Prijenos video signala korištenjem HLS protokola

Tadić, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

PRIJENOS VIDEO SIGNALA KORIŠTENJEM HLS PROTOKOLA

Diplomski rad

Luka Tadić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 12.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Luka Tadić
Studij, smjer:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. Pristupnika, godina upisa:	D-1382, 07.10.2021.
OIB studenta:	37594143682
Mentor:	prof. dr. sc. Marijan Herceg
Sumentor:	,
Sumentor iz tvrtke:	Zvonimir Kaprocki
Predsjednik Povjerenstva:	prof. dr. sc. Mario Vranješ
Član Povjerenstva 1:	prof. dr. sc. Marijan Herceg
Član Povjerenstva 2:	izv. prof. dr. sc. Ratko Grbić
Naslov diplomskog rada:	Prijenos video signala korištenjem HLS protokola
Znanstvena grana diplomskog rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	HLS (engl. HTTP live streaming) je protokol za prijenos video signala s prilagodljivom brzinom prijenosa. Podrška za protokol široko je rasprostranjena i koristi se u različitim internetskim preglednicima, mobilnim uređajima, medijskim poslužiteljima, itd. U okviru diplomskog rada potrebno je razviti web poslužitelj/aplikaciju na ugradbenoj platformi zasnovanoj na Linux operacijskom sustavu korištenjem Django (Python) web radnog okvira, koji će korištenjem HLS protokola omogućiti prijenos video signala s ugradbene platforme od krajnjeg korisnika. Web aplikacija treba omogućiti registraciju novih korisnika, prijavu registriranih korisnika, podešavanje parametara kamere, brzinu prijenosa, tip kodera, itd. Sumentor iz tvrtke: Zvonimir Kaprocki
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	12.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2023.

Ime i prezime studenta:	Luka Tadić
Studij:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1382, 07.10.2021.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Prijenos video signala korištenjem HLS protokola**

izrađen pod vodstvom mentora prof. dr. sc. Marijan Herceg

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

S ubrzanim rastom interneta i sve većom popularnošću multimedijskih sadržaja, video strujanje (engl. *streaming*) postalo je sastavni dio svakodnevnog života. Mogućnost gledanja visokokvalitetnih videozapisa na zahtjev ili gledanja uživo promijenilo je način na koji se koriste mediji i dovelo do pojave brojnih platformi za strujanje. Za besprijekornu isporuku ovih videozapisa na različitim uređajima i mrežnim uvjetima ključni su učinkoviti protokoli za strujanje videozapisa. Jedan takav protokol koji je široko prihvaćen je HTTP (engl. *Hyper Text Transfer Protocol*) Live Streaming (HLS) protokol [1]. HLS je protokol za strujanje koji je razvio Apple Inc., dizajniran za isporuku video sadržaja putem interneta na pouzdan i prilagodljiv način. Multimedijski sadržaj se razbija u manje datoteke koje se nazivaju segmenti, te se kodiraju u određenom formatu. Svaki segment sadrži nekoliko sekundi video ili audio sadržaja. HLS protokol nudi nekoliko prednosti, uključujući široku kompatibilnost uređaja, prilagodljivu brzinu protoka i robustan oporavak od pogreške. Kao rezultat toga, HLS je zapravo postao standard za video strujanje na iOS uređajima, a široko je podržan i na drugim platformama. Strujanje videozapisa korištenjem HLS protokola susreće se s raznim izazovima poput povećanog kašnjenja, odnosno latencije što predstavlja vrijeme od dohvaćanja video sadržaja do njegovog posluživanja korisniku. Drugi problemi se odnose na opterećenje poslužitelja i zahtjeve za većom propusnosti.

Zadatak diplomskog rada je razviti web aplikaciju/poslužitelj na *Raspberry Pi 3* računalu zasnovanom na *Linux* operacijskom sustavu. Web aplikacija treba omogućiti prijenos uživo video signala uhvaćenog s kamere korištenjem HLS protokola s *Raspberry Pi 3* računala koji predstavlja web poslužitelj do krajnjeg korisnika. Također, web aplikacija omogućuje registraciju novih korisnika, prijavu registriranih korisnika, mijenjanje postavki strujanja videozapisa poput rezolucije, brzine prijenosa (engl. *bitrate*), tipa kodera i sl. Aplikacija je izrađena u *Python* programskom jeziku korištenjem *Django* web radnog okvira (engl. *web framework*).

Rad je strukturiran na sljedeći način: u drugom poglavlju su opisani određeni protokoli za strujanje videozapisa Internet mrežom te su navedene prednosti i nedostatci pojedinih protokola. U trećem poglavlju objašnjen je cjelokupan proces izrade web aplikacije od razvoja algoritma za strujanje videozapisa s kamere, konfiguriranja poslužitelja, dodavanja dodatnih funkcionalnosti poput registracije i prijave registriranih korisnika do implementacije liste korisnika koji gledaju prijenos videozapisa uživo. Na kraju trećeg poglavlja, opisana je implementacija web aplikacije na ugradbenu platformu *Raspberry Pi 3*. U četvrtom poglavlju objašnjen je postupak implementacije algoritma za strujanje pomoću MPEG-DASH (engl. *MPEG Dynamic Adaptive*

Streaming over HTTP) protokola u web aplikaciju te usporedba i analiza performansi strujanja korištenjem HLS i MPEG-DASH protokola. U petom poglavlju iznesen je zaključak rada.

2. PROBLEM PRIJENOSA VIDEO SIGNALA INTERNET MREŽOM

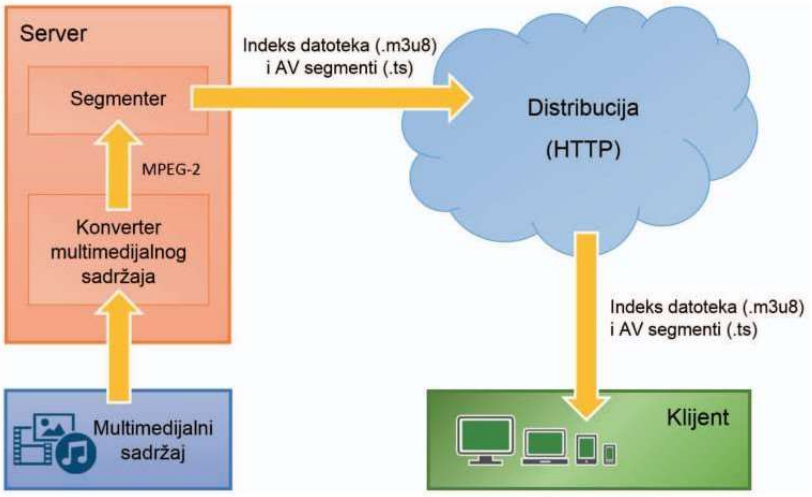
2.1. Protokoli za prijenos video signala Internet mrežom

Protokol za strujanje predstavlja skup standardiziranih metoda i pravila za prijenos multimedijskog sadržaja (obično video i audio) preko interneta. U osnovi, protokol za strujanje videozapisa prenosi dijelove (engl. *chunk*) video sadržaja od jednog uređaja do drugog. Osim toga, on definira način na koji se ti dijelovi ponovno formiraju u oblik za reprodukciju na drugoj strani komunikacijskog kanala. Da bi to sve bilo moguće, izlazni uređaj video sadržaja i korisnik moraju podržavati protokol za strujanje. Primjerice, ukoliko neki poslužitelj generira HLS strujanje, a video reproduktor na korisničkom uređaju ne podržava HLS protokol, strujanje neće raditi. Prije nego li se započne sa prijenosom sadržaja, isti bi se trebao komprimirati pomoću kodeka (primjerice H.264) kako bi se mogli uštediti mrežni resursi. Video datoteke se prije prijenosa također moraju spremiti u tzv. format spremnika (engl. *container format*) poput *.mp4* ili *.avi*. Izvor video sadržaja može biti kamera u slučaju strujanja uživo ili statičke datoteke (engl. *static files*) u slučaju videa na zahtjev (engl. *video on demand, VOD*). Danas su dostupni brojni protokoli za strujanje od kojih se neki i dalje razvijaju, a neki su relativno novi. U nastavku poglavlja je dan je opis pojedinih protokola za strujanje videozapisa internetom.

2.1.1. HLS

HLS predstavlja komunikacijski protokol koji je najrašireniji za strujanje video sadržaja, a zasnovan je na HTTP protokolu. Razvio ga je Apple 2009. godine kako bi izbacili *Flash* iz *Iphone-a* i isprva je bio usmjeren na iOS mobilnim uređajima ili osobnim računalima s OS X operacijskim sustavom. HLS je dizajniran na način da se efikasnije koristi mrežna veza. Efikasnost je omogućena preuzimanjem sadržaja više segmenata sa samo jednim HTTP zahtjevom. HLS također podržava strujanje s prilagodljivom brzinom prijenosa (engl. *adaptive-bitrate streaming*). Ovo je tehnologija koja omogućuje dinamičku isporuku videozapisa kako bi se osigurala najbolja moguća kvaliteta videozapisa za krajnje korisnike. Jedini veliki nedostatak povezan s HLS protokolom mogu biti visoke latencije povezane s njim. Zbog toga je Apple uveo proširenje koje se naziva HLS niske latencije (engl. *Low-Latency HLS*) koji može omogućiti latenciju na 2 sekunde ili manje. Ova latencija veliko je poboljšanje u odnosu na 15-30 sekundi latencije s kojom su općenito povezani HLS prijenosi uživo. HLS protokol omogućava dijeljenje cjelokupnog transportnog toka podataka na segmente koji se kasnije preuzimaju pomoću HTTP protokola [2],[3].

Sustav zasnovan na HLS protokolu se sastoji od tri dijela: poslužiteljske komponente (engl. *server*), distribucijske komponente (engl. *distribution*) i klijentske komponente (engl. *client*). Poslužiteljska komponenta je odgovorna za pretvaranje ulaznog multimedijskog toka u digitalni oblik, enkapsulaciju tog toka u format pogodan za isporuku te pripremu enkapsuliranog sadržaja za distribuciju. Distribucijsku komponentu predstavlja sustav web poslužitelja koji isporučuju medijske datoteke i datoteke manifesta klijentu HTTP protokolom. Klijentska komponenta šalje zahtjeve kako bi preuzela određene resurse, zatim preuzima te resurse i onda ih ponovno sastavlja u jedinstveni tok podataka. Na slici 2.1. prikazana je arhitektura HLS protokola u kojoj se na poslužiteljskoj strani uzima multimedijski sadržaj, zatim se obrađuje video i audio sadržaj te se dobije MPEG-2 transportni tok podataka. Nakon toga, tok podataka se u djelitelju toka (engl. *stream segmenter*) dijeli u seriju kratkih segmenata transportnog toka podataka. Zatim se segmenti transportnog toka postavljaju na web poslužitelj. Djelitelj toka također generira i ažurira glavnu i sporedne indeksne datoteke ili datoteke manifesta ekstenzije *.m3u8* koje sadrže listu imena segmenata transportnog toka video i audio podataka i putanje do lokacija na kojima ih je moguće preuzeti. Klijentska komponenta preuzima i čita manifest datoteku kako bi dobila informacije o lokaciji segmenata te šalje zahtjeve za preuzimanje segmenata. Nakon preuzimanja segmenata, klijentska strana sastavlja segmente u organizirani tok medijskog sadržaja koji reproducira bez pauza i zastoja između pojedinih segmenata. U slučaju gledanja videa na zahtjev, taj proces se nastavlja dok se u indeksnoj datoteci ne pojavi oznaka EXT-X-ENDLIST. Ukoliko ona nije prisutna u indeksnoj datoteci, radi se o strujanju uživo. Tokom toga, klijent periodično učitava novu verziju manifest datoteke te tako dohvaća lokacije novih segmenata te ih stavlja u popis za reprodukciju [2], [4], [5].



Slika 2.1. Arhitektura HLS protokola [2]

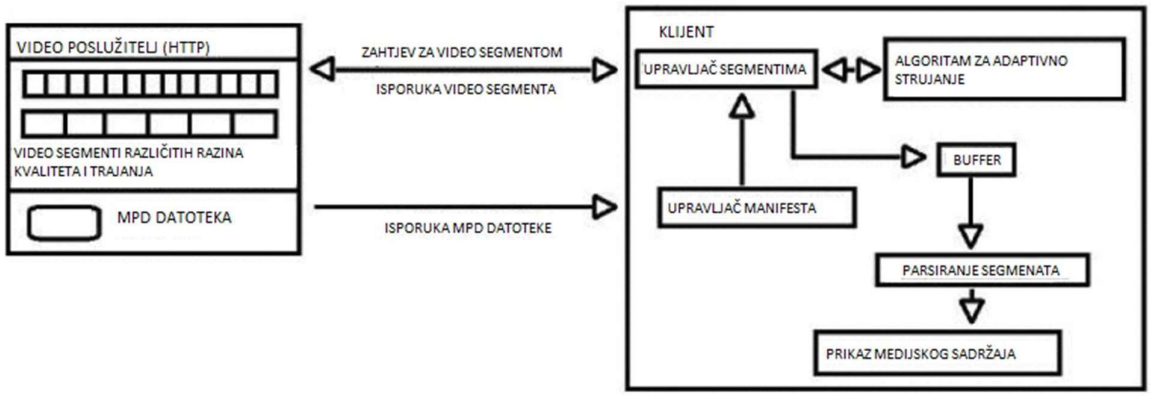
Kao što je ranije rečeno, HLS protokol podržava sustav strujanja s adaptivnom brzinom prijenosa. To znači da cijeli sustav ovisi o propusnosti mreže. Zbog toga se audio i video podatci pretvaraju u više tokova podataka različitih kvaliteta. Ovisno o propusnosti mreže bit će reproduciran sadržaj određene kvalitete. Da bi to funkcioniralo, multimedijski sadržaj za strujanje pretvara se u više tokova podataka različite kvalitete i generira se glavna manifest datoteka koja sadrži lokacije sporednih manifest datoteka. Svaka sporedna manifest datoteka sadrži popis segmenata određene kvalitete i putanje do njihovih lokacija na web poslužitelju. Klijentska strana ima zadaću pratiti propusnost mreže te ovisno o istoj bira multimedijski sadržaj s najvećom mogućom kvalitetom prilikom strujanja. Osim lokacija sporednih manifest datoteka, glavna manifest datoteka može sadržavati lokacije manifest datoteka alternativnih tokova segmenata sadržaja. To može biti zvuk na drugom jeziku, prijevod na drugom jeziku ili alternativni video (npr. prikaz iz drugog kuta). Zato je klijentska strana odgovorna i za izbor alternativnih tokova segmenata prilikom reprodukcije. HLS protokol omogućuje sustav za kriptiranje i dekriptiranje podataka. Kriptiranje/dekriptiranje vrši se pomoću AES (engl. *Advanced Encryption Standard*) algoritma. Ključ za kriptiranje i dekriptiranje se generira i razmjenjuje pomoću metode sigurne distribucije ključeva za kriptiranje pomoću HTTPS protokola. HLS podržava H.264 i H.265 video kodeke, dok od audio kodeka podržava AAC i MP3. Format prijenosnog toka je MPEG-2 TS.

2.1.2. MPEG-DASH

MPEG-DASH standard slično kao i HLS definira pravila za prijenos multimedijskog sadržaja različitih kvaliteta preko interneta. Ovaj protokol također omogućava strujanje s prilagodljivom brzinom prijenosa propusnosti mreže. MPEG-DASH određuje format medijskih datoteka primjeren za adaptivno strujanje koje omogućuje efektivno i neprekidno izmjenjivanje između različitih transportnih tokova osiguravajući prilagodbu nestabilnim mrežnim uvjetima. Multimedijski sadržaj se generira i čuva na poslužiteljskoj strani te se do krajnjeg korisnika prenosi pomoću HTTP protokola. Sadržaj koji se čuva sastoji se od dva dijela: segmenata transportnog toka multimedijskog sadržaja i manifest datoteke ili opisa medija prezentacije (engl. *Media Presentation Description, MPD*) napisane u XML (engl. *eXtensible Markup Language*) jeziku koja opisuje multimedijski sadržaj, odnosno njegove varijante s različitim rezolucijama i kodnim brzinama, lokacije na kojima se segmenti nalaze i ostale karakteristike [6], [7].

Kada DASH klijent započinje strujanje multimedijskog sadržaja, prvo mora dohvatiti MPD manifest datoteku. Klijent parsirajući MPD datoteku dobiva različite informacije o multimedijskom sadržaju poput njegove dostupnosti, varijantama sadržaja u smislu rezolucija i kodnih brzina, minimalnoj i maksimalnoj propusnosti, lokacijama segmenata u mreži i sl. Pomoću

toga DASH klijent odabire odgovarajuću varijantu toka podataka te počinje s preuzimanjem istog putem HTTP GET zahtjeva. Klijent odabire adaptacijske skupove (engl. *adaptation sets*) ovisno o primljenim podatcima iz MPD datoteke. Unutar svakog adaptacijskog skupa klijent odabire jednu reprezentaciju toka podataka na temelju opsega propusnosti u mreži. Nakon toga, klijent generira listu segmenata za svaku reprezentaciju kako bi unaprijed bio poznat redoslijed segmenata za formiranje HTTP GET zahtjeva. Na osnovu te liste segmenata, klijent šalje zahtjeve za segmente HTTP poslužitelju. Prvotno se podatci prije početka reprodukcije prikupljaju u međuspremniku (engl. *buffer*) kako bi se osiguralo da strujanje ispočetka nema prekida. Nakon toga, klijent nastavlja s preuzimanjem sljedećih segmenata dok u isto vrijeme promatra promjene propusnosti mreže. Ukoliko se stanje propusnosti u mreži pogorša, klijent će početi preuzimati segmente s manjom kodnom brzinom. Važno je znati da su segmenti različitih kodnih brzina poravnati u vremenu tako da je omogućeno izmjenjivanje bez zastoja s jedne kvalitete sadržaja na drugu. Na slici 2.2. nalazi se prikaz arhitekture MPEG-DASH protokola. MPEG-DASH za razliku od HLS-a podržava sve audio i video kodeke, a podržava MPEG-2 TS i fMP4 formate prijenosnog toka podataka. U odnosu na HLS, MPEG-DASH nije podržan na iOS i MacOS sustavima [6], [7], [8].



Slika 2.2. Arhitektura MPEG-DASH protokola [6]

2.1.3. Ostali protokoli za prijenos video signala Internet mrežom

RTMP (engl. *Real-Time Messaging Protocol*) je protokol koji se prethodno koristio za isporuku videozapisa u *Adobe Flash* reproduktoru. RTMP je razvila *Macromedia* s primarnom svrhom za rad s *Adobe Flash* reproduktorom, ali *Flash* reproduktor se više ne koristi. Budući da se RTMP nalazi na vrhu protokola za kontrolu prijenosa (engl. *Transmission Control Protocol*, TCP), on započinje proces uspostavljanja veze u tri koraka između RTMP klijenta i RTMP poslužitelja prilikom prijenosa podataka. Inicijator (klijent) traži od poslužitelja da započne vezu;

poslužitelj odgovara; tada inicijator potvrđuje odgovor i održava sesiju između oba kraja. Iz tog razloga, RTMP je prilično pouzdan. RTMP danas ima ograničenu podršku za reprodukciju. Umjesto toga, RTMP se sada koristi za unos (engl. *ingest*) iz kodera u online video platformu. RTMP unos omogućuje korisnicima da iskoriste podršku za jeftine RTMP kodere. Velik dio industrije online video strujanja, uključujući vodeći softver za strujanje i online video platforme, još uvijek je kompatibilan s RTMP unosom. Kada je uparen s HLS isporukom, RTMP unos proizvodi strujanje niske latencije. RTMP je i dalje dobar izbor jer može podržati nisku latenciju, što je glavni razlog zašto je RTMP unos ostao popularan. Još jedan glavni razlog zbog kojeg je RTMP unos trenutno najpopularniji protokol za unos je kompatibilnost. HLS unos, na primjer, još uvijek nije široko podržan od strane usluga strujanja [3], [8], [9].

