TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Joonas Praks 154838IABB

EESTI VABARIIGI AVALIKU VIDEOMATERJALI TÖÖTLEMINE JA ESITAMINE TASKUHÄÄLINGUNA

Bakalaureusetöö

Juhendaja: Gunnar Piho
PhD

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Joonas Praks

06.01.2020

Annotatsioon

Töö eesmärgiks oli luua veebiteenus, mis eraldab jooksvalt heli Eesti Vabariigi Valitsuse ja Riigikogu YouTube´i videokeskkonda üles laetud videomaterjalist, kohandab selle omadusi tarbimise hõlbustamiseks ning riputab saadud meediafailid valitud taskuhäälingu keskkonda, ent pakub neid ka RSS-i vahendusel sisukoondusteenustele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 21 leheküljel, 5 peatükki, 5 joonist.

Abstract

Processing and presenting public videos of the Republic of Estonia as a podcast

The purpose of this thesis was to create a web service that continuously extracts audio from videos uploaded to the YouTube channels of the Government and parliament of Estonia, adjusts its properties to provide a better listening experience and uploads the results to a selected podcasting service while also serving the files to RSS feed aggregators.

The thesis is in Estonian and contains 21 pages of text, 5 chapters, 5 figures.

Lühendite ja mõistete sõnastik

ACID	*Atomicity, Consistency, Isolation, Durability*, Andmebaasisüsteemi töökindluse printsiibid
API	*Application Programming Interface*, liides, mille alusel kasutab programm teise programmi teenuseid
dB	Detsibell, helitugevuse ühik
FFmpeg	Komplekt heli- ja videotöötlusvahendeid
HTTP	*Hypertext Transfer Protocol*, rakendusprotokoll teabe jagamiseks veebis kliendi ja serveri vahel
HTTPS	*Hypertext Transfer Protocol Secure*, HTTP laiendus, mis tagab, et vahendatav info oleks autenditud ja krüpteeritud
ID3	Meta-andmete konteiner, MP3-e *de facto* meta-andmete standard
JSON	*JavaScript Object Notation*, tekstistandard andmete struktureerimiseks, alternatiiv XMLile
Kodek	Programm, mis kodeerib või dekodeerib andmeid
kbit/s	Kilobitti sekundis
Lubadus	*Promise*, JavaScripti objekt, mis sisaldab mingil ajahetkel lõppeva asünkroonse protsessi tulemust.
MP3	Heli kodeerimisformaat
RSS	*Really Simple Syndication*, XML-il rajatud vorming, millega sisupakkujad sissekandeid jagavad
RSS-voog	*RSS feed*, uudisvoog, mille alusel pakuvad RSS-i sisukoondajad tarbijale uut teavet
Sisukoondaja	*Content aggregator*, teenus või rakendus, mis pakub korraga erinevate sisupakkujate sissekandeid
Taskuhääling	*Podcast*, episoodiline helifailidele rajatud netisaade
URL	*Uniform Resource Locator*, veebiaadress
Veebiaadressi päring	*URL query component*, *?*-märgiga algav võti-väärtus paaridest koonsev sõne veebiaadressis pärast teed
Veebiaadressi tee	*URL path component*, */*-märkitega segmenteeritud sõne veebiaadressis pärast porti
XML	*Extensible Markup Language*, märgistuskeel andmete struktureerimiseks, alternatiiv JSONile

Sisukord

1 Sissejuhatus 9
1.1 Probleem 9
1.2 Eesmärk 9
2 Metoodika 9
2.1 Ülevaade objektist 9
2.2 Ülevaade tööriistadest 10
2.3 Ülevaade protsessist 12
3 Tulemus 12
3.1 Testversioon 13
3.2 Moodulid 14
3.3 *Server* 15
3.4 *Subscriber* 15
3.5 *Endpointhandler* 16
3.5.1 Tellimuse kinnitamine 16
3.5.2 YouTube'i teadete vastuvõtmine 16
3.5.3 RSS-voo loomine 19
3.5.4 Failide voogesitamine 19
3.5.5 Kodulehe kuvamine 20
3.6 *AudioProcessor* 20
3.6.1 *EditAudio* 20
3.6.2 *EditAudioMetadata* 21
3.7 *PodBeanManager* 21
3.7.1 *StartUploading* 22
3.7.2 *UpdatePodcast* 22
3.8 *LocalFileManager* 22
3.8.1 *CreateDescription* 23
3.8.2 *CreateRSS* 23
3.8.3 *CreateHTML* 23
3.8.4 *RemoveOldContent* 23
3.8.5 *GetAudioById* 23
3.8.6 *GetAudioFiles* 23

3.8.7 *GetAudioListSortedByDate* 23
3.8.8 *GetDescriptionFileOfAudio* 24
3.8.9 *GetIdOfAudio* 24
3.8.10 *GetMetadataFromAudio* 24
3.8.11 *GetProcessingAudioFile* 24
4 Analüüs ja järeldused 24
4.1 Vead 24
4.2 Põhjendused erinevatele otsustele 26
4.2.1 PodBean'i ja RSS-i kattuvus 26
4.2.2 Failitöötluse parameetrid 26
4.2.3 Andmete talletamine 27
4.2.4 Süsteemi piirangud 27
4.2.5 Verifikatsiooni puudumine 28
4.3 Mida tulevikus teha 28
4.3.1 Süsteemi oleku hindamine 28
4.3.2 Failide edasi kerimine 28
4.3.3 Täiendatud ffmpeg käsk 28
4.4 Alternatiivtoode 29
5 Kokkuvõte 29
Kasutatud kirjandus 30

Jooniste loetelu

Joonis 1. Teenustevaheline suhtlus uue video puhul 13
Joonis 2. Moodulite sõltuvused 15
Joonis 3. Teatiste tellimine 16
Joonis 4. Uue materjali lisamine 17
Joonis 5. Näide Pubsubhubbub Hub'i teatise formaadist [19] 18

1 Sissejuhatus

1.1 Probleem

Eesti Vabariigi valitsus ja Riigikogu peavad regulaarselt vastavalt pressikonverentse ja istungeid, mis talletatakse videoplatvormil YouTube [1] [2]. Mainitud keskkond pole aga mugav, kui üles laetud materjali tahetakse tarbida liikvel olles, kuna YouTube näitab oma ainest tavakasutajale vaid juhul, kui seadme, millel rakendus töötab, ekraan on sisse lülitatud [3]. Aeg-ajalt võib videomaterjalis kohata ka sisutühje pause [4] ning helitaset, mis on liiga vaikne, et seda mürarohkes keskkonnas tulusalt kuulata. Samuti on valitsuse ja Riigikogu videote jälgimine üleliia kulukas, sest kasutatud andmemaht on suurem kui ilma videota ning kuuldavat teavet täiendab pilt marginaalselt.

1.2 Eesmärk

Pakkumaks lahendust loetletud murekohtadele otsustati luua veebiteenus, mis lubab tarbijal riigi videomaterjali taskuhäälingu- või RSS-i põhise sisukoondusteenuse kaudu kuulata. Lisaks on valmiva teenuse siht eemaldada tühjad lõigud, parendada helitugevust ning vähendada pakutavate audiofailide andmemahtu.

2 Metoodika

2.1 Ülevaade objektist

Lähtudes toodud punktidest, peab loodav teenus vastama järgnevatele funktsionaalsetele nõuetele:

- Teenus ootab teateid Riigikantselei ja Riigikogu YouTube´i kanalitelt
- Teatise saamisel eraldatakse ja talletatakse vastavast videost heli

- Heli laetakse üles mõnda taskuhäälingu keskkonda
- Heli on võimalik tarbida RSS-i sisukoondusteenuse abil
- Heli salvestatakse koos vastava video pealkirja ning kirjeldusega
- YouTube´i video pealkirja ja kirjelduse muutudes peavad muutuma ka talletatud heli ning taskuhäälingu keskkonna eksemplari pealkiri ja kirjeldus

Lisaks peab teenus vastama järgnevatele mittefunktsionaalsetele nõuetele:

- Juhul kui YouTube´i videot uuendatakse enne helitöötluse valmimist, tuleb teade ajutiselt tagasi lükata
- Teenuse RSS-voogu pakkuv lõpp-punkt peab kasutama HTTPS protokolli

2.2 Ülevaade tööriistadest

Teenus kirjutati NodeJS raamistikus, kasutades ES6 Javascripti standardit. Valiku tegemisel lähtuti programmeerija varasemast kokkupuutest Javascriptiga ning võimalusest NodeJS-i abil väikese ajakuluga üles seada veebiserver, mis täidaks püstitatud eesmärkideks vajalikku veebiülest suhtlust.

Vaadeldavates YouTube´i kanalites toimunud muudatuste jälgimiseks kasutati WebSub protokolli, mis lubab huvipakkuvatele teemadele tellida teatisi [5]. WebSub on rajatud webhooks kontseptioonile, mis vastandub varasemale tehnoloogiale – pollimisele. Erinevalt viimasest, kus tarbija teeb valitud intervalli järel huviobjektile värskenduspäringuid, teavitab valitud teenus ise teatise tellijat [6]. WebSub lisab webhooksile turvaelemendi. Tellimuse saatmisel lisatakse päringusse salasõne ning iga teavituse päis kaasab HMAC-tüüpi sõnumiautentimiskoodi [7], mille abil tellija saab kontrollida sõnumi allika õigsust. WebSubi protokollil on mitmeid realisatsioone, käesolev töö kasutas Google'i teenust PubSubHubbub Hub (edaspidi Hub) [8].

Heli käsitsemiseks ning MP3 failide meta-andmete lugemiseks kasutati FFmpeg projekti programme ffmpeg ja ffprobe [9]. Valiku tegemisel aitas FFmpeg-ile rajatud ffmpeg-fluent-nimeline API Javascriptiga integreerimiseks.

Taskuhäälingu lehekülg, mida perioodiliselt uue materjaliga täiendada oli PodBean.com. Ligipääsuks piiramatutele üleslaadimistele ning massmälule soetati PodBean'i tasuline konto.

Versioonihaldus vahendiks valiti Git. Koodi hoiustati GitHub'i kaugkoodihoidlas.

Arenduskeskkonnana kasutati rakendust Visual Studio Code Windowsi operatsioonisüsteemil. Tegemist on tasuta ning kergekaalulise tarkvaraga, mida saab lihtsasti täiendada teiste arendajate loodud pistikprogrammidega [10].

Tagamaks, et teenus taaskäivitub pärast võimalikku kokkujooksmist rakendati forever aplikatsiooni.

Teenus seati sisse Vultri privaatserverisse. Alternatiivina oleks võinud kasutada DigitalOceani, ent arenduse ajal pakkus Vultr esmakordsetele liitujatele piiratud aja raames tasuta majutust. Virtuaalmasina operatsioonisüsteemiks valiti Ubuntu 19.10 x64.

Veebimajutus.ee'ga registreeriti domeen riigipodcast.ee, et RSS-voog oleks RSS-i tarbijatele pilkupüüdvam ja kergemini leitav.

Töös kasutati mitmeid teeke, mida hallati NodeJS-iga kaasa pakendatud npm paketihalduri abil.

- fluent-ffmpeg: 2.1.2, API FFmpeg-i hõlspamaks kasutamiseks Javascriptiga [11]
- rss: 1.2.2, koostab korrektse XML-iga RSS-i [12]
- simple-node-logger: 18.12.23, logide koostamiseks [13]
- superagent: 5.1.0, teek AJAX kutsete tegemiseks [14]
- xml2js: 0.4.19, XMLi teisendamiseks [15]
- ytdl-core: 0.29.5, YouTube'i videote laadimiseks ja meta-andmete pärimiseks [16]

2.3 Ülevaade protsessist

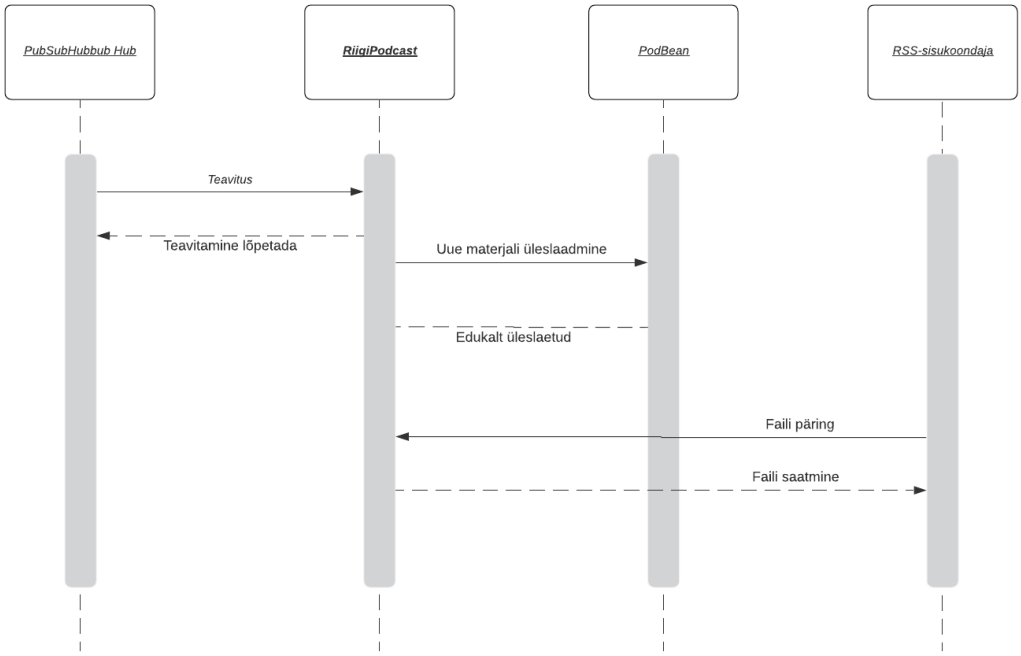
Tarkvara arendati VSCode'is praktiseerides Javascriptiga funktsionaalset programmeerimist. Koostati konfiguratsioonifail, mille abil hoiti lahus arendus- ja tooteeksemplarid. Lisati logisüsteem, mille abil paremini hinnata toimunud muutuseid töötavas süsteemis. Uue komponendi valmimisel või vea parandamisel laeti kood Git'i abil GitHub'i kaugkoodihoidlasse. Seejärel logiti ennast kaugvirtuaalarvutisse ning laeti alla hoidlas talletatud kood. Kood käivitati esmalt arenduseksemplarina, tulemuste kontrollimiseks muudeti arenduseksemplariga seotud YouTube'i kanali videote pealkirju ja kirjeldusi. Teenuse rahuldaval käitumisel võeti maha endine tooteeksemplar ning asendati see forever rakenduse abil uue versiooniga.

3 Tulemus

Töö tulemusena valmis veebiteenus RiigiPodcast. Teenus töötab ning vastab püstitatud eesmärkidele. Lähtekood asub https://github.com/Joonaspraks/riigitaskuh22ling aadressil. RiigiPodcast on saanud positiivset tagasisidet Riigikogu administratsioonilt.

Kasutades Hub'i tellib teenus endale Valitsuse ja Riigikogu YouTube'i kanalitelt teatised. Kui üks mainitud kanalitest muudab või lisab videoid, saab RiigiPodcast toimingust teate. Teate alusel hangib teenus videovoo ning töötleb selle ffmpeg rakenduse abil mugandatud helifailiks. Seejärel laetakse tulemus taskuhäälingu veebisaidile, PodBean'i (Joonis 1).

Tingimusel, et teatises mainitud video on süsteemis juba olemas, uuendatakse olemasoleva faili pealkirja ja kirjeldust. Samuti muudetakse vastava taskuhäälingu episoodi andmeid.



Joonis 1. Teenustevaheline suhtlus uue video puhul

Teenuse loodud helifaile saab kasutada RSS sisukoondajate abil, lisades allikate hulka URL-i riigipodcast.ee/feed. RiigiPodcast loob ja tagastab RSS-voo 20st värskeimast helifailist.

3.1 Testversioon

RiigiPodcast'i tööd on võimalik proovida testversiooni abil. Selleks on vaja siseneda kaugarvutisse aadressil riigipodcast.ee, et käivitada teenus. Seejärel siseneda testimiseks loodud YouTube'i kasutajasse ja muuta mõnda sinna laetud videot. Muutuse tulemus on mõninga aja pärast nähtaval aadressil http://riigipodcastdemo.podbean.com/.

1) Sisenege SSH-ga aadressile riigipodcast.ee, kus kasutajanimi on *demouser* ja salasõne on *demouser1234*

2) Minge kausta riigitaskuh22ling/src

3) Käivitage testversioon kirjutades käsureale *NODE_ENV=dev forever start server.js*

4) Sisenege YouTube’i, kasutajanimi on *riigipodcastdemo@gmail.com* ja salasõne on *Demo1234*

5) Aadressil https://studio.youtube.com/channel/UC0FfpQ9PI9TSjuDl4byyjKQ/videos muutke mõne või mõlema faili pealkirja või kirjeldust

6) Tulemust näeb aadressil http://riigipodcastdemo.podbean.com/.

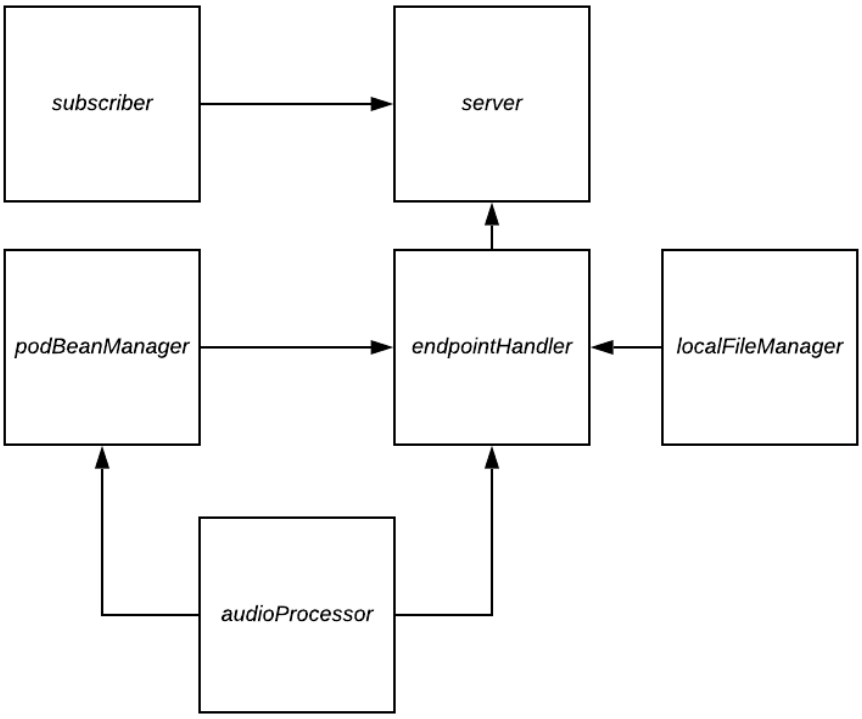
7) Kohalikult salvestatud faile näeb kaustas /riigitaskuh22ling/src/audioStorage/devAudio

Kui enne tehtud toiminguid faile süsteemis ei eksisteerinud, tekib kummagi video kohta üks kohalik helisalvestis ning üks taskuhäälingu episood. Kui failid olid varem olemas, on nende kirjeldus ja pealkiri muutunud.

Testimiseks kasutatav PodBean’i kasutaja ei ole tasuline variant. Seepärast võib päevas faile üles laadida vaid piiratud arv kordi.

3.2 Moodulid

Teenus jaguneb kuueks mooduliks: *server, subscriber, endpointHandler, audioProcessor, podBeanManager, localFileManager.*



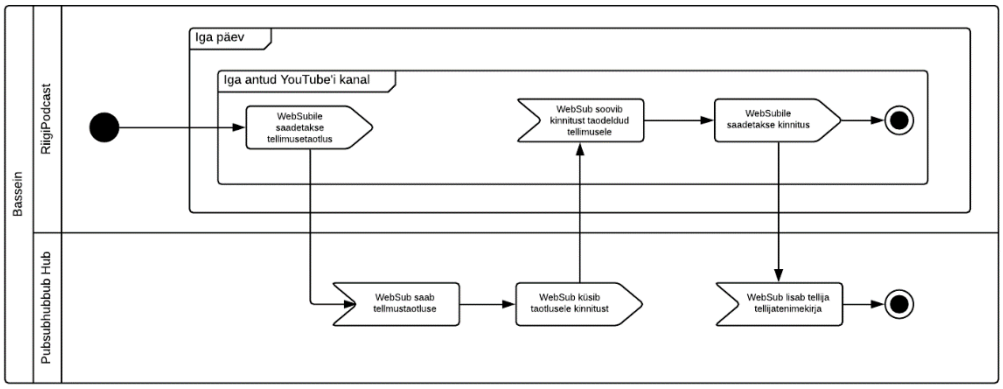
Joonis 2. Moodulite sõltuvused

3.3 *Server*

Server kutsub välja *subcsriber* mooduli ja seab püsti veebiserverid, mis veebikutseid vastu võtavad ning *endpointhandler* moodulile edastavad. Kontrollitakse, kas teenus on käivitatud arengu- või lõpptooterežiimis. Arengujärje korral luuakse üks server, mis võtab vastu ja delegeerib HTTP kutseid. Lõpptooterežiimis moodustatakse üks HTTPS ning üks HTTP server. HTTPS server võtab vastu ja delegeerib HTTPS kutseid. HTTP server suunab kõik HTTP kutsed ümber HTTPS protokollile.

3.4 *Subscriber*

Subscriber saadab Hub'ile taotluse tellida teateid etteantud YouTube'i kanalitelt. Taotlusele liidetakse sõne, mille abil server hiljem sissetulevaid teatiseid autendib. Lisatud on taimer, mis kordab *subscriber*'i tegevust kindla intervalli järel, sest teavitusi ei saa Google'i WebSubi teostuse abil määramata ajaks tellida.



Joonis 3. Teatiste tellimine

3.5 *Endpointhandler*

Endpointhandler võtab vastu ja vastab veebikutsetele. Kirjeldatud lõpp-punktid käivitavad erinevaid süsteemi protsesse.

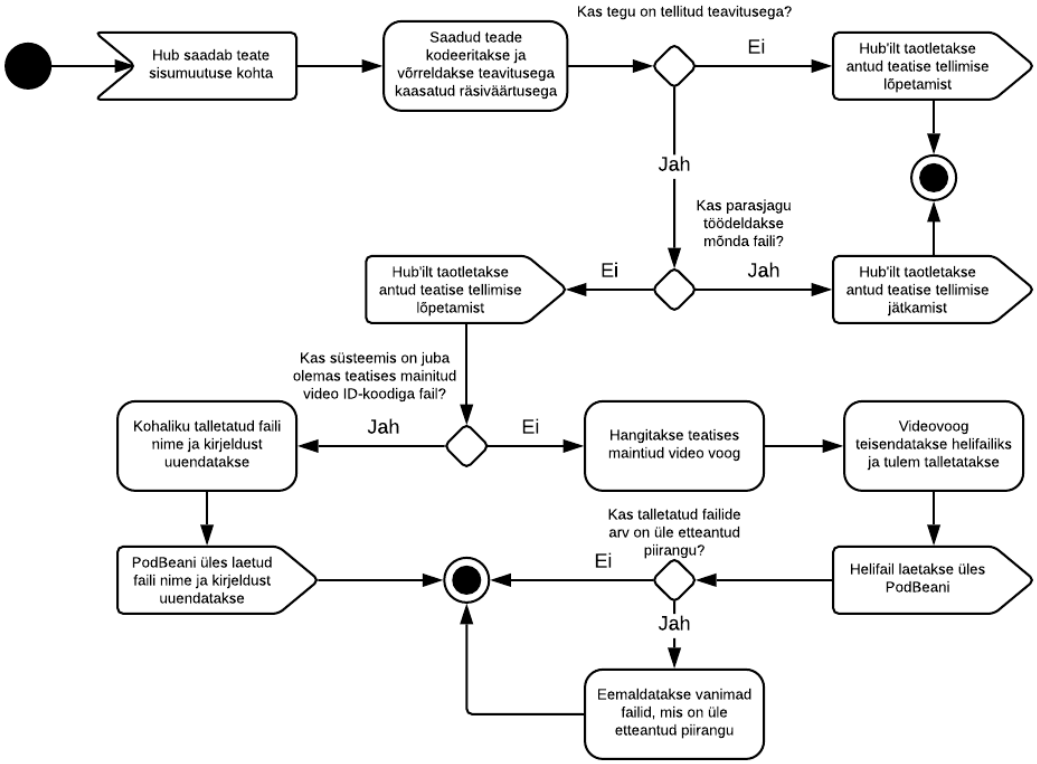
3.5.1 Tellimuse kinnitamine

Protsess kinnitab Hub'i saadetud kinnitussoovi taotletud tellimusele. See käivitub, kui päringumeetod on GET ning päringu aadressis on väli *topic*, mille väärtus sisaldab soovitud YouTube'i kanali ID-koodi.

Tellimuse kinnitamise märgiks saadetakse WebSub konventsioonile vastavalt vastuseks päringu aadressi *challenge* välja väärtus ning HTTP olekukood 200 [5]. Tellimuse kinnitus käivitub pärast seda, kui *subscriber* komponent on saatnud tellimuse taotluse.

3.5.2 YouTube'i teadete vastuvõtmine

Protsess veendub saadetud teatise õigsuses ning loob uue helifaili, mis laetakse taskuhäälingu veebisaidile, või uuendab olemasolevat faili (Joonis 4). See käivitub, kui päringumeetod on POST ning URL-i päise *link* väli sisaldab viidet Hub'ile.



Joonis 4. Uue materjali lisamine

Saadetava teate alusel genereerib Hub eelnevalt tellimustaotluses esitatud salasõne abil HMAC allkirja [17]. Hub kasutab räsimiseks SHA-1 algoritmi [17]. Allkiri lülitatakse teatise päise *x-hub-signature* välja.

Teatis kodeeritakse lokaalselt samade meetmetega, mis Hub: SHA-1 algoritm ja salasõne. Tulemit võrreldakse *x-hub-signature* väärtusega. Räsiväärtuste mittevõrdsuse puhul vastatakse Hub'ile olekukoodiga 200, andes märku, et antud teatise tellimusi soovitakse katkestada ning protsess lõpeb [17].

Hub'i saadetud teatis on XML formaadis (Joonis 5). Xml2js mooduli abil teisendatakse teade JSON-formaati. Andmeid on sel viisil kergem eraldada, sest standardiseeritud JavaScript toetab JSON-it natiivselt [18].

```
<feed xmlns:yt="http://www.youtube.com/xml/schemas/2015"
         xmlns="http://www.w3.org/2005/Atom">
  <link rel="hub" href="https://pubsubhubbub.appspot.com"/>
  <link rel="self"
href="https://www.youtube.com/xml/feeds/videos.xml?channel_id=
CHANNEL_ID"/>
  <title>YouTube video feed</title>
  <updated>2015-04-01T19:05:24.552394234+00:00</updated>
  <entry>
    <id>yt:video:VIDEO_ID</id>
    <yt:videoId>VIDEO_ID</yt:videoId>
    <yt:channelId>CHANNEL_ID</yt:channelId>
    <title>Video title</title>
    <link rel="alternate"
href="http://www.youtube.com/watch?v=VIDEO_ID"/>
    <author>
     <name>Channel title</name>
     <uri>http://www.youtube.com/channel/CHANNEL_ID</uri>
    </author>
    <published>2015-03-06T21:40:57+00:00</published>
    <updated>2015-03-09T19:05:24.552394234+00:00</updated>
  </entry>
</feed>
```

Joonis 5. Näide Pubsubhubbub Hub'i teatise formaadist [19]

Teatest eraldatakse YouTube'i kanali ID-kood *yt:channelId*, seda võrreldakse oodatud YouTube'i kanalite ID-koodi loendiga. Kui puudub kood, mis oleks võrdne teatises antud kanali koodiga, operatsioon lõpeb. Vastasel juhul eraldatakse teatisest lisaks kanali koodile veel video pealkiri *title* ning video ID-kood *yt:videoId*.

Järgmiseks veendutakse, et süsteem ei ole parasjagu hõivatud failitöötlusega. Videot töödeldes luuakse ajutine *.tmp*-lõpuline fail (vt peatükk 3.6.1). Leides kohalike talletatud andmete seast sellise faili, ei vastatakse olekukoodiga 503. See annab Hub'ile märku, et teenus pole hetkel saadaval ja antud teatise tellimust ei soovita katkestada ning protsess lõpeb [17].

Jäädvustatud andmete hulgast otsitakse faili, mille nimi oleks võrdne sissetulnud teate video ID-koodiga. Olemasoleva faili puudumisel luuakse uus helifail, mis laetakse taskuhäälingu saidile. Juhul kui sellise nimega fail on juba olemas, värskendatakse kohaliku ning taskuhäälingu saidile laetud faili andmeid. Enne protsessi lahknemist hangitakse ytdl-core mooduliga teatise video kirjeldus ning taskuhäälingu API kasutamiseks vajalikud ID-kood ja salasõne.

Uue YouTube'i video korral saadakse ytdl-core'i abil kätte videovoog, mis töödeldakse helifailiks (vt peatükk 3.6.1). Faili nimeks saab vastava YouTube'i video ID-kood. Faili ümbertöötamise lõpul saadetakse Hub'ile vastuseks olekukood 200, andes märku, et antud teatise tellimusi soovitakse katkestada [17]. Töötluse tulemust asutakse laadima valitud taskuhäälingu veebisaidile (vt peatükk 3.7.1). Seejärel luuakse videokirjelduse põhjal tekstfail ning eemaldatakse vanimad kohalikud failid, mis on üle määratud kogusepiirangu (vt peatükk 3.8.1, 3.8.4).

Tingimusel, et YouTube'i teatise video alusel on varem juba helifail loodud, saadetakse Hub'ile vastuseks olekukood 200, andes märku, et antud teatise tellimusi soovitakse katkestada [17]. Faili pealkiri muudetakse sissetuleva videoteate pealkirjaks (vt peatükk 3.6.2). Faili kirjeldus asendatakse uue kirjeldusega (vt peatükk 3.8.13.8). Hangitakse failile vastava taskuhäälingu saidi episoodi ID-kood (vt peatükk 3.8.10), mille kaudu uuendatakse taskuhäälingu pealkirja ning kirjeldust (vt peatükk 3.7.2).

3.5.3 RSS-voo loomine

Protsess edastab päringu saatjale salvestatud helifailidele vastava RSS-voo. See käivitub, kui päringumeetod on GET ning URL-i aadressi tee on võrdne sõnega */feed*.

RSS-voo koostamine on täielikult delegeeritud *localFileManager*'ile (vt peatükk 3.8.2). Vastuse sisutüübiks kirjutatakse RSS-i meediatüüp *application/rss+xml* ning voog saadetakse päringu esitajale [20].

3.5.4 Failide voogesitamine

Protsess esitab päringu saatjale soovitud helifaili andmevoo. See käivitub, kui päringumeetod on GET ning sisse tulnud URL-i päringus esineb *file* võti.

File välja väärtuse abil otsitakse vastava nimega helifaili. Faili leidmisel kirjutatakse vastuse sisutüübiks *audio/mpeg* ning märgitakse ära faili pikkus [21] [22]. Helifaili põhjal luuakse andmevoog ja see suunatakse vastusesse. Kui heli ei suudetud *file* välja alusel leida, saadetakse vastusena HTTP olekukood 404, et teavitada kasutajat otsitud andmete puudumisest [23].

3.5.5 Kodulehe kuvamine

Protsess esitab päringu saatjale veebilehekülje, mis osutab taskuhäälingu saidi ning RRS-voo ressursside poole. See käivitub, kui päringumeetod on GET ning päringu aadressi rajakomponent on võrdne sõnega /.

HTML lehekülje genereerib *localFileManager* (vt peatükk 3.8.3). Tulem saadetakse päringu saatjale.

3.6 *AudioProcessor*

AudioProcessor töötleb videod sobilikeks helifailideks ja hoiustab helifailide meta-andmete väljadel vajalikku infot. Moodulil on kaks funktsiooni: *editAudio* ning *editAudioMetadata*.

3.6.1 *EditAudio*

Funktsioon töötleb pakutud videovoo MP3-formaadis helifailiks ning talletab video nime loodud faili meta-andmete *title* märgise väärtusena. *EditAudio* nõuab kolme parameetrit: videovoog, video nimi ja video ID-kood.

Fluent-ffmpeg moodulist kutsutakse funktsioon *ffmpeg*, mis lubab programmiliselt luua ja käivitada ffmpeg rakenduse kutse. Töötluse allikaks määratakse videovoog, formaadiks pannakse MP3, bitiedastuskiiruseks seatakse 96 kbit/s.

Lisatakse kaks helifiltrit: *silenceremove* ja *dynaudnorm*. *Silenceremove* eemaldab osa helist, mis vastab etteantud reeglitele [24]. Parameetrid, mis 3.4.1 e*ditAudio* filtrile annab on:

- *stop_periods*: mis ajahetkedel peaks vaikust välja lõikama? [24]
- *stop_duration*: kui pikk peab vaikus olema, et seda välja lõigata? [24]
- *stop_threshold*: millist helitaset peaks programm pidama vaikuseks? [24]

Parameetrite väärtusteks antakse vastavalt -1, 3 ning -35dB.

Dynaudnorm vähendab heli dünaamilisust, tuues vaiksemad ja valjemad lõigud sarnasemale helitugevusele [25]. Kuna heli normaliseerimine teeb valjemad hetked

vaiksemaks, saab kogu heli tugevust suurendada [25]. *Dynaudnorm* filtrile e*ditAudio* parameetreid ei esita.

Lõplik väljundfaili nimi koostatakse ühendades kausta, kus faile talletatakse aadress, video ID-kood ning *.mp3*-nimelaiend. *Ffmpeg*-i väljundnimeks märgitakse ajutine nimi, mis saadakse liites eelnevalt loodud nimele *.tmp*-nimelaiendi. Valmiva MP3-faili meta-andmete märgendi *title* väärtuseks palutakse seada video pealkiri. Pärast mainitud täpsustusi formuleerib ning käivitab fluent-ffmpeg tagaplaanil ffmpeg-ile sobiva kutse. Töö lõppedes nimetatakse tulemus ümber ajutiselt nimelt lõplikule nimele.

3.6.2 *EditAudioMetadata*

Funktsioon lisab etteantud MP3 faili soovitud meta-andmete märgendile uue väärtuse. *EditAudioMetadata* nõuab kolme parameetrit: failinimi, meta-andmete märgend ja märgendi väärtus.

Fluent-ffmpeg moodulist kutsutakse funktsioon *ffmpeg*, mis lubab programmiliselt luua ja käivitada *ffmpeg* rakenduse kutse. Töötluse allikaks määratakse faili aadress, formaadiks pannakse MP3, audiokodeki väärtuseks sätestatakse *copy*.

Ffmpeg-i väljundnimeks märgitakse ajutine nimi, mis saadakse liites olemasolevale nimele *.tmp*-nimelaiendi. Valmiva MP3-faili etteantud meta-andmete märgendi väärtuseks palutakse seada edastatud märgendi väärtus. Pärast mainitud täpsustusi formuleerib ning käivitab fluent-ffmpeg tagaplaanil ffmpeg-ile sobiva kutse. Töö lõppedes kustutatakse algne sisendfail ning ffmpeg-i tulemus nimetatakse ümber ajutiselt nimelt algsele nimele.

3.7 *PodBeanManager*

PodBeanManager laeb PodBeani leheküljele uut materjali ning uuendab aegunud taskuhäälingu episoode. Moodulil on kaks avalikku funktsiooni: *startUploading* ning *updatePodcast*.

Mõlemad funktsioonid juhinduvad tulemuste saavutamiseks PodBeani API-st ning selle dokumentatsioonist [26]. Seetõttu on enamus koodist nendes protsessides triviaalne ning üksikasjalikult seda ei kirjeldata. Kõik HTTP-kutsed konstrueeritakse ning saadetakse superagent mooduli abil.

3.7.1 *StartUploading*

Funktsioon laeb saidile http://riigipodcast.podbean.com/ üles valitud helifaili ja avaldab selle taskuhäälinguna. PodBean'ilt saadav taskuhäälingu ID-kood salvestatakse kohaliku helifaili meta-andmete *TIT3* märgise väärtusena. *StartUploading* nõuab nelja parameetrit: video ID-kood, video pealkiri, video kirjeldus ning PodBean'i API kasutamiseks vajalikud ID-kood ning salasõne.

Esmalt taotletakse ligipääsuluba, kasutades API ID-koodi ja salasõne. Teiseks küsitakse voli faili üleslaadimiseks, saates PodBean'ile ligipääsuloa, faili nime ja faili suuruse. Eduka pöördumise puhul talletatakse vastusest saadud URL, mida tuleb kasutada faili üleslaadimise sihtaadressina ning failivõti, mille alusel hiljem üleslaetud episoodi avaldada. Järgmisena laetakse eelnevalt nimetatud URL-i abil helifail üles. Fail avaldatakse taskuhäälingu episoodina kasutades varem saadud failivõtit ja lisades taotlusesse video pealkiri ning kirjeldus.

Õnnestunud avaldamine tagastab taskuhäälingu episoodi ID-koodi. See hoiustatakse kohalikult helifaili meta-andmete *TIT3* märgise väärtusena. Meta-andmeid aitab salvestada *audioProcessor* (vt 3.6.2).

3.7.2 *UpdatePodcast*

Funktsioon uuendab varem üles laetud taskuhäälingu nime ning kirjeldust. *UpdatePodcast* nõuab nelja parameetrit: PodBean'i taskuhäälingu episoodi ID-kood, helifaili pealkiri, helifaili kirjeldus ning PodBean'i API kasutamiseks vajalikud ID-kood ning salasõne.

Esmalt taotletakse ligipääsuluba, kasutades API ID-koodi ja salasõne. Seejärel uuendatakse käesoleva failiga seotud taskuhäälingu episoodi, liites siht-URL-ile lõppu episoodi ID-koodi ning lisades taotlusesse helifaili pealkiri ning kirjeldus.

3.8 *LocalFileManager*

LocalFileManager loob ja haldab andmeid. Enamik mooduli funktsioonidest on lihtsad abimeetodid, mis muundavad sõnesid või tagastavad faile. *LocalFileManager*'is asuvad RSS-voo, HTML-i ja kirjeldusfaili looja. Samuti leiab siit funktsiooni, mis eemaldab vanimad kohalikud failid, mis on üle määratud kogusepiirangu.

3.8.1 *CreateDescription*

Funktsioon loob uue tekstfaili, mis sisaldab helifaili kirjeldust. *CreateDescription* nõuab kaht parameetrit: helifaili nime ning kirjeldust. Kui sama nimega fail on olemas, kirjutatakse vana fail üle.

3.8.2 *CreateRSS*

Funktsioon tagastab lubaduse kujul talletatud kohalike andmete alusel uue RSS-voo.

Luuakse uus voog, millele helifaili viiteid lisada. Iga helifaili (vt peatükk 3.8.6) kohta leitakse selle pealkiri (vt peatükk 3.8.10), kirjeldusfaili sisu ning helifaili sünnihetk. Kõigi kolme protsessi asünkroonilisuse tõttu tagastatakse nende vastused lubadustena. Kui kõik kolm on töö lõpetanud, tagastatakse nende vastused ühise objektina.

Tingimusel, et iga audiofaili kohta on tagastatud andmeobjekt need sorteeritakse ajaliselt kahanevas järjekorras. Objektide alusel luuakse helifailidele viited ja need lisatakse voogu.

3.8.3 *CreateHTML*

Funktsioon tagastab sõne, mille veebilehitsejad visualiseerivad HTML-lehena. Lehekülg osutab taskuhäälingu saidi ning RRS-voo ressursside poole.

3.8.4 *RemoveOldContent*

Funktsioon sorteerib helifailid ajaliselt (vt peatükk 3.8.7) ning eemaldab koos neile vastavate kirjeldusfailidega kõik, mis ei mahu etteantud kogusepiirangusse.

3.8.5 *GetAudioById*

Funktsioon leiab kõik helifailid (vt peatükk 3.8.6) ning tagastab faili, mille failinimi on võrdne sissetulnud ID-koodiga. *GetAudioById* nõuab üht parameetrit: helifaili nimi. Faili puudumisel tagastatakse *undefined*.

3.8.6 *GetAudioFiles*

Funktsioon tagastab kõik helifailid.

3.8.7 *GetAudioListSortedByDate*

Funktsioon tagastab kõik helifailid ajaliselt kahanevas järjekorras sorteerituna.

Luuakse jada kõikidest talletatud helifailidest (vt peatükk 3.8.6). Igale failile lisatakse ajutiselt juurde tema loomise hetk. Toimub aja järgi sorteerimine ning pärast ajaväärtuste eemaldamist failid tagastatakse.

3.8.8 *GetDescriptionFileOfAudio*

Funktsioon tagastab helifailile vastava kirjeldusfaili. *GetDescriptionFileOfAudio* nõuab üht parameetrit: helifail.

3.8.9 *GetIdOfAudio*

Funktsioon tagastab helifailile vastava failinime. *GetIdOfAudio* nõuab üht parameetrit: helifail.

3.8.10 *GetMetadataFromAudio*

Funktsioon tagastab lubaduse kujul antud helifaili meta-andmete märgise väärtuse. *GetMetadataFromAudio* nõuab kaht parameetrit: helifail ning meta-andmete märgis. Fluent-ffmpeg moodulist kutsutakse funktsioon *ffprobe*, mis lubab programmiliselt luua ja käivitada ffprobe rakenduse kutse.

3.8.11 *GetProcessingAudioFile*

Funktsioon tagastab faili, mille nimelaiend on *.tmp*. Faili puudumisel tagastatakse *undefined*.

4 Analüüs ja järeldused

4.1 Vead

Pärast töö valmimist on ilmnenud mõned puudused, mida oleks saanud süsteemi arendamise ja disainimise perioodil vältida. Järgnevalt on toodud tehtud vead ning kuidas neid tulevikus vältida proovitakse.

- Hub'i dokumentatsioon näeb ette, et teatise saamisel vastaks klient nii kiiresti kui võimalik [27]. RiigiPodcast võiks sellest nõudmisest eeskujulikumalt kinni

pidada. Süsteemis esineb kohti, kus Hub'ile peaks varem vastama ning kui-lauseid, mis jätavad vastuse üldse andmata.

- Arenduse jooksul on tulnud ette olukordi, kus jälgitavate YouTube'i kanalite haldaja muudab mõnda videot ning väga väikese ajavahemiku järel teeb seda uuesti. Hub'i asünkroonsuse tõttu võib tekkida olukord, kus esimene parandus jõuab loodud teenuseni pärast teist ning kirjutab üle õiged andmed. Sellise tulemuse vältimiseks oleks RiigiPodcast võinud hoiustada Hub'i teatises sisalduvat redigeerimisaega ning selle alusel kindlaks teinud kas muudatusi on vaja jõustada või ei.

- Teenusega töötamise jooksul on pakutav RSS-voog sisaldanud andmeid, mis on seal vaid arenduse testimiseks ning andmeid, mille vorm on muutunud. Sisukoondajad jäädvustavad aga pidevalt koormuse vähendamiseks leitud voogude sisu. Seega võib mitme sellise teenuse puhul ette tulla, et RiigiPodcast'i RSS-voo lisamisel leidub seal viiteid failidel, mida tegelikult ei eksisteeri või mille veebiaadress pole enam kehtiv. Tulevikus tuleb enne oma andmete üles laadimist olla ettevaatlikum ja teadlikum võimalikest tagajärgedest.

- Vanemate failide veebiaadressid võivad olla loodud nende pealkirjade järgi. Helifailide individuaalsed URL-id põhinesid algselt faili vanusest tingitud järjekorranumbritel. See tähendas, et samale veebiaadressile võis erineval ajahetkel vastata erinev helifail. Käesolevas süsteemis on URL-id muudetud YouTube'i ID-koodi-põhisteks. Kahjuks ei talletatud varem failide ID-koode ning seetõttu on vanemate failide ID-koodideks nende helifailide pealkirjad.

- Teenus oleks võinud rohkem ära kasutada asünkroonseid programmeerimisvõimalusi. Koodimisele oleks pidanud eelnema diagramm, mille järgi hinnata, kas teatud protsessid võivad toimuda paralleelselt või mitte. Käesolev lahendus lähtub pigem koodimise lihtsusest kui optimeeritud lõpptulemusest.

4.2 Põhjendused erinevatele otsustele

Tööd alustades ja selle käigus langetati mitmeid tehnilisi ja disainialaseid otsuseid, mis ei pruugi ilma täpse kontekstita ilmselged olla. Järgnevad punktid aitavad mõista nende valikute tagamaid.

4.2.1 PodBean'i ja RSS-i kattuvus

RiigiPodcast pakub helisalvestiste kuulamise võimalust kahel väga sarnasel rindel – taskuhäälingu hostimissaidil ning läbi RSS-voo. Kummalgi on aga oma eelised.

PodBean'i on kergem levitada ja jagada, sest see ei eelda, et kuulaja teab mis RSS on ja kuidas seda sisukoondajatega ühendada. Voo jagamisel ei pea aga inimesed, kes juba tarbivad oma huvisid sisukoondajate kaudu uusi meediaallikaid kasutama hakkama. Lisaks ei katke teenuse pakkumine, kui mõnel põhjusel peaks PodBean'i toetamine lõppema.

4.2.2 Failitöötluse parameetrid

Töödeldes videovoogusid helifailideks kasutatakse erinevaid võtteid, et tagada lõpptulemusele nõuetes nimetatud omadused.

Bitikiiruseks määratakse 96 kbit/s, et vähendada failisuurust. Sellel tasemel kõlab inimkõne veel dünaamiliselt ja orgaaniliselt [28].

Silenceremove helifiltrile antav *stop_threshold* väärtus ei tohi olla liiga suur ega väike. Vastasel juhul lõigatakse välja lisaks vaikusele ka kellegi vaiksem kõne või jäetakse faili pikkus üldse muutmata. -35dB väärtus leiti helitöötlus rakendusse Audacity laetud istungeid ja pressikonverentse analüüsides. Selline helitase saavutati siis, kui ainukene heliallikas oli üleüldine müra.

Meta-andmete muutmisel määratakse helifaili kodekiks *copy* [29]. See annab ffmpeg-ile mõista, et faili ei pea dekodeerima ja kodeerima. *Copy* miinuseks on, et käsus ei saa kasutada helifiltreid [29]. Seepärast ei kasutata seda lähenemist ka videovoo töötlemisel helifailiks.

4.2.3 Andmete talletamine

Andmete salvestamine, pärimine ja muutmine toimub operatsioonisüsteemi failisüsteemis. Neist andmetest pealkiri ning taskuhäälingu episoodi ID-kood jäädvustatakse MP3-e ID3-nimelisse meta-andmete konteinerisse. Kuigi pealkirja saab hoida *title* nimelisel ettenähtud märgisel, pole ID-koodi jaoks vastavat meta-andmete märgist olema [30] s. ID-kood salvestatakse *TIT3* märgisele, mis pole aga intuitiivne ning valmistab raskusi süsteemi mõistmisel.

Esialgne disain nägi ette ka helifaili kirjelduse salvestamist ID3-e. Praktikas ilmnes aga, et meta-andmete lugemiseks kasutatud fluent-ffmpeg teegi *ffprobe* funktsioon tagastab sõnesid, mille lugemine on lõpetatud pärast esimest reavahetust. Video kirjeldused on tihti mitmerealised, seepärast otsustati need talletada eraldi tekstifailidena.

Alternatiivina failisüsteemile kaaluti andmebaaside kasutamist, et tagada andmete stabiilsus läbi ACID põhimõtte ning hoida andmete struktuur kergemini loetav. Andmebaasisüsteemi tarvitamine oleks aga suurendanud üleliigselt süsteemi keerukust ning tekitanud juurde ajalist kulu. Failisüsteemi kasutamiseks vajaliku NodeJS'i *fs* mooduliga oli arendajal varasem kogemus ning uute tööriistade kasutamist polnud seega juurde vaja õppida.

4.2.4 Süsteemi piirangud

Teenus näeb ette, et korraga tegeletakse vaid ühe videovoo töötlemisega. Süsteemi sisenevad YouTube'i videovood võivad olla mitmete tundide pikkused. Mitut sellist voogu paralleelselt töödelda võib olla süsteemile koormav.

Kaaluti varianti, kus mitme töödeldava voo puhul ressursside hulka piiratakse, et ära hoida süsteemi kokkujooksmist. Kuna RiigiPodcast pole ajaintensiivne teenus ning Hub'ile on sisse ehitatud korduskutsete süsteem, on lihtsam variant sissetulev teatis tagasi lükata ning oodata hiljem tema naasmist.

Teine riistvaraline piirang lasub süsteemi piiratud massmälul. Teenus ei suuda salvestada lõpmatut hulka andmeid ning helifailide arvu piiranguks on määratud 20. Kuna helisalvestised kätkevad endas päevakajalisi teemasid, pole see suureks miinuseks, et vanimad nende seast aja jooksul eemaldatakse.

4.2.5 Verifikatsiooni puudumine

Teste süsteemile ei lisatud, sest enamus teenuse funktsionaalsusest toimub erinevate süsteemide koostööl. Sellisel juhul oleks tulnud teenuse verifitseerimiseks kirjutada integratsiooniteste või ühikteste, mis kasutavad teisi süsteeme matkivaid makette. Selliste testide kirjutamine hinnati aga mahult liiga ajakulukaks ettevõtmiseks.

4.3 Mida tulevikus teha

RiigiPodcast täidab oma eesmärki, ent tal esineb murekohti ja laiendamisvõimalusi millele tulevikus pilk heita. All toodud punktid aitavad teenusel tõsta kasutajate heaolu ning vähendada võimalikke süsteemirikkeid.

4.3.1 Süsteemi oleku hindamine

Parajasti toetuvad teenuse protsessid eeldusele, et tehnilisi rikkeid töö ajal ei toimu. Kui pärast helifaili loomist programm kokku peaks jooksma, siis ilma füüsilise sekkumiseta antud faili taskuhäälingu saidile üles ei laeta. Sellise probleemi lahendamiseks tuleks kirjutada funktsioon, mis igal RiigiPodcast'i käivitamisel võrdleb kõiki kohalikke faili kõigi taskuhäälingu episoodidega ning laeb puuduvad helifailid üles.

4.3.2 Failide edasi kerimine

Käesolev RiigiPodcast'i versioon lubab faile kuulates sisu edasi kerida vaid nii palju, kui palju materjali on alla laetud. See tähendab, et klikkides kursoriga tunniajase helifaili edenemisriba keskele ei hakka fail mängima poole tunni pealt, vaid nihkub vaid veidi edasi algsest asukohast. Mugavaks edasikerimiseks tuleb implementeerida failide voogesitamine kasutades olekukoodi 206 ning päise välja *Content-Range* [31].

4.3.3 Täiendatud ffmpeg käsk

Tühjad vaikused videodes pole ainsad lõigud, mis kuulajate aega raiskavad. Riigikogu saalikutse on monotoonselt korduv helisignaal, mis võib kõlada kümneid sekundeid [4]. Ffmpegi käsku võib tulevikus täiendada, et eemaldatud ei oleks ainult liigne vaikus, vaid ka helilõigud, milles kordub etteantud muster.

4.4 Alternatiivtoode

Teine valik RiigiPocast'i asemel on YouTube Vanced, mis lubab kuulata YouTube'i videoid lukustatud ekraaniga. Kui suletud ekraan aga välja arvata, ei tegele Vanced'il ülejäänud probleemidega, nagu vaikuse eemaldamine, helivaljuse suurendamine, faili suuruse vähendamine.

Lisaks peab YouTube Vanced'i kasutamiseks alla laadima eraldi aplikatsiooni. Kuna Google Play rakenduste jaotusteenus seda ei paku, tuleb Android telefonide puhul kasutajal eraldi nõustuda tundmatute failide allalaadimisega [32].

5 Kokkuvõte

Töö sihiks oli luua teenus RiigiPodcast, mis levitaks internetis helifailidena Eesti Vabariiki puudutavaid ametlikke videoid. Teenus kirjutati NodeJS raamistikus ES6 JavaScripti standardis.

Programmi toimimiseks on vajalik koostöö mitme teise teenusega. Huvipakkuvad institutsioonid laevad oma videod üles YouTube'i, mis saadab teavitusteenus Pubsubhubbub Hub'i kaudu Riigipodcast'ile teate uuest materjalist. Kirjutatud teenus eraldab seejärel saadud videovoost heli, lõikab välja vaiksed osad, suurendab helitugevust ning vähendab faili suurust. Saadud tulemus laetakse üles PodBean'i-nimelisele taskuringhäälingusaidile. Failide muutumisel YouTube'is muutuvad ka RiigiPodcast'i talletatud kohalik helifail ning PodBean'i taskuhäälingu episood.

Teenuse loodud helifaile saab kasutada RSS sisukoondajate abil, lisades allikate hulka URL-i riigipodcast.ee/feed. RiigiPodcast loob ja tagastab RSS-voo 20st värskeimast helifailist.

Kasutatud kirjandus

[1] „Pressikonverentsid,“ [Võrgumaterjal]. Available: https://www.valitsus.ee/et/uudised-istungite-info/pressikonverentsid. [Kasutatud 8. detsember 2019].

[2] „Videod,“ [Võrgumaterjal]. Available: https://www.riigikogu.ee/infoallikad/multimeedia/videod/. [Kasutatud 8. detsember 2019].

[3] „Using YouTube Premium benefits,“ YouTube, [Võrgumaterjal]. Available: https://support.google.com/youtube/answer/6308116#background_play. [Kasutatud 4. jaanuar 2020].

[4] Riigikogu, „Riigikogu istung, 16. detsember 2019,“ 16. detsember 2019. [Võrgumaterjal]. Available: https://www.youtube.com/watch?v=3u66aX0gxyw. [Kasutatud 4. jaanuar 2020].

[5] J. Genestoux, „WebSub,“ 23 jaanuar 2018. [Võrgumaterjal]. Available: https://www.w3.org/TR/websub/. [Kasutatud 15. detsember 2019].

[6] J. Lindsay, „Web hooks to revolutionize the web,“ 3. mai 2017. [Võrgumaterjal]. Available: https://web.archive.org/web/20180630220036/http://progrium.com/blog/2007/05/03/web-hooks-to-revolutionize-the-web/. [Kasutatud 15. detsember 2019].

[7] J. Genestoux, „Signing-content,“ 23 jaanuar 2018. [Võrgumaterjal]. Available: https://www.w3.org/TR/websub/#signing-content. [Kasutatud 15. detsember 2019].

[8] „https://pubsubhubbub.appspot.com,“ Google, 2017. [Võrgumaterjal]. Available: https://pubsubhubbub.appspot.com. [Kasutatud 5. jaanuar 2020].

[9] ffmpeg, „FFmpeg,“ FFmpeg team, [Võrgumaterjal]. Available: https://ffmpeg.org/. [Kasutatud 10. oktoober 2019].

[10] „Visual Studio Code,“ Microsoft, [Võrgumaterjal]. Available: https://code.visualstudio.com/. [Kasutatud 4. detsember 2020].

[11] N. Joyard ja V. Stefan, „node-fluent-ffmpeg,“ [Võrgumaterjal]. Available: https://github.com/fluent-ffmpeg/node-fluent-ffmpeg. [Kasutatud 4. jaanuar 2020].

[12] D. Greene, „node-rss,“ [Võrgumaterjal]. Available: https://github.com/dylang/node-rss. [Kasutatud 7 oktoober 2019].

[13] D. West, „simple-node-logger,“ [Võrgumaterjal]. Available: https://github.com/darrylwest/simple-node-logger. [Kasutatud 8. detsember 2019].

[14] visionmedia, „superagent,“ [Võrgumaterjal]. Available: https://github.com/visionmedia/superagent. [Kasutatud 12. september 2019].

[15] M. Kubica, „node-xml2js,“ [Võrgumaterjal]. