VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

MINISTERSTVO VNITRA ČESKÉ REPUBLIKY

Příjemci podpory:

Vysoké učení technické v Brně
Fakulta informačních technologií

Poskytovatel:

Ministerstvo vnitra ČR

BAZAR: Budování komunity k problematice bezpeněženkových temných tržišť

Identifikační kód *VJ01030004*

Název předkládaného výsledku:

Software pro sběr dat a metadat z darknet tržišť

Typ výsledku dle UV č. 837/2017	Evidenční číslo (příjemce)	Rok vzniku
R software	R1	2021
ISBN-ISSN	**Webový odkaz na výsledek**	**Kde a kdy publikováno**
	https://www.fit.vut.cz/research/product/709/	web

Stručná anotace k výsledku:

Software periodicky přistupuje na vybraná darknet tržiště (Monopoly Market). Z neveřejného obsahu tržišť pak stahuje seznamy nabízených zboží, jejich ceny a stav skladových zásob, informace o prodejcích i nakupujících. Všechny tyto údaje ukládá do relační databáze umožňující hromadné zpracování a analýzu.

Řešitelský tým:

Daniel Dolejška, Vladimír Veselý (manažer a hlavní řešitel), Matěj Grégr

Příjemci podpory: ***Poskytovatel:***

Obsah

Úvod 1
Instalace 1
Spuštění aplikace 1
Monitoring 1
Uživatelský manuál 3
Runtime aplikace 4
API aplikace 4
Vlastní služby 5
Programátorská dokumentace 5
Užitečné akce 6
Konfigurace přístupu k API 6
Registrace záznamů služeb 8

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

MINISTERSTVO VNITRA ČESKÉ REPUBLIKY

Příjemci podpory: ***Poskytovatel:***

Úvod

Tento projekt si klade za cíl poskytnout přizpůsobitelný nástroj pro sběr webových dat snadno použitelný v síti Tor. Primárním cílem této implementace je umožnit automatizované a snadné sledování obsahu webových stránek.

Instalace

Doporučený přístup je naklonovat tento repozitář na lokální disk, vytvořit obraz virtuálního stroje a poté jej spustit v Dockeru.

```
git clone …
```

Pomocí `make` se interaktivně vytvoří přizpůsobená místní konfigurace (může být provedeno samostatně pomocí `make init`) a poté se vytvoří místní obraz virtuálního stroje (může být provedeno samostatně pomocí `make image`).

```
cd torscraper
make
```

Další konfiguraci lze provést v adresáři `config/app`. Především úprava proměnných prostředí aplikace v souboru `config/app/.env`. Prostředí jiných souvisejících aplikačních kontejnerů by nemělo být nutné.

Před spuštěním aplikace mohou být další zásuvné moduly zkopírovány do adresáře `plugin`. Po vytvoření místní konfigurace a přípravě pluginů by měl být projekt připraven k nasazení pomocí Dockeru. To lze provést jednoduše pomocí `docker-compose up -d`. Aplikace při prvním spuštění automaticky inicializuje databázi (platí i pro všechny dynamicky načítané pluginy).

Po spuštění programu by měly být přidány služby zásuvných modulů a poté aplikace restartována. Jak na to je popsáno v sekci registrace záznamů služeb.