SRT (engl. *Secure Reliable Transport*) je relativno novi protokol za strujanje tvrtke *Haivision*, vodećeg igrača u online prostoru za strujanje. SRT je protokol otvorenog koda koji je vjerojatno budućnost strujanja uživo. Ovaj video protokol za strujanje poznat je po svojoj sigurnosti, pouzdanosti i niskoj latenciji strujanja. SRT osigurava vezu i kontrolu te pouzdan prijenos sličan TCP-u. Međutim, to čini na aplikacijskom sloju koristeći UDP (engl. *User Datagram Protocol)* protokol kao temeljni transportni sloj. Podržava oporavak paketa uz održavanje niske latencije (zadano: 120 ms). SRT također podržava šifriranje pomoću AES-a. SRT je još uvijek prilično futuristički jer još uvijek postoje neka ograničenja kompatibilnosti s ovim protokolom. Sam protokol je otvorenog koda i vrlo je kompatibilan, ali hardver i softver koji podržava SRT protokol još je u procesu razvitka. Podržava sve video i audio kodeke te formate prijenosnog toka [3], [8], [9].

MSS (engl. *Microsoft Smooth Streaming*) je protokol za strujanje koji je *Microsoft* razvio 2008. kako bi zadovoljio rane potrebe za prilagodljivom brzinom prijenosa. Ovaj protokol za strujanje video sadržaja bio je poznat po isplativosti, smanjenju spremanja u međuspremnik i ponudi optimiziranih performansi. MSS je protokol s adaptivnom brzinom prijenosa dizajniran s DRM sigurnosnim mjerama za zaštitu sadržaja od piratstva. To je hibridna metoda isporuke medija koja funkcionira poput strujanja, ali se oslanja na HTTP progresivno preuzimanje. MSS protokol omogućuje brzu isporuku na sve Microsoftove uređaje. Protokol se ne može natjecati s drugim formatima koji se temelje na HTTP-u. Video kodeci koje podržava su H.264 i VC-1 te podržava AAC, MP3, WMA audio kodeke. Omogućava latenciju 6-30 sekundi [8].

WebRTC (engl. *Web Real-Time Communication*) je standard otvorenog koda za komunikaciju u stvarnom vremenu kojeg podržavaju gotovo svi moderni preglednici, uključujući

Safari, *Google Chrome*, *Firefox*, *Operu* i druge. WebRTC podržava visokokvalitetne VP8 i VP9 kodeke (osim starog H.264), kao i *Opus* audio kodek. U bliskoj budućnosti, protokol će dobiti podršku za potpuno novi video kodek AV1. Predviđa se da će protokol zamijeniti telefoniju i postati stup komunikacijskih usluga. U početku razvijen samo za aplikacije temeljene na *chat-u* i korištenje *VoIP-a*, postao je poznat po upotrebi u aplikacijama za *videochat* i konferencije nakon što ga je kupio *Google*. Neke od najčešćih aplikacija za potrošače današnjice, kao što su *Google Meet*, *Discord*, *Houseparty*, *Gotomeeting*, *WhatsApp* i *Messenger*, koriste WebRTC. Ono što WebRTC čini jedinstvenim je njegovo oslanjanje na *peer-to-peer* ili P2P strujanje. To je također poželjno rješenje kada strujanje zahtijeva strujanje niske latencije. WebRTC podržava adaptivno strujanje brzine prijenosa na isti način na koji to čine HLS i MPEG-DASH. Na isti način kao i HLS, WebRTC također koristi dinamičku prilagodbu kvalitete tijekom prijenosa kako bi korisnicima s različitim brzinama veze omogućio optimalno iskustvo pri gledanju video sadržaja, neovisno o tome jesu li njihove veze sporije ili brže. Jedna od najvećih prednosti WebRTC-a je ta što pretvara milijune preglednika u terminale za strujanje bez potrebe za instaliranjem dodatnih dodataka. Štoviše, WebRTC podržava latenciju ispod sekunde, što znači da nema više kašnjenja. Na kraju, protokol koristi tehnologiju prilagodljive brzine prijenosa, koja mu omogućuje automatsku prilagodbu kvalitete videa i sprječava bilo kakve padove i prekide. Nedostatak je nestabilnost zbog latencije od manje od sekunde [3], [8], [9].

RTSP (engl. *Real Time Streaming Protocol*) je još jedan protokol za strujanje koji su razvili stručnjaci iz *RealNetworks*, *Netscape* i sveučilišta *Columbia* oko 1996. godine. Dizajniran je za kontrolu servera za strujanje koji se koriste u zabavnim i komunikacijskim sustavima. RTSP radi s postavkom 'klijent-medijski poslužitelj' gdje obje strane mogu izdavati naredbe u VHS stilu RTSP poslužitelju. Naredbe uključuju *'play'*, *'record'* i *'pause'* što RTSP čini prikladnim za strujanje medijskih podataka uživo. RTSP može funkcionirati u oba smjera: može kontrolirati strujanja s poslužitelja na klijenta (video na zahtjev) kao i strujanja s klijenta na poslužitelj (snimanje glasa). RTSP se obično koristi za IP (engl. *Internet Protocol*) strujanje putem kamere jer, obično, CCTV ili IP kamera proizvodi RTSP tok. Kako bi se postigao dosljedan prijenos, RTSP koristi dva druga mrežna komunikacijska protokola—TCP za izdavanje i primanje kontrolnih naredbi (npr. zahtjev za reprodukciju) i UDP za isporuku zvuka, videa i podataka. Ova postavka omogućuje aplikaciji na strani klijenta da počne koristiti RTSP strujanje dok se strujanje preuzima. RTSP se može koristiti i za videozapise uživo i za videozapise na zahtjev. Korisnik može uživo gledati sadržaj koji je identičan onome kako bi gledao TV. U isto vrijeme, on ili ona mogu gledati video na zahtjev bez čekanja da se cijela datoteka preuzme [3], [8].

FTL (engl. *Faster Than Light*) protokol je razvila platforma za strujanje *Mixer* u vlasništvu *Microsoft-a*. FTL je protokol za strujanje u stvarnom vremenu, što znači da podržava latenciju ispod sekunde. To omogućuje uključivanje i komuniciranje sa gledateljima u stvarnom vremenu gotovo bez kašnjenja. FTL podržavaju najpopularnije aplikacije za strujanje, uključujući *XSplit* i *OBS Studio*. Također je unaprijed integriran u *Windows 10 OS* i *Xbox One*. Koristi *Opus* audio kodek i H.264 video kodek kako bi omogućio dobru kombinaciju kvalitete, glatke reprodukcije i niske latencije. Loša strana korištenja FTL-a je ta što će strujanje malo izgubiti kvalitetu. *Mixer* preporučuje smanjenje brzine prijenosa na 7 Mbps u usporedbi s RTMP-ovim 10 Mbps. Još jedan od nedostataka FTL-a je nedostatak stabilnosti. Za razliku od svojih predaka, FTL protokol još nije prošao kroz mnogo ispravljanja grešaka, što znači da može biti pomalo nepredvidljiv. Iako je FTL potpuno nov, već je potpuno integriran u *Restream* ekosustav. Do sada samo *Mixer* i *Restream* podržavaju FTL protokol [9].

2.2. Pregled postojećih radova na temu prijenosa video signala korištenjem HLS protokola

U radu [10] je predstavljena društvena obrazovna platforma za strujanje namijenjena prijenosu uživo velikih videozapisa kako bi se olakšala online nastava. Za strujanje velikih videozapisa koristi se HLS protokol koji se temelji na običnim HTTP transakcijama i može proći kroz bilo koji vatrozid ili *proxy* poslužitelj koji dopušta standardni HTTP promet. Upravo strujanje velikih videozapisa pomoću HLS protokola i prethodna pohrana u oblaku (engl. *cloud*) predstavlja novost u radu. Za prethodnu obradu videozapisa koristi se softver *FFmpeg* čije su biblioteke u većini softverskih video reproduktora. *FFmpeg* dolazi s koderima i dekoderima za većinu vrsta audio i video datoteka, što ga čini idealnim za pretvaranje uobičajenih i neuobičajenih medijskih datoteka. Nakon obrade videozapisa, isti se pohranjuje u oblak te se nakon toga krajnjim korisnicima video poslužuje pomoću HLS protokola. Učinkovitost platforme za strujanje može se mjeriti na temelju ciklusa zahtjeva i odgovora te broja odgovora koje može obraditi u sekundi. Za testiranje opterećenja platforme koristi se biblioteka *Locust*. Test je izveden na 100 korisnika i *hatch rate-u* od 2 korisnika/s. HLS protokol omogućuje video reproduktoru prilagodbu nestabilnim mrežnim uvjetima bez zaustavljanja reprodukcije. Izvorni videozapis se kodira na više različitih razina kvalitete kako bi se video reproduktoru omogućilo prilagođavanje propusnosti mreže. Kako se doda novi korisnik u sustav, srednje vrijeme odgovora raste. Dakle, kako se sve više korisnika pridružuje strujanju, vrijeme odgovora se povećava te je za odgovor na zahtjev potrebno je više vremena.

U radu [11] je predstavljen distribuirani online sustav za strujanje uživo baziran na SRS-u (engl. *Simple Realtime Server*) koji obuhvaća više kampusa na sveučilištu Xi'an Jiaotong u gradu Xi'an u Kini. Dinamičkim distribuiranim slanjem (engl. *push*), razdvajanjem uzlaznog i silaznog prometa te mehanizmom uravnoteženja opterećenja na više razina, sustav može zadovoljiti potrebu za normalnim automatskim i nenadziranim video prijenosom nastave uživo u velikim učionicama. U sustavu su komponente distribuirane na svakom kampusu i komuniciraju redovima čekanja poruka i putem HTTP API-ja. Za razliku od tradicionalnog sustava emitiranja, predloženi distribuirani sustav za strujanje uživo ne suočava se s problemima ograničene skalabilnosti, sposobnosti otpornosti na greške i lošeg korisničkog iskustva. Arhitektura distribuiranog sustava za prijenos signala uživo sastoji se od poslužitelja za kontrolu upravljanja, distribuiranog *cluster-a* za slanje videa, distribuirane platforme za prijenos uživo i uravnoteženja opterećenja na više razina. Protokoli koji se koriste u sustavu su RTMP, HTTP-FLV i HLS te se mrežni sustav za strujanje temelji na SRS-u. Rješenje je testirano na *SRSBench* alatu za testiranje opterećenja, a eksperimentalno okruženje je postavljeno u virtualnoj mašini. Test je izveden na videozapisu kojeg *FFmpeg* šalje na lokalni poslužitelj putem RTMP-a s brzinom prijenosa od 2 Mb/s. Platforma za prijenos uživo podržava RTMP, HTTP-FLV i HLS u skladu s različitim zahtjevima raznih platformi. Sustav sadrži mehanizam za uravnoteženje opterećenja na više razina kako bi se poboljšala cjelokupna izvedba sustava i izbjegao kvar na jednoj točki. Prilikom prijenosa videa uživo HLS protokolom može doći do povećane latencije kod distribuirane pohrane objekata.

U radu [12] predložen je učinkovit lokalni mrežni sustav koji pruža visokokvalitetna mrežna okruženja za učenje velikom broju studenata. Predloženi sustav kombinira novo umrežavanje usmjereno na sadržaj (engl. *Content centric networking*, CCN) i prevladavajući HLS protokol. Uvodi se lokalni mrežni sustav za rješavanje problema kvalitete komunikacije u strujanju uživo. Za predloženi sustav usvojen je protokol mrežnog sloja u nastajanju pod nazivom *Information or Content-Centric Networking* (ICN/CCN) koji inherentno podržava *multicasting* i predmemoriranje unutar mreže (engl. *in-network caching*). Sustav se sastoji od dvije mreže: IP mreže koja pruža korisničke aplikacije implementirane od strane HLS-a i CCNx mreže koja inherentno podržava *multicasting* i predmemoriranje unutar mreže implementirano od strane *Cefore-a*. Ove mreže su povezane prevoditeljem između HLS i CCN, koji se naziva HLS-CCNx prevoditelj (HCT). *Cefore* je softver otvorenog koda koji kombinira praktičnost RFC-kompatibilne CCNx komunikacije s lakoćom proširenja novih funkcija za istraživanje. Lokalni mrežni sustav demonstriran je na 60-minutnoj online nastavi u suradnji s Nacionalnim institutom za tehnologiju, *Akashi College*. Nastavnik je snimio video od 2 Mbps koristeći svoj tablet (*iPad*), a studenti (njih

30) su gledali strujanje videozapisa na svojim tabletima. U prilog rješenju ide to da predloženi sustav može smanjiti opterećenje prometa za približno 89% u usporedbi sa sustavom za strujanje videa koji se temelji na HLS-u korištenjem funkcija *multicast-a* i predmemoriranja unutar mreže. Druga prednost je ta da promet može biti stabilniji jer je manje konkurentskog prometa unutar mreže. S druge strane, neki su potrošači iskusili dugo vrijeme kašnjenja preko jedne sekunde zbog nedostatka inteligentne kontrole retransmisije u HCT-u na strani potrošača. S trenutnom jednostavnom politikom ponovnog slanja, kada se paket podataka izgubi, HCT mora čekati do vremenskog ograničenja za odgovarajući zahtjev. U budućem radu mora se razviti inteligentnija kontrola retransmisije.

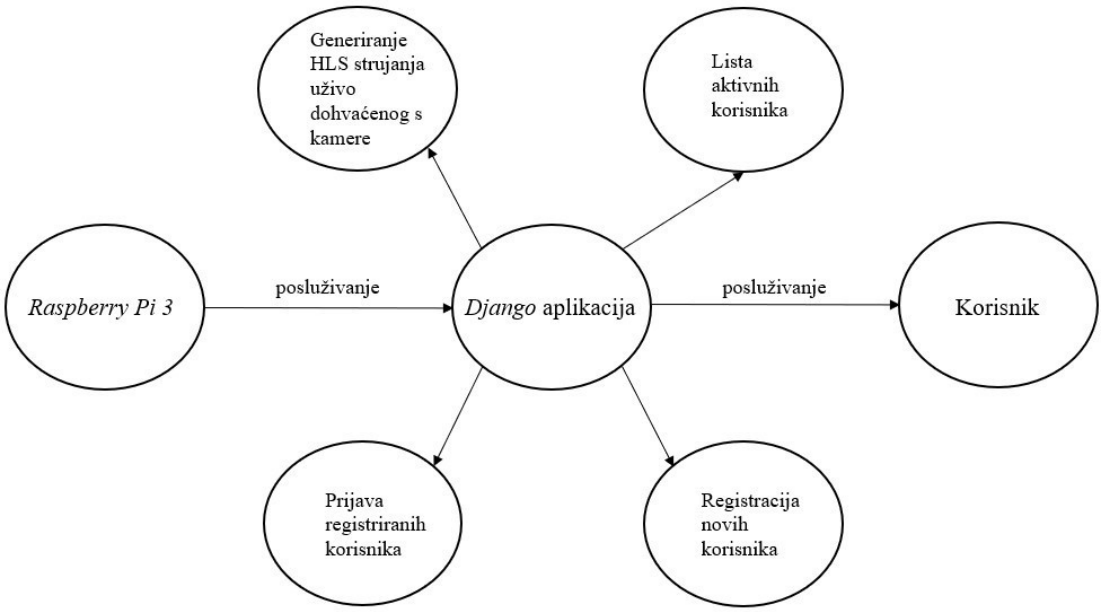
U radu [13] je predstavljen novi način video prijenosa koji maksimizira kvalitetu kada je propusnost komunikacije ograničena. Temeljem HLS protokola, predložena interpolacija na razini segmenta dopušta stvaranje video sekvence koja se prilagođava kvaliteti, a istovremeno smanjuje potrebu za više prostora za pohranu na medijskim poslužiteljima i optimizira iskorištavanje dostupne propusnosti. Drugim riječima, predstavljen je alternativni način za poboljšanje kvalitete HLS protokola potpunim korištenjem dane širine pojasa bez povećanja zahtjeva za pohranom na poslužiteljskom sustavu. Umjesto pohranjivanja više sekvenci strujanja različite kvalitete, predstavljena je shema interpolacije na razini segmenta. Sustavu se dodaju interpolirajući podređeni uređaji, od kojih svaki uključuje samo indeksne podatke za označavanje dodatnih nizova. Za razliku od prethodnog protokola koji uvijek upravlja strujanjem skupova parova indeks-segment, interpolirajući podređeni uređaji u ovom radu koriste segmente iz dva postojeća skupa. Predložena metoda interpolira dva susjedna niza za strujanje na razini segmenta bez stvaranja novog para indeks-segment. Za testiranje je korišteno okruženje otvorenog koda u *FFmpeg-u* za realizaciju HLS protokola koji rukuje s četiri vrste video uzoraka: animacija, predavanje, film i priroda. Ograničavanjem komunikacijske propusnosti, za različite skupove strujanja, mjerila se SSIM (engl. *Structural Similarity Index Measure*) metrika i iskorištenje propusnosti, definirano kao omjer veličine prenesenih podataka i dopuštene veličine. Rezultati simulacije pokazuju da predloženi pristup drastično poboljšava kvalitetu usluga temeljenih na HLS-u korištenjem propusnosti do 102,67% u usporedbi s konvencionalnim rješenjem. Kod videozapisa predavanja, nema poboljšanja SSIM-a za jedan segmentni interpolirani niz što vrijedi i kod iskorištenja komunikacijske propusnosti.

U radu [14] predložena je metoda koja poboljšava kvalitetu strujanja videozapisa uživo pomoću CDN-a (engl. *Content Delivery Network*). Globalna CDN infrastruktura koja se koristi je *Cloudfront*, a podržava ga AWS (engl. *Amazon Web Services*). Poslužitelj se nalazi u Bandungu u

Indoneziji, a klijent je smješten u Tokiju u Japanu. HLS format izvodi se za prijenos sadržaja. Rad uvodi mogućnost korištenja oblak infrastrukture za CDN uslugu. Odabran je AWS jer je popularan davatelj usluga u oblaku. Za strujanje videozapisa uživo izabran je HLS protokol jer održava kvalitetu videa tijekom sesije strujanja. Arhitektura rješenja sastoji se od video kamere za video snimanje. Nakon toga videozapis će se kodirati i pretvoriti u odgovarajući format pomoću OBS-a i *AWS MediaLive-a*. Zatim će se video obraditi putem *AWS MediaStore-a* s HLS-om kao izlaznim video formatom. Rješenje je testirano na spomenutoj arhitekturi u dva scenarija: prvom u kojem se radi HLS strujanje uživo u mreži bez implementacije CDN-a gdje su podatci za strujanje poslani izravno od poslužitelja krajnjem korisniku koji se nalazi u Tokiju i drugom u kojem je implementirana CDN infrastruktura. U tom scenariju kada se prikazuje strujanje videozapisa uživo, događa se kopiranje sadržaja te se iste kopije prenose CDN-u koja to prenosi krajnjem korisniku. U *Wireshark-u* je mjerena propusnost i gubitak paketa. Rezultati testiranja pokazuju da HLS video prijenos uživo s CDN-om daje 4452,6 kbps za prosječnu propusnost. S druge strane, HLS video strujanje uživo bez korištenja CDN-a proizvodi 3990,4 kbps za prosječnu propusnost. HLS video strujanje uživo s CDN-om generira gubitak paketa od 0,08%. S druge strane, HLS video strujanje uživo bez korištenja CDN-a generira gubitak paketa od 0,33%. Za prijenos video sadržaja uživo koristeći CDN infrastrukturu moraju se potrošiti resursi za memoriju CDN servera.

