TEHNIČKO VELEUČILIŠTE U ZAGREBU

STRUČNI STUDIJ INFORMATIKE

Antonio Mrkoci

Međuprogram za upravljanje i komunikaciju između RTSP poslužitelja i uređivača video prijenosa

ZAVRŠNI RAD br. 3648

Zagreb, srpanj, 2022.

TEHNIČKO VELEUČILIŠTE U ZAGREBU

STRUČNI STUDIJ INFORMATIKE

Antonio Mrkoci

JMBAG: 0246087629

Međuprogram za upravljanje i komunikaciju između RTSP poslužitelja i uređivača video prijenosa

ZAVRŠNI RAD br. 3648

Zagreb, srpanj, 2022.

Sažetak

Ovim radom objasnit će se proces obavljanja streama kombinacijom poslužitelja za primanje prijenosa također poznat pod imenom RTSP Simple Server, čija kratica RTSP predstavlja „Real time streaming protocol“ te besplatnog uređivača prijenosa otvorenog koda OBS Studio. Za početak, objasnit će se teoretske potrepštine neophodne za korištenje ove usluge, postavke poslužitelja i uređivača video prijenosa, popis potrebnih biblioteka pri izradi rada te detaljan opis programskog jezika te naredbi u istoimenom radu.

Tehnologije korištene u ovom radu su programski jezik node.js, glavni programski jezik rabljen unutar svrhe ovog rada, a u sklopu samog node.js također su i korištene biblioteke primjerice Axios – bibliotečki nastavak koji služi za pružanje mogućnosti slanja API zahtjeva poslužitelju, Socket.io – također bibliotečki nastavak korišten za osposobljavanje komunikacije između klijenta te njemu dostupnog poslužitelja te Express – posljednji bibliotečki nastavak rabljen u ovom radu čija je uloga posluživanje klijentske HTML aplikacije. Zatim korišten je Visual Studio Code čija je funkcionalnost bila namijenjena pisanju te uređivanju samih kodova te naravno samoj provjeri sintaksnog zapisa koda.

Opisan je proces obavljanja prijenosa uživo, engleski zapisano kao „Live stream“, novodolazeće grane industrijskog prijenosa sadržaja uporabom interneta dostupnog široj publici jednostavnim procesom uspostavljanja prijenosa rabeći gore spomenutu tehnologiju.

Ključne riječi: node.js, OBS Studio, RTSP Simple Server, live stream

Sadržaj

Popis oznaka i kratica

OBS - Open Broadcaster Software

RTSP - Real Time Streaming Protocol

RTMP - Real Time Messaging Protocol

HTTP - HyperText Transfer Protocol

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

API - Application Programming Interface

Popis slika

1. Uvod

U današnje vrijeme pristupačni su mnogobrojni profesionalni programi s funkcionalnošću te pruženom uslugom prijenosa audio i video zapisa. Naime, istoimeni profesionalni programi imaju veliku manu sve istaknutijom napretkom tehnologije, a ta mana je prikazana u obliku visokih cijena koje postaju sve teže dostižne prosječnom korisniku. Srećom, plaćanje je apsolutno moguće izbjeći zahvaljujući rastu te dolasku mnogobrojnih besplatnih alternativnih solucija otvorenog koda. Samim time što su alternativne solucije otvorenog koda znači da sam programer omogućava besplatno korištenje svojeg koda koji također pruža korisnicima mogućnost proširenja i napretka usluge te prilagođavanja programa svojim potrebama.

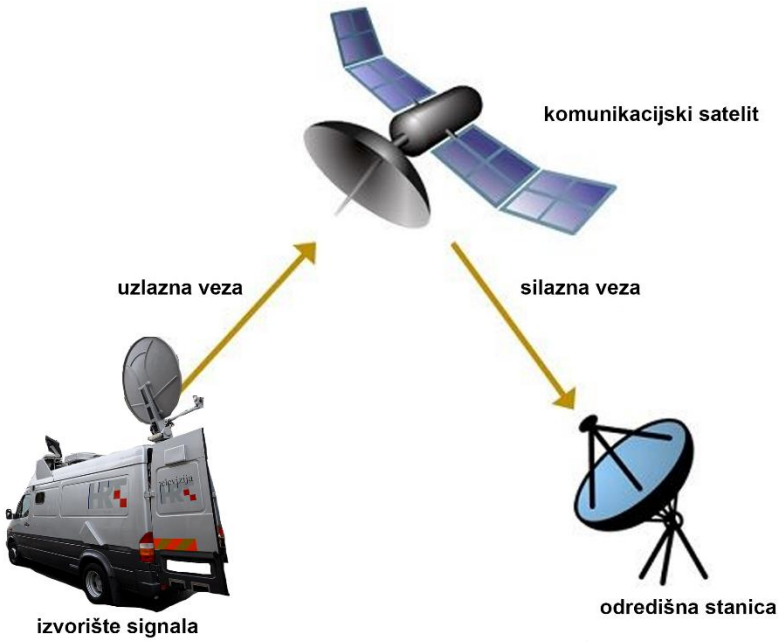
Ovaj rad će replicirati značajku primanja prijenosa programa koju popularni uređivači prijenosa skrivaju iza plaćanja pomoću nekolicine programa otvorenog koda te vlastitog programa. Povodom toga, ovaj rad će se oslanjati isključivo na besplatne programe otvorenog koda kako bi se dokazalo da cijena nije presudan faktor pri omogućavanju korisnicima pristupu potrebnim funkcionalnostima neophodnim za izvedbu prijenosa uživo. Za obavljanje prijenosa uživo korišten je OBS Studio. OBS Studio je program otvorenog koda koji je zahvaljujući svojoj širokoj uspio postati industrijski standard.

Za značajku primanja prijenosa je birano između tri programa: FFmpeg, Simple Realtime Server i RTSP Simple Server. FFmpeg je vrlo moćan alat za obradu razne multimedije, između ostalog koriste ga VLC media player i Open Broadcasting Software Studio. Njegov nedostatak je taj što je potrebno mnogo znanja da bi se mogao optimizirano koristiti. Nedostatak Simple Realtime Servera je taj što je razvijen u Kini što samo po sebi uzrokuje probleme s čitanjem dijela dokumentacije pošto je pisan na mandarinskom. Zbog navedenih razloga RTSP Simple Server je bio očiti izbor te ujedno ima pregršt dokumentacije dostupne na engleskom jeziku.

2. Tehnika rada s video prijenosima

U televizijskom svijetu se koriste dva načina za prijenos signala, satelitski i internetski. S obzirom na to da većina televizijskih organizacija nema dovoljno resursa da pokrije svaki prijenos s vlastitim ljudima, alternativa su im produkcijske kuće koje se mogu unajmljivati po potrebi. Kvaliteta produkcijskih kuća varira ali sukladno s njom i cijena pa ovisno o važnosti prijenosa nije nužno uvijek odabrati najbolje. Produkcijske kuće čija je usluga kvalitetnija najčešće imaju kamion koji sadrži prostor za rad režisera i audio inženjera, te na krovu satelit pomoću kojeg mogu prenositi signal. S druge strane produkcijske kuće niže kvalitete najčešće postave svu opremu u neposrednoj blizini kamera i prijenos rade pomoću interneta koristeći rtmp protokol. Za slanje podataka preko rtmp protokola televizija mora imati mogućnost primiti taj signal što može napraviti pomoću programa npr. Livestream Studio ili ovog rada.

2.1. Satelitski prijenos



Slika 1 - putanja signala u satelitskom prijenosu

Satelitski prijenos je staromodan, ali i dan danas uporabljivi oblik prijenosa. Ovakav oblik prijenosa omogućava korisniku da pristupi prijenosu spajanjem svojeg uređaja sa satelitom. Ulazna veza uspostavlja se između korisnika i satelita putem kablova – uređaj će se nalaziti kod korisnika dok će se antena najčešće nalaziti montirana na vrhu kamiona na kojem će se nalaziti montirani komunikacijski satelit.(Slika 1) Kamioni sa satelitom su dostupni za prodaju svakodnevnim korisnicima s relativno pristupačnom cijenom s obzirom na to da uslugu koju pružaju uz iščekivanu određenu količinu znanja koja dozvoljava korisniku uporabu spomenutih satelita. Kada je riječ o komunikacijskim satelitima, nalaze se na visini od 35900 kilometara. Riječ je o geostacionarnoj poziciji pošto moraju održavati konzistentnu udaljenost od zemlje kako bi funkcionirali ispravno. Sama geostacionarna udaljenost je dakle riječ koja opisuje objekt koji se nalazi na kontinuiranoj udaljenosti naspram zemlje unutar njezine orbite. Kako bi se uspješno

obavio prijenos signala, koristi se takozvani opseg „**K_u**“. Opseg „**K_u**“ indicira radnu frekvenciju čiji se valni interval nalazi između 12,000 i 18,000Hz, odnosno radi lakšeg zapisa 12GHz i 18GHz. Sam naziv, „**K_u**“, potječe iz njemačkog jezika te znači „Kurz-unten“ što bi u prijevodu predstavljalo donji dio „K“ opsega čija namjena varira od komunikacijskih satelita, izmjena informacija s međunarodnom svemirskom postajom pa sve do policijskih radara za brzinu, odnosno reprezentira nižu vrijednost samog opsega.

