programovací jazyky. To však klade větší náročnost na znalosti uživatele pro použití.

## Webdriver

Selenium Webdriver nám umožňuje stejně jako předchůdce RC ovládat pomocí API instanci prohlížeče a simulovat tak chování uživatele na stránce. Avšak používá k tomu rozdílnou architekturu. U RC jsme potřebovali mít Selenium server, který bude přijímat požadavky a vracet odpovědi, u Webdriveru vše běží na lokální stroji. (wikipedia.com 2020) Toto zjednodušení vede ke zlepšení rychlosti zpracování skriptů. Díky tomu se také ujal jako nástroj pro scraping dat, tedy automatizovaný sběr dat na webu.

Webdriver je možné ovládat skrze API v jazycích PHP, Python, Ruby, .NET, Perl a Java. V našem případě zvolíme Python API jednoduše z důvodu, že autor práce je s tímto jazykem nejlépe seznámen.

Velkou výhodou oproti IDE je počet dostupných platforem. Webdrivery jsou dostupné do všech 5 zmíněných prohlížečů a z archivu lze získat webdriver pro většinu stable verzí každého z nich.

## Osobní zkušenost

Během svého bakalářském studia jsem několikrát zaslechl termín „Vývoj řízený testy“. Moje zvědavost mě přivedla ke knize Test-Driven Development with Python od Harry J. W. Percivala. Tato kniha se snaží přátelsky popsat proud práce skrze testy řízeného vývoje webové aplikace. Jelikož jsem vytvářel vlastní projekt aplikace ve frameworku Django, který autor na svých příkladech také používá, použil jsem knihu jako názorného průvodce do tohoto způsobu vývoje.

Nejdříve jsme si určili use case dané stránky a tok uživatele skrze web. Poté jsme právě pomocí Selenia definovali objekty, se kterými bude uživatel provádět interakci. Tyto testy byly doplněny klasickými jednotkovými testy. Teprve poté se přešlo na samotný vývoj aplikace.

## Alternativy

Zde si představíme další 2 nástroje, které jsou velmi populární v data science komunitě při sběru dat na webu.

### Scrapy

Scrapy je ustálený framework pro crawling webu. Slouží ke stavbě spiderů, kteří prochází určitou skupinu stránek a sbírají data. Nástrojumožňuje práci s html, css a podporuje také Xpathové vyjádření. Je vybudován na základě Twisted, síťového frameworku, díky kterému umožňuje multiprocesové asynchronní zpracování požadavků. (Kocman 2020) Díky tomu je nejrychlejší ze všech 3 zmíněných scrapingových nástrojů.

Tento framework nemá pověst uživatelsky přívětivého nástroje a dle mého názoru chaotická dokumentace na tom hraje také svou roli. Mezi největší nevýhody patří chybějící podpora pro Javascript. S nástupem JS frontendových frameworků hraje tento faktor o to větší roli.

### BeautifulSoup

Hlavní účel tohoto nástroje je práce s HTML a XML dokumenty. Jedná se o nástroj jednoduchý na ovládání s dobrou dokumentací. Díky tomu je často vstupní branou pro světa scrapingu.

Jelikož není hlavním účelem nástroje samotný crawling, neumí posílat http požadavky ani parsovat xml dokument. Je proto nutné spolu s ní využít další knihovny. Také nemá nativní podporu multitaskingu. Za tímto účelem rychlého zpracování musíme využít například knihovnu multithreading, která ovšem není už tak snadná na použití. Další nevýhodou je neschopnost stejně jako scrapy sbírat data na stránkách spolehajících z velké části na javascript.

### Srovnání

Nejjednoduším nástrojem vhodným pro jednoduchou manipulaci s html je jednoznačně BeautifulSoup a díky dalším knihovnám lze zvýšit i jeho rychlost. Skutečně specializovaný nástroj na crawling je až Scrapy, který je však kvůli své robusnosti složitější na použití. Avšak nativní asynchroní zpracování z něj činní nejrychlejšího z představených nástrojů.

Selenium je tedy nejpomalejší z nástrojů. Avšak je multifunkční, je to primárně nástroj na testování. Jako největší výhodu bereme zvládnutou práci s weby spoléhající na javascript. Jako jedinný ze zmíněných nástrojů si poradí s „nekonečným“ načítáním i se Single Page aplikacemi. Proto u některých projektů, ačkoliv je primárně používán Scrapy, či BeautifulSoup, je k vyřešení těchto problémů také připojeno Selenium a zvyšuje se tim komplexita celého projektu.

# Praktická část

Tato část se zaměruje na praktické využití nástroje Selenium Webdriver. V první části části si představíme zdroj a charakter sbíraných dat. Poté sestavíme prostředí a nainstalujeme nástroje. V kapitolách průběh vývoje a problémy budou popsány principy, jakými je třeba takový crawlingový script řídit a které problémy jsme během vývoje museli řešit. V závěrečné části se podíváme na výsledný výstup a zhodnotíme kvalitu.

## Cíl praktické části

Cílem této části je automatizovat sběr dat z webové stránky www.github.com pomocí představeného nástroje Selenium a dalších pomocních nástrojů. Sběr bude probíhat pomocí skriptu, který je potřeba sestavit na míru struktuře stránky github.

## Zdroj dat

Zdroj našich dat je stránka [www.github.com](http://www.github.com) založená v roce 2008, nyní patřící pod společnost Microsoft Corporation. Jedná se webovou službu pro vývoj softwaru spoléhající na verzovací nástroj Git. Nabízí bezplatný hosting pro IT projekty. (wikipedia.com 2020) Služba je svojí funkcionalitou podobná sociálním sítím (uživatelé, diskuze) a je rozšířená převážně mezi open source komunitou. Uživatelé tu mohou diskutovat nad kódem, přispívat do projektů či nahlašovat případné bugy.

Naší snahou bude získat data o populárních repozitářích. Tak se nazívají zde umístěné projekty. K vyhledání těchto frekventovaných projektů využijeme podstránku „Github Collections“, která sdružuje podobné repozitáře do kategorií. Najdeme zde kupříkladu kategorie repozitářů pro zlepšení produktivity vývoje nebo tzv. lintery pro kontrolu kvality kódu.

Detailní stránka každého repozitáře obsahuje rozličné informace o něm. Dozvíme se počet hvězdiček, které značí jeho oblíbenost, dále počet „watchers“, tedy uživatelů, kteří odebírají novinky o repozitáři. Zajímavé je také informace o poměru používaných programovacích jazyků v projektu. V podkategoriích jsou umístěné diskuze otevřenými bugy, či žádosti o sloučení přídavku od přispěvatelů. Zde nás bude zajímat kolik takových vláken v těchto diskuzích existuje. Samozřejmě nezapomeneme na důležité informace jako celkový počet přízpěvků (commitů) nebo kolik lidí již přispělo svým kódem.

## Příprava prostředí

### Selenium Webdriver

V naší práci použijeme webdriver nejpopulárnějšího prohlížeče Google Chrome. Nástroj je dostupný ke stažení na adrese <https://chromedriver.chromium.org/downloads>. Je nutné se nejdříve podívat na verzi lokálně nainstalovaného prohlížeče. Tak provedeme, pokud napíšeme příkaz „<chrome://settings/help>“ do vyhledávacího pole. Náš prohlížeč běží na verzi 83.0.4103.61. Ve zmíněném archivu jsou dostupné verze pro všechny stabilní buildy prohlížeče. My si stáhneme ovladač pro naši konkrétní verzi Chromu a operačního systému, jímž je MS Windows.

V archivu nalezneme soubor chromedriver.exe. Soubor uložíme například do složky našeho projektu. Cestu k souboru budeme muset později ve skriptu explicitně definovat. To bylo s ohledem na Webdriver vše, co je třeba vyřešit.

### Python prostředí

Skript budeme psát v dynamickém jazyce Python. Selenium API je podporovány verzemi 2.7, 3.5 a výše. My máme na lokálním stroji nainstalovanou distribuci Anaconda s Python verzí 3.7 dostupnou na adrese <https://www.anaconda.com/products/individual>, která přichází spolu s grafickým rozhraním a užitečným manažerem balíků nazvaným Conda.