3. IZRADA VLASTITOG RJEŠENJA ZA PRIJENOS VIDEO SIGNALA INTERNET MREŽOM

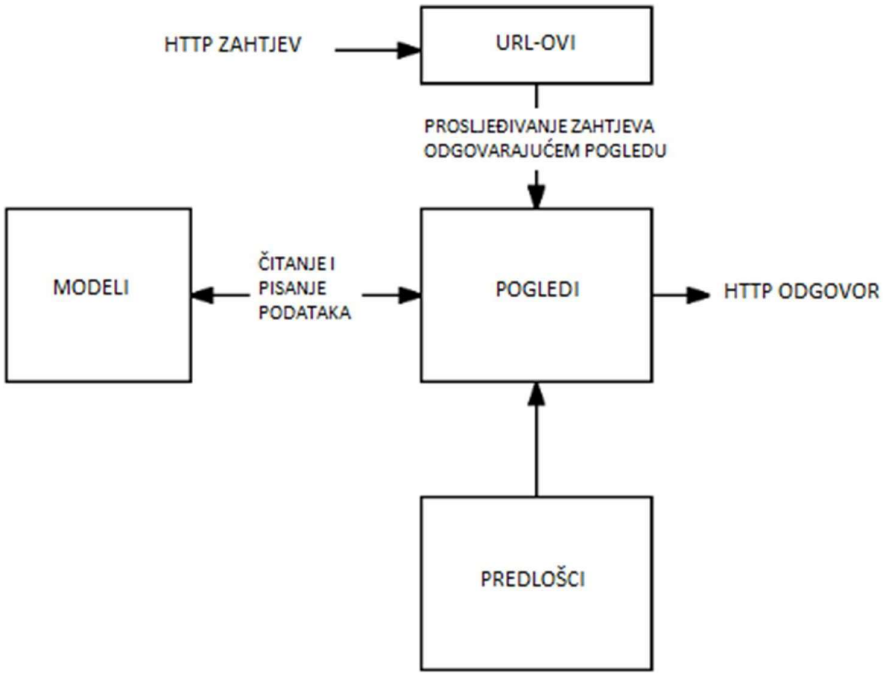
U ovom poglavlju opisan je postupak izrade web aplikacije koja omogućuje strujanje video signala korištenjem HLS protokola. Računalo *Raspberry Pi 3* treba imati ulogu poslužitelja web aplikacije te će se na njemu generirati HLS strujanje. Osim gledanja HLS strujanja, aplikacija treba omogućiti registraciju novih korisnika i prijavu registriranih korisnika. Također, kada korisnik pristupi dijelu aplikacije za gledanje videozapisa, uz videozapis trebao bi i vidjeti listu svih aktivnih korisnika. Aplikacija treba biti izvedena u *Python* programskom jeziku u *Django* web radnom okviru. Uz to, potrebno je koristiti različite web servere za konfiguriranje posluživanja web aplikacije te kako bi lista aktivnih korisnika ispravno funkcionirala. Na slici 3.1. prikazana je blok shema sustava za prijenos video signala Internet mrežom. Aplikacija je prvotno izvedena u virtualnom uređaju u *Linux Mint 21* operacijskom sustavu prije nego li je implementirana na *Raspberry Pi 3*. U potpoglavlju 3.1. opisane su sve korištene tehnologije i alati za izradu web aplikacije. U potpoglavlju 3.2. i njegovim potpoglavljima detaljno je objašnjen postupak rada i izrade web aplikacije od generiranja i reproduciranja strujanja, konfiguriranja servera za posluživanje web aplikacije, dodavanja funkcionalnosti registracije i prijave korisnika do implementacije liste aktivnih korisnika na dijelu aplikacije za gledanje video signala. U potpoglavlju 3.3. opisan je postupak implementacije web aplikacije na računalo *Raspberry Pi 3*.



Slika 3.1. Blok shema vlastitog rješenja za prijenos video signala Internet mrežom

3.1. Opis korištenih tehnologija i alata za izradu rješenja

Za izradu web aplikacije korišten je *Django* web radni okvir otvorenog koda visoke razine koji omogućuje brz razvoj i čist, pragmatičan dizajn web stranica. *Django* se može koristiti za izradu gotovo svih vrsta web stranica od sustava za upravljanje sadržajem do društvenih mreža i stranica s vijestima. Može raditi s bilo kojim okvirom na strani klijenta i isporučiti sadržaj u gotovo svim formatima (uključujući HTML, RSS izvore, JSON i XML). *Django* pomaže programerima da izbjegnu mnoge uobičajene sigurnosne pogreške pružajući okvir koji je projektiran da "čini prave stvari" za automatsku zaštitu web stranice. *Django* koristi arhitekturu temeljenu na komponentama gdje je svaki dio arhitekture neovisan o drugima, te se stoga može zamijeniti ili promijeniti ako je potrebno. *Django* kod je napisan korištenjem načela dizajna i obrazaca koji potiču stvaranje koda koji se može održavati i ponovno koristiti. Napisan je u *Python-u* koji radi na mnogim platformama. To znači da programer nije vezan ni za jednu određenu poslužiteljsku platformu i može pokretati svoje aplikacije na mnogim verzijama *Linux-a*, *Windows-a* i MacOS-a. *Django* web aplikacije obično grupiraju kod u četiri zasebne datoteke: URL-ovi (*urls.py*), *Models* (*models.py*), *Views* (*views.py*) i *Templates* (npr. *index.html*). URL maper koristi se za preusmjeravanje HTTP zahtjeva na odgovarajući pogled (engl. *View*) na temelju URL zahtjeva. URL maper također može uskladiti određene uzorke nizova ili znamenki koje se pojavljuju u URL-u i proslijediti ih funkciji prikaza kao podatke. Pogled je funkcija rukovatelja zahtjevima koja prima HTTP zahtjeve i vraća HTTP odgovore. Pogledi pristupaju podacima potrebnim za zadovoljenje zahtjeva putem modela (engl. *Models*) i delegiraju oblikovanje odgovora predlošcima (engl. *Templates*). Modeli su *Python* objekti koji definiraju strukturu podataka aplikacije i daju mehanizme za upravljanje (dodavanje, mijenjanje, brisanje) i upite zapisa u bazi podataka. Predložak je tekstualna datoteka koja definira strukturu ili izgled datoteke (kao što je HTML stranica), s rezerviranim mjestima koja se koriste za predstavljanje stvarnog sadržaja. Pogled može dinamički stvoriti HTML stranicu pomoću HTML predloška, popunjavajući je podacima iz modela. Predložak se može koristiti za definiranje strukture bilo koje vrste datoteke koja ne mora biti HTML tipa. Na slici 3.2. nalazi se dijagram toka HTTP zahtjeva u *Django* aplikaciji [15], [16].



Slika 3.2. Dijagram toka HTTP zahtjeva u *Django* aplikaciji [15]

Za instalaciju *Django* web radnog okvira, u *Linux* terminalu potrebno je upisati sljedeću naredbu:

python -m pip install django

dok je za kreiranje *Django* projekta potrebno upisati sljedeću naredbu u terminalu:

django-admin startproject nazivprojekt

Za pokretanje *Django-ovog* razvojnog servera koji se koristi pri razvijanju web aplikacije kako bi se ista mogla vidjeti u web pregledniku potrebno je upisati sljedeću naredbu u *Linux* terminalu:

python manage.py runserver

FFmpeg softver predstavlja skup biblioteka koje sadrže funkcije za obradu audio i video sadržaja. To je besplatni multimedijski radni okvir koji pruža mogućnost kodiranja, dekodiranja, transkodiranja, multipleksiranja, demultipleksiranja, strujanja, filtriranja i reproduciranja većine multimedijskog sadržaja. Sloj hardverske apstrakcije visoke razine pruža jednostavnost korištenja te eksplicitna upotreba registra nije potrebna zbog visoke razine apstrakcije. Razvijen je za *GNU/Linux* operacijski sustav iako se može koristiti i na ostalim operacijskim sustavima te je većinom napisan u *C* programskom jeziku. Podržana je većina mikroprocesora i platformi poput

x86, *PowerPC*, ARM, MIPS i sl. *FFmpeg* multimedijski radni okvir sadrži biblioteke *libavcodec*, *libavutil*, *libavformat*, *libavfilter*, *libavdevice*, *libswscale* i *libswresample* koje se mogu koristiti u aplikacijama. Uz to, *FFmpeg* sadrži i nekoliko pomoćnih alata poput *ffmpeg-a*, *ffserver-a*, *ffplay-a* i *ffprobe-a*. Alat *ffmpeg* koristi se za konvertiranje audio/video formata i u ovom radu on se koristi za generiranje segmenata HLS strujanja te manifest datoteke ekstenzije *.m3u8*. Alat *ffserver* je HTTP ili RTSP multimedijski poslužitelj namijenjen emitiranju multimedijskog sadržaja. Aplikacija *ffplay* omogućuje reprodukciju multimedijskih sadržaja, a temelji se na *FFmpeg* bibliotekama i SDL biblioteci. Aplikacija *ffprobe* omogućuje prikaz osnovnih informacija o multimedijskom sadržaju. Za instalaciju *FFmpeg* softvera potrebno je upisati sljedeću naredbu u *Linux* terminalu [2], [17], [18] :

sudo apt install ffmpeg

Nginx je web server otvorenog koda koji omogućuje korištenje predmemorije (engl. *caching*), uravnoteženje opterećenja (engl. *load balancing*), strujanje medijskog sadržaja te služi kao obrnuti *proxy* poslužitelj. Izvorno je dizajniran kao web poslužitelj s visokim performansama i pouzdanošću. Osim što funkcionira kao HTTP poslužitelj, *Nginx* djeluje kao *proxy* poslužitelj za e-poštu (IMAP, POP3 i SMTP). *Nginx* zahtjeva nisku upotrebu memorije i ima visoku konkurentnost. Umjesto stvaranja novih procesa za svaki web zahtjev, *Nginx* koristi asinkroni pristup vođen događajima gdje se zahtjevima rukuje u jednoj niti. Uz *Nginx*, jedan glavni proces može kontrolirati više radnih procesa. Glavni proces održava radne procese, dok radni procesi obavljaju stvarnu obradu. Budući da je *Nginx* asinkron, svaki zahtjev može izvršiti radni proces istovremeno bez blokiranja drugih zahtjeva. U većini slučajeva, preporučena *Nginx* konfiguracija jednog radnog procesa po CPU jezgri predstavlja najučinkovitiju upotrebu hardverskih resursa. Broj radnih procesa može se prilagoditi postavljanjem direktive *worker_processes* u *Nginx* konfiguraciji. Kada je *Nginx* poslužitelj aktivan, samo je jedan radni proces zauzet. Svaki radni proces je jednonitni i izvodi se neovisno za dobivanje i obradu novih veza. Procesi mogu komunicirati korištenjem zajedničke memorije za dobivanje zajedničkih podataka predmemorije, trajnih podataka sesije i drugih zajedničkih resursa. Postoje četiri koncepta prilikom rada s *Nginx* konfiguracijom: direktive, blokovi, kontekst i poslužiteljski blokovi. Direktive su upute koje govore *Nginx-u* što treba učiniti. Sastoje se od naziva i jednog ili više parametara. Direktive se mogu postaviti na različite razine u konfiguracijskoj datoteci, a razina na kojoj se direktiva postavlja određuje njezin opseg i način na koji se tumači. Blokovi su zbirke direktiva koje su grupirane zajedno. Okruženi su vitičastim zagradama i mogu sadržavati druge blokove. Blokovi se mogu ugnijezditi kako bi se stvorila hijerarhija konfiguracijskih postavki. Kontekst u koji je

postavljena direktiva određuje njezino značenje i način na koji se primjenjuje. *Nginx* ima nekoliko različitih konteksta, uključujući glavni kontekst, koji se primjenjuje na cijeli *Nginx* poslužitelj, i kontekst poslužitelja, koji se primjenjuje na određeni blok poslužitelja. Blok poslužitelja je blok direktiva koje definiraju konfiguraciju za određeni virtualni poslužitelj. Virtualni poslužitelji omogućuju posluživanje više web stranica na jednoj *Nginx* instanci određivanjem različitih konfiguracija za svaki blok poslužitelja. U radu se *Nginx* koristi kao virtualni poslužitelj za *Django* web aplikaciju. Kako bi se instalirao *Nginx*, potrebno je u *Linux* terminalu upisati sljedeću naredbu [19], [20], [21]:

sudo apt install nginx

Green Unicorn, skraćeno, *Gunicorn*, implementacija je poslužitelja WSGI (engl. *Web Server Gateway Interface*) koja se obično koristi za pokretanje *Python* web aplikacija. WSGI pruža most za komunikaciju između web poslužitelja i web aplikacije. To je skup pravila koja omogućuju WSGI kompatibilnom poslužitelju da radi s WSGI kompatibilnom *Python* aplikacijom. WSGI također upravlja skaliranjem za web poslužitelje kako bi mogao obraditi tisuće zahtjeva tako da programer ne mora razmišljati o prihvaćanju više zahtjeva odjednom. *Gunicorn* prima zahtjeve poslane web poslužitelju od klijenta i prosljeđuje ih na *Python* aplikacije ili web radne okvire (kao što su *Flask* ili *Django*) kako bi se pokrenuo odgovarajući kod aplikacije za zahtjev. *Gunicorn* zna kako pokrenuti web aplikaciju temeljenu na poveznici (engl. *hook*) između WSGI poslužitelja i WSGI kompatibilne web aplikacije. *Gunicorn* s *Django-om* funkcionira tako da unutar programskog koda treba kreirati funkciju koja se može pozvati i koja će za *Gunicorn* biti ulazna točka u aplikaciju. Kada je *Django* projekt kreiran, unutar njega treba postojati datoteka *wsgi.py* koja sadrži varijablu *application*. *Django* izlaže varijablu *application* putem *wsgi.py* datoteke tako da WSGI poslužitelj može koristiti *application* kao poveznicu za pokretanje web aplikacije. U ovom radu, *Gunicorn* predstavlja poveznicu između web aplikacije i *Nginx* web poslužitelja. *Gunicorn* se instalira upisivanjem sljedeće naredbe u *Linux* terminalu [22], [23], [24] [25] :

pip install gunicorn

Daphne je poslužitelj HTTP, HTTP2 i *WebSocket* protokola za ASGI i ASGI-HTTP, razvijen za pokretanje *Django* kanala (engl. *Django Channels*). ASGI (engl. *Asynchronous Server Gateway Interface*) nasljednik je WSGI-ja, namijenjen pružanju standardnog sučelja između asinkronih *Python* web poslužitelja, okvira i aplikacija. ASGI dopušta višestruke dolazne događaje i odlazne događaje za svaku aplikaciju, ali također dopušta pozadinsku korutinu (engl. *coroutine*)

tako da aplikacija može raditi druge stvari. ASGI je također dizajniran da bude iznad WSGI-ja i postoji definiran način prevođenja između njih dvoje, što omogućuje pokretanje WSGI aplikacija unutar ASGI poslužitelja putem omotača prijevoda (koji se nalazi u biblioteci *asgiref*). Skup niti se može koristiti za pokretanje sinkronih WSGI aplikacija izvan asinkrone petlje događaja. S *Django-om* funkcionira tako da unutar programskog koda treba kreirati funkciju koja se može pozvati i koja će za *Daphne* biti ulazna točka u aplikaciju. Kada je *Django* projekt kreiran, unutar njega treba postojati datoteka *asgi.py* koja sadrži varijablu *application*. Django izlaže varijablu *application* putem *asgi.py* datoteke tako da ASGI poslužitelj može koristiti *application* kao poveznicu za pokretanje web aplikacije. Jedna od primarnih prednosti korištenja *Daphne-a* je njegova sposobnost rukovanja velikim brojem istodobnih veza na asinkroni način, pružajući izvrsnu skalabilnost i odziv. To se postiže korištenjem arhitekture vođene događajima, što omogućuje da istovremeno obrađuje više zahtjeva bez blokiranja ili trošenja prekomjernih resursa sustava. *Daphne-ova* integracija s *Django* kanalima čini je posebno prikladnom za razvoj aplikacija koje zahtijevaju komunikaciju u stvarnom vremenu, kao što su aplikacije za *chat*, sustavi obavijesti ili alati za suradnju. Uz *Daphne*, može se jednostavno koristiti *WebSocket* protokol za uspostavljanje dvosmjerne komunikacije između poslužitelja i klijenta, omogućujući trenutna ažuriranja i interakcije. U radu se *Daphne* ASGI poslužitelj koristi za implementaciju dijela aplikacije vezan uz listu aktivnih korisnika te služi kao veza između *Nginx* poslužitelja i *Django* kanala u web aplikaciji. Za instalaciju *Daphne* ASGI poslužitelja u *Linux* terminalu potrebno je napisati sljedeću naredbu [26], [27]:

python -m pip install daphne

Django kanali (engl. *Django Channels*) obuhvaćaju *Django-ovu* izvornu podršku za asinkroni prikaz, omogućujući *Django* projektima da rukuju ne samo HTTP-om, već i protokolima koji zahtijevaju dugotrajne veze: *WebSockets*, MQTT, *chatbot-ovi* i sl. To čini uz očuvanje *Django-ove* sinkrone prirode i jednostavne za korištenje, omogućujući programeru da odabere kako će pisati svoj kod: sinkroni u stilu poput *Django* pogleda, potpuno asinkroni ili korištenjem oba načina. Povrh toga, pruža integraciju s *Django-vim* sustavom autentikacije, sustavom sesija i s još mnogo toga, što olakšava proširenje HTTP projekta na druge protokole. Kanali također povezuju ovu arhitekturu vođenu događajima sa slojevima kanala (engl. *channel layers*), sustavom koji omogućuje jednostavnu komunikaciju između procesa i odvajanje projekta u različite procese. *Django* kanali i ASGI dijele dolazne veze u dvije komponente: opseg (engl. *scope*) i niz događaja (engl. *event*s). Opseg je skup detalja o jednoj dolaznoj vezi kao što je putanja s kojeg je poslan web zahtjev, ili izvorna IP adresa *WebSocket-a*, ili korisnik koji šalje poruku *chatbot-u*. Opseg

ostaje prisutan tijekom cijele veze. Za HTTP, opseg traje samo jedan zahtjev. Za *WebSocket*, traje za vrijeme trajanja *socket-a* (ali se mijenja ako se *socket* zatvori i ponovno poveže). Potrošač (engl. *consumer*) je osnovna jedinica koda *Django* kanala. Naziva se tako jer konzumira događaje, ali ga se može gledati kao manju aplikaciju. Kada stigne novi HTTP zahtjev ili *socket*, kanali će slijediti svoju tablicu usmjeravanja (engl. *routing table*), pronaći pravog potrošača za tu dolaznu vezu te pokrenuti njegovu kopiju. Više potrošača mogu se kombinirati u jednu veću aplikaciju koja predstavlja projekt pomoću usmjeravanja (engl. *routing*). U aplikaciji, *Django* kanali se koriste za osvježavanje liste aktivnih korisnika. Za instalaciju *Django* kanala, potrebno je upisati sljedeću naredbu u *Linux* terminalu [28]:

pip install django-channels

WebSocket je dvosmjeran, *full-duplex* protokol koji se koristi u istom scenariju komunikacije između klijenta i poslužitelja, a za razliku od HTTP-a, počinje s *ws://* ili *wss://*. To je protokol s praćenjem stanja, što znači da će veza između klijenta i poslužitelja ostati živa sve dok je ne prekine bilo koja strana (klijent ili poslužitelj). Nakon zatvaranja veze od strane klijenta i poslužitelja, veza se prekida s oba kraja. Web preglednik koji predstavlja klijenta šalje WS zahtjev poslužitelju za uspostavu veze. Nakon uspostavljanja komunikacijske veze, razmjena poruka odvijat će se u dvosmjernom načinu sve dok veza između klijenta i poslužitelja traje. *WebSocket* se često koristi u aplikacijama u stvarnom vremenu, *gaming* aplikacijama i *chat* aplikacijama. U radu se *WebSocket* koristi kao *front-end* dio aplikacije za osvježavanje liste aktivnih korisnika [29].

