Práce s YouTube API a její vizualizace

Semestrální práce předmětu PP

Milan Horínek

13. prosince 2021

Obsah

1 Úvod

V posledním měsící došlo na sociální video platformě YouTube (*zkráceně také YT*) ke změně, která odranila viditelný počet záporných reakcí diváků na video, tzv. *disliků*. I přesto, že tato změna přinesla pozitiva pro provozovatele sítě z biznisového, tak zároveň spoustu uživatelů pobouřila, protože má negativní dopad i na *edukativní* či jinak užitečný vliv obsahu. Dle této metriky například divák mohl vědět, zda má smysl věnovat čas danému videu nebo jestli se jedná jen o ztrátu času. Užitečné bylo toto číslo zejména u návodů, které mohou být často zavádějící čí zmatečné a bez této metriky se uživatelům hůře určuje kredibilita jednotlivých videí.

Uživatelé sítě začali tuto situlaci řešit po svém a zároveň vytvářet obkliky. Jedním z vlivů této změny je větší frekvence nových komentářů u videí, ve kterých uživatelé vyjadřují svůj názor. S tím je spojena jedna z oblik, kterou uživatelé vymysleli a to, že že přidají komentář, která posléze slouží jako *dislike* tlačítko pro ostatní.

Účelem této práce by tedy mělo být zkusit implementovat některé z těchto oblik jakožto funkci ze strany tvůrce pomocí aplikace, která by mohla běžet neustále na serveru. YouTube sice zatím stále posílá tuto statistiku do datové vrstvy frontendu a tedy přes patřičné rozšíření do webového prohlížeče lze tuto metriku stále zobrazit veřejně, tak by backend měl brzi přestat tuto hodnotu posílat. Kde se však dí získat také je jejich API, kde sice ze strany veřejnosti není také jistota, že tento údaj zůstane neskrytý, avšak YouTube slibuje, že ze strany tvůrců bude tento údaj i nadále přístupný, a protože API nabízí přístup do účtu i na této úrovni, můžete tuto metodu z pohledu tvůrce považovat za spolehlivou.

1.1 Volba technologií

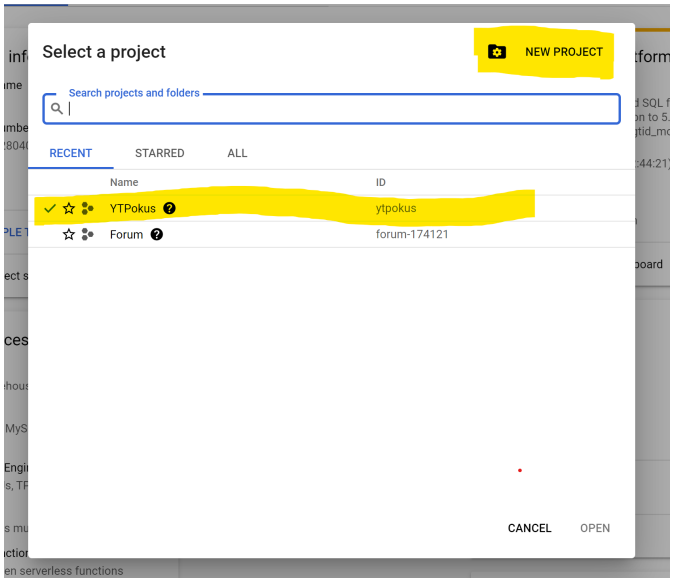
Naše aplikace bude obsahovat jak back-end tak front-end. Back-end aplikace bude postaven na jazyce **Python**, který se běžně nehodí pro produkční aplikace, protože většina exekucí v rámci něj je neefektivních, avšak jedná se o vhodné řešení pro demonstraci naší problematiky a zároveň jde jednoduše rozšířit pro případné další využití tohoto kódu. Front-end aplikace bude webová stránka, postavena na klasické trojici HTML pro markup značek, CSS pro základní styly a JavaScriptu pro veškeré kódování. S back-endem bude spojen skrze technologii WebSocket na bázi klient-server.

2 Popis technologií a požadavky

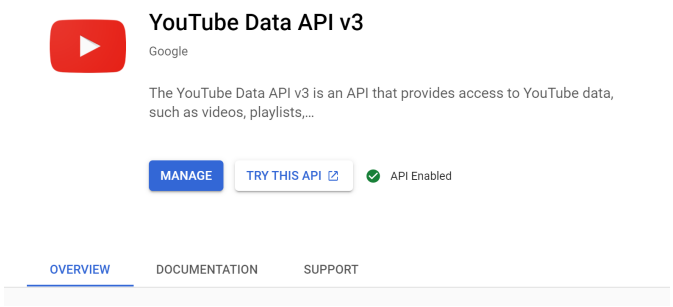
V první řadě se musíme seznámit s technologiemi, se kterými budeme pracovat, abychom znali jejich možnosti a věděli, jak je implementovat do našeho projektu. Stejně tak se v tomto kroce seznámíme s požadavky jednotlivých technologií a provedeme prvotní konfiguraci, kterou posléze budeme v projektu využívat.