Vytvoříme si virtuální prostředí „conda create -n 4iz470-env“, které bude obsahovat knihovny potřebné pro tento projekt. Pokud chceme replikovat prostředí hotového projektu, můžeme nainstalovat všechny potřebné balíky pomocí příkazu „pip install -r requirements.txt“. Jinak provedeme instalaci balíků klasicky pomocí „pip install package“ nebo „conda install package“.

### Build-In knihovny

Knihovny jsou součástí pythonu a není potřeba je instalovat.

* **Pathlib** – pro práci s objekty v souborovém systému
  + Dokumentace: <https://docs.python.org/3/library/pathlib.html>
* **Re** – pro vyhledávání textových řetězců podle regex vzorů
  + Dokumentace: <https://docs.python.org/3/library/re.html>
* **Time** – pro práci s časem uvnitř skriptu
  + Dokumentace: <https://docs.python.org/3/library/time.html>
* **Itertools** – pro vytváření efektivních iterátorů
  + Dokumentace: <https://docs.python.org/3/library/itertools.html>

### Externí knihovny

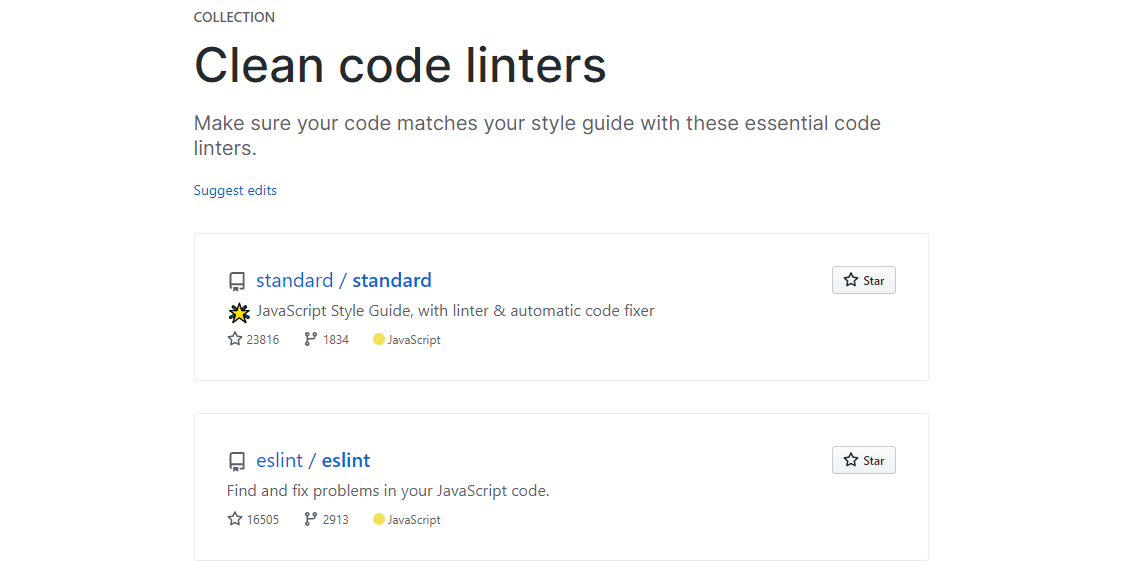
Tyto knihovny je potřeba nainstalovat, nejsou součástí zabudovaných balíčků.

* **Selenium** – API pro ovládání webdriveru
  + Dokumentace: <https://selenium-python.readthedocs.io/>
* **Jupyter** – vývojové prostředí pro vývoj
  + Dokumentace: <https://jupyter-notebook.readthedocs.io/en/stable/>
* **Pandas** – knihovna pro transformaci dat
  + Dokumentace: <https://pandas.pydata.org/docs/>
* **Seaborn** – knihovna pro vizualizaci
  + Dokumentace: <https://seaborn.pydata.org/>

## Průběh vývoje

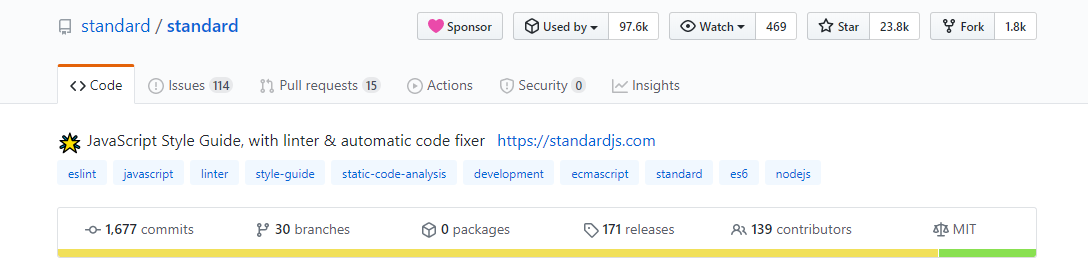
Pro vývoj je použit jupyter notebook, jelikož umožňuje vývoj per partes a průběžně kontrolovat obsah proměných. Na začátku skriptu naimportujeme všechny používané knihovny.

Nejdříve budeme pracovat jen s jedním repozitářem, abychom vyhledali elementy uchovávající informace. Snažíme se projít cestu skrze seznam kolekcí, přes seznam repozitářů v kolekci (zobrazen na obrázku) až do stránky detailu onoho projektu.



V okně klasickém prohlížeče otevřeme nástroje pro vývojáře. V prvním kroku se snažíme identifikovat elementy jako jsou odkazy a tlačítka, se kterými budeme provádět interakci a navigovat se po stránkách. Ve skriptu si tyto tlačítka uložíme do proměných a postupně provádíme akci kliknutí. V tomto kroku jsme schopni se dostat z katalogu kolekcí k detailu prvního repozitáře.

Dalším krokem je sesbírat cestou textový obsah elementů. Identifikujeme jednotlivé boxy s informacemi o stavu hvězdiček, commitů apod. Zjišťujeme, že na stránce repozitáře je počet pull requestů a issues zaokrouhlen, musíme tyto kategorie ještě ručně proklikat. Na dalším obrázku je zobrazena část detailu stránky s sbíranými údaji.



Již jsme schopni se proklikat na detail repozitáře a cestou posbírat údaje. Teď je na čase vytvořit loop, který tak učiní pro všechny kolekce a všechny repozitáře uvnitř. K tomu použijeme vrchní loop pro procházení listu kolekcí a vnořený loop pro procházení repozitářů v kolekci. Na konci každého vnořeného loopu vše vložíme do listu. Nyní můžeme již spustit skript a sbírat data. Avšak se setkáváme s několik problémy, které je třeba vyřešit. Ty jsou popsány ve speciální kapitole.

Po dokončení sběru dat převedeme list na datovou strukturu pandas dataframe a provedeme kontrolu dat.

## Problémy

V této kapitole jsou popsány problémy, které bylo potřeba vyřešit v rámci praktické části práce.

### Identifikátory elementů

Stránka github používá ve svých elementech velmi modulární CSS design. To nám z pohledu scrapingu znepříjemňuje život, protože elementy nemají unikátní názvy tříd ani ID. Tento problém je zvláště znatelný u značky odkazu <a>, které uchovávají informace a slouží k navigaci po webu.

Tento problém vyřešíme použitím dalších atributů. Kupříkladu se díváme kam nás “href” u linku odkazuje. Nebo využíváme githube implementované atributy s prefixem “data-”. V těchto atributech prohledáme obsah, zdali obsahuje klíčová slova. K tomu využíváme xpath s funkcí contains. Kupříkladu string „//a[contains(@data-selected-links,'repo\_issues')]“ vyhledá všechny odkazy, které nás odkazují na diskuzi ohledně bugů v projektu.

### Dynamický počet elementů

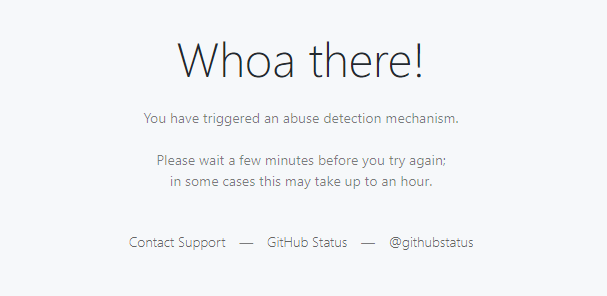
Některé repozitáře mají zapnutou speciální funkcionalitu, která přidává tlačítka a odkazy na stránku. Jelikož je zde předchozí problém s indetifikátory, je ve skriptu využíváno na některých místech prosté indexování. Kupříkladu víme, že informace o počtu odběratelů je obsahem prvního odkazu v horním menu. Avšak v 30. repozitáři je zapnutá funcionalita sponzorování, která je v tom případě na prvním místě. Proto jsme museli častokrát opravovat skript a zpřesňovat identifikaci elementů.