Redis je *in-memory* struktura pohrane podataka otvorenog koda koja se koristi kao baza podataka, predmemorija, broker poruka i mehanizam za strujanje. *Redis* omogućuje kreiranje struktura podataka kao što su nizovi, *hash-ovi*, popisi, skupovi, sortirani skupovi s upitima raspona, bitmape, hiperlogovi, geoprostorni indeksi i tokovi. Na ovim tipovima se mogu izvoditi operacije poput dodavanja u niz, povećanje vrijednosti u *hash-u*, stavljanje elementa u listu, presjek, unija i razlika računskog skupa i dobivanje člana s najvišim rangom u sortiranom skupu. Kako bi postigao vrhunsku izvedbu, *Redis* radi s *in-memory* skupom podataka. Ovisno o slučaju korištenja, *Redis* može zadržati podatke bilo povremenim ispisivanjem skupa podataka na disk ili dodavanjem svake naredbe u zapisnik na disku. *Redis* podržava asinkronu replikaciju, s brzom sinkronizacijom bez blokiranja i automatskim ponovnim povezivanjem s djelomičnom ponovnom sinkronizacijom na podjeli mreže. *Redis* se u *Django* aplikaciji koristi sa slojevima kanala (engl. *Channel Layers*) koji omogućuju komunikaciju između različitih instanci aplikacije (potrošača). *Redis* sloj kanala

je jedini službeni sloj kanala koji *Django* podržava za produkciju. Primarna svrha *Redis-a* u Django kanalima je pohranjivanje potrebnih informacija potrebnih različitim instancama potrošača za međusobnu komunikaciju. *Redis* se koristi kao sloj za pohranu naziva kanala i naziva grupa. One su pohranjene unutar *Redis-a* tako da im se može pristupiti s bilo koje potrošačke instance. U radu *Redis* čuva informaciju o grupi kanala koji izmjenjuju informacije kada se neki korisnik prijavljuje ili odjavljuje. Za instalaciju *Redis* servera treba upisati sljedeću naredbu u *Linux* terminalu [30], [31]:

sudo apt install redis-server

3.2. Opis rada web aplikacije i algoritma za strujanje video signala

Web aplikacija za strujanje video signala korištenjem HLS protokola sadrži nekoliko funkcionalnosti na različitim HTML dokumentima: stranica za prijavu registriranih korisnika, stranica za registraciju novih korisnika te stranica za gledanje videozapisa uživo s web kamere koja također sadrži listu aktivnih korisnika. Na stranici za prijavu registriranih korisnika nalaze se polja za upis adrese elektroničke pošte i lozinke korisnika, tipka za prijavu i odlazak na dio stranice za gledanje videozapisa te poveznica za odlazak na stranicu za registraciju novog korisnika. Kada se dođe na stranicu za registraciju korisnika, mogu se vidjeti polja za unos punog imena korisnika, korisničkog imena korisnika, adrese elektroničke pošte, lozinke te polje za ponovni unos lozinke, odnosno njezinu potvrdu. Također, stranica sadrži tipku za registraciju i istovremenu prijavu ukoliko su svi podatci dobro uneseni te poveznicu za povratak na stranicu za prijavu korisnika. Kada se pritisne gumb za prijavu ili registraciju i ukoliko su uneseni podatci u redu, odlazi se na stranicu za gledanje videozapisa uživo s web kamere koja je prvotno spojena na osobno računalo, a kasnije na *Raspberry Pi 3*. Na stranici za gledanje video signala uživo nalazi se HTML 5 reproduktor za reprodukciju HLS strujanja, lista korisnika koji upravo gledaju videozapis, tipka za odjavu s kojom se odlazi na početnu stranicu za prijavu te oznaka s imenom prijavljenog korisnika. Na slici 3.3. nalazi se dijagram toka navigacije kroz web aplikaciju.

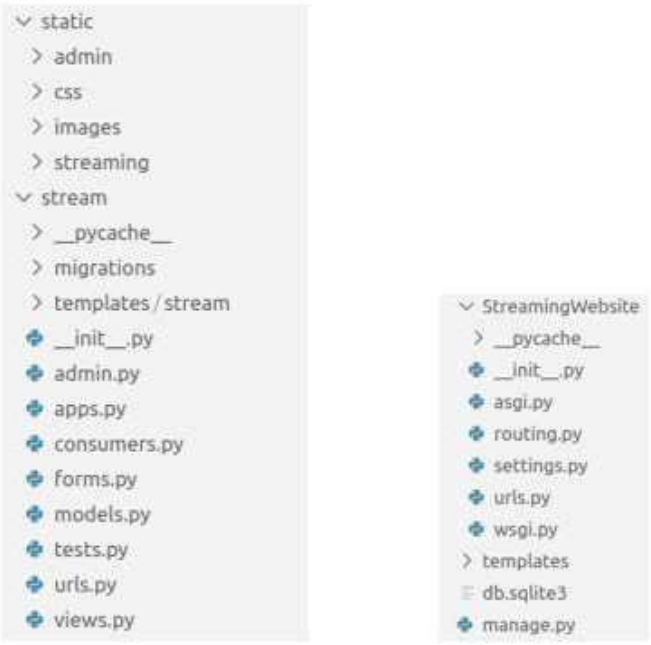
Slika 3.3. Dijagram toka navigacije kroz web aplikaciju

Na slici 3.4. prikazao je stablo direktorija *Django* projekta web aplikacije koji se naziva *StreamingWebsite*. U prvom dijelu stabla u prvom direktoriju *static* nalaze se statičke datoteke (engl. *static files*) koje krajnjem korisniku poslužuje izravno *Nginx* web poslužitelj. U podmapi *admin* nalaze se statičke datoteke osnovnog projekta koje se dobiju upisivanjem sljedeće naredbe:

python manage.py collectstatic

Podmapa *css* sadrži CSS (engl. *Cascading Style Sheets*) datoteku projekta, podmapa *images* sadrži fotografije koje se nalaze u projektu, a direktorij *streaming* sadrži manifest datoteku i generirane segmente HLS strujanja dohvaćenog uživo s kamere. Direktorij *stream* predstavlja jedinu aplikacijsku komponentu projekta koja definira većinu funkcionalnosti projekta. Poddirektorij *migrations* sadrži sve migracije, odnosno promjene unutar *SQLite* baze podataka projekta. Poddirektorij *stream* unutar direktorija *templates* sadrži HTML dokumente koji definiraju elemente vidljive na stranicama. U poddirektoriju *admin.py* registriraju se nove tablice u bazi podataka kako bi bile vidljive u administratorskom panelu u bazi podataka. U *apps.py* definira se aplikacijska komponenta *stream* kako bi bila vidljiva u projektu. Datoteka *consumers.py* definira funkcionalnosti za *Django* kanale, odnosno potrošače koji omogućavaju osvježavanje liste s aktivnim korisnicima. Unutar *forms.py* definirana je forma za registraciju korisnika, odnosno koja će polja biti u formi za registraciju korisnika. Datoteka *models.py* definirane modele, odnosno tablice u bazi podataka. U *urls.py* definirani su URL-ovi koji su povezani sa svakim pojedinim pogledom. Unutar *views.py* se nalaze pogledi koji definiraju većinu funkcionalnosti stranice i koji

renderiraju vidljivi sadržaj web aplikacije, odnosno HTML dokumente. Na slici 3.4. u b) dijelu stabla direktorija *Django* projekta nalazi se poddirektorij *StreamingWebsite* koji predstavlja glavni poddirektorij *Django* aplikacije. Datoteka *asgi.py* predstavlja sučelje za asinkronu komunikaciju između *Django* aplikacije i ASGI servera poput *Daphne-a* koji dalje komunicira s *Nginx* web poslužiteljem. Ta datoteka pokreće asinkroni dio aplikacije vezan uz osvježavanje liste korisnika. U *Django* kanalima, datoteka *routing.py* koristi se za definiranje konfiguracije usmjeravanja za *WebSocket* veze. Također, određuje kako treba rukovati dolaznim *WebSocket* vezama i usmjeravati ih odgovarajućim potrošačima definiranim u datoteci *consumers.py*. Datoteka *settings.py* u *Django-u* je *Python* modul koji sadrži različite postavke i konfiguracije za *Django* projekt i predstavlja ključnu datoteku za prilagodbu ponašanja *Django* aplikacije. U *urls.py* unutar *StreamingWebsite* direktorija uključeni su URL obrasci iz aplikacijske komponente *stream* zajedno s URL-om koji pokazuje na administratorski panel za bazu podataka. Datoteka wsgi.py u *Django-u* koristi se za posluživanje *Django* aplikacije koristeći WSGI protokol. Njezina glavna svrha je pružiti objekt koji se može pozvati i koji služi kao ulazna točka za *Gunicorn* WSGI poslužitelj za interakciju s *Django* aplikacijom. Pozivanjem funkcije *get_wsgi_application() Gunicorn* server pokreće preostali dio aplikacije. Unutar *wsgi.py* također se nalazi kod za generiranje segmenata HLS strujanja i manifest datoteke direktno s web stranice. Unutar direktorija *templates* nalazi se glavni HTML dokument naziva *main.html* koji predstavlja glavnu strukturu za čitav projekt. Ostali predlošci ga proširuju ili uključuju kako bi naslijedili njegovu strukturu i sadržaj. Oznake *{% block content %}* i *{% endblock %}* definiraju blok koji drugi predlošci mogu prepisati kako bi umetnuli svoj određeni sadržaj unutar predloška *main.html*. Datoteka *db.sqlite3* odnosi se na zadanu *SQLite* datoteku baze podataka koja se automatski stvara kada se postavi *Django* projekt. *SQLite* je baza podataka bez poslužitelja koja pohranjuje podatke u jednu datoteku. Tijekom razvoja, mogu se koristiti *Django* ugrađene upravljačke naredbe, kao što su *migrate* i *makemigrations*, za upravljanje promjenama sheme baze podataka i izvođenje operacija baze podataka koristeći *SQLite*. Zadnja datoteka unutar direktorija projekta je *manage.py* datoteka koje se obično koristi za obavljanje administrativnih zadataka, kao što je pokretanje razvojnog poslužitelja, izvršavanje migracija baze podataka, stvaranje superkorisnika i više. Zapravo služi kao ulazna točka za upravljanje i kontrolu *Django* projekta iz naredbenog retka.

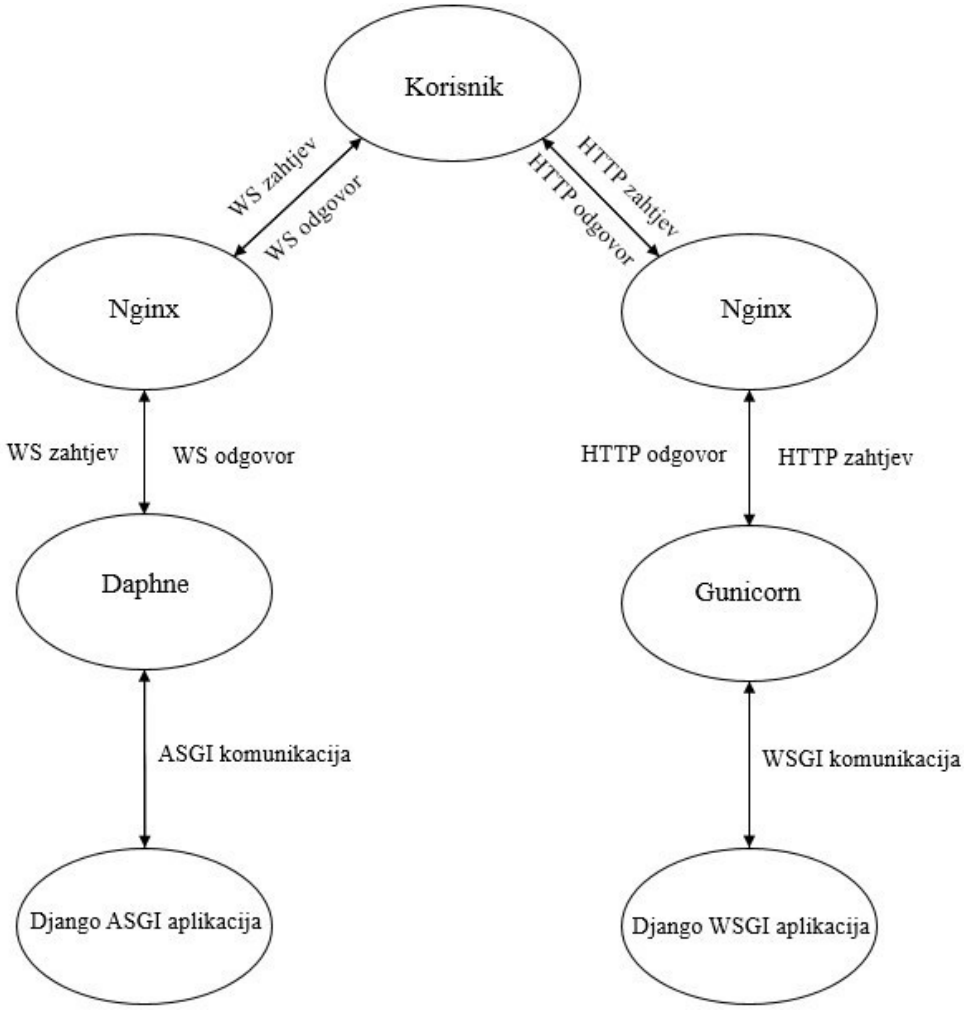


a) b)

Slika 3.4. Stablo direktorija Django projekta web aplikacije, a) prvi dio stabla, b) drugi dio stabla

Rad web aplikacije za HLS strujanje videozapisa uživo dohvaćenog s web kamere u produkciji sadrži nekoliko točaka. Na vrhu hijerarhije nalazi se *Nginx* poslužitelj koji ima tri različite funkcije. Prva je da predstavlja obrnuti *proxy* poslužitelj koji zahtjeve od klijenta prosljeđuje *Gunicorn* WSGI poslužitelju koji komunicira s *Django* aplikacijom WSGI protokolom. Na taj način je omogućena većina funkcionalnosti *Django* aplikacije koje se nalaze u pogledima te je omogućeno generiranje segmenata i manifest datoteke HLS strujanja. Druga funkcija *Nginx-a* koju obavlja je obrnuti *proxy* poslužitelj koji *WebSocket* klijentske zahtjeve za osvježivanjem liste aktivnih korisnika prosljeđuje *Daphne* ASGI poslužitelju koji komunicira s *Django* aplikacijom ASGI protokolom. Tako je omogućeno pokretanje asinkronog dijela *Django* aplikacije koji omogućava osvježavanje liste aktivnih korisnika. Treća funkcija *Nginx-a* je posluživanje statičkih datoteka koje *Nginx* izravno prosljeđuje klijentu preko HTTP zahtjeva. Kao što je ranije rečeno, u statičke datoteke spadaju fotografije, CSS datoteke, *Javascript* datoteke i ostale medijske datoteke. Među tim medijskim datotekama nalaze se manifest datoteka i video segmenti HLS strujanja koje *Nginx* poslužitelj prosljeđuje HLS reproduktoru te je na taj način omogućeno HLS strujanje. Ispod *Nginx* poslužitelja nalaze se *Gunicorn* i *Daphne* poslužitelj. *Gunicorn* poslužitelj dohvaća objekt WSGI aplikacije koji se nalazi unutar *wsgi.py* datoteke koja također sadrži kod za generiranje HLS strujanja s tri kvalitete koji se pokreće nakon što se pokrene WSGI dio *Django* aplikacije. *Daphne*

poslužitelj dohvaća objekt ASGI aplikacije koji se nalazi unutar *asgi.py* datoteke. Time je omogućeno da se lista trenutno aktivnih korisnika svakog prijavljenog korisnika na stranici za gledanje video signala osvježava pri svakoj odjavi ili prijavi na stranicu za strujanje. To se događa jer pri svakoj prijavi ili odjavi korisnika, *Django* aplikacija šalje poruku prijavljenim korisnicima putem *Django* kanala, a na korisničkoj strani se pomoću *WebSocket* komunikacije dobiva ta ista poruka. Na prijem te poruke pokreće se kod za osvježivanje liste aktivnih korisnika. Kako bi poruke dolazile svim aktivnim korisnicima, a ne samo onom koji se prijavljuje, koriste se slojevi kanala. Slojevi kanala za svakog klijenta stvaraju kanal, a svi kanali su povezani u grupu kanala. Tako se pri prijavi ili odjavi korisnika šalje poruka svim kanalima u grupi te je tako omogućeno osvježivanje liste svim aktivnim korisnicima. Kako bi informacije o grupi kanala i svakom pojedinom kanalu bile sačuvane, koristi se *Redis* koji predstavlja *backend* dio za slojeve kanala. Na slici 3.5. nalazi se dijagram toka rada web aplikacije.



Slika 3.5. Dijagram toka rada web aplikacije

3.2.1. Generiranje manifest datoteke i segmenata videa za HLS strujanje korištenjem *FFmpeg* alata

Kao što je navedeno u poglavlju 3.1., za generiranje manifest datoteke ekstenzije *.m3u8* i segmenata videa HLS strujanja koristi se softver *FFmpeg* i njegov alat *ffmpeg*. Video se dohvaća s web kamere *Fantech Luminuos C30* čija je video rezolucija 2560x1440p, a omogućuje generiranje 25 okvira/s. Kod za generiranje HLS strujanja nalazi se u *wsgi.py* datoteci i pokreće se nakon što se pokrene WSGI dio *Django* aplikacije. Budući da se *ffmpeg* kod uobičajeno piše u naredbenom retku *Linux* terminala, bilo je potrebno pronaći način kako ga pokrenuti unutar *Python* skripte u *Django* aplikaciji. Rješenje je postignuto uporabom *Python-ovog* modula *subprocess* koji pruža način pokretanja novih procesa, spajanja na njihove ulazne cijevi, izlazne cijevi i cijevi pogrešaka te dobivanja njihovih povratnih kodova. Omogućuje interakciju sa sistemskim naredbama i drugim programima unutar *Python* koda. Prije nego li se pokrene kod za generiranje HLS strujanja, potrebno je obrisati sve što se prethodno nalazilo u poddirektoriju *streaming* unutar direktorija *static* kako reproduktor ne bi dohvatio segmente koji su generirani prije pokretanja aplikacije. Brisanje datoteka unutar direktorija *streaming* postiže se pomoću *Python* koda danog u prilogu P.3.1. Zatim je potrebno otići do direktorija *streaming* kako bi se osiguralo da se segmenti i manifest datoteka generiraju unutar tog direktorija. To se postiže sljedećom naredbom:

os.chdir('/home/luka/StreamingWebsite/static/streaming')

Nakon toga se pomoću *subprocess* modula otvara novi proces u kojem se pokreće kod za generiranje HLS strujanja s tri različite kvalitete koji se nalazi u prilogu P.3.2. U tom kodu ulazni format videa je YUV format boja s 4:2:2 shemom poduzorkovanja. Format *v4l2* koristi se dohvaćanje videa s kamere. Rezolucija ulaznog videa je 1920x1080. Dio koda */dev/video0* predstavlja ulazni video uređaj s kojeg se dohvaća video. Opcija *preset ultrafast* govori da se koristi najbrže moguće kodiranje uz najmanju moguću kompresiju. Tako je osigurano da kašnjenje strujanja bude što manje. Dio koda *-g 25 -sc_threshold 0* postavlja veličinu GOP-a (engl. *Group Of Pictures*) na 25 okvira te prag promjene scene na 0. Nakon toga se ulazni video skalira i mapira u tri različita strujanja s različitim rezolucijama i kodnim brzinama. Prvo strujanje ima rezoluciju ulaznog videa i kodira se kodnom brzinom od 3600 kbit/s. Drugo strujanje ima rezoluciju 1280x720 i kodnu brzinu 2424832 bit/s dok treće strujanje ima rezoluciju 854x480 i kodnu brzinu 964608 bit/s. Kao standard za kodiranje sva tri strujanja koristi se H.264. Zatim se sa *-f hls* određuje HLS kao izlazni format. Opcija *hls_time 0.5* znači da će trajanje segmenta HLS strujanja iznositi 0.5 s. Dio koda *hls_segment_type mpegts* označava da su segmenti tipa MPEG TS (engl. *Transport Stream*). Zastavica *delete_segments* označava da će se sačuvati samo nekoliko

posljednje kreiranih segmenata kako bi se sačuvala memorija. Ovim kodom sačuvat će se šest posljednje generiranih segmenata strujanja. Oznaka *-master_pl_name master.m3u8* pokazuje kako će se zvati glavna manifest datoteka. Dio koda *-var_stream_map "v:0 v:1 v:2" stream_%v.m3u8* kreira sporedne manifest datoteke mapiranjem pojedine kvalitete HLS strujanja. Proces generiranja HLS strujanja se događa neprekidno sve dok radi WSGI dio aplikacije kojem pripadaju sve funkcionalnosti koje obrađuju pogledi. *Nginx* će generirane segmente i manifeste dohvatiti kao statičke datoteke te će ih poslužiti HLS reproduktoru na korisničkoj strani.

3.2.2. Implementacija reproduktora za reprodukciju HLS strujanja

Za implementaciju reproduktora za prikazivanje video signala HLS strujanja uživo dohvaćenog s web kamere korištene su dvije *Javascript* biblioteke: *Plyr* [32] i *hls.js* [33]. *Plyr* je popularna JavaScript biblioteka za izradu prilagođenih video reproduktora na webu. Omogućuje dosljedno i prilagodljivo sučelje za kontrolu i prikaz video sadržaja na različitim platformama i uređajima. Uz *Plyr* se može izraditi i stilizirati vlastiti video reproduktor koristeći HTML, CSS i *JavaScript*. Podržava širok raspon video formata i pruža značajke kao što su reprodukcija/pauza, kontrola glasnoće, način rada preko cijelog zaslona, titlovi i još mnogo toga. Upravo se i u web aplikaciji *Plyr* omogućava izgled web reproduktora i njegove opcije. Biblioteka *hls.js* omogućuje reprodukciju HLS strujanja u web preglednicima koji izvorno ne podržavaju HLS. Funkcionira pomoću *JavaScript-a* za analizu i rukovanje HLS manifestima i segmentima, a zatim renderiranje videozapisa omogućuje pomoću HTML5 video elemenata. Pruža robusno i pouzdano rješenje za reprodukciju HLS videa na različitim preglednicima i platformama. U web aplikaciji, *hls.js* omogućuje automatsko izmjenjivanje kvaliteta strujanja na temelju mrežne propusnosti te omogućuje korisničko mijenjanje kvalitete po izboru. U glavnom predlošku *main.html* nalaze se sve *script* oznake za *hls.js* i *Plyr* bez kojih biblioteke ne bi ispravno radile. Uz to, nalazi se i *link* oznaka za uključivanje CSS datoteka za *Plyr*. U poddirektoriju *stream* direktorija *templates* aplikacijske komponente *stream* nalazi se HTML dokument *stream.html* u kojemu je napisan HTML i *Javascript* kod za HLS reproduktor. HTML kod reproduktora sastoji se od jednog *video* elementa i *div* elementa unutar kojeg je isti video element enkapsuliran. *Javascript* kod za reproduktor napisan je unutar *script* elementa. Na početku se postavlja izvor strujanja, odnosno putanja do manifest datoteka i segmenata HLS strujanja. Zatim je postavljena konfiguracija reproduktora kojom je omogućeno da reproduktor radi u modu niske latencije te da dohvaća zadnji generirani segment. Time je omogućeno da kad se korisnik prijavi na stranicu za gledanje video signala, bit će mu prikazani najnoviji dijelovi strujanja. Na taj način kašnjenje strujanja će biti smanjeno. Nadalje se u kodu inicijalizira reproduktor i dodjeljuje mu se izvor strujanja. Nakon

toga se na događaj parsiranja manifesta inicijaliziraju opcije reproduktora poput reprodukcije, pauze, premotavanja, načina rada preko cijelog ekrana i sl. Zatim se uvodi funkcionalnost za izmjenu kvaliteta pri čemu se prvo dohvaćaju dostupne kvalitete koje se nalaze u manifestu. Omogućeno je da kad se u izborniku za odabir kvalitete odabere opcija AUTO, tada je uključeno automatsko izmjenjivanje kvaliteta. Osim toga, omogućen je izbor bilo koje kvalitete koja je generirana. Funkcija *updateQuality(newQuality)* omogućuje korisnički odabir kvaliteta pri čemu korisnik može odabrati i opciju AUTO za automatsku izmjenu kvaliteta.

Na slici 3.6. prikazan je reproduktor web aplikacije za HLS strujanje videozapisa uživo dohvaćenog s web kamere. Na reproduktoru od opcija je ponuđena reprodukcija/pauza, premotavanje naprijed i nazad, traka s vremenom, trajanje, način rada preko cijelog zaslona te gumb za postavke. Kada se pritisne gumb za postavke, ponuđeni su izbornici za promjenu brzine reprodukcije i promjenu kvalitete. Izbornik za promjenu kvalitete vidljiv je na slici 3.6. gdje je označena opcija za automatsku izmjenu kvaliteta. U zagradi je kraj opcije AUTO prikazana je kvaliteta strujanja koja se trenutno reproducira. Ispod se nalaze sve kvalitete koje su generirane. Prvo vrijeme slijeva koje je prikazano odnosi se na vrijeme koje je proteklo otkada korisnik gleda videozapis bez prekida ili pauze dok drugo vrijeme prikazuje koliko je trajanja HLS strujanja generirano također gledajući od trenutka kada je korisnik započeo gledanje video signala. Budući da je razlika ta dva vremena dvije sekunde, može se zaključiti da je minimalno kašnjenje strujanja dvije sekunde.

Slika 3.6. HLS reproduktor s prikazanim izbornikom za izmjenu kvaliteta

3.2.3. Konfiguriranje *Nginx* web poslužitelja i *Gunicorn* WSGI poslužitelja

Nakon što se prethodno instaliraju pomoću naredbi navedenih u poglavlju 3.1., potrebno je napraviti valjanu konfiguraciju jednog i drugog poslužitelja kako bi *Django* WSGI aplikacija bila ispravno postavljena u produkciju. Pri tome je preporučeno da se *Gunicorn* instalira unutar virtualnog okruženja pomoću alata *virtualenv*. Ono stvara izolirano *Python* okruženje u kojem se mogu instalirati određene verzije paketa i biblioteka bez utjecaja na *Python* instalaciju u sustavu ili drugim projektima. Kod konfiguriranja *Gunicorn-a*, prvo je potrebno u *Linux* naredbenom retku stvoriti *Python* datoteku koja će predstavljati konfiguraciju. Unutar *home* direktorija to će se postići idućom naredbom:

nano conf/gunicorn_config1.py

gdje je *conf* poddirektorij unutar kojeg se nalazi *Python* datoteka *gunicorn_config1.py* kojom će se pokrenuti WSGI dio *Django* aplikacije. Unutar datoteke je navedeno sljedeće:

```
command = '/home/luka/django-env/bin/gunicorn'
pythonpath = '/home/luka/StreamingWebsite'
bind = '10.0.2.15'
workers = 3
```

gdje varijabla *command* predstavlja putanju do naredbe za pokretanje *Gunicorn* poslužitelja. Varijabla *pythonpath* predstavlja putanju do *Django* aplikacije, a varijabla *bind* sadrži IP adresu na kojoj će *Gunicorn* osluškivati nadolazeći promet od *Nginx* poslužitelja. Varijabli *workers* pridružen je broj *Gunicorn-ovih* radnih procesa koji obrađuju dolazne zahtjeve i služe ih *Django* aplikaciji. *Gunicorn* koristi više radnih procesa za obradu istodobnih zahtjeva i poboljšanje ukupne izvedbe i skalabilnosti *Django* aplikacije. Nakon toga potrebno je konfigurirati *Nginx* web poslužitelj.

Prije nego li se konfigurira *Nginx* za dohvaćanje statičkih datoteka i prosljeđivanja HTTP zahtjeva *Gunicorn-u*, potrebno je unutar *Django* aplikacije u *settings.py* navesti URL statičkih datoteka pomoću sljedeće naredbe:

STATIC_URL = 'static/'

Osim toga, u istoj datoteci je potrebno navesti IP adresu računala/poslužitelja koje će smjeti pristupati *Django* aplikaciji što se postiže sljedećom naredbom:

ALLOWED_HOSTS = ['10.0.2.15']

Nakon toga, korištenjem naredbe *sudo nano /etc/nginx/sites-available/StreamingWebsite* kreira se konfiguracijska datoteka za *Django* aplikaciju koja je navedena u prilogu P.3.3. U konfiguraciji je navedeno da će *Nginx* poslužitelj slušati na *portu* 80. S dijelom *server_name* određeno je ime domene koje je jednako IP adresi računala, odnosno virtualne mašine. S blokom location */static/* određeno je da će *Nginx* posluživati statičke datoteke s putanje */home/luka/StreamingWebsite/static/*. Blok *location /* je vezan za prosljeđivanje zahtjeva *Gunicorn-u*. *Gunicorn* će slušati zahtjeve na *portu* 8000. Nakon što se napiše konfiguracijska datoteka potrebno je naredbom *cd /etc/nginx/sites-enabled/* otići do direktorija *sites-enabled* koji obično se koristi za pohranjivanje konfiguracijskih datoteka za pojedinačne web aplikacije. Ovaj direktorij sadrži simboličke veze na konfiguracijske datoteke koje se nalaze u direktoriju *sites-available*. Zatim je potrebnu napraviti simboličku vezu na sljedeći način:

sudo ln -s /etc/nginxsites-available/StreamingWebsite

Odvajanjem pojedinačnih konfiguracija web aplikacija u zasebne datoteke u direktoriju *sites-available* i njihovim omogućavanjem pomoću simboličkih veza u direktoriju *sites-enabled* lakše je upravljati i organizirati *Nginx* konfiguraciju za više web aplikacija. Nakon toga potrebno je resetirati *Nginx* poslužitelj kako bi se primijenile promjene zbog konfiguriranja što se postiže sljedećom naredbom:

sudo service nginx restart

Kako bi se pokrenula web aplikacija u produkciji, odnosno WSGI dio *Django* aplikacije u kojoj se nalaze sve stranice koje se izmjenjuju u aplikaciji, potrebno je upisati sljedeću naredbu u *Linux* naredbenom retku:

gunicorn -c conf/gunicorn_config1.py StreamingWebsite.wsgi

3.2.4. Dodavanje funkcionalnosti vezane za registraciju i prijavu korisnika

Kod dodavanja funkcionalnosti za registraciju novih korisnika i prijavu registriranih korisnika koristi se *Django-ov* ugrađeni sustav autentikacije koji omogućuje upravljanje autentikacijom i autorizacijom korisnika u web aplikaciji. Sustav provjere autentičnosti obrađuje zadatke kao što su registracija korisnika, prijava, odjava, poništavanje lozinke itd. *Django* nudi zadani korisnički model, ali je u aplikaciji izmijenjen kako bi se korisnici prijavljivali s adresom elektroničke pošte i lozinkom, a ne korisničkim imenom i lozinkom. Zatim je za tu izmjenu u *models.py* datoteci napravljena nova migracija u bazi pomoću naredbi *makemigrations* i *migrate*.

Zatim je u *forms.py* napravljena nova forma za registraciju korisnika naziva *MyUserCreationForm* koja je kao klasa naslijedila ugrađenu formu za kreiranje novog korisnika. Time je omogućeno da pri registraciji postoje polja za unos punog imena korisnika, njegovog korisničkog imena i adrese elektroničke pošte te njegove lozinke i polja za potvrdu lozinke.

Nakon toga je potrebno kreirati poglede koji će odraditi funkcionalnost prijave, registracije, ali i odjave korisnika. Pogled koji odrađuje funkcionalnost prijave korisnika te renderira sadržaj stranice za prijavu predstavljen je kao funkcija *loginPage(request)* koja prima POST zahtjev kao argument koji se šalje od forme za prijavu korisnika, a navedena je u prilogu P.3.4. Najvažnije naredbe koje se mogu istaknuti su naredba *authenticate* koja autenticira korisnika, *login* funkcija koja prijavljuje korisnika te funkcija *render* koja renderira HTML predložak te njemu kao argument šalje *Python* rječnik koji sadrži s*tring* koji daje informaciju o tome treba li renderirati sadržaj stranice za prijavu ili registraciju budući da i jedna i druga imaju sadržaj unutar istog HTML dokumenta. Osim toga, sadrži funkciju *redirect* koja nakon uspješne prijave preusmjerava korisnika na stranicu za gledanje video signala uživo, a kao argument prima naziv URL-a stranice za gledanje videozapisa. Pogled kojim se odrađuje funkcionalnost registracije novog korisnika te renderiranje sadržaja stranice za registraciju nalazi se u prilogu P.3.5. Pogled za registraciju korisnika definiran je kao funkcija *registerPage(request)* koja također prima POST zahtjev koji se šalje od forme za registraciju. Bitne funkcije koje se mogu istaknuti su funkcija *save* koja sprema korisnika u bazu ukoliko su sva polja za unos ispravno unesena, funkcija *login* koja nakon registracije prijavljuje korisnika u stranicu za gledanje videosignala te funkcija *render* koja renderira sadržaj stranice za registraciju te u HTML dokument šalje *Python* rječnik unutar kojeg se nalazi forma za registraciju korisnika. Također, stranica za registraciju sadrži funkciju *redirect* koja nakon uspješne registracije preusmjerava korisnika na stranicu za gledanje video signala, a kao argument prima naziv URL-a stranice za gledanje video signala. Kada korisnik želi otići sa stranice za gledanje video signala, može se odjaviti pritiskom na tipku *Logout*. Funkcionalnost odjave odrađuje pogled *logoutUser(request)* koji kao argument također prima POST zahtjev koji se šalje klikom na tipku za odjavu, a pogled sadrži sljedeći kod:

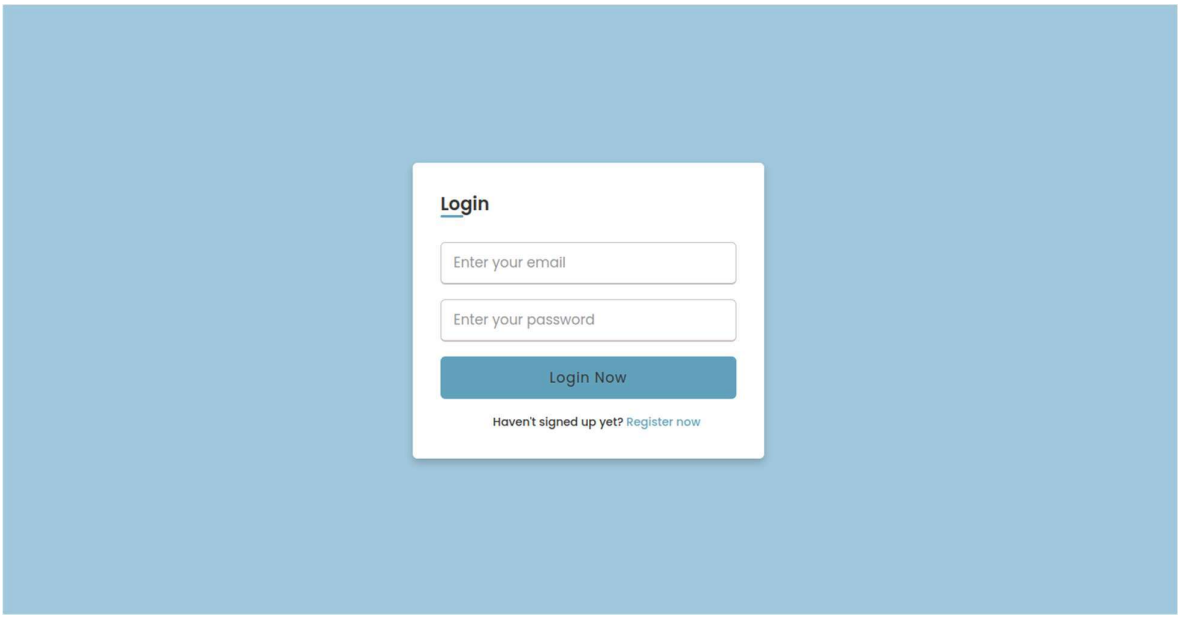
```
def logoutUser(request):
    logout(request)
    return redirect('login')
```

Pogled sadrži dvije funkcije: funkciju *logout (request)*, koja odjavljuje korisnika, a kao argument prima POST zahtjev te funkcija *redirect* koja preusmjerava korisnika na stranicu za prijavu jer kao argument prima naziv URL-a stranice za prijavu. Nakon što su napisani svi pripadajući pogledi potrebno je napraviti URL-ove koji će preusmjeravati HTTP zahtjeve na određeni pogled za prijavu, registraciju ili odjavu. To se postiže na sljedeći način:

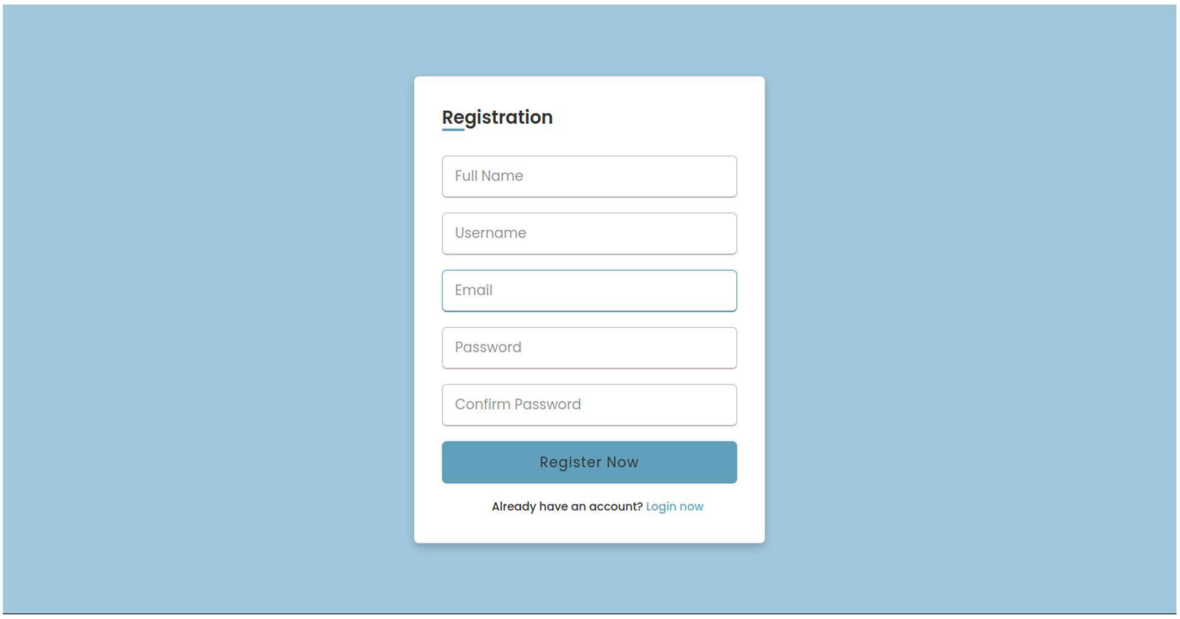
```
urlpatterns = [
path('stream/', views.stream, name='stream'),
path('', views.loginPage, name="login"),
path('register/', views.registerPage, name="register"),
path('logout/', views.logoutUser, name="logout"),
]
```

Prvi URL je vezan za pogled koji renderira kompletan sadržaj stranice za gledanje videozapisa, drugi je URL ulazne stranice u aplikaciju (stranica za prijavu korisnika), treći URL se odnosi na pogled koji renderira stranicu za registraciju korisnika i četvrti URL je vezan za pogled koji odjavljuje korisnika iz stranice za gledanje videozapisa. Nakon toga, potrebno je napraviti sadržaj stranica za prijavu i registraciju unutar jednog HTML dokumenta koji se naziva *login_register.html*. Dokument sadrži HTML kod za obje stranice, a koji će se renderirati ovisi o primljenom *string-u* iz pogleda. Pogled za registraciju ne šalje nikakav *string* pa će se u tom slučaju odmah prijeći na *else* blok koji sadrži HTML kod stranice za registraciju. Također, pogled za registraciju šalje formu kao argument te će se njezina polja prikazati u HTML kodu. Ukoliko je pogled za prijavu korisnika renderirao dokument *login_register.html*, kao argument će doći *string* s kojim će početni *if* uvjet biti zadovoljen te će se prikazati sadržaj stranice za prijavu korisnika.

Na slici 3.7. prikazano je kako izgleda stranica za prijavu korisnika dok je na slici 3.8. prikazano kako izgleda stranica za registraciju novog korisnika. Kao što je ranije navedeno u poglavlju 3.1., stranica za prijavu sadrži polja za unos adrese elektroničke pošte i lozinke korisnika, gumb za prijavu i poveznicu za odlazak na stranicu za registraciju korisnika. Stranica za registraciju novog korisnika sadrži polja za unos punog imena korisnika, korisničkog imena, adrese elektroničke pošte, lozinke, polje za potvrdu lozinke, poveznicu za povratak na stranicu za prijavu korisnika te gumb za registraciju.



Slika 3.7. Stranica za prijavu korisnika



Slika 3.8. Stranica za registraciju korisnika

3.2.5. Implementacija liste aktivnih korisnika na web stranici za gledanje video signala

Prije nego li se definiraju pogledi koji će renderirati sadržaj stranice za gledanje videozapisa na kojoj se nalazi i lista aktivnih korisnika, potrebno je načiniti novi model unutar baze podataka koji će predstavljati prijavljenog korisnika. Taj model se naziva *LoggedUser* i kao

klasa sadrži funkcije za upis u tablicu prijavljenih korisnika i brisanje iz iste. Te funkcije su povezane sa signalima (engl. *Django signals*) koji se aktiviraju u određenim događajima poput prijave ili odjave korisnika. Tako će se pri prijavi korisnika aktivirati signal koji će pokrenuti funkciju upisa korisnika u tablicu prijavljenih korisnika. Na isti način će se pri odjavi korisnika signalizirati pokretanje funkcije za brisanje korisnika iz liste prijavljenih korisnika.

Nakon toga definirana su dva pogleda: prvi naziva *stream* koji renderira sadržaj cijele stranice za gledanje videozapisa koji se nalazi u dokumentu *stream.html* te drugi naziva *users* koji renderira samo listu aktivnih korisnika koja se naziva u dokumentu *users-list.html*. Upravo taj drugi pogled će se pozivati kada bude trebalo osvježiti listu te će on opet renderirati sadržaj liste bez da se cijela stranica za gledanje videozapisa osvježava. Time je spriječeno prekidanje u gledanju videozapisa. Pogledi su dani u prilogu P.3.6. Svaki pogled prije renderiranja dohvaća sve prijavljene korisnike i sprema ih u *Python* rječnik kako bi korisnik pri dolasku na stranicu za gledanje videozapisa vidio korisnike koji su prijavljeni od ranije. Svaki pogled iznad sebe ima dekorator *@login_required(login_url='login')* koji govori da se tim pogledima ne može pristupiti ako korisnik nije prijavljen, odnosno ako prethodno nije došao sa URL-a koji se naziva *login*. Također je bitno naglasiti da je HTML dokument za listu korisnika *users-list.html* uključen unutar *stream.html* dokumenta naredbom *{% include 'stream/users-list.html' %}*. Nakon toga potrebno je u *urls.py* dodati URL koji će biti povezan s pogledom *users*. To se postiže sljedećom naredbom:

url_patterns += path('users-list/', views.users, name="users")

Nakon toga, potrebno je napraviti vidljivi sadržaj liste u dokumentu *users-list.html* kako bi ona bila vidljiva na stranici za strujanje te će korisnik pri dolasku na stranicu moći vidjeti sve korisnike koji su ranije prijavljeni.

Međutim, korisnici koji od ranije gledaju videozapis, neće moći vidjeti kada novi korisnik dođe gledati videozapis jer je potrebno ponovno renderirati listu s aktivnim korisnicima. Također kada neki korisnik napusti stranicu za gledanje video signala, ostalim korisnicima će on i dalje ostati na listi aktivnih korisnika sve dok ne osvježe cijelu stranicu. Zato je potrebno uvođenje *Django* kanala koji će na prijavu ili odjavu korisnika svim prijavljenim korisnicima poslati poruku na čiji će se prijem *Websocket-om* pokrenuti *Ajax* (engl. *Asynchronous JavaScript and XML*) asinkroni poziv koji će dohvatiti URL pogleda koji renderira listu s korisnicima. Nakon instalacije *Django* kanala u virtualno okruženje, potrebno ih je uključiti u aplikacije unutar *settings.py* tako da se unutar INSTALLED_APPS liste unese *'channels'*. Potom je potrebno uključiti ASGI dio aplikacije naredbom *ASGI_APPLICATION = 'StreamingWebsite.asgi.application'*. Zatim je

unutar postavki projekta potrebno uključiti slojeve kanala koji će omogućiti da se informacija o prijavi ili odjavi korisnika pošalje svim aktivnim korisnicima. Bez njih informacija bi se slala samo onom koji se prijavljuje ili odjavljuje. Slojevi kanala uključuju se na sljedeći način:

```
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            'hosts': [('10.0.2.15', 6379)]
        },
    }
}
```

Iz gore navedenog može se zaključiti kako će se koristiti *Redis* kao *backend*, odnosno on će čuvati informaciju o pojedinima kanalima i nazivu grupe u kojoj se kanali nalaze. Također je navedeno da će *Redis* poslužitelj slušati na IP adresi 10.0.2.15 i *portu* 6379. Nakon što se instalira *Redis* naredbom iz poglavlja 3.1., potrebno je otići do konfiguracijske datoteke *Redis-a* naredbom *sudo nano /etc/redis/redis.conf* kako bi promijenili IP adresu na kojoj će *Redis* slušati u gore iznad navedenu. *Port* će ostati isti jer je to uobičajeni *port* na kojem *Redis* sluša.

Kada se to napravi, može se početi implementacijom *Django* kanala. Prvotno se unutar aplikacijske komponente *stream* definira datoteka *consumers.py* koja će predstavljati potrošače. Unutar nje definirana je klasa *RefreshConsumer* koja nasljeđuje klasu *WebsocketConsumer* kako bi potrošač mogao rukovati *WebSocket* vezama i događajima. Unutar klase *RefreshConsumer* definirane su tri metode: *connect(self)*, *disconnect(self)*, *chat_message(self,event)*. Funkcija *connect(self)* se pokreće kada s korisničke strane dođe zahtjev za *WebSocket* konekcijom. Tada ona korisnika, odnosno njegov kanal stavlja u grupu kanala te potom svim članovima u grupi šalje poruku da se novi korisnik prijavio kako bi se na njihovoj strani pomoću *WebSocket-a* pokrenulo ponovno renderiranje sadržaja liste s korisnicima. Funkcija *disconnect(self)* se pokreće kada s korisničke strane dođe zahtjev za prekidom *WebSocket* konekcije. Tada ona korisnika, odnosno njegov kanal izbacuje iz grupe kanala te potom svim članovima u grupi šalje poruku da se korisnik odjavio. Tada se na njihovoj strani pomoću *WebSocket-a* pokreće *Ajax* poziv koji će dohvatiti URL koji je povezan s pogledom te će isti ponovno renderirati listu s korisnicima. Nakon definiranja

potrošača, potrebno je definirati usmjeravanje za potrošače unutar datoteke *routing.py*. Usmjeravanje se kao i URL-ovi određuje na sličan način:

```
websocket_urlpatterns = [
    path('', consumers.RefreshConsumer.as_asgi())
]
```

Nakon toga je potrebno u datoteci *asgi.py* dodati sljedeće naredbe:

```
application = ProtocolTypeRouter({
  'http': get_asgi_application(),
  'websocket': AuthMiddlewareStack(
        URLRouter(
            StreamingWebsite.routing.websocket_urlpatterns
        )
    ),
})
```

Aplikacijskom objektu dodaje se *ProtocolTypeRouter* koji je odgovoran za usmjeravanje različitih protokola prema odgovarajućoj ASGI aplikaciji. Linija *'websocket':AuthMiddlewareStack* specificira da se *WebSocket* protokol treba usmjeriti na URL usmjeravanje za rukovanje *WebSocket* vezama. Unutar *URLRouter-a* je navedeno gdje se nalazi datoteka usmjeravanja za potrošače. Nakon što su implementirani *Django* kanali, potrebno je unutar *stream.html* dokumenta unutar *script* oznaka definirati *WebSocket* komunikaciju koja će se odvijati na korisničkoj strani. Prvo je potrebno uspostaviti *WebSocket* konekciju pomoću sljedeće naredbe:

var socket = new WebSocket('ws://' + window.location.host +':8001');

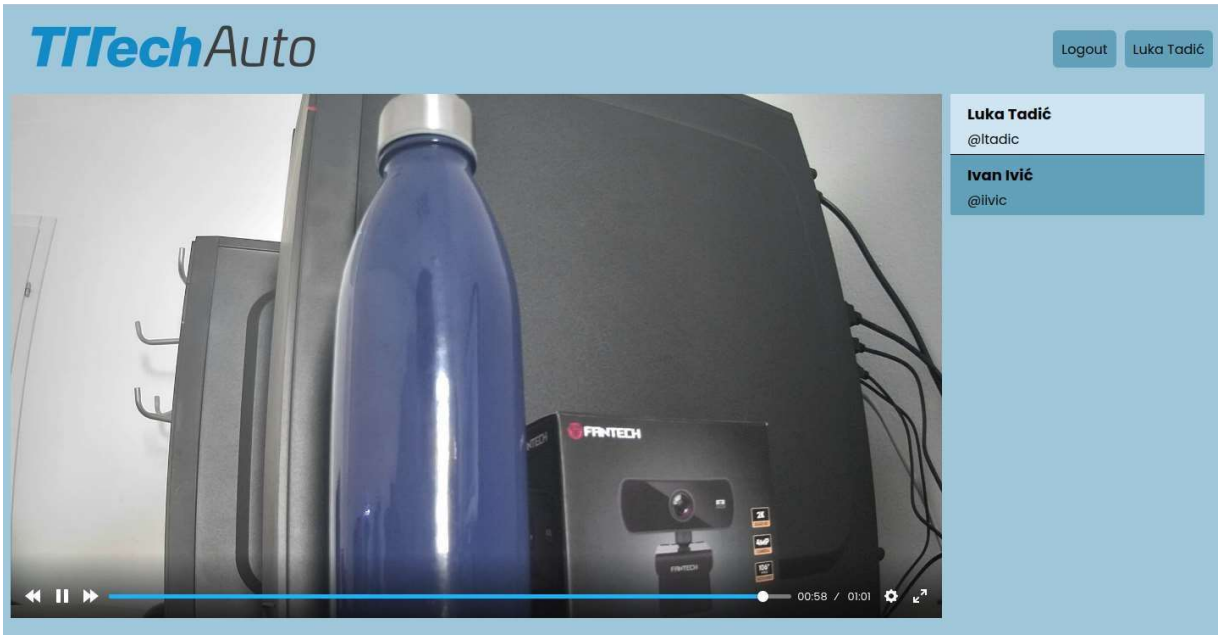
kojom se stvara novi *socket* tako što se prvo doda prefiks *ws://* što predstavlja *WebSocket* konekciju, zatim se na to povezuje IP adresa poslužitelja web aplikacije, odnosno IP adresa navedena u konfiguraciji *Nginx* poslužitelja te se zadnje dodaje *port* na kojem će se odvijati *WebSocket* komunikacija. Na tom portu će *Nginx* poslužitelj slušati za nadolazeće *WebSocket* zahtjeve. Nakon toga potrebno je definirati funkciju kojom će *Websocket* na primitak poruke od *Django* kanala o prijavi ili odjavi korisnika pokrenuti *Ajax* zahtjev koji će preko URL-a pozvati pogled koji će ponovno renderirati listu s korisnicima. To se postiže sljedećim skupom naredbi:

```
socket.onmessage = function message(event) {
  console.log("Refreshing user list");
  $.ajax({
            url: "{% url 'users' %}",
            success: function(data) {
                $('#users').html(data);
            }
        });
};
```

Kako bi lista s aktivnim korisnicima na stranici za gledanje video signala radila ispravno u produkciji, potrebno je napraviti izmjenu konfiguracije *Nginx* poslužitelja. U konfiguraciju je potrebno dodati značajke navedene u prilogu P.3.7. U dijelu konfiguracije u prilogu P.3.7. direktiva *upstream* koristi se za definiranje poslužitelja za *WebSocket* komunikaciju koji će slušati na IP adresi računala/poslužitelja i *portu* 8001. Blokom *location /ws/* određeno je gdje će *Nginx* poslužitelj prosljeđivati *WebSocket* zahtjeve, a ti zahtjevi će biti proslijeđeni *Daphne* poslužitelju. Nakon što se konfiguracija izmjeni, potrebno je ponovno resetirati *Nginx* naredbom navedenom u potpoglavlju 3.2.3. Kako bi se u produkciji pokrenuo ASGI dio *Django* aplikacije koji će omogućiti osvježivanje liste aktivnih korisnika, potrebno je uključiti *Daphne* poslužitelj sljedećom naredbom:

daphne -b 10.0.2.15 -p 8001 StreamingWebsite.asgi:application

gdje opcija *–b 10.0.2.15* označava za koju će se adresu vezati *Daphne* poslužitelj, *–p 8001* označava na kojem će *portu* slušati *Daphne*, a *StreamingWebsite.asgi:application* predstavlja koju ASGI aplikacija *Daphne* treba uključiti. Na slici 3.9. nalazi se prikaz stranice za gledanje videozapisa s listom aktivnih korisnika. Lista za svakog korisnika prikazuje puno ime i korisničko ime.



Slika 3.9. Stranica za gledanje video signala s listom aktivnih korisnika

3.3. Implementacija vlastitog rješenja na *Raspberry Pi 3*

Prije nego li je web aplikacija implementirana na *Raspberry Pi 3*, na istom je instaliran 32-bitni operacijski sustav *Raspbian*. Unutar sustava je potrebno omogućiti SSH komunikaciju kako bi se *Raspberry-ju* moglo pristupiti s vanjskog računala. Nakon toga se pomoću sučelja *Putty* korištenjem SSH protokola spoji na *Raspberry Pi 3* upisivanjem imena računala, te ako je to uspješno potrebno je upisati korisničko ime i lozinku korisnika na *Raspberry-ju*. Nakon toga mogu se početi instalirati svi potrebni alati i tehnologije navedene u poglavlju 3.1. s time da se sve što se instalira *pip* instalacijskim paketom stavi u virtualno okruženje. Nakon dovršenih instalacija, pomoću alata *WinSCP* se na sličan način s SSH protokolom spoji na *Raspberry Pi*, te se nakon uspješne konekcije može prenijeti *Django* projekt.

Zatim je potrebno napraviti sve izmjene u konfiguracijama. *Raspberry Pi 3* je spojen na lokalnu mrežu *ethernet* kablom i ima IP adresu 10.30.10.45. Tu adresu potrebno je postaviti na sva mjesta gdje se nalazi IP adresa virtualnog uređaja na kojem je razvijana aplikacija. Prvo je potrebno u konfiguraciji *Nginx* poslužitelja promijeniti IP adresu na dva mjesta na koju će *Nginx* prosljeđivati HTTP zahtjeve za *Gunicorn* te *WebSocket* zahtjeve za *Daphne* poslužitelj. Osim toga, kao domenu *Nginx* poslužitelja (direktiva *server_name*) potrebno je također staviti IP adresu *Raspberry-ja*. Kada je promijenjena konfiguracija, *Nginx* poslužitelj se može ponovno pokrenuti.