2.1 Google Cloud

Dříve než se vrhneme na popis samotného YT API, tak potřebujeme znát šablonu, ze které vychází a prerekvizity, které musíme splnit. Toto API vychází z běžné šablony pro API, které Google využívá. Pokud chcete získat k některému z Google API přístup, musíte mít nejdříve zřízen u Googlu vývojářský účet a vytvořený projekt na platformě Google Cloud. Tato platforma nabízí zejména virtuální hostovací služby pro databáze, servery, atp. a mimo jiné také nabízí tvorbu přístupu k jednotlivým API. V administraci pro tuto platformu Google Cloud Console si tedy vytovoříme projekt, se kterým budeme následně pracovat (viz. obr. 1). Z vyhledávání v rámci této správy projektu si do něj přidáme API pojmenované **YoutTube Data API v3** (viz. obr. 2).



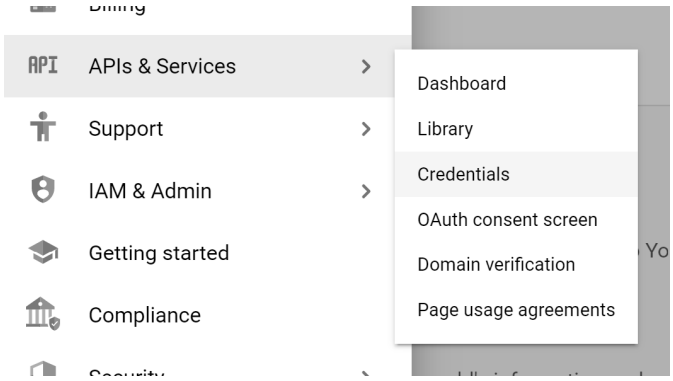
Obrázek 1: Přehled projektů v rámci Google Cloud Platform. *Označené:* Možnost vytvořit nový projekt a aktuální projekt, se kterým budeme pracovat.



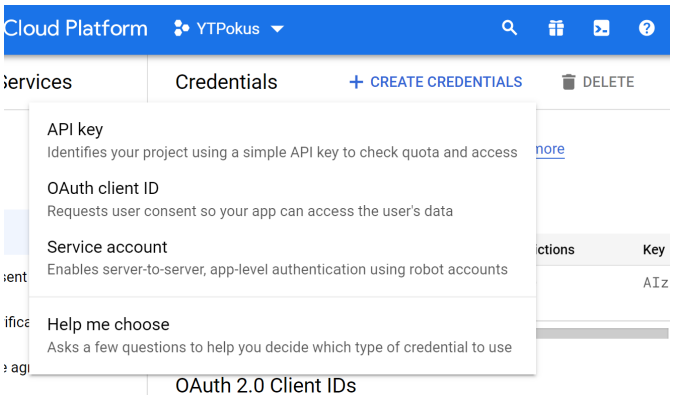
Obrázek 2: Nabídka API v rámci Google Cloud Console a možnost jeho přidání do projektu.

2.1.1 Klíče

Po přídání YT API do projektu potřebujeme vytvořit 2 (resp. 3) klíče. Prvním z nich bude **API klíč**, pomocí kterého se budeme na API dotazovat a identifikujeme se tak k němu jako náš projekt. Vzhledem k tomu, že se ale jedná o relativně veřejný údaj, můžeme klíč limitovat pro použití jen z konkrétních druhů zařízení. Nám zde tedy bude stačit jeden. Druhým klíčem bude **ID klienta**, které bude využívat aplikace při pro získání přístupu do API v rámci **autorizace** (*vysvětleno později*, 2.2). Tento klíč by měl být vždy specificky vytvořen pro každé zařízení, avšak tuto vlastnost nebudeme implementovat. Součástí **klíče klienta** je i jeho **soukromý klíč**, který nebudeme nikde zveřejňovat a aplikace se jím bude pouze prokazovat. Pro získání klíčů se lze odnavigovat po postranní navigaci rozhraní do podsekce *API - Credentials* (viz. obr. 3). Tlačítkem *Create Credentials* (obr. 4) poté můžeme přidat oba klíče. Pro vytvoření API klíče není potřeba žádný další vstup a pro vytvoření klíče zařízení bude rozhraní chtít vědět pouze *druh zařízení*. V sekci *Credentials* můžeme vidět ještě třetí položku *Service Accounts*, která slouží k vytvoření Google účtů pro potřeby projektu, avšak i přesto, že v běžných případech by nám tyto účty usnadnili práci, tak se spojení s YT API je nelze využít.