### Zaokrouhlování

Github má ve zvyku na detailní stránce dost zaokrouhlovat na tisíce. Děje se tak u počtu hvězdiček, forků, problémů, pull requestů a dále. Proto je potřeba rozklikávat i další podstránky detailu, abychom se dostali k přesným číslům.

### Omezení pro počet requestů

Největším problémem je ale mechanismus na stránce, který detekuje anomální chování uživatel. Jelikož posíláme v krátkém časovém úseku velké množství požadavků, server po čase zjišťuje, že jsme program a zablokuje nám další tok dat.



Tomu se potřebujeme vyhnout. Z diskuzního fora reddit psal člen technické podpory, že přihlášením na stránku se počet možných požadavků navýší. Také přidáme do našeho skriptu časové rozestupy mezi jednotlivými akcemi. Skript tedy bude sbírat data déle, ale to v našem případě nehraje roli, jelikož se jedná o jednorázový sběr. Tento problém by bylo potřeba vyřešit, pokud bychom sbírali data pravidelně, například několikrát denně. To bychom například prováděli další experimenty za účelem optimalizace počtu requestů za určitý úsek času.

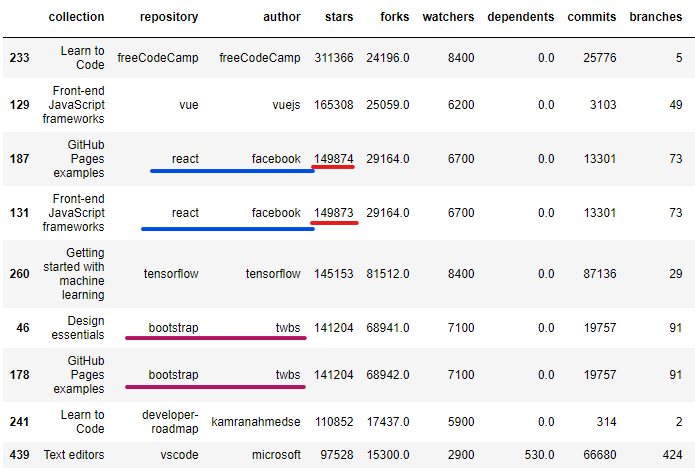
### Načítání výsledků

Github nepoužívá při zobrazení dlouhých seznamů stránkování. K načítání dalšího obsahu je použit AJAX, který do původní stránky vloží nově načtené položky. Je potřebné, aby skript sledoval, zdali má možnost načíst další výsledky a případně je prošel. Naštěstí Selenium tento problém řeší velmi dobře. S alternativními nástroji bychom zde ovšem narazili. Nemožnost načíst další kolekce a další repozitáře by nás omezila na základních 20 výsledků na 1 stránku. Přišli bychom o přibližně 40 % výsledků.