Nakon toga, potrebno je u konfiguraciji *Gunicorn-a* za *bind* varijablu postaviti IP adresu *Raspberry-ja*. Zatim je u datoteci *settings.py* unutar *Django* projekta potrebno dodati IP adresu *Raspberry-ja* unutar liste ALLOWED_HOSTS te se mora promijeniti IP adresa koja stoji u slojevima kanala na kojoj će *Redis* slušati zahtjeve. Nakon toga se unutar konfiguracije *Redis-a* postavi da će *Redis* slušati na IP adresi *Raspberry-ja*. Potom je potrebno resetirati *Redis*. Zadnje što je potrebno promijeniti kako bi kompletna aplikacija ispravno funkcionirala je IP adresa unutar naredbe kojom se pokreće *Daphne* poslužitelj navedenoj u poglavlju 3.2.5. Također, na to mjesto je potrebno upisati IP adresu *Raspberry-ja*. Aplikaciji koja se pokrene na računalu *Raspberry Pi 3* u web pregledniku će moći pristupiti sva računala spojena na istu lokalnu mrežu.

Performanse računala *Raspberry Pi 3* nisu dovoljno dobre da bi se kao ulazna rezolucija uzelo 1920x1080 elemenata slike, stoga je potrebno smanjiti kvalitete strujanja kako bi strujanje zadržalo jednako kašnjenje i kako ne bi bilo prekida u strujanju. Uzeta je manja ulazna rezolucija videa te su izlazne kvalitete strujanja smanjenje, odnosno smanjene su rezolucije i kodne brzine. Generiranje HLS strujanja na računalu *Raspberry Pi 3* postiže se *Python* kodom unutar *wsgi.py* datoteke, a koji je dan u prilogu P.3.8. Kao ulazna veličina video uzima se 720x480 elemenata slike. Ulazna kvaliteta se mapira u tri različita izlazna strujanja kao i na virtualnom uređaju. Prvo strujanje ima rezoluciju ulaznog videa te kodnu brzinu od 1500 kbit/s , drugo strujanje ima rezoluciju 640x360 elemenata slike i kodnu brzinu 1200 kbit/s, a treće strujanje ima rezoluciju 320x240 elemenata slike i kodnu brzinu 800 kbit/s. Ostale stvari u naredbi su nepromijenjene.

4. IMPLEMENTACIJA MPEG-DASH STRUJANJA I USPOREDBA PERFORMANSI S HLS PROTOKOLOM

U ovom poglavlju opisan je postupak implementacije MPEG-DASH strujanja u *Django* aplikaciju na *Raspberry Pi 3* računalu i zatim je napravljena usporedba performansi s HLS protokolom. To je napravljeno na način da je *FFmpeg* kod za generiranje HLS strujanja u datoteci *wsgi.py* zamijenjen s *FFmpeg* kodom za generiranje MPEG-DASH strujanja. Osim toga, potrebno je implementirati MPEG-DASH reproduktor umjesto HLS reproduktora koji će moći dohvatiti segmente i manifest datoteku MPEG-DASH strujanja. Od performansi strujanja, mjerena je latencija i varijacija kašnjenja (engl. *jitter*). U potpoglavlju 4.1. opisan je *FFmpeg* kod za generiranje segmenata i manifest datoteke MPEG-DASH strujanja. Zatim je u potpoglavlju 4.2. opisan reproduktor koji će reproducirati MPEG-DASH strujanje dohvaćeno s kamere. Nakon toga, u potpoglavlju 4.3. su analizirane performanse HLS strujanja dok su u potpoglavlju 4.4. prikazani rezultati MPEG-DASH strujanja.

4.1. Generiranje manifest datoteke i segmenata videa za MPEG-DASH strujanje korištenjem *FFmpeg* alata

Za generiranje manifest datoteke i segmenata video signala MPEG-DASH strujanja također se koristi alat *ffmpeg* opisan u poglavlju 3.1. i web kamera *Fantech Luminous C30*. Kod za generiranje MPEG-DASH strujanja nalazi se unutar *wsgi.py* datoteke. Prije samog generiranja strujanja, također se briše sve što se nalazi unutar poddirektorija *streaming* te se nakon toga odlazi do tog direktorija na isti način kao u poglavlju 3.2.1. Generiranje MPEG-DASH strujanja s tri kvalitete dohvaćenog s web kamere postiže se pomoću Python koda iz priloga P.3.9. Kod za generiranje strujanja se također pokreće pomoću *subprocces* modula i sadrži većinu istih stavki kao i kod u poglavlju 3.3. Razlike između tih kodova počinju poslije mapiranja ulaznog videa u tri izlazna strujanja. MPEG-DASH strujanja na računalu *Raspberry Pi 3* imaju iste kvalitete kao HLS strujanja na tom računalu. Linija koda *use_timeline 1* omogućuje korištenje oznaka vremenske crte u generiranom MPEG-DASH manifestu. Dio koda *use_template 1* predstavlja korištenje segmentacije temeljene na predlošku koja omogućuje učinkovitije predstavljanje segmentiranih medijskih datoteka u MPEG-DASH manifestu. Opcija *window_size 5* određuje broj segmenata (pet) koji će se uključiti u klizni prozor za segmentaciju temeljenu na predlošku. Dio koda *adaptation_sets "id=0,streams=0 id=1,streams=1 id=2,streams=2"* definira adaptacijske skupove u MPEG-DASH manifestu. Adaptacijski skupovi se koriste za grupiranje povezanih strujanja zajedno, kao što su različiti prikazi videa, zvuka ili titlova. Ovom naredbom je definirano

tri adaptacijska skupa koji sadrže tri kvalitete strujanja. Linija koda *f dash -seg_duration 1 manifest.mpd* određuje da je izlazni format MPEG-DASH i da je trajanje segmenta jedna sekunda te također kreira izlaznu manifest datoteku koja se naziva *manifest.mpd*.

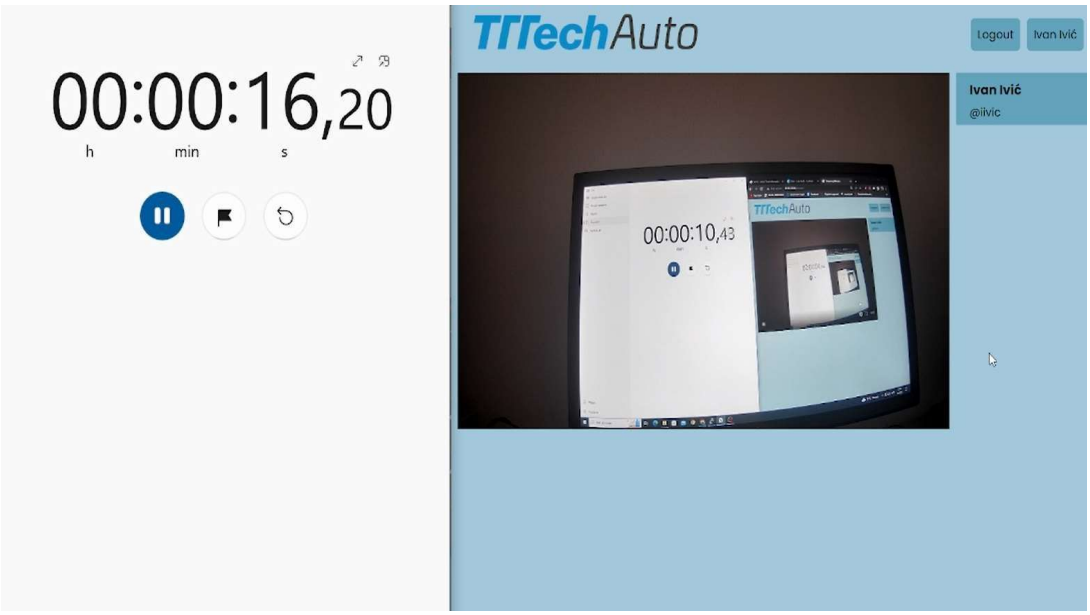
4.2. Implementacija reproduktora za reprodukciju MPEG-DASH strujanja

Za implementaciju reproduktora za reprodukciju MPEG-DASH strujanja uživo dohvaćenog s web kamere koristi se *Javascript* biblioteka otvorenog koda koja se naziva *Shaka Player* [34]. *Shaka Player* omogućuje reprodukciju adaptivnog strujanja sadržaja poput HLS-a ili MPEG-DASH strujanja. Pruža fleksibilno i prilagodljivo rješenje za reprodukciju medija za web aplikacije. U *main.html* predlošku se pomoću *script* oznaka uključuje *Shaka Player* te se *link* oznakom uključuje CSS datoteka za izgled *Shaka Player-a*. Unutar *stream.html* se nalazi HTML i *Javascript* kod za reproduktor. HTML kod se sastoji od jednog *div* elementa koji uokviruje *video* element koji predstavlja reproduktor. *Javascript* kod za reproduktor nalazi se unutar *script* oznaka. Prvo se dohvaća izvor strujanja, odnosno putanja do direktorija *streaming*. Nakon toga se unutar asinkrone funkcije *initPlayer()* obavlja inicijalizacija reproduktora tako što se prvo konfiguriraju kontrole i izgled reproduktora, zatim se radi konfiguracija strujanja te se na kraju učitava manifest datoteka unutar danog izvora strujanja, odnosno direktorija *streaming*. Funkcionalnosti automatske izmjene kvalitete i izmjene kvalitete korisničkim odabirom implementirane su unutar uključene biblioteke. Na slici 4.1. nalazi se prikaz MPEG-DASH reproduktora s istaknutim izbornikom za odabir kvaliteta u kojem se također može odabrati opcija *Auto* za automatsku izmjenu kvaliteta ili se može odabrati neka od kvaliteta. Osim toga, ima implementirane opcije pauze i nastavka reprodukcije, način rada preko cijelog zaslona te oznaku da je reproduktor trenutno u prijenosu uživo koja zapravo prikazuje i minimalno kašnjenje strujanja.

Slika 4.1. MPEG-DASH reproduktor s prikazanim izbornikom za izmjenu kvaliteta

4.3. Analiza performansi strujanja pomoću HLS protokola

Nakon implementacije *Django* aplikacije za strujanje na računalo *Raspberry Pi 3*, na klijentskom računalu mogu se izvesti mjerenja performansi algoritma strujanja. U radu je mjereno kašnjenje strujanja i varijacija kašnjenja strujanja (engl. *jitter*). Kada se na *Raspberry-ju* pokrene *Django* aplikacija koju poslužuje *Nginx*, klijentsko računalo spojeno na istu lokalnu mrežu kao i *Raspberry Pi 3* može otvoriti aplikaciju u web pregledniku. Mjerenje kašnjenja i varijacije kašnjenja radi se tako da se uz otvorenu aplikaciju u web pregledniku najprije otvori i aplikacija za štopericu, ali da su i stranica za strujanje i štoperica vidljivi na zaslonu računala. Kamera koja snima video za strujanje, a spojena je na *Raspberry Pi 3*, okrene se prema zaslonu računala kako bi snimala zaslon klijentskog računala. Na računalu se pokrene štoperica i ona počinje brojati vrijeme koje se također vidi unutar reproduciranog strujanja u reproduktoru na stranici za gledanje video signala. Razlika vremena koje se vidi na štoperici i onog koje se vidi na videu u reproduktoru, predstavlja kašnjenje strujanja, dok razlika između susjednih uzoraka kašnjenja predstavlja varijaciju kašnjenja. Za lakše uzimanje uzoraka kašnjenja, dok traje brojanje vremena na štoperici te strujanje videa snimljenog zaslona kamerom, snima se zaslon računala korištenjem alata *OBS Studio*. Iz videa koji je snimljen alatom *OBS Studio*, uzeto je 30 uzoraka kašnjenja i 29 uzoraka varijacije kašnjenja jer je za to uzeta razlika između prethodnog i sljedećeg uzorka kašnjenja. Na slici 4.2. prikazano je kako izgleda zaslon računala pri mjerenju kašnjenja i varijacije kašnjenja strujanja.



Slika 4.2. Zaslon klijentskog računala prilikom mjerenja kašnjenja i varijacije kašnjenja

Tablica 4.1. prikazuje srednje kašnjenje i srednju varijaciju kašnjenja HLS strujanja s pripadajućim standardnim devijacijama za različite kvalitete i broj korisnika koji gleda video signal na toj kvaliteti. Mjerenje je također izvedeno za slučaj kada je u aplikaciji odabrano automatsko izmjenjivanje kvaliteta.

Tablica 4.1. Srednje kašnjenje i varijacija kašnjenja HLS strujanja s pripadajućim standardnim devijacijama za različite kvalitete i broj korisnika

	Prosječno kašnjenje [s]	**Prosječni jitter [s]**
Kvaliteta 240p, jedan korisnik	2,62 ± 0,13	0,06 ± 0,04
Kvaliteta 240p, dva korisnika	2,89 ± 0,07	0,05 ± 0,04
Kvaliteta 240p, tri korisnika	2,95 ± 0,06	0,05 ± 0,05
Kvaliteta 360p, jedan korisnik	2,92 ± 0,07	0,06 ± 0,04
Kvaliteta 360p, dva korisnika	2,91 ± 0,04	0,05 ± 0,04
Kvaliteta 360p, tri korisnika	2,93 ± 0,05	0,05 ± 0,03
Kvaliteta 480p, jedan korisnik	2,75 ± 0,16	0,06 ±0,06
Kvaliteta 480p, dva korisnika	2,87 ± 0,05	0,05 ± 0,06
Kvaliteta 480p, tri korisnika	2,86 ± 0,05	0,05 ± 0,04
Kvaliteta auto, jedan korisnik	2,74 ± 0,13	0,06 ± 0,05
Kvaliteta auto, dva korisnika	2,74 ± 0,15	0,06 ± 0,07
Kvaliteta auto, tri korisnika	2,92 ± 0,04	0,04 ± 0,04

Budući da su strujanja različitih kvaliteta generirana u istom trenutku, razlike u kašnjenju manifestiraju se zbog različitog vremena potrebnog za dohvaćanje segmenata svakog strujanja na što utječu mrežni uvjeti. Za strujanje veće kvalitete reproduktoru je potrebno više vremena da dohvati njegov sadržaj pa je time kašnjenje veće, ali rezultati u tablici pokazuju da to nije uvijek tako. U slučaju strujanja kvalitete 360p s jednim korisnikom kašnjenje je 2,92 s dok je u slučaju strujanja kvalitete 480p s jednim aktivnim korisnikom kašnjenje jednako 2,75 s. To se dogodilo jer je u trenutku snimanja kašnjenja strujanja kvalitete 360p mreža bila više opterećena pa je kašnjenje bilo veće nego za kvalitetu 480p. S povećanjem broja korisnika koji gledaju videozapis iste kvalitete, raste i samo kašnjenje strujanja. Međutim, u slučaju strujanja kvalitete 360p s jednim prijavljenim korisnikom i istom strujanju s dva prijavljena korisnika rezultati pokazuju da je kašnjenje manje u slučaju kada je jedan korisnik gleda video signal. To je također posljedica

opterećenije mreže kada je rađeno snimanje strujanja s jednim korisnikom. Najmanje izmjereno kašnjenje je u slučaju strujanja kvalitete 240p s jednim aktivnim korisnikom i iznosi 2,62 s dok je najveće izmjereno kašnjenje u slučaju strujanja kvalitete 360p s tri aktivna korisnika i iznosi 2,93 s. Prema teoriji, najveće kašnjenje bi trebalo biti u slučaju strujanja kvalitete 480p s tri aktivna korisnika, ali tada mreža nije bila toliko opterećena kao u slučaju strujanja s kvalitetom 360p i tri korisnika na strujanju. Varijacija kašnjenja u većini slučajeva ima vrijednost 50 ms pri čemu je najveća varijacija kašnjenja 60 ms u više slučajeva, a najmanja varijacija kašnjenja je pri strujanju s automatskom kvalitetom i tri aktivna korisnika i iznosi 40 ms.

4.4. Analiza performansi strujanja pomoću MPEG-DASH protokola

Mjerenja performansi MPEG-DASH strujanja u *Django* aplikaciji implementiranoj na računalu *Raspberry Pi 3* izvedena su na isti način opisan u poglavlju 4.3. Mjerenja su također izvedena u slučajevima kada videozapis neke kvalitete gleda jedan, dva ili tri korisnika i u slučaju kada je u aplikaciji odabrano automatsko izmjenjivanje kvaliteta. Tablica 4.2. prikazuje srednje kašnjenje i srednju varijaciju kašnjenja MPEG-DASH strujanja s pripadajućim standardnim devijacijama za različite kvalitete i broj korisnika koji gleda video signal na toj kvaliteti.

Tablica 4.2. Srednje kašnjenje i varijacija kašnjenja MPEG-DASH strujanja s pripadajućim standardnim devijacijama zarazličite kvalitete i broj korisnika

	Prosječno kašnjenje [s]	**Prosječni jitter [s]**
Kvaliteta 240p, jedan korisnik	5,77 ± 0,09	0,05 ± 0,04
Kvaliteta 240p, dva korisnika	5,84 ± 0,07	0,07 ± 0,07
Kvaliteta 240p, tri korisnika	5,92 ± 0,17	0,06 ± 0,05
Kvaliteta 360p, jedan korisnik	5,42 ± 0,23	0,17 ± 0,26
Kvaliteta 360p, dva korisnika	5,71 ± 0,13	0,08 ± 0,13
Kvaliteta 360p, tri korisnika	5,61 ± 0,11	0,07 ± 0,05
Kvaliteta 480p, jedan korisnik	5,33 ± 0,11	0,09 ± 0,07
Kvaliteta 480p, dva korisnika	5,79 ± 0,24	0,14 ± 0,22
Kvaliteta 480p, tri korisnika	5,78 ± 0,06	0,07 ± 0,04
Kvaliteta auto, jedan korisnik	5,72 ± 0,05	0,05 ± 0,04
Kvaliteta auto, dva korisnika	5,37 ± 0,17	0,08 ± 0,07
Kvaliteta auto, tri korisnika	5,98 ± 0,05	0,07 ± 0,05

Može se zaključiti da je kašnjenje MPEG-DASH strujanja otprilike duplo veće od strujanja korištenjem HLS protokola. Kao i kod HLS strujanja, za strujanje veće kvalitete reproduktoru je potrebno više vremena da dohvati njegov sadržaj pa je time kašnjenje veće te kašnjenje raste s povećanjem broja korisnika koji gledaju videozapis iste kvalitete. Međutim, strujanje kvalitete 240p s jednim aktivnim korisnikom ima veće kašnjenje nego strujanje kvalitete 360p s jednim korisnikom na strujanju. Također, prilikom strujanja kvalitete 360p s dva korisnika dobiva se veće kašnjenje u odnosu kada se na istom strujanju nalaze tri korisnika. Najveće kašnjenje je dobiveno prilikom strujanja s automatskim izmjenjivanjem kvalitete i tri korisnika na strujanju te iznosi 5,98 s. Najmanja vrijednost kašnjenja iznosi 5,33 s i postiže se u slučaju strujanja s kvalitetom 480p i jednim aktivnim korisnikom. Najveća vrijednost varijacije kašnjenja je u slučaju strujanja kvalitete 360p s jednim aktivnim korisnikom i iznosi 0,17 s. Najmanja vrijednost varijacije kašnjenja iznosi 50 ms i javlja se pri strujanju kvalitete 240p s jednim korisnikom i strujanju s automatskom izmjenom kvalitete te jednim korisnikom na strujanju.

5. ZAKLJUČAK

U diplomskom radu izrađena je web aplikacija za strujanje videosignala Internet mrežom korištenjem HLS protokola. Aplikacija se sastoji od stranice za prijavu registriranih korisnika, stranice za registraciju novih korisnika te stranice za gledanje videozapisa uživo. Na stranici za strujanje uz reproduktor za reprodukciju strujanja nalazi se i lista korisnika koji uživo prate strujanje videosignala. Aplikacija je uspješno izvedena u *Django* web radnom okviru *Python* programskog jezika. Za generiranje segmenata i manifest datoteke HLS strujanja iz videa dohvaćenog s web kamere korišten je *ffmpeg* alat. Za postavljanje aplikacije u produkciji, korišten je *Nginx* web poslužitelj koji rukuje korisničkim zahtjevima i prosljeđuje ih *Gunicorn* WSGI poslužitelju koji komunicira s WSGI dijelom *Django* aplikacije u kojem se nalazi skoro cijela funkcionalnost aplikacije. Za implementaciju sustava prijave i registracije korisnika koristi se *Django-ov* ugrađeni sustav za autentikaciju s izmijenjenim modelom korisnika tako da se novi korisnik može prijaviti s adresom elektroničke pošte i lozinkom umjesto korisničkim imenom i lozinkom. Osvježavanje liste aktivnih korisnika postignuto je uz korištenje *Django* kanala i *Websocket* komunikacije na korisničkoj strani. Budući da se radi o asinkronoj komunikaciji, taj dio je implementiran u ASGI dio *Django* aplikacije koji poslužuje *Daphne* ASGI poslužitelj. Kompletna aplikacija implementirana je računalo *Raspberry Pi 3* koje predstavlja poslužitelj za aplikaciju svim računalima koja su spojena na istu lokalnu mrežu kao i *Raspberry Pi 3*. Zbog lošijih performansi računala *Raspberry Pi 3* u odnosu na računalo na kojem aplikacija prvotno razvijana, bilo je potrebno smanjiti rezoluciju ulaznog videa te kvalitete izlaznih strujanja. Nakon toga je na *Raspberry Pi 3* implementirano MPEG-DASH strujanje kako bi se mogla napraviti usporedba s HLS strujanjem. Zamijenjen je kod za generiranje segmenata i manifest datoteke HLS strujanja i HLS reproduktor s *ffmpeg* kodom za generiranje segmenata i manifest datoteke MPEG-DASH strujanja i MPEG-DASH reproduktorom. Zatim je napravljeno mjerenje kašnjenja i varijacije kašnjenja HLS i MPEG- DASH strujanja. Rezultati pokazuju da se prosječno kašnjenje HLS strujanja kreće od 2,62 s do 2,95 s dok je varijacija kašnjenja HLS strujanja najčešće 50 ms. Srednje kašnjenje MPEG-DASH strujanja ima vrijednosti od 5,33 s do 5,98 s dok se vrijednosti varijacije kašnjenja MPEG-DASH strujanja kreću od 50 ms do 170 ms.

LITERATURA

[1] https://developer.apple.com/streaming/, dostupno na: https://developer.apple.com/streaming/ [21.8.2023.]

[2] R. Vulin, T. Samardžić, Đ. Simić, B. Kovačević, One software solution for processing WebVTT subtitles during playback of hls streams using FFmpeg libraries, 2015 23rd Telecommunications Forum Telfor (TELFOR), 2015.

[3] Video Streaming Protocols – What Are They & How Do They Work?, CDNetworks, 19.12.2021., dostupno na: https://www.cdnetworks.com/media-delivery-blog/video-streaming-protocols/ [27.6.2023.]

[4] Understanding the HTTP Live Streaming Architecture, Apple, dostupno na: https://developer.apple.com/documentation/http-live-streaming/understanding-the-http-live-streaming-architecture [27.6.2023.]

[5] A. Parashar; S. Taterh, Playback protection in HTTP live streaming, 2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS), 2016.

[6] J. Vlaović, S. Rimac-Drlje, F. Vranješ; R. Pečkai Kovač, Evaluation of Adaptive Bitrate Selection Algorithms for MPEG DASH, 2019 International Symposium ELMAR, 2019.

[7] .] K. Lazić, M. Milošević, G. Miljković, N. Ikonić, J. Kovačević, One Implementation of Dynamic Adaptive Streaming over HTTP, 20th Telecommunications forum TELFOR, 2012.

[8] E. Krings, Streaming Protocols for Live Broadcasting: Everything You Need to Know, Dacast, 1.6.2023., dostupno na: https://www.dacast.com/blog/streaming-protocols/ [27.6.2023

[9] Video streaming protocols explained: RTMP, WebRTC, FTL, SRT, Restream, 26.5.2020., dostupno na: https://restream.io/blog/streaming-protocols/ [28.6.2023.]

[10] M. Prasad , P. Venkateshwari, Social Educational Streaming Platform Using HTML Live Streaming, 2022 International Mobile and Embedded Technology Conference (MECON), 2022.

[11] Q. Wang, B. She, J. Cui, Z. Qin, G. Li, Design and Implementation of Distributed Video Live-Streaming System Based on SRS, 2021 10th International Conference on Educational and Information Technology (ICEIT), 2021.

[12] A. Ooka, Y. Hayamizu, H. Asaeda, HLS and CCNx Based High-Quality Live Streaming on On-Premises Network System, 2022 IEEE International Conference on Communications Workshops (ICC Workshops), 2022.

[13] J. G. Min, Y. Lee, High-Quality HTTP Live Streaming System for Limited Communication Bandwidth, 2020 International SoC Design Conference (ISOCC), 2020.

[14] W. Edli Shabrina, D. Wisaksono Sudiharto, E. Ariyanto, M. Al Makky, The QoS Improvement Using CDN for Live Video Streaming with HLS, 2020 International Conference on Smart Technology and Applications (ICoSTA), 2020.

[15] Django introduction, MDN Web Docs, dostupno na : https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction [29.6.2023.]

[16] DjangoThe web framework for perfectionists with deadlines., dostupno na: https://www.djangoproject.com/ [29.6.2023.]

[17] D. Radović, M. Čupić, S. Stefanović, D. Majstorović, Internet Radio Player Implementation Using FFmpeg Software Support, 2017 International Conference on Smart Systems and Technologies (SST), 2017.

[18] About FFmpeg, FFmpeg, dostupno na: https://ffmpeg.org/about.html [30.6.2023.]

[19] nginx, Nginx, dostupno na: https://nginx.org/en/ [30.6.2023.]

[20] Understanding NGINX, solo.io, dostupno na: https://www.solo.io/topics/nginx/ [30.6.2023.]

[21] What Is Nginx? A Basic Look at What It Is and How It Works, Kinsta, 26.1.2022., dostupno na: https://kinsta.com/knowledgebase/what-is-nginx/ [30.6.2023.]

[22] What is Gunicorn?, Serdar Irarslan, 21.5.2022., dostupno na: https://medium.com/@serdarilarslan/what-is-gunicorn-5e674fff131b [30.6.2023.]

[23] Green Unicorn (Gunicorn), Full Stack Python, dostupno na: https://www.fullstackpython.com/green-unicorn-gunicorn.html [30.6.2023.]

[24] Gunicorn - WSGI server, dostupno na: https://docs.gunicorn.org/en/stable/ [30.6.2023.]

[25] gunicorn, dostupno na: https://gunicorn.org/#docs [30.6.2023.]

[26] How to use Django with Daphne, dostupno na: https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/daphne/ [30.6.2023.]

[27] ASGI Documentation, dostupno na: https://asgi.readthedocs.io/en/latest/ [30.6.2023.]

[28] Introduction, dostupno na: https://channels.readthedocs.io/en/stable/introduction.html [30.6.2023.]

[29] What is web socket and how it is different from the HTTP?, Arpit Asati, dostupno na: https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/ [30.6.2023.]

[30] Introduction to Redis, dostupno na: https://redis.io/docs/about/ [30.6.2023.]

[31] Channel Layers, dostupno na: https://channels.readthedocs.io/en/stable/topics/channel_layers.html [30.6.2023.]

[32] Plyr, dostupno na: https://plyr.io/ [5.7.2023.]

[33] hls.js, dostupno na: https://github.com/video-dev/hls.js/ [5.7.2023.]

[34] shaka-project/shaka-player, dostupno na: https://github.com/shaka-project/shaka-player [10.7.2023.]

SAŽETAK

U ovom diplomskom radu izrađena je web aplikacija/poslužitelj za strujanje videosignala korištenjem HLS protokola. Aplikacija je izrađena korištenjem *Django* web radnog okvira u *Python* programskom jeziku. U aplikaciji je omogućena prijava registriranih korisnika i registracija novih korisnika prije nego li se pristupi stranici za gledanje videozapisa. Segmenti videa i manifest datoteka HLS strujanja generirani su iz videa dohvaćenog s web kamere korištenjem *ffmpeg* alata te ih HLS reproduktoru *Nginx* web poslužitelj poslužuje kao statičke datoteke. Većina aplikacije se poslužuje putem *Gunicorn* WSGI poslužitelja koji prima korisničke zahtjeve od *Nginx* poslužitelja i prosljeđuje ih *Django* aplikaciji. Prijava i registracija korisnika odrađena je korištenjem ugrađenog *Django-ovog* sustava autentikacije s izmijenjenim modelom korisnika. Za osvježivanje liste aktivnih korisnika na stranici za strujanje uvedena je asinkrona komunikacija pomoću *Django* kanala i *WebSocket* protokola. Nakon toga aplikacija je implementirana na računalo *Raspberry Pi 3* s izmijenjenim kodom za generiranje strujanja zbog lošijih performansi *Raspberry-ja*. Također je implementirano MPEG-DASH strujanje kako bi se napravila usporedba dvaju protokola. Izmjereno je kašnjenje i varijacija kašnjenja te rezultati pokazuju da HLS strujanje ima otprilike duplo manje kašnjenje nego MPEG-DASH strujanje dok varijacije kašnjenja imaju slične vrijednosti.

Ključne riječi: *Django* aplikacija, *ffmpeg* alat, *Nginx* web poslužitelj, *Raspberry Pi 3*, strujanje videosignala uživo korištenjem HLS protokola

TRANSMISSION OF VIDEO SIGNAL USING THE HLS PROTOCOL

ABSTRACT

In this master's thesis, a web application/server for streaming video signals using the HLS protocol was developed. The application was built using the Django web framework in the Python programming language. The application allows registered users to log in and new users to register before accesing the video streaming page. Video segments and HLS streaming manifest files were generated from video captured from a webcam using the ffmpeg tool, and the Nginx web server serves them as static files to the HLS player. Most of the application is served through the Gunicorn WSGI server, which receives user requests from the Nginx server and forwards them to the Django application. User login and registration were implemented using the built-in Django authentication system with a modified user model. Asynchronous communication using Django Channels and the WebSocket protocol was introduced to refresh the list of active users on the streaming page. Afterward, the application was implemented on a Raspberry Pi 3 computer with modified streaming code due to the Raspberry Pi's lower performance. MPEG-DASH streaming was also implemented to compare the two protocols. Delay and jitter were measured, and the results show that HLS streaming has approximately half the delay compared to MPEG-DASH streaming, while jitter has similar values.

Keywords: Django application, ffmpeg tool, Nginx web server, Raspberry Pi 3, live video streaming using HLS protocol

ŽIVOTOPIS

Luka Tadić rođen je 7.8.1999. godine u Novoj Gradiški. Osnovnu školu završio je u Cerniku pokraj Nove Gradiške i 2014. upisuje Opću gimnaziju u Novoj Gradiški. 2018. godine završava srednju školu i polaže državnu maturu nakon čega upisuje preddiplomski studij elektrotehnike i informacijskih tehnologija na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. 2021. godine stječe akademski naziv sveučilišni prvostupnik (lat. *baccalaureus*) inženjer elektrotehnike i informacijskih tehnologija. Te godine upisuje i diplomski sveučilišni studij elektrotehnike, smjer komunikacije i informatika, izborni blok mrežne tehnologije. 2022. godine postaje stipendist tvrtke TTTech-Auto.

Potpis:

_____

PRILOZI

Prilog P.3.1. *Python* kod za brisanje datoteka unutar direktorija *streaming*

```
folder = '/home/luka/StreamingWebsite/static/streaming/'
for filename in os.listdir(folder):
    file_path = os.path.join(folder, filename)
    try:
        if os.path.isfile(file_path) or os.path.islink(file_path):
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)
    except Exception as e:
        print('Failed to delete %s. Reason: %s' % (file_path, e))
```

Prilog P.3.2. Primjer *Python* koda za generiranje HLS strujanja s tri različite kvalitete

```
p = subprocess.Popen(
        '     ffmpeg \
            -input_format yuyv422 \
            -f v4l2 -video_size 1920x1080 \
            -i /dev/video0  \
            -preset ultrafast -g 25 -sc_threshold 0 \
            -filter_complex \
            "[0:v]split=3[v1][v2][v3]; \
            [v1]copy[v1out];                [v2]scale=w=1280:h=720[v2out];
[v3]scale=w=854:h=480[v3out]" \
            -map [v1out] -c:v:0 libx264 -b:v:0 3600k  \
            -map [v2out] -c:v:1 libx264 -b:v:1 2424832  \
            -map [v3out] -c:v:2 libx264 -b:v:2 964608 \
            -f hls \
            -hls_time 0.5 \
            -hls_segment_type mpegts \
            -hls_flags delete_segments \
            -master_pl_name master.m3u8 \
            -var_stream_map     "v:0     v:1     v:2"     stream_%v.m3u8',
stdout=subprocess.PIPE, shell=True)
```

Prilog P.3.3. Konfiguracijska datoteka *Nginx* poslužitelja za *Django* aplikaciju

```
server{
   listen 80;
   server_name 10.0.2.15;
location = /favicon.ico { access_log off; log_not_found off; }
location /static/ {
   alias /home/luka/StreamingWebsite/static/;
   autoindex on;
}
location / {
   proxy_pass http://10.0.2.15:8000 ;
   proxy_set_header X-Forwarded-Host $server_name;
   proxy_set_header X-Real-IP $remote_addr;
   proxy_redirect off;
   add_header P3P 'CP="ALL DSP COR PSAa OUR NOR ONL UNI COM NAV"';
   add_header Access-Control-Allow-Origin *;
}
```

Prilog P.3.4. Pogled *loginPage (request)* za prijavu korisnika i renderiranje sadržaja stranice za prijavu

```
      def loginPage(request):
          page = 'login'
          if request.user.is_authenticated:
              return redirect('stream')
          if request.method == 'POST':
              email = request.POST.get('email').lower()
              password = request.POST.get('password')
              try:
                  user = User.objects.get(email=email)
              except:
                  messages.error(request, 'User does not exist')
              user = authenticate(request, email = email, password =
password)
              if user is not None:
                  login(request, user)
                  return redirect('stream')
              else:
                  messages.error(request, 'Username or password does not
exist')
          context = {'page':page}
          return render (request, 'stream/login_register.html', context)
```

Prilog P.3.5. Pogled *registerPage (request)* za registraciju korisnika i renderiranje sadržaja stranice za registraciju

```
def registerPage(request):
    form=MyUserCreationForm()
    context={'form':form}
    if request.method == 'POST':
        form=MyUserCreationForm(request.POST)
        if form.is_valid():
            user=form.save(commit=False)
            user.username = user.username.lower()
            user.save()
            login(request,user)
            return redirect('stream')
        else:
            messages.error(request, "An error occured during registration")
    return render (request, 'stream/login_register.html', context)
```

Prilog P.3.6 Pogled *stream (request)* za renderiranje sadržaja stranice za gledanje videozapisa i pogled *users(request)* za renderiranje liste trenutno aktivnih korisnika

```
@login_required(login_url='login')
def stream(request):
    logged_users=[user.user for user in LoggedUser.objects.all()]
    context = {'logged_users':logged_users}
    print(context)

    return render (request, 'stream/stream.html',context)

@login_required(login_url='login')
def users(request):
    logged_users=[user.user for user in LoggedUser.objects.all()]
    context = {'logged_users':logged_users}
    return render(request,'stream/users-list.html', context)
```

Prilog P.3.7. Dodatak konfiguracijskoj datoteci *Nginx* poslužitelja za ispravan rad ASGI dio *Django* aplikacije

```
upstream websockets{

    server 10.0.2.15:8001;

}

location /ws/ {

        proxy_pass http://websockets;

        proxy_set_header Host                $http_host;

        proxy_set_header X-Real-IP           $remote_addr;

        proxy_set_header X-Forwarded-For     $proxy_add_x_forwarded_for;

        proxy_set_header X-Forwarded-Host    $server_name;

        proxy_set_header X-Forwarded-Proto   $scheme;

        proxy_set_header X-Url-Scheme        $scheme;

        proxy_redirect off;

        proxy_http_version 1.1;

        proxy_set_header Upgrade $http_upgrade;

        proxy set header Connection "upgrade";

    }

}
```

Prilog P.3.8. Primjer *Python* koda za generiranje HLS strujanja s tri različite kvalitete na računalu *Raspberry Pi 3*

```
p = subprocess.Popen(
        '     ffmpeg \
            -input_format yuyv422 \
            -f v4l2 -video_size 720x480 \
            -i /dev/video0  \
            -preset ultrafast -g 25 -sc_threshold 0 \
            -filter_complex \
            "[0:v]split=3[v1][v2][v3]; \
            [v1]copy[v1out];                  [v2]scale=w=640:h=360[v2out];
[v3]scale=w=320:h=240[v3out]" \
            -map [v1out] -c:v:0 libx264 -b:v:0 1500k   \
            -map [v2out] -c:v:1 libx264 -b:v:1 1200k  \
            -map [v3out] -c:v:2 libx264 -b:v:2 800k \
            -f hls \
            -hls_time 0.5 \
            -hls_segment_type mpegts \
            -hls_flags delete_segments \
            -master_pl_name master.m3u8 \
            -var_stream_map     "v:0     v:1     v:2"     stream_%v.m3u8',
stdout=subprocess.PIPE, shell=True)
```

Prilog P.3.9. Primjer *Python* koda za generiranje MPEG-DASH strujanja s tri različite kvalitete na računalu *Raspberry Pi 3*

```
p = subprocess.Popen(
     '      ffmpeg \
               -input_format yuyv422 \
               -f v4l2 -video_size 720x480 \
               -i /dev/video0  \
               -preset ultrafast -g 25 -sc_threshold 0 \
               -filter_complex \
               "[0:v]split=3[v1][v2][v3]; \
               [v1]copy[v1out];                [v2]scale=w=640:h=360[v2out];
[v3]scale=w=320:h=240[v3out]" \
               -map [v1out] -c:v:0 libx264 -b:v:0 1500k  \
               -map [v2out] -c:v:1 libx264 -b:v:1 1200k  \
               -map [v3out] -c:v:2 libx264 -b:v:2 800k \
               -use_timeline 1 \
               -use_template 1 \
               -window_size 5 \
               -adaptation_sets        "id=0,streams=0        id=1,streams=1
id=2,streams=2" \
               -f dash -seg_duration 1 manifest.mpd', stdout=subprocess.PIPE,
shell=True)
```