2.2. Internetski prijenos

Za kvalitetan prijenos putem interneta je vrlo važno da se ne događaju prekidi koji mogu nastati u slučaju da se odašilje s mjesta koje ima lošu pokrivenost. Na mjestima koja imaju dobar signal i male smetnje prijenos se može izvoditi koristeći prijenosne umjerivače koje nudi svaki telekom, ali za mjesta s lošom pokrivenosti signalom ili za prijenose za koje je apsolutno da se prijenos ni u jednom trenutku ne prekine potrebni su usmjerivači koji mogu koristiti više telekoma istovremeno te u slučaju da jedan iskusi smetnje automatski se prebacuje na drugi ili u slučaju lošeg signala može kombinirati više različitih telekoma kako bi postigao veću brzinu.

3. Programi potrebni za funkcioniranje ovog rada

Engl. „dependencies“ naziv za programe na koji se neki drugi program oslanja kako bi funkcionirao. U ovom radu to su RTSP Simple Server, OBS Studio i node.js programski jezik.

Počevši s RTSP Simple Serverom i njegovim zavisnostima, za razliku od ostalih spomenutih alata RTSP Simple Server se ne oslanja, odnosno ne koristi apsolutno nikakve zavisnosti već je, kao što je naglašeno, u potpunosti samostalan u svojoj funkcionalnosti.

OBS Studio, za razliku od RTSP Simple Servera, za koji je spomenuto da je samostalan, nije samostalan te ovisi o nekoliko srodnih značajki te zavisnosti. Među najbitnijim zavisnostima uporabljenim u sklopu OBS Studija, valjalo bi istaknuti „FFmpeg“ koji se također koristi u primijenjenoj kombinaciji s izvođačem audio te video zapisa „VLC player“, zatim uređivač video zapisa „DaVinci Resolve“ i poznate internetske stranice čije su uslužne djelatnosti posluživanje video zapisa kao što su YouTube i TikTok.

Node.js, koji isto kao OBS Studio, a opet za razliku od RTSP Simple Servera, sadrži nekoliko zavisnosti među kojima se ističe „V8 javascript engine“ kao najbitniji te najučinkovitiji element pri stizanju zavisnosti čija je upečatljiva karakteristika ta što je zapravo pisan u C++ programskom jeziku. „V8 javascript engine“ je ključni sastavljač koda, engleski poznatije kao „assembler“ čija je funkcija prevođenje javascript programskog jezika na binarne numeričke vrijednosti jedinica i nula koje sklopovi, za razliku od javascript programskog jezika, mogu razumjeti te primijeniti u svojoj zadanoj funkciji.

3.1. Poslužitelj za primanje prijenosa

RTSP Simple Server je serverska aplikacija (Ros, 2022) napisana u “Golang” programskom jeziku koji je razvijen u Google-u 2007 [2]. Aplikacija omogućuje objavljivanje, čitanje i prosljeđivanje video i audio prijenosa uživo kroz razne protokole i

kodeke. Može se pokretati na sva tri standardna operativna sustava (Linux, macOS i Windows). Postavke se učitavaju iz yml konfiguracijske datoteke te je također podržan "hot reload" pristup, što jednostavno znači da se mogu izmjenjivati za vrijeme rada aplikacije i automatski primijeniti. S obzirom na mnogo značajki od kojih mnoge neće biti korištene, u nastavku će biti pokazana nekolicina postavki koje su važne za uspješno uspostavljanje funkcionalnosti ovog rada.

```
# Number of read buffers.
# A higher number allows a wider throughput, a lower number allows to save RAM.
readBufferCount: 512
```

U trenutnoj situaciji ova postavka može ostati zadana zato što se u radu koristi samo s jednim video prijenosom i dobrom internetskom vezom, ali kada bi se radilo s više prijenosa koji bi dolazili s lokacija koje nemaju adekvatnu internetsku vezu, povećanje međuspremnika ("readBufferCount") može riješiti ili umanjiti probleme tj. prekidanja.

```
# Enable the HTTP API.
api: no
# Address of the API listener.
apiAddress: 127.0.0.1:9997
```

Postavka "api" može imati vrijednost "yes" kao omogućeno i "no" kao neomogućeno. Api u ovoj aplikaciji poslužuje podatke o aktivnim video prijenosima. Podaci se mogu zatražiti na lokalnoj adresi na priključku 9997 pod /v1/paths/list.

Kada se podatke zatraži dobiju se u formatu prikazanom na slici (Slika 2). Od dobivenih podataka za rad je potreban samo naziv prijenosa i ključ. U primjeru na slici "live4464" bi bio naziv prijenosa, a "test" bi bio ključ. S obzirom na to da je u planu poboljšati rad tj. dodati mogućnost rada s više prijenosa istovremeno u naziv prijenosa je upisan i broj priključka za OBS primjer koji će biti pojašnjen u nastavku.

Slika 2 – izgled APIja koji RTSP Simple Server poslužuje

Iako je RTSP Simple Server vrlo moćna aplikacija i dalje je open source te samostalno ju razvija programer pod nazivom "aler9" na Github-u. Samim time greške nisu rijetkost. Tako je za vrijeme razvijanja ovog rada bio otkriven "bug" unutar API (Slika 2) gdje bi prijenos ostao na popisu iako više podaci ne dolaze na server. Nakon konzultacije s programerom i dolaska do zaključka da je stvarno riječ o grešci, programer se odmah uputio u rješavanje problema. Unutar dokumentacije je pronađen i drugi API koji prikazuje manje podataka i unutar rada s drugim dok greška i ne bude popravljena.

Slika 3 – prikaz u komentara u kojem programer potvrđuje grešku u kodu te obavještava o ispravci

Na slici (Slika 3) se vidi da je programer popravio grešku unutar samo šest dana od prijavljivanja problema i objašnjenja kako ga rekreirati. Što je rijetkost zato što većina programera na projektima otvorenog koda rade u slobodno vrijeme te često nisu u mogućnosti odvojiti dovoljno vremena za održavanje projekata.

```
# RTSP parameters

# Disable support for the RTSP protocol.
rtspDisable: no
# List of enabled RTSP transport protocols.
# UDP is the most performant, but doesn't work when there's a NAT/firewall
between
# server and clients, and doesn't support encryption.
# UDP-multicast allows to save bandwidth when clients are all in the same LAN.
# TCP is the most versatile, and does support encryption.
# The handshake is always performed with TCP.
protocols: [udp, multicast, tcp]
# Encrypt handshake and TCP streams with TLS (RTSPS).
# Available values are "no", "strict", "optional".
encryption: "no"
# Address of the TCP/RTSP listener. This is needed only when encryption is "no"
or "optional".
rtspAddress: :8554
```

Real Time Streaming Protocol (RTSP) je protokol za prijenos slike i zvuka, zamišljen 1997. godine. [3] Namjenjen je za prijenos u lokalnoj mreži. Također podržava slanje zahtjeva za upravljanje. [4]Najčešće ga koriste nadzorne kamere, a u ovom radu bit će korišten kako bi se pristupilo prijenosu koji dolazi na RTSP Simple Server - u ovom slučaju na priključku 8554.