Spuštění aplikace

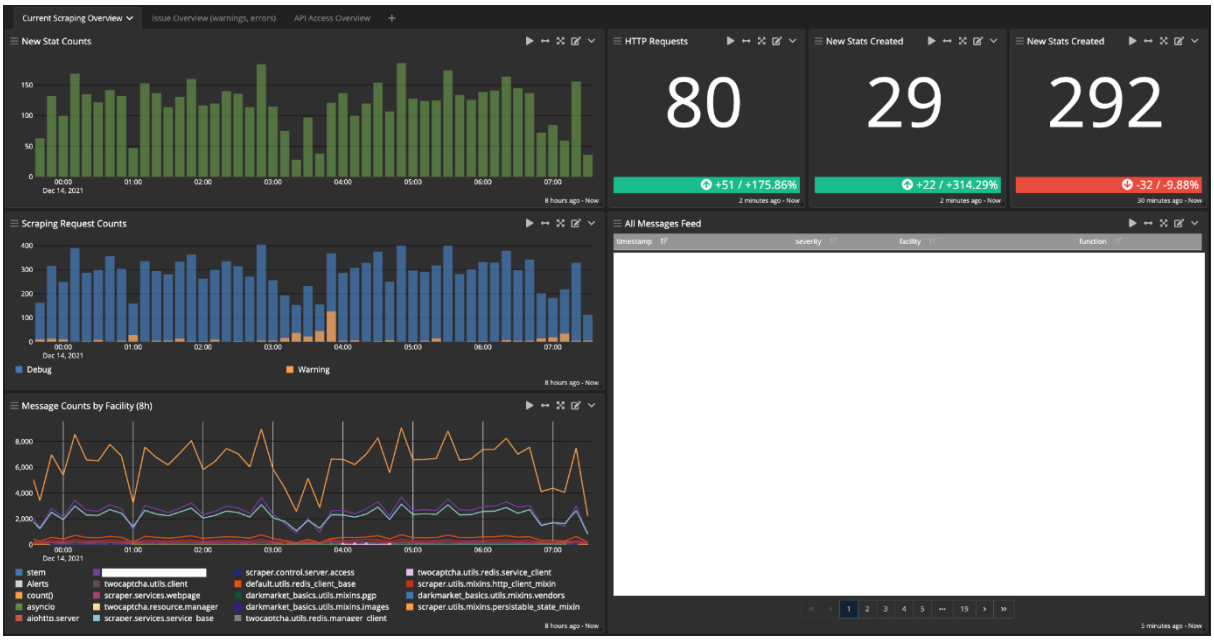
Nasazení celého souboru služeb pomocí Dockeru je přímočaré.

```
make all-up
# nebo
docker-compose up -d
```

Monitoring

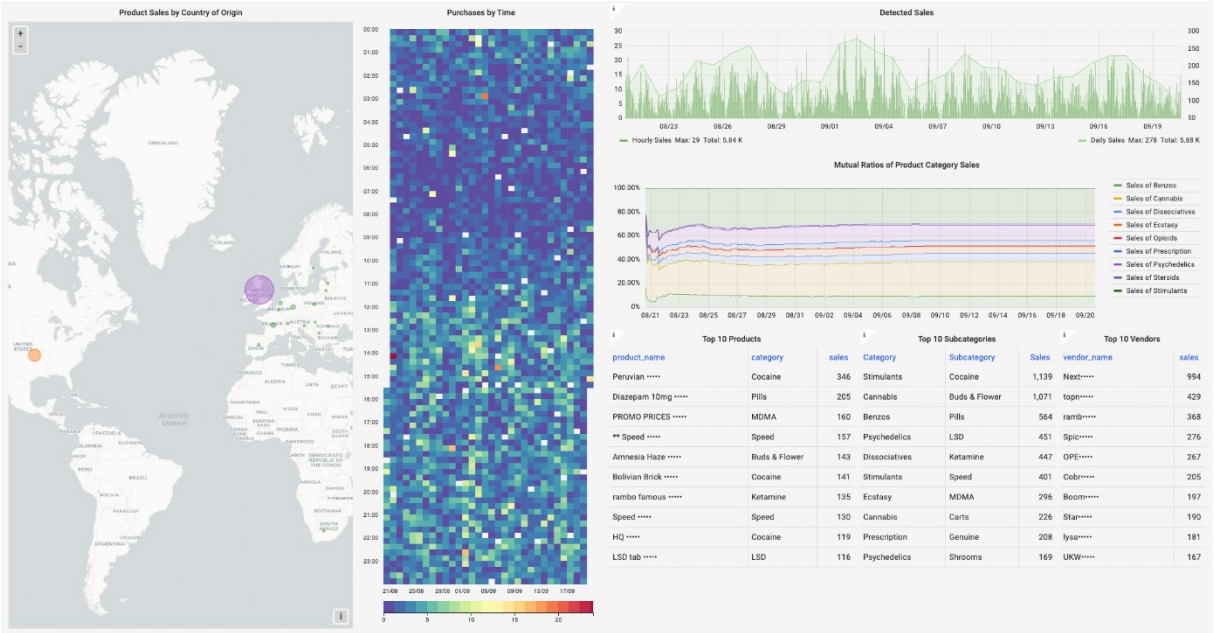
Aplikace umožňuje podrobné sledování téměř všech jejích akcí. Aby bylo možné tuto funkci povolit, musí být Graylog správně nakonfigurován (povolen jako obslužná rutina protokolování v `config/app/logging.yaml`) a nastaven (se spuštěnou službou a