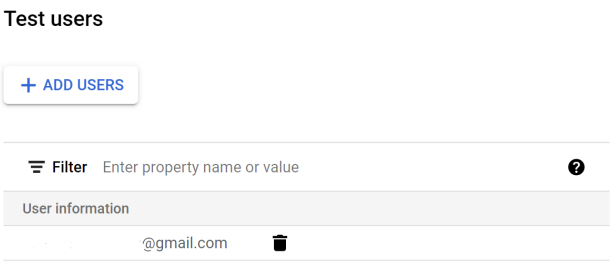
Obrázek 3: Správa ověřovacích údajů v navigaci Google Cloud Console.



Obrázek 4: Přidání klíčů pro přístup k API.

2.1.2 Testovací uživatel

Protože se aktuálně aplikace nachází ve stádiu *testování*, nelze se k ní ale přihlásit běžným Google účtem. Toto přihlášení potřebujeme, protože chceme získat přístup k YouTube kanálu, který je právě s některým z Google účtů spojen. Aplikaci však nikdy publikovat neplánujeme a fáze *testování* nám vyhovuje, protože pro publikaci by musela projít schvalovacím procesem. Aktuálně nám bude stačit přidat v záložce *OAuth consent screen* náš Google účet jakožto testovací účet, který poté budeme využívat pro přihlášení do aplikace (obr. 5).



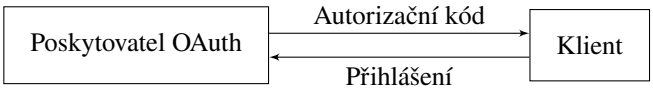
Obrázek 5: Testovací účty v záložce *OAuth consent screen*.

2.2 OAuth 2

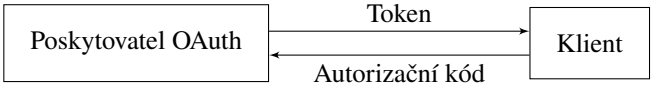
Ještě než přistoupíme k samotnému YT API, musíme absolvovat zmíněný proces autorizace. Ten je zde realizován pomocí technologie OAuth 2.0, která funguje ve 2 (resp. 3) krocích:

1. **Získání autorizačního kódu**
2. **Vyzvednutí tokenu**
3. **Obnovení tokenu**

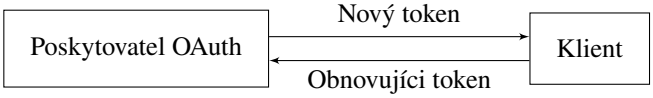
Hlavním prvkem této autorizace je **token**, který aplikace získá na bázi ověření uživatele u poskytovatele služby. Tento token bude poté vždy předán API společně s dotazem, aby byla ověřena totožnost uživatele a práva na danou operaci.



Obrázek 6: OAuth - Krok 1



Obrázek 7: OAuth - Krok 2



Obrázek 8: OAuth - Krok 3

2.2.1 Krok 1 - Získání autorizačního kódu

V prvním kroku autorizace potřebujeme od poskytovatele získat tzv. *autorizační kód*. Ten získáme tak, že odkážeme uživatele na autorizační URL:

```
https://accounts.google.com/o/oauth2/v2/auth
```

K této URL poté připojíme dále specifikované `GET` parametry. Google definuje více způsobů jak k jejich autorizaci přistupovat. My zde budeme využívat tzv. *přístup nainstalované aplikace*. Parametry jsou ukázány v tabulce 1. První z parametrů, který nás bude zajímat je `response_type`. Ten nám určuje výstup, který nám poskytoval odešle zpět. Jak jsme již řekli, námi požadovanou odpovědí bude *autorizační kód*. Do tohoto požadavku bude také vloženo **ID Klienta**, které jsme již diskutovali (2.1.1). Podle tohoto *ID* určí poskytovatel z jakého druhu naší aplikace požadavek pro autorizaci pochází. Důležitým parametrem pro nás bude tzv. *scope*, což je úroveň práv, kterou dostaneme k přihlášenému účtu. Tato práva definuje Google v YT API dokumentaci. Pro nás projekt budeme využívat následující úroveň:

```
https://www.googleapis.com/auth/youtube.force-ssl
```