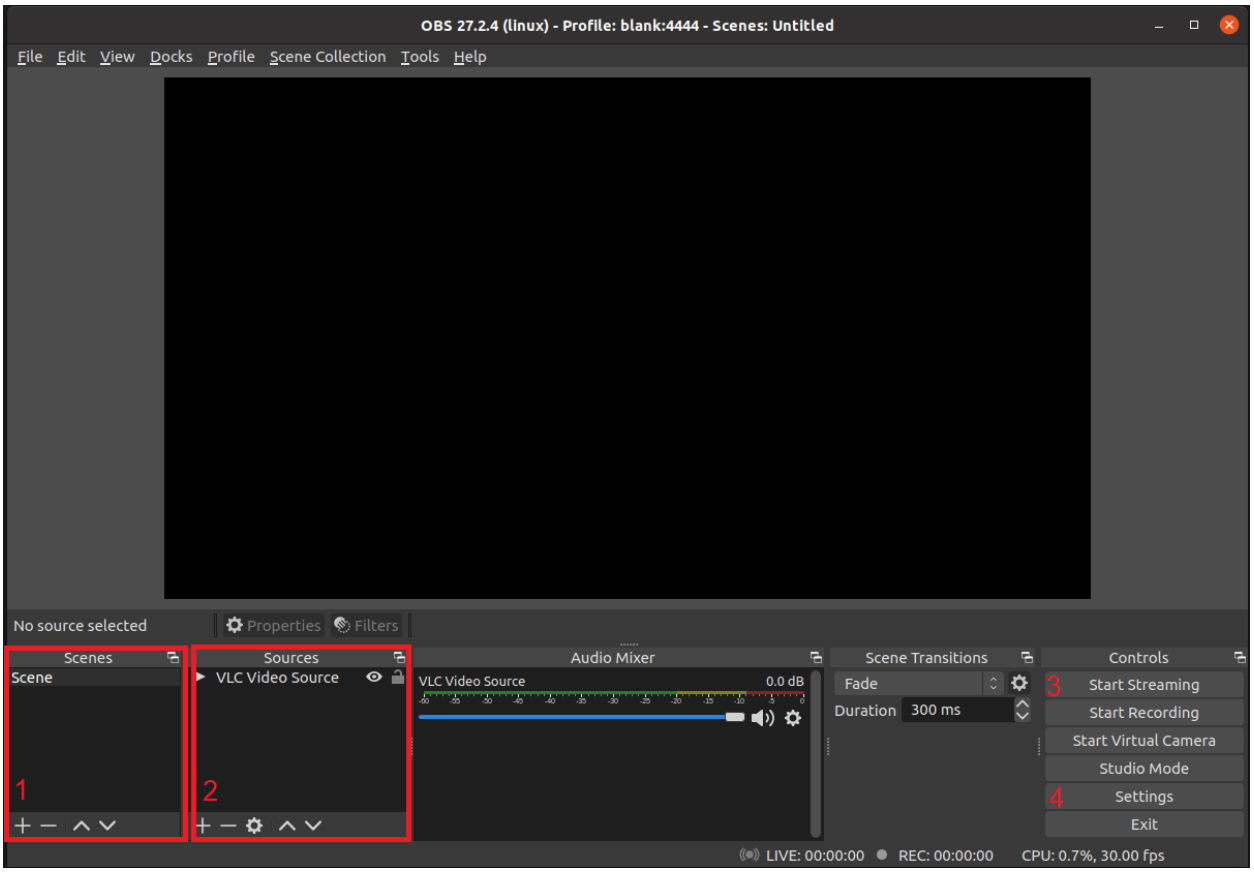
```
# RTMP parameters

# Disable support for the RTMP protocol.
rtmpDisable: no
# Address of the RTMP listener.
rtmpAddress: :1935
```

Real Time Messaging Protocol (RTMP) [5]je također protokol za prijenos zvuka i videa. Razvijen je u firmi „Macromedia“ za prijenos između „Flash player“ i poslužitelja koju je kasnije kupio Adobe. Iako je ”Flash player” dosegnuo kraj života 2020. godine [6] zbog pojave boljih alternativa i mnogih sigurnosnih propusta RTMP je i dalje jedan od najzastupljenijih protokola za prijenos uživo te ga se između ostalog koristi i za prijenos na YouTube.

3.2. Uređivač video prijenosa

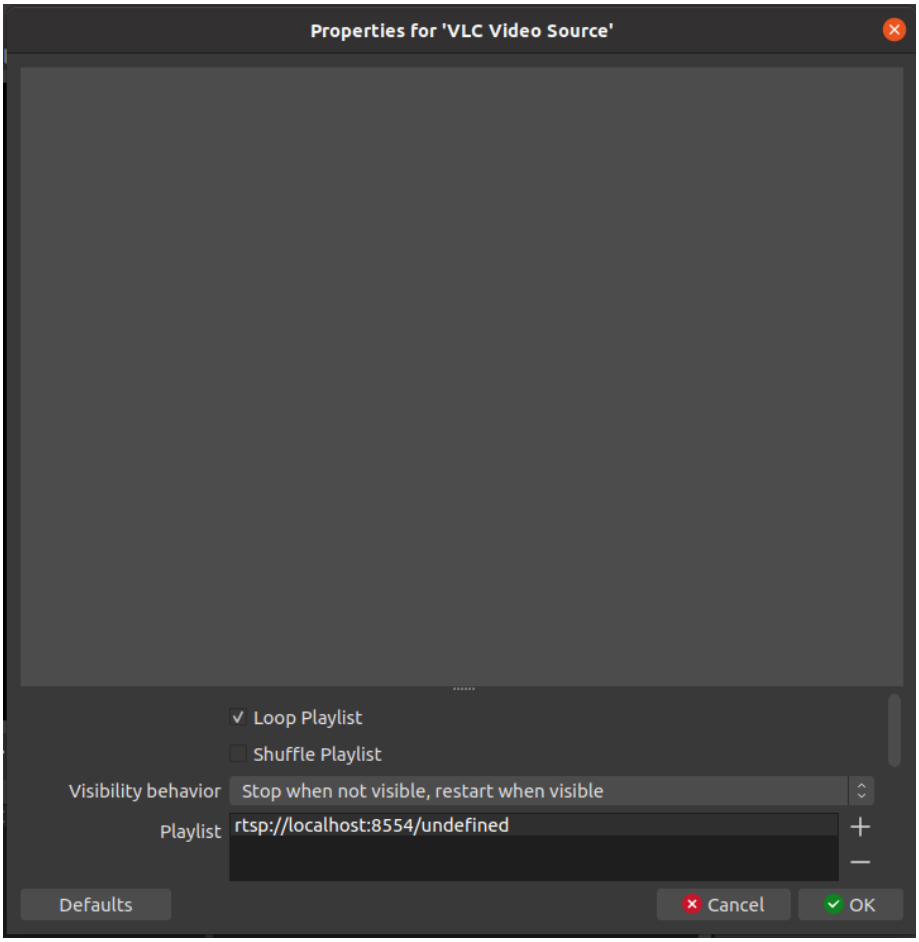
OBS Studio je besplatan “open source” program korišten za snimanje i prijenos video i audio signala. Pisan je u programskom jeziku C++ te je samim time odlično optimiziran. Dostupan je za Linux, macOS i Windows. Iako je neiskusnom korisniku teze započeti korištenjem OBS programa nego, primjerice, konkurentske proizvode kao što su Camtasia i Bandicam, važno je imati na umu da mnoge značajke nadmašuju cijenu vremena koje se treba uložiti da bi se mogao kvalitetno koristiti. Može se preuzeti sa službene “OBS project” (https://obsproject.com/) stranice ili sa githuba (https://github.com/obsproject/obs-studio) gdje se također može vidjeti izvorni kod.



Slika 4 – snimka zaslona početnog pogleda u grafičko sučelje OBS Studija

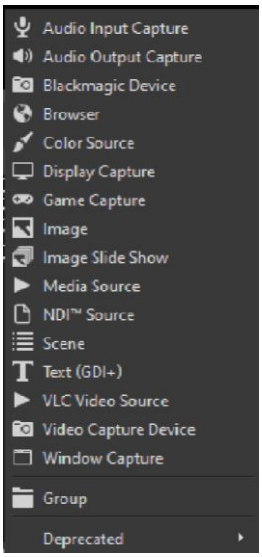
Na slici (Slika 4) je prikaz OBS-ovog grafičkog sučelja u kojem će se prikazati i objasniti funkcije korištene u ovom radu.

U okviru pod brojem „1“ nalazi se popis scena. Svaka scena je skup izvora raspoređenih po platnu. Nova scena se može dodati klikom na plus ikonu, a postojeća ukloniti odabirom i klikom na minus ikonu. Također im se može mijenjati redoslijed odabirom i uporabom ikona strelica. Broj „2“ prikazuje popis izvora, novi izvor se može dodati klikom na plus ikonu, a postojeća ukloniti odabirom i klikom na minus ikonu. Mijenjanjem redoslijeda odabirom i uporabom ikona strelica se također mijenja koji će izvor biti ispred kojeg. Postavke izvora se mogu mijenjati desnim klikom na izvor i odabirom svojstva. Pod brojem „3“ je gumb koji započinje i/ili završava prijenos. Gumb ispod broja „4“ otvara novi prozor u kojem se nalaze postavke.