příslušným balíčkem obsahu). Obsahový balíček pro Graylog s předkonfigurovanými vstupy a řídicími panely lze nalézt v adresáři `config/graylog`.



Totéž platí pro aplikaci Grafana. Protože se Grafana připojuje přímo k databázi, musí být spuštěná instance databáze přidána do webové aplikace Grafana jako nový datový zdroj. To lze provést prostřednictvím webového uživatelského rozhraní na výchozí adrese `http://127.0.0.1:8082/grafana/datasources/new`. Příslušné přístupové údaje naleznete v automaticky generovaném souboru `config/app/app.yaml` (při inicializaci pomocí make init). Různé předkonfigurované monitorovací dashboardy opět naleznete v odpovídajícím konfiguračním adresáři - `config/grafana`.

Příjemci podpory: ***Poskytovatel:***



Uživatelský manuál

Tato aplikace je primárně určena pro programátory, proto je hlavním referenčním bodem programátorská dokumentace. Aktuální kapitola stručně vysvětlí proces běhu aplikace a načítání rozšíření.

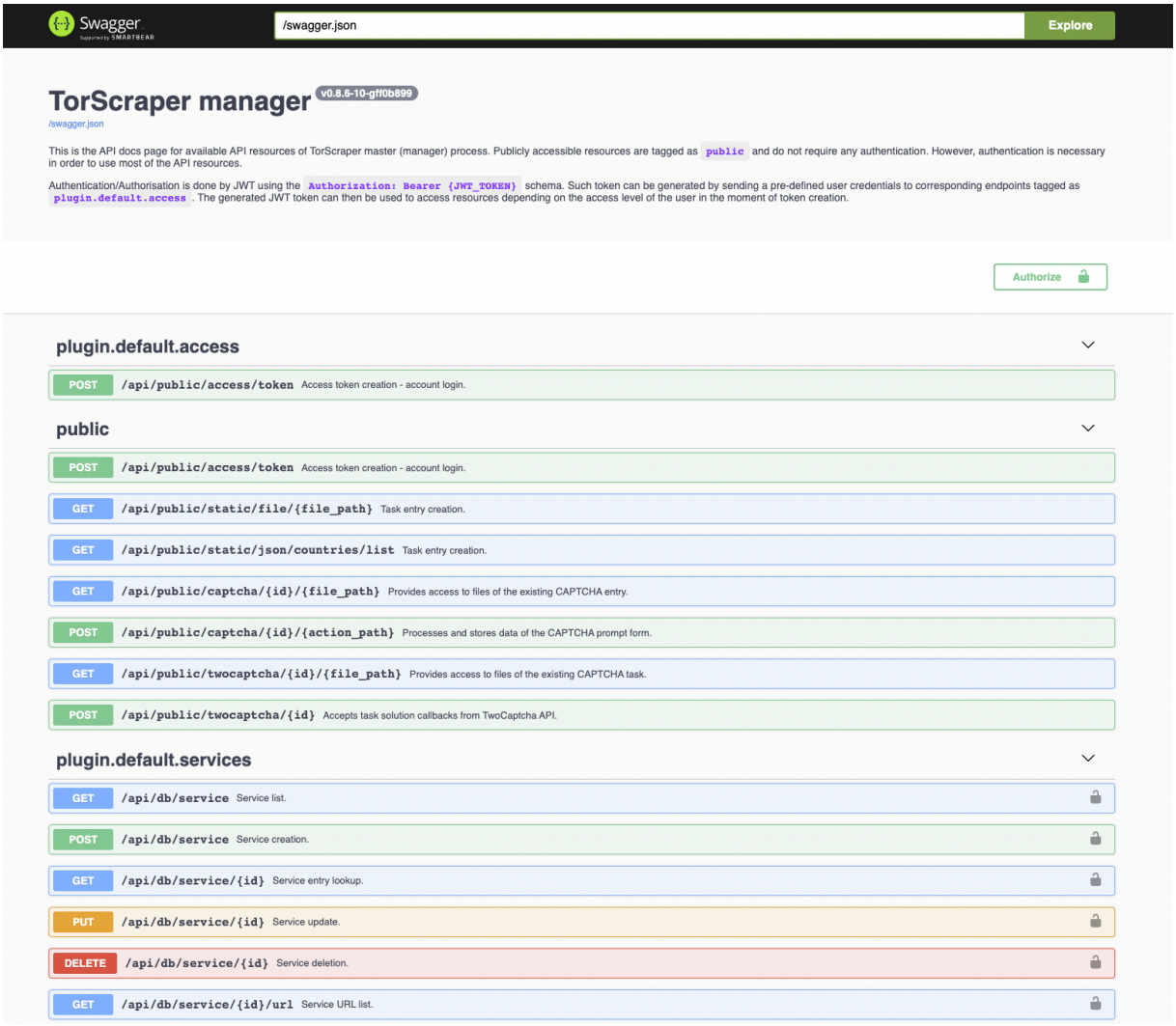
Runtime aplikace

Program nejprve spustí instanci správce, která spustí zásuvné moduly související s manažery (pro databázi, prostředky a nástroje). Poté, co je vše inicializováno, jsou načteny moduly servisních zásuvných modulů. Jejich odpovídající konfigurace jsou staženy z databáze a služby jsou konkretizovány. Pokud pro žádný z načtených servisních modulů není v databázi nalezena konfigurace služby, příslušný modul se vůbec nespustí!

Služby jsou spouštěny v samostatných procesech a jsou monitorovány instancí hlavního manažera. Další konfiguraci libovolné dané služby lze provést pomocí databázového pole `additional` u každého odpovídajícího záznamu služby (`Service`).

API aplikace

Programové rozhraní aplikace standardně naleznete ve výchozím stavu na adrese `http://127.0.0.1:8080`. Aplikaci lze ovládat z webové stránky pomocí webového uživatelského rozhraní Swagger přítomného na dané URL adrese (pokud není výslovně zakázáno v konfiguraci aplikace).