Aplikace může požádat i o více úrovní najednou, což by bylo potřeba pokud bychom například na YouTube chtěli nahrávat videa. Pro zobrazení práv, které nám uživatel dal můžeme využít parametr `include_granted_scopes`, který nám vrátí jejich souhrn v rámci odpovědi poskytovatele. Parametr `state` poté slouží pro ověření zda odpověď pochází od skutečného serveru poskytovatele. Zde můžeme například vložit hash pro ověření nebo zpětný autorizační klíč pro klientský server. V našem

případě tuto ochranu vynecháme. `access_type` poté slouží k určení, jak bude token uložen. U nás tedy *offline*. Posledním parametrem je **přesměrovací adresa**. Ta má být běžně URI, na kterou server poskytovatele odešle *autorizační kód*. Může to být například server nebo by nám postačilo aby náš program poslouchal lokálně na nějakém portu, kam by tento klíč by odeslán. Protože však chceme tento proces, co nejvíce zjendodušit, můžeme využít speciální URI, která nám poslouží k tomu, že se autorizační klíč zobrazí uživateli po přihlášení a ten ho zkopíruje do našeho programu. Toto řešení je vhodné zejména pro autorizaci do serverových aplikací jako je ta naše, kdy k ní uživatel nemá jiný přístup jak vzdálený.

Parametr	Hodnota
`response_type`	`code`
`client_id`	*ID Klienta* (2.1.1)
`scope`	*Požadovaný scope*
`include_granted_scopes`	`true` / `false`
`state`	`pass-through value`
`access_type`	`offline`
`redirect_uri`	`urn:ietf:wg:oauth:2.0:oob`

Tabulka 1: Parametry autorizačního URL.

2.2.2 Krok 2 - Vyžádání tokenu

Pomocí získaného *autorizačního kódu* si nyní aplikace od serveru poskytovatele může vyžádat token. Ten lze získat pomocí `POST` dotazu na adresu:

```
https://oauth2.googleapis.com/token
```

Právě v tomto dotazu předáme v rámci parametrů (viz. tab. 2) serveru získaný **autorizační kód** (2.2.1), dále **ID klienta** společně s jeho **privátním klíčem** (viz. sekce 2.1.1). Stejně jako v předchozím kroku bude jedním z parametrů **přesměrovací adresa** se stejnou hodnotou, tedy v tomto případě nás nikam napřesměruje a posledním parametrwm je tzv. `grant_type`, tedy druh autorizace ze strany klienta. Zde se ověřujeme právě **autorizačním kódem** získaným v přechozím kroku.

Parametr	Hodnota
`code`	*Autorizační kód*
`grant_type`	`authorization_code` / `refresh_token`
`client_id`	*ID Klienta*
`client_secret`	*Privátní klíč klienta*
`redirect_uri`	`urn:ietf:wg:oauth:2.0:oob`
`refresh_token`	*Obnovující token*

Tabulka 2: Parametry URI pro vyžádání tokenu.

Pokud jsou všechny údaje poskytnuté klientem korektní, obržíme od serveru odpověď, jejíž struktura je je definovaná v dokumentaci a také stručně popsána v tabulce 3. Námi potřebný **token** je v tabulce označen právě jako `access_token` (tedy *přístupový token*) a má určitou **platnost**. Ta je většinou 3600 sekund, tedy jednu hodinu. Po skončení této doby je **přístupový token** neplatný, avšak lze i po uplynutí této doby získat nový pomocí **obnovujícího tokenu**, který je také jedním z parametrů odpovědi od serveru. Postup obnovy tokenu bude vysvětlen v následujícím kroku (2.2.3). Po tomto kroku již můžeme komunikovat s API, protože máme platný token.

Parametr	Význam
`access_token`	Přístupový token
`expires_in`	Doba platnosti tokenu (sekundy)
`refresh_token`	Obnovující token
`token_type`	Druh tokenu

Tabulka 3: Parametry parametry odpovědi serveru na úspěšné vyžádání tokenu.

2.2.3 Krok 3 - Obnova tokenu

Jak bylo vysvětleno v předešlém kroku, tak **přístupový token** má danou platnost a po jejím uplnutí je potřeba projít procedurou obnovení (obr. 8). K tomu využijeme získaný **obnovující token** a dotážeme se na stejnou adresu jako v přechozím kroku (viz. tab. 2), jen s tím, že změníme parametr `grant_type` na hodnotu `refresh_token` a právě ten přidáme mezi parametry namísto *autorizačního kódu*. Zbytek parametrů ponecháme. Odpověď serveru bude také shodná s předchozí. V rámci ní získáme nový **přístupový token** společně s novým **obnovujícím tokenem**. Po vypršení přístupového tokenu bude potřeba tento krok zopakovat. Obnovující token se nemění!