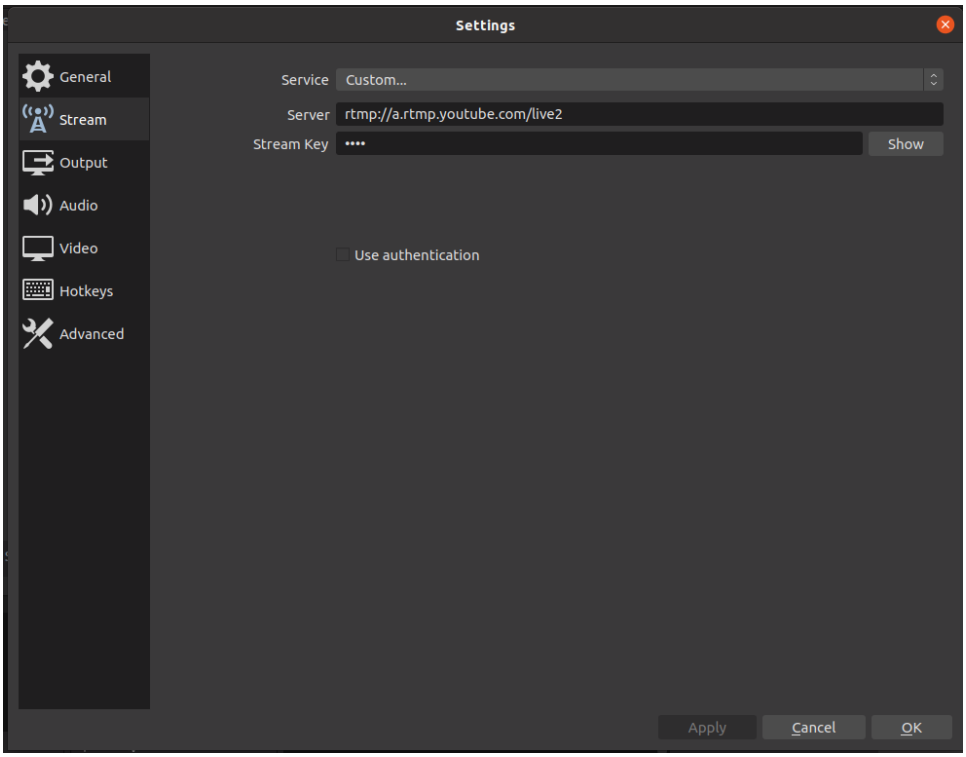
Slika 5 – snimka zaslona postavki multimedijskog izvođača

Kao što sam naišao na grešku u RTSP Simple Serveru tako sam također naišao na jedan u OBS Studiju koji je na sreću bilo lagano zaobići. Problem je bio u tome što OBS-ov "Media Source" (izvor koji može reproducirati datoteke ili u ovome slučaju RTSP prijenose) nakon nekoliko minuta počne imati problema s reprodukcijom tj. sinkronizacijom slike i zvuka. Taj problem je zaobiđen na tako da se umjesto "Media Source" koristi "VLC Video Source" (Slika 5) koji se pojavi medu izvorima nakon instalacije VLC media playera.



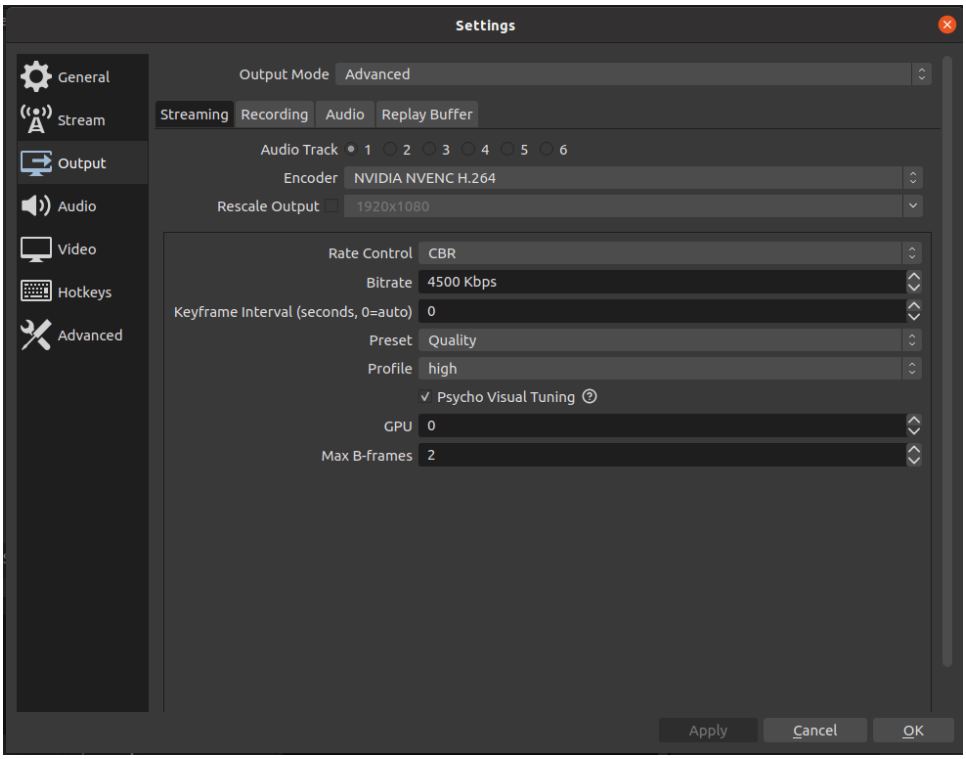
Slika 6 – snimka zaslona koja prikazuje popis mogućih izvora

Na ovoj slici (Slika 6) su prikazani ponuđeni izvori. S obzirom na to da je slika napravljena na računalu koje se profesionalno koristi za prijenose uživo, u odnosu na većinu drugih vjerojatno će nedostajati neke opcije ako se isti izbornik otvori na nekom drugom.



Slika 7 – snimka zaslona koja prikazuje postavke za prijenos uživo na YouTube

Slika (Slika 7) prikazuje postavke prijenosa. U ovom radu će se prenositi na YouTube pa kao uslugu se može odabrati “YouTube RTMS” ili “Custom”. U slučaju da se odabere “YouTube RTMS” jedini podatak koji je potrebno upisati je ključ prijenosa dok s “Custom” opcijom uz ključ streama također potrebno upisati i adresu poslužitelja te je moguće birati između YouTubeovog primarnog ili sekundarnog poslužitelja.

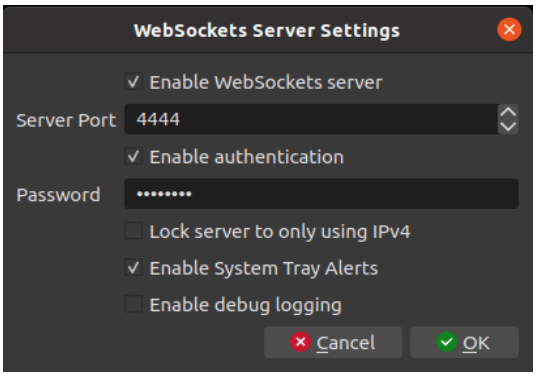


Slika 8 – snimka zaslona koja prikazuje postavke kodera

Opcija “Output” (Slika 8) u izborniku nudi opcije kodera u jednostavnom prikazu se ne prikazuju opcije potrebne za uglađivanje prijenosa na YouTube [7] pa je potrebno odabrati napredni prikaz (“Advanced” na slici). U tom pogledu se mogu izmijeniti potrebne opcije: “Encoder”, “Bitrate”, “Keyframe interval”, “GPU” i “Max B-frames". “Encoder” opcija pokazuje listu kodeka koji su dostupni na računalu (trenutno su najefikasniji Nvidia NVENC i Intel QuickSync), “Bitrate” određuje koliku količinu podataka koder šalje po sekundi, “Keyframe interval” [7] je opcija koja određuje koderu koliko često mora poslati referentni okvir, GPU određuje grafičku karticu koja će kodirati video u slučaju da ih je vise ugrađeno u računalo (brojanje počinje od 0) i zadnja bitna opcija je “Max B-frames” koja određuje maksimalnu količinu okvira koji se generiraju na osnovu prethodnog i budućeg okvira.

3.2.1. Proširenja uređivača

Proširenja ("plugins") u OBS Studiju služe kako bi korisnici vješti u programiranju mogli proširiti funkcionalnost programa i dodati njima interesantne ili generalno potrebne značajke. [8]. Jedan od primjera se mogao vidjeti na slici s izvorima gdje se kao izvor vidi "NDI source" (NDI je protokol koji prenosi sliku u visokoj kvaliteti preko mreže) koji planiram koristiti u budućoj verziji ovog rada uz program za grafike (casparCG). Dodatak koji koristim u ovom radu se zove "obs-websocket" dostupan je na službenom OBS forumu ili na githubu. "Websocket" je stalna veza između klijentske i poslužiteljske aplikacije koja omogućuje komunikaciju.