Vlastní služby

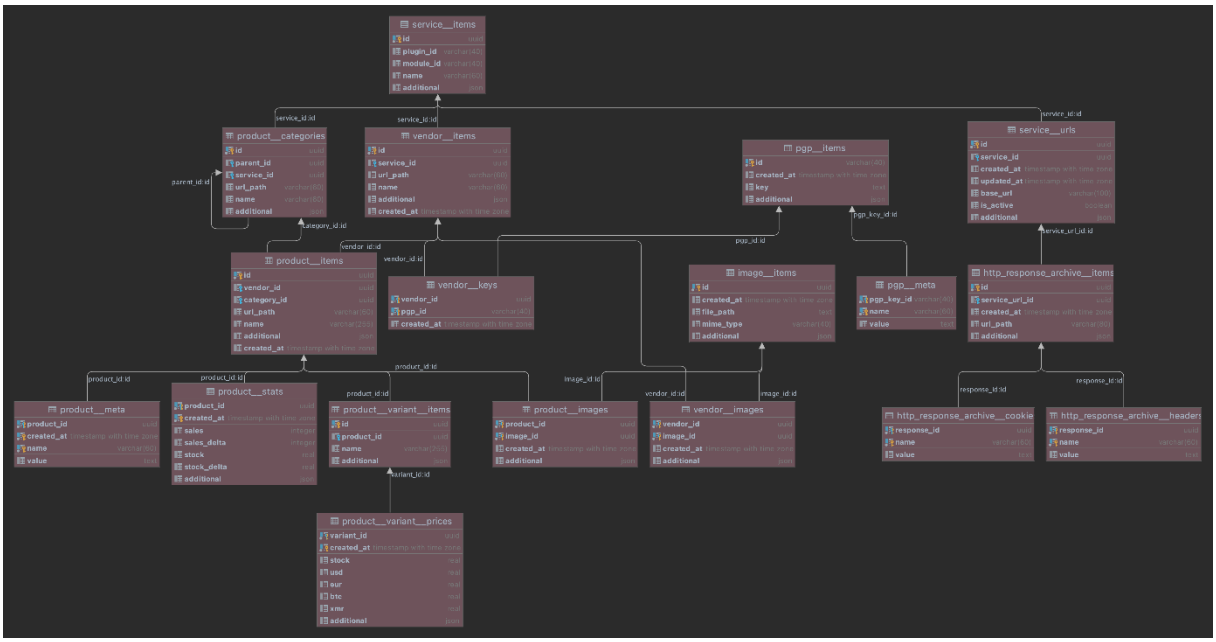
Vlastní služby lze snadno implementovat vytvořením podtříd poskytovaných tříd `scraper.services.CrawlerServiceBase` a `scraper.services.ScraperServiceBase` v odpovídajících složkách zdrojových souborů (`plugin`). Rozhraní výše uvedených základních tříd je dále popsáno v programátorské dokumentaci.

Programátorská dokumentace

Lze jej nalézt v adresáři `docs` po použití knihovny `pdoc3` ke generování ze zdrojových souborů aplikace.

```
pipenv run pdoc3 -o docs --html . --force --skip-errors
```

Schéma databáze je dynamicky definováno různými zahrnutými zdrojovými soubory databázových zásuvných modulů. Výsledné schéma při použití všech poskytnutých pluginů:



Užitečné akce

Tato kapitola zahrnuje různé užitečné akce s příklady a úryvky příkazů.

Konfigurace přístupu k API

Vytváření autorizovaných účtů

Nové účty API lze nakonfigurovat v `config/app/auth_users.yaml`. Tento konfigurační soubor očekává následující konfigurační strukturu:

```
users:
    - name: USER_NAME
      key: PRIVATE_KEY
      scopes:
          - SCOPE1
          - SCOPE2
          - ...
    - ...
```

Dostupné rozsahy uživatelského přístupu:

- `read` (globální)
- `write` (globální)
- `$MODULE_NAME:read` (v rámci celého modulu)

- `$MODULE_NAME:write` (v rámci celého modulu)
- `$MODULE_NAME/$RESOURCE_NAME:read` (přístup ke konkrétnímu zdroji modulu)
- `$MODULE_NAME/$RESOURCE_NAME:write` (přístup ke konkrétnímu zdroji modulu)
- `$MODULE_NAME/$RESOURCE_NAME[/$SCOPE1[/$SCOPE2[...]]]` (přístup ke konkrétní akci vybraného zdroje modulu)

Rozsahy lze nalézt a specifikovat v definicích operací prostředků pluginu:

```
class CustomResource(ResourceBase):
    @check_permissions([
        "read",                                  # accept global scope
        "custom_module:read",                    # accept module-wide sc
ope
        "custom_module/custom_resource:read",    # accept resource-wide
scope
        "custom_module/custom_resource/operation",  # accept operation-spec
ific scope
    ], comparison=match_any)
    def operation(self):
        pass
```

Přístupové tokeny lze poté vygenerovat odesláním požadavku POST do příslušného koncového bodu (klíč `PRIVATE_KEY` je klíč uložený v konfiguračním souboru uživatele):

```
APP_HOST="http://127.0.0.1:8080"

ACCESS_TOKEN=$(curl -s -X POST "$APP_HOST/api/public/access/token" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -d '{"user_key": "PRIVATE_KEY"}' \
    | python3 -c 'import sys, json; d=json.load(sys.stdin); print(d["token"
]) if "token" in d else print(d["reason"], file=sys.stderr)')
```

Vygenerované JWT se pak použijí ve všech požadavcích následujícím způsobem:

```
curl -s -X POST "$APP_HOST/api/..." \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $ACCESS_TOKEN" \
    -d '{...}'
```

Dočasné zakázání ověřování

Chcete-li zakázat ověřování na všech koncových bodech prostředků, přidejte položku `/*` do `control_server.auth.whitelist` v konfiguračním souboru `app.yaml` a restartujte program. Položka `control_server.auth.credentials_required` musí být nastavena na hodnotu no (nebo `false`).

Registrace záznamů služeb

Služby představují instance zásuvných modulů, které mají být spuštěny – každá služba představuje jednu instanci služby zásuvného modulu, kterou spustí manažerský program. Nejprve povolte neomezený přístup ke zdrojům webového rozhraní API pomocí následujícího příkazu (příkaz odkomentuje `/*` položku v konfiguraci `control_server.auth.whitelist` – je tam ve výchozím stavu):

```
sed -E -i "" "s|#- /\*|- /\*|g" config/app/app.yaml
```

Pomocí následujících příkazů vytvořte položky `Service` a `ServiceUrl` v databázi (použijte vlastní názvy modulů a pluginů). Použitím následující strukturu vytvořit servisní pluginy, názvy znamenají:

- /plugin
  - /plugin_package_name
    - /plugin
      - /SERVICE_MODULE_NAME
        - /services
          - /SERVICE_PLUGIN_NAME.py
      - /SERVICE_MODULE_NAME
        - /services
          - /SERVICE_PLUGIN_NAME.py
      - …
  - …

```
APP_HOST="http://127.0.0.1:8080"
SERVICE_MODULE="" # přidat název modulu služby, kterou si přejete povolit
SERVICE_PLUGIN="" # přidat název submodulu služby, kterou si přejete povoli
t

# vytvořit položku služby
SERVICE_ID=$(curl -s -X POST "$APP_HOST/api/db/service" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -d '{"name": "Service Name", "module_id": "'"$SERVICE_MODULE"'", "plugi
n_id": "'"$SERVICE_PLUGIN"'"}' \
    | python3 -c 'import sys, json; d=json.load(sys.stdin); print(d["servic
e"]["id"]) if "service" in d else print(d["reason"], file=sys.stderr)')

# vytvořit položky ServiceUrl propojené s dříve vytvořeným záznamem Service
curl -s -X POST "$APP_HOST/api/db/service/url" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
```

```
    -d '{"base_url": "http://service-base-url.onion", "service_id": "'"$SER
VICE_ID"'"}'
```

V případě lokálního nasazení, kde je nutné ruční řešení CAPTCHA:

```
# modify an already created Service record
curl -s -X PUT "$APP_HOST/api/db/service/$SERVICE_ID" \
    -H "accept: application/json" \
    -H "Content-Type: application/json" \
    -d '{"additional": {"captcha_solver": "manual"}}'
```

Nakonec zakažte neomezený přístup k rozhraní API:

```
sed -E -i "" "s|- /\*|#- /\*|g" config/app/app.yaml
```

Po přidání nových nebo úpravách položek služeb je nutné kontejner restartovat. To lze kdykoli snadno provést pomocí `docker-compose restart scraper`. Nakonfigurované služby budou automaticky spuštěny aplikací při jejím spuštění.