2.3 YouTube API

Datové API YT nabízí komplexní možnosti práce s platformou. Zde bude vypsáno pouze několik z nich, které budeme potřebovat při zpracovávání našeho programu. API se dělí do různých sekcí podle toho, jakou informaci chceme z API získat případně do ní poslat. Jednotlivé úkony také využívají různé druhy HTTP dotazů. Konkrétně se tu setkáme s následujícími druhy dotazu a jejich příklady:

- `GET` - Získání informací z API
- `PUT` - Zápis do API
- `POST` - Nahrání videa nebo náhledového obrázku
- `DELETE` - Smazání videa

API je uloženo na následující adrese, za kterou bude následovat identifikátor konkrétní sekce, ke které chceme přistupovat:

`https://youtube.googleapis.com/youtube/v3/` + *sekce*

Pro nás budou důležité následující sekce API:

Sekce	Účel
`search`	Možnost vyhledat videa, kanály, nebo playlisty ze zadaných parametrů
`videos`	Tato sekce reprezentuje YT video a veškeré jeho možnosti
`comments`	Obsahuje informace o konkrétních komentářích pod videi
`commentThreads`	Obsahuje inforamce o komentáři a všech jeho odpovědích

Tabulka 4: Sekce YT API.

Každý z dotazů má různé parametry. Zde budou vysvětleny všechny najednou a jejich konkrétní význam budeme diskutovat při implementaci, protože dotazy mezi sebou mohou mít jistou souvislost.

Parametr	Význam
`part`	Seznam parametrů, které chceme od API získat nebo do něj zapsat
`order`	Řazení dle
`mine`	Zobrazit pro vlastní kanál
`maxResults`	Maximální počet výsledlů (*max. 50*)
`channelId`	ID kanálu
`id`	ID videa

Tabulka 5: Parametry dotazů pro YT API.

Seznam parametrů v rámci položky `part` poté najdeme v odpovědi API. Tento rámec nemusí představovat pouze odpověď API, ale také vkládaná data. Datový rámec je popsán v tab. 6.

Parametr	Význam
`id`	Identifikátory
`snippet`	Základní informace, má vlastní strukturu dle sekce
`replies`	Odpovědi
`statistics`	Statistické a analytické údaje
`contentDetails`	Informace o obsahu

Tabulka 6: Datový rámec pro zprávu z nebo do YT API.

Pro přístup do API má účet v *bezplatné verzi* k dispozici 10 000 přístupových jednotek denně. Každý z dotazů na API má svou vlastní cenu. Většina běžných dotazů má cenu 1 jednotku, zápisy pomocí metody `PUT` poté běžně stojí 50 jednotek, vyhledávání 100 jednotek a například nahrání videa 1800 jednotek.A

3 Řešení

3.1 Back-end

Program je seskládaný z několika dalších modulů, kdy každý z nich řeší určitou problematiku. První z nich například poskytuje veškeré funkce potřebné pro autorizaci uživatele výše popsanou technologií OAuth2 (sekce 2.2).

3.1.1 Autorizace

Autorizačním modulem je soubor `auth.py`, v rámci kterého máme definované jednotlivé URI adresy pro práci s autorizační technologií Googlu a zároveň zde definujeme třídu `AuthClient`, která bude představovat objekt autorizace klienta a bude mít v rámci sebe uložené hodnoty jako **ID klienta**, **privátní klíč klienta** a také jeho získaný **token**, který bude v podobné podobě jako JSON objekt, jenž dostaneme od serveru. Objekt v sobě obsahuje veškeré informace ukázané v tab. 3.

Program budeme chtít navrhnout tak, že **token** si uložíme do souboru `token.json`, protože Python nenabízí lokální ukládání dat jako jiné jazyky. Pokud by s programem měl pracovat koncový uživatel, nejednalo by se o bezpečné řešení, avšak tento program je určený pro běh na serveru, kde bezpečnost nebude problémem. O načetení a zápis tokenu do souboru se starají funkce `save_token` a `load_token`.

První krok - autorizace Tento krok obstarává funkce `show_auth_code_uri`, která vypíše do konzole odkaz, na který uživatle klikne a proběhne proces popsaný v sekci 2.2.1, kdy uživatel se přihlásí přes bránu Googlu svým účtem, který je spojen YT kanálem, který pro běh této aplikace chce využívat. Na další straně vybere tento daný YT kanál a zobrazí se mu **autorizační kód**, který zkopíruje a vloží do konzole. Následně ho potvrdí klávesou *Enter*.

Druhý krok - vyžádání tokenu Se získaným **autorizačním kódem** můžeme nyní požádat o **token**. Jak bylo vysvětleno v sekci 2.2.2, jedná se o pouhý `POST` dotaz s JSON odpovědí. Tento proces zašťituje funkce `request_token`, jejíž vstupem je *autorizační kód*. Vzhledem k tomu, že refresh tokenu je stejný proces, bude velice podobná i funkce `refresh_token`, která na bázi obnovující tokenu získá nový přístupový token, stávající token vymění za něj a uloží společně s ostatními daty do souboru.