## Výsledky

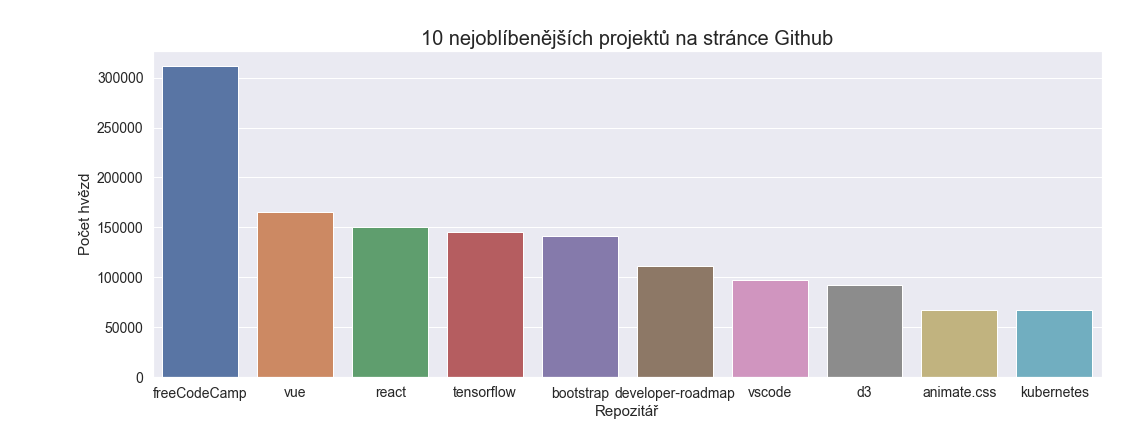
V této chvíli jsme úspěšně prošli kolekce a vybrali z repozitářů data. Některé kolekce se velmi lišili svou povahou obsahu a způsobili pád skriptu. Naštěstí lze crawler jednoduše korigovat, aby začal procházet stránky nadcházející od té problematické. Na konci jsme měli kvůli těmto problémům několik částečných datových setů. Sloučili jsme je dohromady a upravili datové typy. Prázdné hodnoty jsme doplnili číslem 0.

Při kontrole kvality jsme zjistili, že se zde nachází duplicity (na obrázku modře a fialově). To se dalo očekávat, některé repozitáře mohou být zařazeny ve více kategoriích. Také si můžeme všimnout, že jelikož se jednotlivé kategorie procházeli v různý čas, může se míra u stejného repozitáře lišit (na obrázku červeně).

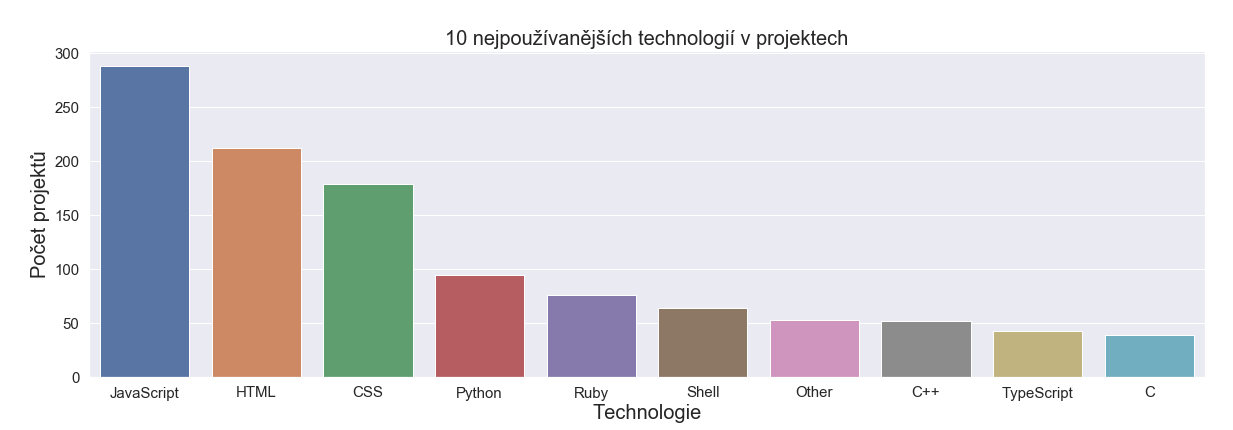


Odebereme tedy duplicity. Také si můžeme všimnout, že některé jazyky jsou sice v našich nashromážděných datech, tudíš byli použity u projektu, ale jeho hodnoty jsou ve všech bodech nulové. Je to proto, že tato technologie mohla být v malém měřítku použitá v projektu, ale jelikož github zaokrouhluje hodnoty na jednu desetinou čárku, výsledek je 0. Tato chyba není v našich silách napravit.