Slika 9 – snimka zaslona koja prikazuje postavke poslužiteljskog proširenja pomoću kojeg je izvedena kontrola

OBS Websocket poslužiteljske postavke (Slika 9) omogućavaju odabir priključka na kojem će dodatak slušati zahtjeve za spajanjem i lozinke koju će tražiti da se ne bi bilo tko s lokalne mreže mogao spojiti na njega. [9]

3.3. Programsko okruženje

Asinkrono JavaScript okruženje, "Node.js" je dizajnirano za izradu skalabilnih mrežnih aplikacija. [10] Asinkrono znaci da program može pokretati druge dijelove koda dok čeka da se određeni segment izvrši. Odabran je jer je fleksibilan (ima veliku količinu biblioteka) i jer podržava značajke koje će mi biti potrebne pri proširivanju programa u budućnosti.

3.3.1. Protokol za posluživanje podataka naziva HyperText Transfer Protocol

„HTTP“ odnosno HyperText Transfer Protocol najznačajniji je protokol aplikacijskog sloja na Internetu. On služi prvenstveno za razmjenu hipertekstova odnosno strukturiranih tekstova koji koriste logičke poveznice. Za uspostavljanje veze najčešće koristi TCP protokol transportnog sloja, iako se može prilagoditi i primijeniti i za UDP veze (npr. HTTPU, SSDP). UDP, kao i TCP, je protokol koji se koristi kod transportnog sloja, ali je od TCP-a znatno jednostavniji i nema kontrolu razmjene podataka (pogodan za komunikacije gdje su pogreške prihvatljive)”. U radu se koristi za prijenos podataka iz API-ja i za posluživanje HTML dokumenta preko kojeg se može upravljati poslužiteljskom stranom. [11]

3.3.2. Biblioteka za slanje API zahtjeva naziva Axios

Jedna od biblioteka koja olakšava rad s “API-jima” i čini slanje zahtjeva iznimno jednostavno. [12] Također podržava obećanja eng. Promises što je značajka koja zaustavlja rad funkcije dok npr. se ne dobiju podaci. Vrlo je popularan jer je jednostavno dodati uvjete uz zahtjev koji se šalje npr. da se zahtjev otkaže ako poslužitelj ne odgovori u zadanom vremenu, dakle jednostavno ne pošalje nikakav odgovor.

3.3.3. Biblioteka za komunikaciju između klijentske i poslužiteljske aplikacije naziva Socket.io

Biblioteka za ostvarivanje komunikacije između poslužitelja i klijenta te razmjenu podataka. [13] Koristi “http” protokol za uspostavu veze te kasnije šalje podatke preko "tcp” protokola. S obzirom na popularnost uz node.js i javascript u pregledniku dostupan je i za jezike/ okruženja: Java, C++, Swift, Dart, Python i .NET.

3.3.4. Biblioteka za posluživanje podataka naziva Express

Biblioteka koja sadrži skup značajki koje omogućuju posluživanje raznih podataka preko zahtjeva na “HTTP” protokolu. Funkcionira na principu da se na poslužitelju

odabere jedan priključak koji se već ne koristi te se onda na njemu sluša dolazni zahtjev. Ovisno o putanji zahtjeva nazad se šalje određeni odgovor s traženim podacima.

4. Uspostavljanje komunikacije između poslužiteljskih aplikacija

S obzirom na to da RTSP Simple Server nema mogućnosti dodavanja grafika ili ikakvog rada s prijenosom osim prosljeđivanja i pretvaranja u različite formate, a OBS Studio nema mogućnosti primanja prijenosa odlučio sam napraviti međuprogram koji će na osnovu podataka iz RTSP Simple Servera upravljati OBS Studiom. Osnovna premisa je bila napraviti funkciju koja će se izvoditi u određenom intervalu te na osnovu podataka tj. stanja u datom trenutku izvršavati određene naredbe. Radi bolje preglednosti i modularnosti rad sam raspodijelio na vise datoteka. Glavna datoteka u kojoj je sva logika ima naslov „controller.js“, uz nju su još dvije „apiGet.js“ i „obsFje.js“ koje sadržavaju funkcije koje šalju i vraćaju podatke te „control.html“ pomoću koje se upravlja.

4.1. Dokument koji sadrži funkcije za rad s API-jima naziva apiGet.js

```
const axios = require('axios');

function apiGet(url){
    return axios.get(url)
        .catch(error =>{
            console.log(error);
        })
    }

function apiKick(url){
    return axios.post(url)
        .catch(error =>{
            console.log(error);
        })
    }
module.exports = {
    apiGet,
    apiKick,
};
```

Funkcije u ovom dokumentu koristim za komunikaciju s RTSP Simple Serverom. Jedina biblioteka koja se koristi u ovom dokumentu je „Axios“ koji iako ima mnoge značajke koje su nabrojane ovdje nisu potrebne i sva komunikacija se izvršava koristeći „get“ i „post“ zahtjeve na specifične putanje.

4.2. Dokument koji sadrži funkcije za kontrolu OBS Studio programa naziva obsFje.js

Ovaj dokument sadrži funkcije i pripremljene objekte za interakciju s OBS Studiom koristeći biblioteku obs-websocket-js. Za ovaj rad su važne tri metode iz biblioteke. Metoda „connect“ za spajanje na primjer OBS Studija, „send“ za slanje naredbe i pratećeg objekta (u slučaju da se neka postavka izmjenjuje) koji sadrži postavke te „disconnect“ za prekid veze.

```
const OBSWebSocket = require('obs-websocket-js');
const obs = new OBSWebSocket();
```

Za korištenje potrebno je uvesti biblioteku s funkcijom „require“ i dodijeliti novi primjer konstanti u ovom slučaju naziva „obs“.

```
async function connection(port, pass){
    await obs.connect({address: 'localhost:'+port, password: pass})
    .catch((error) => {
        console.log(error);
        return error.status;
    });
}

async function letItGo(){
    obs.disconnect();
}
```

Prije nego sto je moguće slanje naredbi OBS Studiju preko websocket plugina, potrebno je započeti vezu. Veza se započinje tako da pomoću „connect“ metode pošalje objekt s adresom i lozinkom websocket poslužitelja. Pošto radim na inačici ovog

međuprograma koji će moći imati vise primjera OBS Studija koji će raditi istovremeno samo je IP adresa (localhost) „uklesana u kamen“, a priključak i lozinka ovise o varijablama koje budu prenesene u funkciji. Kako bi se oslobodio procesorski prostor nakon svakog bloka funkcija potrebno je prekinuti vezu pomoću metode „disconnect“.

```
async function streamServer(url, skey){

    var streamSettings = {
        settings: { key: skey, server: url },
        type: 'rtmp_custom',
    }

    await obs.send('SetStreamSettings', streamSettings).then(data => {
        console.log('Settings set: ', data);
    }).catch((error) => {
        console.log(error);
    })

}
```

Za postavljenje postavki prijenosa potrebno je pomoću metode „send“ poslati „string“ koji navodi postavku koja se mijenja i objekt u kojem je obavezno navesti tip servisa koji se koristi za prijenos te ključ za prijenos i adresa servisa u slučaju da je prilagođen („rtmp_custom“).

```
async function vlcConnect(key){

    var vlcSettings = {
        "sourceName": 'VLC Video Source',
        "sourceType": 'vlc_source',
        "sceneName": 'Scene',
        "sourceSettings":
         { "loop": true,
           "network_caching": 5000,
           "playback_behavior": 'stop_restart',
           "playlist": [{
                "hidden": false,
                "selected": true,
                "value": 'rtsp://localhost:8554/' + key,
            }],
           "shuffle": false,
           "subtitle": 1,
           'subtitle_enable': false,
           "track": 1 },
        "setVisible": true,
    }

    await obs.send('SetSourceSettings', vlcSettings).then(data => {
        console.log(data);
    }).catch((error) => {
        console.log(error);
    })
}
```

Kada prijenos bude upućen prema RTMP Simple Serveru potrebno je uputiti izvor (VLC Video Source) prema adresi gdje isti može prikazivati. Ova funkcija zahtijeva da je izvor već dodan i ona samo mijenja njegove postavke. Postavka koja usmjerava prema rtsp prijenosu RTSP Simple Servera je `sourceSettings.playlist[0].value` . Prijenos se može prikazivati na priključku 8554 s istom putanjom/ ključem na koju dolazi rtmp prijenos.

```
async function streamStatus(){
    var streamStatus = await obs.send('GetStreamingStatus').then(data => {
        var tmpArr = [data['recording'], data['streaming']];
        return tmpArr;
    }).catch((error) => {
        console.log(error);
    })
    return streamStatus;
}
```