Zaštiťující funkce Protože obě funkce se volají hned po sobě a zároveň pouze v případě, že program nemá uložený token nebo má token neplatný, tak je zahrneme do nadřazené funkce `auth_user`, která bude zavolána právě v těchto případech a zajistí nejen volání obou funkcí, ale zároveň i další funkcionalitu jako výpis do konzole nebo uložení tokenu.

Validace přístupového tokenu Vzhledem k tomu, že programová smyčka může být spuštěna v libovolný čas, tak nemáme přesný důkaz o stáří aktuálního přístupového tokenu, pouze relativní informaci a době expirace. Proto potřebujeme možnost, jak zjistit, že aktuální přístupový token je stále platný. K tomu slouží funkce `validate_token`, která udělá dotaz na adresu `https://www.googleapis.com/oauth2/v3/tokeninfo`. Ta nám dá aktualizované informace o stavu tokenu a to zejména za jak dlouho vyprší. Pokud je tato doba kratší jak 100 sekund nebo informace o tokenu není podána, funkce ho vyhodnotí jako neplatný. V dalších situlacích naopak jako platný.

3.1.2 YT API

Další modul se stará o komunikaci s YT API. Došlo zde k implementaci nejen funkcí nutných pro aktuální program, ale i další pomocné, které by mohlo být potřeba využít v některých případech. Tento modul představuje soubor `yt_api.py` a obsahuje jak přístupovou URL k API tak také třídu `YTApi`, jejíž instanci budeme využívat právě k přístupu k němu. Vstupem konstruktoru je instance autorizačního klienta (3.1.1) a **API klíč**.

HTTP dotazy Pro vytváření `GET` dotazů má třída funkci `query`, která přebírá požadavek, což je většinou jedna ze sekcí (4) API, případně spojená s jednou z jejích akcí. Dále také přebírá parametry dotazu. K těm funkce přidá ještě **API klíč**, což je povinný parametr pro všechny API dotazy a takto ho nemusíme v každém z nich opakovat. Dále také definuje hlavničku dotazu, ve které je významným parametrem *autorizace*, která je zde typu *Bearer*, což vychází z přijetého typu tokenu (viz. tab. 3). K tomuto typu je poté přidán samotný token a slouží k ověření identity. Bez tohoto hlavičkového parametru by nám API nevydalo určité informace nebo neprovedlo specifické úkony. Pro `PUT` dotaz poté slouží funkce `apiPut`. Ta přidává jeden vstupní parametr a to `data`, což je datový rámec (viz. tab. 6), který bude odeslán do API. Pro tyto potřeby má také funkce ještě upravenou hlavičku.

Pomocné funkce Pro zobrazení videí daného kanálu lze využít funkci `showChannelVideos`, která vyžaduje definovat *ID kanálu* pro zobrazení. Pokud chceme zobrazit detaily kanálu, ke kterému jsme přihlášeni, využijeme funkci `myChannelInfo`. Ta využívá právě parametru `mine` a dotazu (resp. *sekce*, 4) `channels`.

ID komentáře Další pomocnou funkcí je `getCommentsOnVideo`, která se dotáže na komentáře pro dané video. Tu potřebujeme, protože pokud chceme zapisovat aktuální počet *disliků* do komentáře, potřebujeme znát *ID* tohoto konkrétního komentáře. To se bohužel z webového rozhraní služby nedá získat. Proto potřebujeme právě tuto funkci, v rámci které zjistíme *ID* daného komentáře.

Načtení komentáře Pokud již známe *ID komentáře*, můžeme stáhnout veškeré jeho detaily pomocí funkce `getCommentByID`. Aktuálně není tato funkce implementována nikde jinde v kódu a slouží pouze pro případné ověření, zda došlo k vybrání právného ID komentáře, jak bylo popsáno výše (3.1.2).

Načtení statistik videa Pomocí funkce `getVideoStatistics` lze pomocí `GET` metody z API vytáhnout parametr `statistics` ze sekce video (tab. 4) pro určité *ID Videa*. Toto ID lze nalézt v rámci parametru `v` URL videa.

Aktualizace komentáře Protože jsme již získali *ID komentáře*, můžeme provést jeho změnu (resp. editaci) a to tak, že do API pošleme parametr `snippet` s objektem `textOriginal`, který v sobě bude obsahovat nový text. O tuto změnu se stará funkce `updateComment`. Změna se na komentáři projeví stejně jako kdyby uživatel editoval komentář přímo z UI YouTube. Pokud předem nebyl editovaný, zobrazí se u něj ozančení *editováno*. Uživatel může editovat pouze vlastní komentáře, tedy v tomto případě můžeme měnit pouze ty vytvořené kanálem, který je do programu přihlášený.