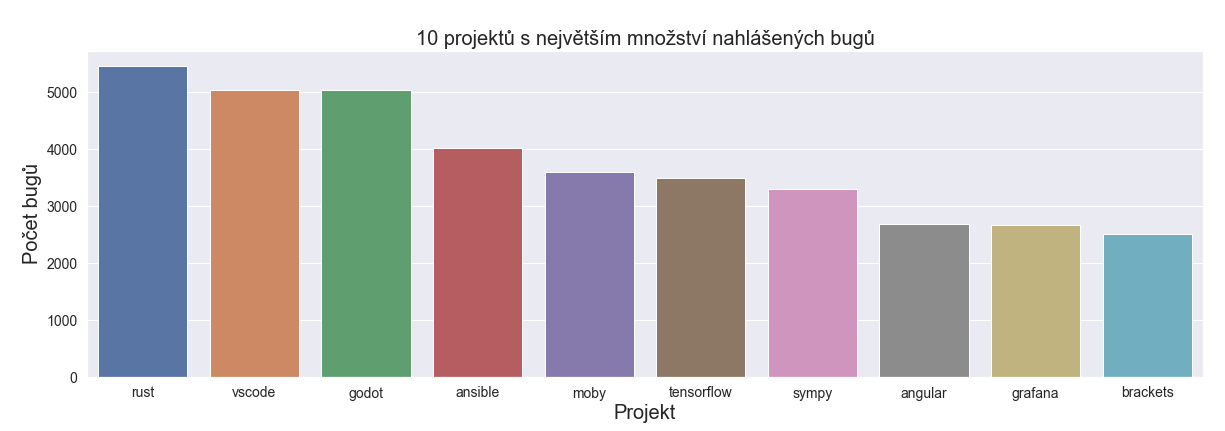
Po odstranění duplicit jsme prošli celkem 35 kolekcí s výjimkou těch, které mají jinou než klasickou strukturu. Z nich jsme posbírali data o 549 repozitářích od 499 autorů. Nyní se můžeme podívat na některé statistiky. Mezi nejoblíbenější repozitáře se zařadil výukový repozitář freeCodeCamp. Na dalších místech vidíme hned dva javascriptové fronendové frameworky.



Na dalším grafu vidíme graf nejpoužívanějších technologií. Vidíme, že nejvyužívanejším jazykem je javascript, používá se ve více než polovině z posbíraných projektů. Webové technologie obsazují i druhou a třetí příčku. Za nimy následují oblíbené skriptovací nástroje.



Z dalšího grafu zjišťujeme repozitáře, které mají největší počet otevřených diskuzních vláken, kde vývojáři popisují zvlášní chování programu. Vysoké číslo nemusí nutně být negativní zprávou, že je projekt plný chyb. Naopak to značí, že komunita kolem těchto projektů je velmi aktivní a snaží se danou technologii vylepšit a optimalizovat.



Závěr

V úvodní části práce jsme si představili sadu nástrojů Selenium. Poté jsme si popsali alternativy využitelné pro naši úlohu. V praktické části byla definována úloha scrapování dat. Prozkoumali jsme cílový zdroj dat a charakterizovali prvky vhodné pro sběr. Připravili jsme si prostředí pro vývoj řešení. Poté jsme vytvořili skript, který automatizuje úlohu sběru dat a popsali postup, podle kterého jsme tak učinili. Ve speciální kapitole jsme vypsali problémy, které bylo nutné pro funkční sběr dat překonat. V poslední fázi jsme zhodnotili výsledky sběru a vytvořili vizualizaci.

Seznam Literatury

KOCMAN, Tomáš, 2020. Představení knihovny Scrapy pro tvorbu web crawlerů. *Root.cz* [online] [vid. 2020-06-04]. Dostupné z: https://www.root.cz/clanky/predstaveni-knihovny-scrapy-pro-tvorbu-web-crawleru/

W3COUNTER.COM, 2020. W3Counter: Global Web Stats. *W3Counter: Global Web Stats* [online] [vid. 2020-06-02]. Dostupné z: https://www.w3counter.com/globalstats.php

WIKIPEDIA.COM, 2020. *GitHub* [online]. [vid. 2020-06-04]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=GitHub&oldid=18109202

WIKIPEDIA.COM, 2020. *Selenium (software)* [online]. [vid. 2020-06-02]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Selenium\_(software)&oldid=956221275