Prije nego se počne upravljati OBS Studiom važno je imati točnu sliku njegovog trenutnog stanja jer uvijek postoji mogućnost da je neki proces prekinut i/ili ostao aktivan. Ova funkcija vraća niz sa stanjem snimanja i prijenosa (aktivan/ neaktivan).

```
async function startStream(){
    var streamStatus = await obs.send('StartStreaming').then(data => {
        console.log(data);
    }).catch((error) => {
        console.log(error);
    })
}
```

Pozivanjem funkcije „startStream“ se započinje prijenos na prethodnu namještenu adresu .

```
async function stopStream(){
    var streamStatus = await obs.send('StopStreaming').then(data => {
        console.log(data);
    }).catch((error) => {
        console.log(error);
    })
}
```

Kao sto pozivanje funkcije „startStream“ započinje prijenos pozivanje funkcije „stopStream“ zaustavlja prijenos.

```
async function gimmeResolutions(){
    var resInfo = await obs.send('GetVideoInfo').then(data => {
        //console.log(data);
        return data;
    }).catch((error) => {
        console.log(error);
    })
    //console.log(obsStats);
    if(resInfo !== undefined){
        const res = [resInfo["baseWidth"] + 'x' + resInfo["baseHeight"],
resInfo["outputWidth"] + 'x' + resInfo["outputHeight"]];
        return res;
    }
}
```

Još jedna prepreka koju je trebalo zaobići tj. na koju je trebalo paziti su rezolucija projekta i rezolucija izlaznog prijenosa. OBS Studio ponekad zna imati probleme s kodiranjem signala ako te dvije rezolucije nisu jednake tj. ako mora kodirati na drugačiju rezoluciju. Ova funkcija vraća obje rezolucije te klijent upozorava da postoji mogućnost da može doći do problema.

```
async function screenshotSource(port){
    await connection(port, "test");
    var screeshot = await obs.send('TakeSourceScreenshot',{saveToFilePath:
"/home/katal/Desktop/zavrsniRad/screenshots/" + port + ".png"}).then(data => {
        console.log(data);
    }).catch((error) => {
        console.log(error);
    })
    letItGo();
}
```

Kako bi se olakšala daljinska kontrola funkcija „screenshotSource“ slika izvor i sprema u direktorij „screenshots“. U poslanom objektu se nalazi samo putanja na kojoj se sprema slika jer je zadana postavka da slika sve izvore iz „Scene“.

```
async function stretchSources(sourceName, resolution){
    var obj = {
                item: {
                   name: sourceName,},
                   bounds: {
                       type: 'OBS_BOUNDS_STRETCH',
                       x: parseInt(resolution.split('x')[0]),
                       y: parseInt(resolution.split('x')[1]),
                        alignment: 1,},
                        position: {x: 0, y: 0,},
                       };
    var resInfo = await obs.send('SetSceneItemProperties', obj).then(data => {

        console.log(data);
        return data;
    }).catch((error) => {
        console.log(error);
    })
}
```

S obzirom na to da je moguće slati različite rezolucije važno je da se ili izvori prilagode rezoluciji projekta ili projekt prilagodi rezoluciji izvora. Funkcija „stretchSources“ rasteže izvor na rezoluciju projekta.

4.3. Dokument za koji sadrži logiku rada naziva controller.js

```
const gimme = require("./apiGet.js");
const obsFje = require("./obsFje.js");
const spawn = require("child_process").spawn;
const execSync = require("child_process").execSync;
const http = require('http').createServer();
const io = require('socket.io')(http, {
    cors: { origin: "*" }
});
const express = require("express");
const app = express();
```

Biblioteke koje se koriste u glavnom programu: apiGet.js, obsFje.js, spawn, execSync,http i socket.io. ApiGet.js i obsFje.js su objašnjene iznad. Spawn služi za pozivanje procesa koji će se izvršavati dulje vrijeme npr. za OBS Studio i aktivno ispisuje

podatke. ExecSync sluzi za pozivanje procesa te sprema podatke u međuspremnik (zadano do 200KB) koje nakon završetka ispisuje. Http određuje adresu i priključak na kojem će se slati zahtjevi koje će socket.io slušati. Pomoću express-a program sluša zahtijeva i šalje klijentsku aplikaciju pomoću koje se upravlja.

```
app.get('/', function(req, res){
    res.sendFile("/home/katal/Desktop/zavrsniRad/control.html")
});
app.use('/screenshots', express.static(__dirname +'/screenshots'));
app.listen(8080, function(){
    console.log("listening on port 8080");
})
```

Kada se pošalje zahtjev na korijensku putanju na priključku 8080 poslužitelj šalje „control.html“ te je također potrebno omogućiti pristup direktoriju sa slikama zaslona na putanji „/screenshots“ kako bi se slike mogle prikazati.

```
(async () => {          //rewriten checkIfActive()
    var defaultPort = 4444;
    for(var i = 0; i < ports.length; i++){
        try{
            var command = 'lsof -i :'+ (defaultPort+i);
            const isPortActive = execSync(command);
            //console.log(isPortActive.toString());
            var pid = isPortAc-
tive.toString().split('NAME')[1].split("     ")[1].split(" ")[0];
            console.log(pid);
            if(pid){
                obsProcessList[ports[i]][5] = await obsFje.connection(4444,
"test");      //wont work if obsFje has uncommented test() function
                obsProcessList[ports[i]][0] = obsProcessList[ports[i]][0] = "ac-
tive";
                obsProcessList[ports[i]][1] = pid;
                obsProcessList[ports[i]][4] = await obsFje.gimmeResolutions();
                obsProcessList[ports[i]][5] = await obsFje.letItGo();
            }
        }catch (err){
            console.log(err);
        }
    }
    console.log(obsProcessList);
})();
```

Prva funkcija koja se izvršava pri pokretanju programa. Posto se izvodi samo jednom ona je anonimna i samopozivajuća. Koristeći „execSync“ se poziva terminalna (konzolni redak za Linux) naredba „lsof -i“ koja provjerava korištene priključke te po završetku izvršavanja vraća tekst s podacima

```
COMMAND     PID  USER   FD   TYPE    DEVICE SIZE/OFF NODE NAME
obs     3712671 katal  123u  IPv6 145262597      0t0  TCP *:4444 (LISTEN)
```

Koji metoda .split razdvaja nakon rijeci „NAME“, druga nakon rijeci „\t“ i treća nakon razmaka te postavlja broj procesa u objekt „obsProcessList“ u koji se također sprema stanje aktivnosti i vrijednosti koje funkcija „gimmeResolutions“ vraća.

```
async function runInLoop() {
    var pathData = await gimme.apiGet(pathUrl);
    pathData = pathData.data.items;
    var pathDataLocation = Object.keys(pathData);
    var diff = [];

    if (tmp !== pathDataLocation) {
        diff = pathDataLocation.filter(x => !tmp.includes(x));
        console.log(diff);
        console.log(obsProcessList);

        if (diff.length) {
            for (var i = 0; i < diff.length; i++) {
                console.log(" A change ");
                if(typeof diff[i] !== 'undefined'){
                    if(obsProcessList[diff[i].substr(4, 4)][0] === "inactive"){
                        obsProcessList[diff[i].substr(4, 4)][2] = diff[i];
                        activateInstance(diff[i].substr(4, 4));
                    }
                    else{
                        await obsFje.connection("4444", "test");
                        await obsFje.vlcConnect(diff[i]);
                        intervalImg = setInterval(function() {
                            obsFje.screenshotSource(4444).catch((err) => {
                                console.log("error", err) }) }, 10000);
                    }
                }
                obsFje.letItGo();
            }
        }

        tmp = pathDataLocation;

    }
}
```