Změna názvu videa Další z akcí, kterou jsme chtěli provést je *změna názvu videa*. To lze pomocí sekce API `videos` a zapsaným parametrem `snippet`. Zde však přichází problém, protože API si v tomto případě nepřevezme pouze změny z tohoto parametru, ale přepíše jím veškerá data. Proto potřebujeme ještě další pomocnou funkci a to `getVideoData`, kde stáhneme

celý `snippet`, ten poté v rámci funkce `changeVideoTitle`, která slouží ke změně názvu videa, změníme v tomto parametru pouze objekt *názvu videa*. Pokud bychom v parametru nechali samotný název videa a poslali ho do API, zbytek nastavení videa by se smazal. Zároveň parametr obsahuje i povinné hodnoty, které si tímto způsobem zjistíme.

3.1.3 WebSocket komunikace

V rámci komunikace přes WebSocket jsem zvolil jako přenosovou zprávu JSON objekt, který má následující strukturu (*hodnoty jsou pouze názorné*):

```
{
    state: "action",
    type: "run",
    holdTime: 5
}
```

Parametr `state` v rámci objektu určuje aktuální stav, ve kterém se program nachází a vizualizace dle něj bude vědět, co zobrazit. Veškeré stavy vizualizace jsou tedy statické a na bázi komunikace mezi nimi bude přepínat. Definované jsou následující stavy a jejich významy:

- `run` - Programová smyčka spuštěna
- `loadVids` - Načten seznam sledovaných videí z interní databáze
- `stats` - Načteny statistiky pro video z YT API
- `cacheSave` - Uložení načtených statistik videa do lokální DB
- `action` - Jedna z akcí pro video, resp. načtené statistiky a jejich zpracování
- `hold` - Programová smyčka pozastavena

Stav `hold` má u sebe ještě parametr `holdTime`, který udává zbývající čas, po který bude programová smyčka pozastavena. Stejně tak má podobný parametr i stav `action` a to `type`, který udává o jakou akci prováděnou s nahranými daty se jedná.

3.1.4 Lokální databáze

SQLite databáze bude mít 2 základní tabulky, kdy jedna z nich (`tracked_videos`) obsahuje seznam sledovaných videí (viz. tab. 7). Z tohoto seznamu si program bude brát jednotlivá videa, pro která bude stahovat statistiky z API a provede s nimi akce, které jsou v záznamu tabulky pro dané video aktivované. Viz. sloupce `action_`. Aktuálně umí program 2 akce a to **zápis disliků do komentáře** (3.1.2) nebo **zápis disliků do názvu videa** (3.1.2). Pro druhou akci má tabulka ještě konfigurační sloupec `conf_title_format`, ve kterém je uložený formát tohoto titulku. Pro akci komentáře je zde zase přítomna konfigurace ID komentáře, které lze získat metodou popsanou v 3.1.2. Tato tabulka tedy slouží pro konfiguraci programu a jsou v ní definována videa, pro které bude program fungovat.

Druhá tabulka `video_cahce` slouží k lokálnímu ukládání načtených statistik videí z API (par. v tab. 8). Toto můžeme využívat jak pokud bychom chtěli jen lokálně zjistit poslední statistiky videí nebo abychom zjistili, zda se naše sledovaná hodnota změnila. Pokud by se od přechozího dotazu nezměnila, nemusíme odesílat změnu zpět do API a plýtvat tak *přístupovými jednotkami*.

Název	Datový typ	Klíč	Význam
`videoID`	STRING	Primární	ID videa
`action_comment`	BOOLEAN		Indikátor akce úpravy komentáře
`action_title`	BOOLEAN		Indikátor akce úpravy názvu videa
`conf_title_format`	TEXT		Formát pro titulek videa při jeho změně
`conf_comment_id`	TEXT		ID komentáře pro změnu

Tabulka 7: Struktura tabulky pro sledovaná videa.

Název	Datový typ	Klíč	Význam
`videoID`	STRING	Primární, Cizí	ID videa
`data`	JSON		Stažená data z API
`updated`	DATETIME		Čas zápisu informace

Tabulka 8: Struktura tabulky pro chache statistik videí.

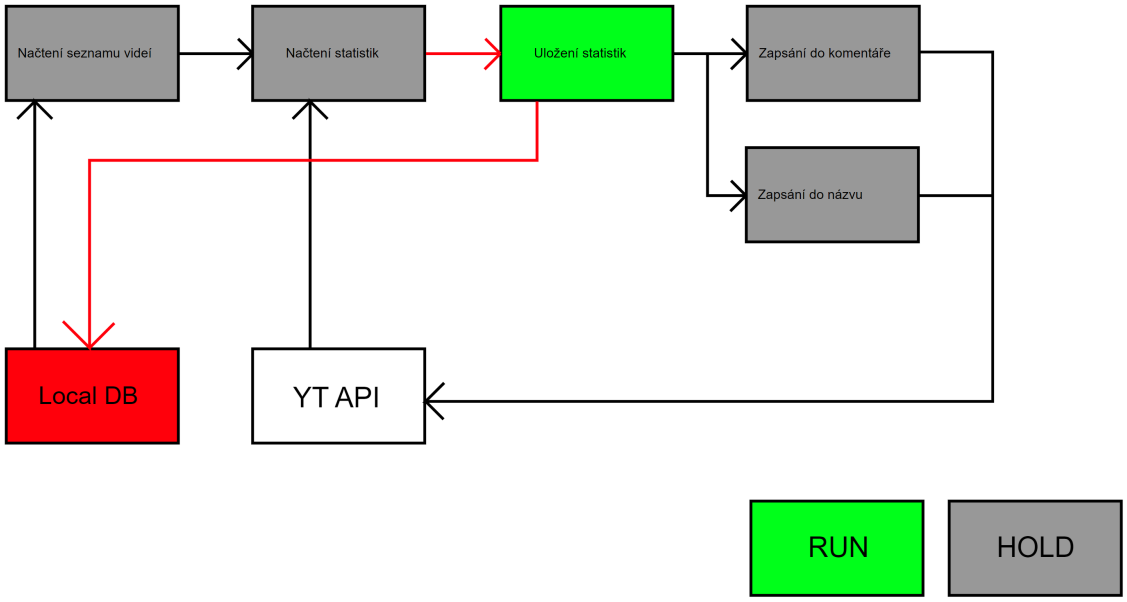
3.1.5 Programová smyčka

Před spuštěním programové smyčky proběhne nejdříve autorizace uživatele dle popsané metody (sekce 2.2). Pokud máme uložený token, načteme si ho a validujeme (viz. 3.1.1). V případě, že token neprojde validací, obnovíme ho pomocí obnovujícího tokenu. Pokud nemáme uložený token nebo je neplatný, vyzveme uživatele k autorizaci v bráně Googlu (popsané v 3.1.1 a 3.1.1). V případě, že by ani jendou cestou program nezískal *přístupový token* uživatele, dojde k jeho ukončení a informování. Pokud se autorizace podaří, otevře se WebSocket na daném portu (*zde pro test byl využit port 63344*). Program bude neustále poslouchat na tomto portu a při připojení klienta na WebSocket se spustí právě **programová smyčka**.

Ta si před každým jejím spuštěním zkontroluje validitu tokenu a pokud není validní, opět dojde k jeho obnovení. Na začátku smyčky odešleme klientovy datový rámec se stavem `run`. Poté následuje načtenmí sledovaných videí z databáze (viz. 3.1.4). Při dokončení této aktivity dojde k odeslání stavu `loadVids`. Zde jsme tedy získali seznam videí, aktuálně budeme projíždět každé z nich a nejdříve pro něj stáhneme z API statistiky (3.1.2). V tomto úseku se odešle stav `stats` a uložíme stažené statistiky do databáze, což je doprovázeno jiným stavem a to `cacheSave`. Tomuto následují akce prováděné se statistikami z jednotlivých videí. V případě zápisu do komentáře (3.1.2) nejdříve dojde k formátování textu dle šablony z databáze. Poté tento text nahrajeme zpět do API (3.1.2) a odešleme aktivitu `comment`. Stejné to bude i v případě změny titulku videa, jen v tom případě se odešle aktivita `title`.

3.2 Front-end

Při načtení vizualizace dojde k připojení na WebSocket programu (3.1.3). Samotná vizualizace je nakreslená jako SVG obrázek (9), který je později modifikován dle stavů přicházejících ze serveru. Jednotlivé bloky v rámci vizualizace představují buďto funkce nebo rozhraní. Šedá barva v diagramu představuje neaktivní bloky. Zelená naopak aktivní bloky a červená aktivní bloky, do kterých se zapisuje. Pokud je nějaký blok nečinný, ale aktivní, je vyznačen bílou barvou. Podobný systém využívají i šipky mezi bloky, kdy aktivní jsou označený červenou barvou. Vyjímku tvoří bloky "RUN"a "HOLD". Ty mají oba vlastní specifikou barvu. Blok *HOLD* navíc ještě mění svůj vnitřní text v závisloti na zbývajícím čase prodlevy programu.



Obrázek 9: Grafika vizualizace