Funkcija „runInLoop“ je druga funkcija po redu koja se izvodi te se izvodi u intervalu od 1500 milisekundi. Pomoću „apiGet“ dobiva API podatke od RTSP Simple Servera te ovisno o tome ako je bilo promjena u odnosu na podatke iz proslog intervala poziva prikladne funkcije. Funkcija radi u dva slučaja, prvi je da nema OBS Studio koji ujedno nije jedan od aktivnih procesa te se onda poziva funkcija „activateInstance“, drugi

slučajje da već postoji aktivni proces OBS Studija te se onda osvježavaju postavke izvora i započinje interval koji svakih 10 sekundi pokreće funkciju „screenshotSource“.

```
async function activateInstance(port){
    const obs = spawn('obs', ['--profile', "blank:"+port, '-m'], {detached: true
    });
    const started = "info: [obs-websocket] server started successfully on port "
+ port;
    obs.stdout.on("data", data =>{
        if(data.toString().trimEnd() == started.trimEnd()){
            obsProcessList[port][0] = ['active'];
            intervalImg = setInterval(function () {
                obsFje.screenshotSource(port).catch((err) => {
                     console.log("error", err) }) }, 10000);
            wsOnStart(port);
        }
    });
    obsProcessList[port][1] = obs.pid;
    obs.stderr.on("data", data =>{
    });

    obs.on("error", (error) =>{
        console.log(`error: ${error.message}`);
    });

    obs.on("close", code =>{
        obsProcessList[port][0] = ["inactive"];
        clearInterval(intervalImg);
        console.log(`child process exited with: ${code}`);
    });
}
```

Za pokretanje procesa OBS Studio unutar funkcije „activateInstance“ se poziva funkcija spawn s naredbom u stringu te objektom u kojem se navodi da se proces pokrene odvojeno od glavnog programa da u slučaja neočekivanog prestanka rada OBS Studio može nastaviti s prijenosom. Nakon započinjanja procesa metodom „.stdout.on“ čitaju se podaci o stanju procesa dok ne dođe podatak o pokretanju websocket poslužitelja. Pokretanje websocket poslužitelja je znak da je OBS Studio uspješno pokrenut i da se može započeti interval koji poziva „screenshotSource“ funkciju te pokrenuti funkcija „wsOnStart“. Pri zatvaranju programa se prekida interval koji poziva „screenshotSource“ funkciju.

```
async function wsOnStart(port){
    await obsFje.connection(port, "test");
        obsProcessList[port][4] = await obsFje.gimmeResolutions();
        await obsFje.vlcConnect(obsProcessList[port][2]);
        await obsFje.stretchSources('VLC Video Source',
      obsProcessList[port][4][0]);


        var tmpArr = await obsFje.streamStatus();
}
```

Pri pokretanju novog primjera OBS Studija izvršava se funkcija „wsOnStart“ koja sadrži skup funkcija koje je potrebno izvršiti da bi se OBS Studio ispravno postavio i da bi program imao sve potrebne podatke za daljnje upravljanje.

```
io.on('connection', (socket) => {
    console.log('a user connected');

    socket.on('command', async(command) =>     {
        switch(command[0]){
            case 'streamserver':
                if(obsProcessList[4444][0] == 'active'){
                    //console.log('doing something');
                    obsProcessList[4444][5] = await obsFje.connection(command[1],
"test");
                    await obsFje.streamServer(command[2], command[3]);
                    obsProcessList[4444][5] = await obsFje.letItGo();
                    io.emit('rtrn',['commandRtrn', `command successful`] );
                }else{
                    console.log('ayo cant set when it aint runnin');
                }
                break;
            case 'stats':
                sendStats(4444);
                break;
            case 'isActive':
                if(obsProcessList[command[1]][0] == 'active'){
                    var data = [true, obsProcessList[command[1]][4]];
                    data.unshift('isActiveRtrn');
                    console.log(obsProcessList[command[1]][4]);
                    io.emit('rtrn', data);
                }
                else if(obsProcessList[command[1]][0] == 'inactive'){
                    io.emit('rtrn', ['isActiveRtrn', false]);
                }
                break;
            case 'toggleActive':
                if(obsProcessList[command[1]][0] == 'inactive'){
                    await activateInstance(command[1]);
                    io.emit('isActiveRtrn', true);
                    //console.log('should make bkg green');
                }
                else if(obsProcessList[command[1]][0] == 'active'){
                    execSync('kill -9 '+ obsProcessList[command[1]][1]);
                    obsProcessList[command[1]] = ["inactive"];
                    io.emit('isActiveRtrn', false);
                }
                break;
```

```
            case 'startOrStopStream':
                if(obsProcessList[4444][0] == 'active'){
                    //console.log(command[1]);
                    obsProcessList[4444][5] = await obsFje.connection(4444,
"test");
                    var tmp2 = await obsFje.streamStats();
                    if(!tmp2["streaming"]){
                        await obsFje.startStream().then(async ()=>{
                            socket.emit('startOrStopFront', [null, await
obsFje.obsStats(), await obsFje.streamStats()]);
                        });
                    }else{
                        await obsFje.stopStream().then(async ()=>{
                            socket.emit('startOrStopFront', [null, await
obsFje.obsStats(), await obsFje.streamStats()]);
                        });
                    }

                    obsProcessList[4444][5] = await obsFje.letItGo();
                }
                break;
        }
    });

});

http.listen(6969, () => console.log('listening on http://localhost:6969') );
```

Ovaj isječak koda prikazuje da program na priključku 6969 čeka da se control.html spoji te da putem socket.io pošalje naredbu koju želi izvršiti.

4.4. Dokument koji sadrži mrežnu kontrolu naziva „control.html“

```
<script>
const socket = io('ws://localhost:6969');
isActive();

function isActive(){
    var port = ['isActive', 4444];
    socket.emit('command', port);
}
```

```
<script>
const socket = io('ws://localhost:6969');
isActive();

function isActive(){
    var port = ['isActive', 4444];
    socket.emit('command', port);
}
```

Nakon učitavanja mrežne stranice koja ima mogućnost kontrolirati dokument „controller.js“ prvo se pokreće funkcija „isActive()“ koja pomoću socket komunikacije na priključku „6969“ koristeći metodu „emit“ šalje ključnu riječ „command“, te niz koji sadrži komandu „isActive“ te broj priključka na kojem OBS Websocket čeka povezanost.

```
function setStreamDestination(){
    var text = ['streamserver', 4444, document.getElementById('urls').value,
document.getElementById('streamKey').value] //send rtmp key for yt
    console.log(text);
    socket.emit('command', text)

}

document.getElementById("startOrStopStream").onclick = () => {
    var text = ['startOrStopStream', 4444]
    socket.emit('command', text)
    socket.once('startOrStopFront', data => {
        if(document.getElementById("startOrStopStream").style.backgroundColor
!== 'red'){
            document.getElementById("startOrStopStream").style.background =
'red';
            document.getElementById("startOrStopStream").innerHTML = "stop
streaming";
        }else{
            document.getElementById("startOrStopStream").innerHTML = "start
streaming";
            document.getElementById("startOrStopStream").style.background =
'white';
        }
    })

}

function streamStats(){
    var text = ['stats', 4444];
    socket.emit('command', text);
}
```

```
function setStreamDestination(){
    var text = ['streamserver', 4444, document.getElementById('urls').value,
document.getElementById('streamKey').value] //send rtmp key for yt
    console.log(text);
    socket.emit('command', text)

}

document.getElementById("startOrStopStream").onclick = () => {
    var text = ['startOrStopStream', 4444]
    socket.emit('command', text)
    socket.once('startOrStopFront', data => {
        if(document.getElementById("startOrStopStream").style.backgroundColor
!== 'red'){
            document.getElementById("startOrStopStream").style.background =
'red';
            document.getElementById("startOrStopStream").innerHTML = "stop
streaming";
        }else{
            document.getElementById("startOrStopStream").innerHTML = "start
streaming";
            document.getElementById("startOrStopStream").style.background =
'white';
        }
    })

}

function streamStats(){
    var text = ['stats', 4444];
    socket.emit('command', text);
}




socket.on('rtrn', data => {
    switch (data[0]){
        case 'isActiveRtrn':
            if(data[1]){
            document.body.style.backgroundColor = "green";
            document.getElementById('toggleActiveB').innerHTML = "kiwf";
            if(data[2][0] != data[2][1]){
                document.getElementById('warning').innerHTML = "base and
output resolution dont match \n" + data[2];
            }else{
                //console.log("all good");
```

```
socket.on('rtrn', data => {
    switch (data[0]){
        case 'isActiveRtrn':
            if(data[1]){
            document.body.style.backgroundColor = "green";
            document.getElementById('toggleActiveB').innerHTML = "kiwf";
            if(data[2][0] != data[2][1]){
                document.getElementById('warning').innerHTML = "base and
output resolution dont match \n" + data[2];
            }else{
                //console.log("all good");
            }
            }
            else{
                document.body.style.backgroundColor = "red";
                document.getElementById('toggleActiveB').innerHTML =
"activate";
            }
            break;

        case 'commandRtrn':
            console.log(data);
            break;

        case 'statsRtrn':
            if(data[1] !== null){
                document.getElementById('obsStats').innerHTML ="framerate: " +
Math.round(data[1]["fps"]) + " cpu: " + Math.round(data[1]["cpu-usage"]) + "%
RAM: " + Math.round(data[1]["memory-usage"]) + "MB";
                if(data[2]["streaming"]){
                    document.getElementById("startOrStopStream").style.backgro
und = 'red';
                    document.getElementById("startOrStopStream").innerHTML =
"stop streaming";
                }else{
                    document.getElementById("startOrStopStream").innerHTML =
"start streaming";
                    document.getElementById("startOrStopStream").style.backgro
und = 'white';
                }
            }
            break;
    }

})
```

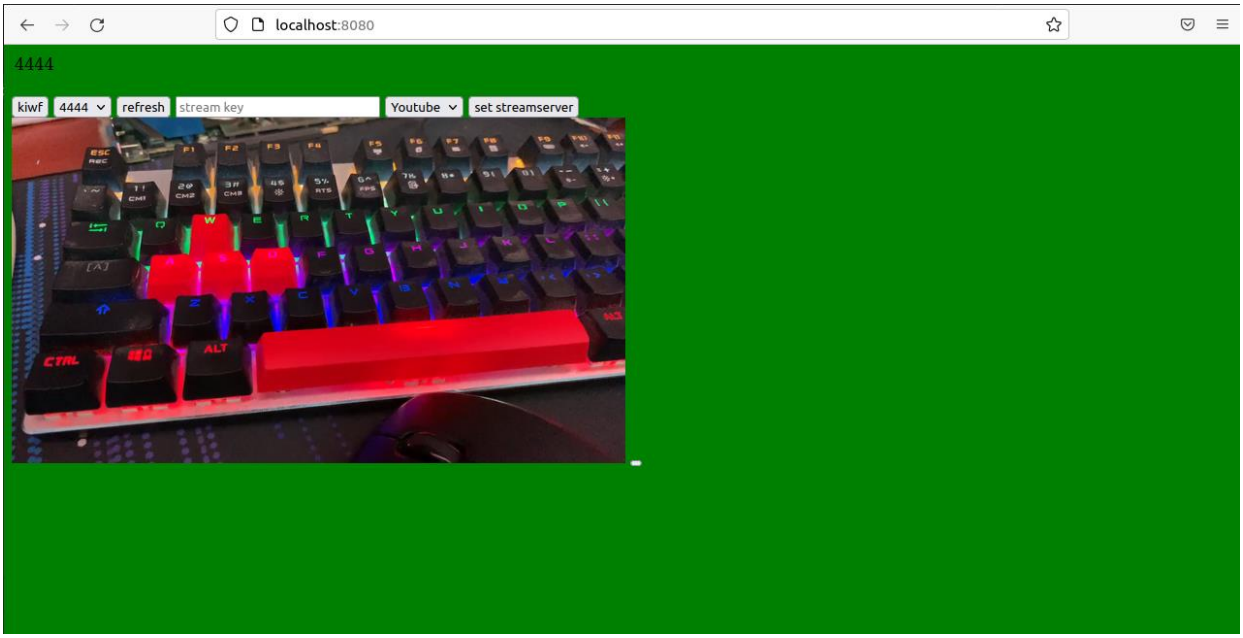
Metoda „io“ na konstanti „socket“ čeka ključnu riječ „rtrn“ i ovisno o podatku tipa „string“ na prvom mjestu u nizu uređuje grafički prikaz pomoću kojeg se kontrolira.

```
function rtmpChange(){
streamStats();
isActive();
}


var intervalId = setInterval(streamStats(), 10000);
var imgRefresh = setInterval(()=>{
    document.getElementById("img").src = "/screenshots/4444.png?t=" + new
Date().getTime(); //date because browser caches images

}, 10000);
</script>
```

Varijabla “intervalId” osvježava podatke svakih 10 sekunci, a variabla “imgRefresh” osvježava sliku



Slika 10 Prikaz kontrole (control.html)

Na slici (Slika 10) kontrola se sastoji od nekolicine tipki koje omogućavaju pokretanje ili zaustavljanje OBS Studio programa, namještanje adrese prijenosa na istom te slike koja pokazuje trenutno stanje OBS Studio programa koja se osvježava u intervalima od 10 sekundi. Pozadina može biti zelena ili crvena, ovisno o tome radi li program ili ne radi.

5. Zaključak

Zahvaljujući tehnološkom napretku, koliko softverskom toliko i hardverskom, novim tehnološkim postignućima te naravno volonterima koji su uložili svoje slobodno vrijeme kako bi doprinijeli napretku i dostupnosti programa diljem svijeta, može se reći da živim u jednom informatičko nezavisnom periodu gdje je svatko u mogućnosti pronaći besplatnu softversku alternativu za svoje potrebe.

Primjer korišten u ovoj dokumentaciji, OBS Studio, odličan je primjer kako je u današnje vrijeme potrebnije uložiti vremena i pokazati dobre volje za postizanje željenih rezultata nego li je sam imovinski nametak koji će se ulagati u softvere jednolikih sposobnosti. Ovo je ujedno i veliki korak za sve samostalne i nezavisne korisnike koji se žele uputiti u svijet streaming-a, ali nisu sigurni kako započeti te kolikim budžetom raspolagati te kako ga podijeliti na edukacijski dio, koji je srećom besplatan i lako dostupan na internetu, a kako na opremu i alate.

S druge strane, stranice poput Githuba jedne su od najvrjednijih izvora informacija, podataka te samih otvorenih kodova koje bilo tko može preuzeti, proučiti te korigirati uspostavom kontakta s developerom kojeg je moguće dohvatiti svega par klikova. Skoro pa uvijek, developeri su više nego voljni pomoći oko svih nejasnoća te samim time doprinose besplatnom znanju dostupnom iz udobnosti kućnih fotelja.

Izradom ovog rada, stečena su nova znanja i novi načini pristupanju prijenosa uživo uz naočigled teško shvatljivih kodova koji su uz malo truda i dodatnog istraživanja zapravo veoma pristupačni te lako uporabljivi. Streaming je jedna od brže rastućih grana unutar informatičkog svijeta prijenosa podataka zbog čega je svaki doprinos ovoj grani te poboljšanju kvalitete i pristupačnosti među publikom značajan korak pri poboljšanju i napretku područja prenošenja uživo, pružajući mnogim ljudima priliku iskušati se u ovom području.

Popis literature

[1] A. Ros, RtspSimpleServer, https://github.com/aler9/rtsp-simple-server [Pristupano 7 9 2022.].

[2] Robert Griesemer, Go, https://go.dev/ [Pristupano 7 9 2022.].

[3] Hamit Demir, AntMedia, https://antmedia.io/rtsp-explained-what-is-rtsp-how-it-works/ [Pristupano 7 9 2022.].

[4] B. Posey, TechTarget, https://www.techtarget.com/searchvirtualdesktop/definition/Real-Time-Streaming-Protocol-RTSP#:~:text=Real%20Time%20Streaming%20Protocol%20(RTSP)%20is%20an%20application%2Dlevel,the%20server%20streaming%20the%20data [Pristupano 7 9 2022.].

[5] Ruether, Traci, Wowza, https://www.wowza.com/blog/rtmp-streaming-real-time-messaging-protocol#what [Pristupano 7 9 2022.] .

[6] Adobe, https://www.adobe.com/products/flashplayer/end-of-life-alternative.html [Pristupano 7 9 2022.] .

[7] W. R., YouTube Pomoć, https://support.google.com/youtube/answer/2853702?hl=hr [Pristupano 7 9 2022.] .

[8] OBS Project, https://obsproject.com/wiki/Getting-Started-with-OBS-Studio-Development [Pristupano 7 9 2022.].

[9] tt2468, WebSocket API for OBS Studio, https://github.com/obsproject/obs-websocket [Pristupano 7 9 2022.].

[10] OpenJS Foundation, node.js, https://nodejs.org/en/about/ [Pristupano 7 9 2022.].

[11] J. Čop, HTTP/2 - protokol prilagođen modernom webu, Rijeka, 2016.

[12] M. Zabriskie, axios, https://axios-http.com/ [Pristupano 7 9 2022.].

[13] G. Rauch, Socket.IO, https://socket.io/docs/v3/ [Pristupano 7 9 2022.].

[14] K. R. Vijayanagar, OTTVerse, https://ottverse.com/i-p-b-frames-idr-keyframes-differences-usecases/#What_is_an_I-frame [Pristupano 7 9 2022.].

[15] T. Holowaychuk, Express, https://expressjs.com/en/guide/routing.html [Pristupano 7 9 2022.].

[16] Winlin, Simple Realtime Server, https://ossrs.io/lts/en-us/ [Pristupano 7 9 2022.].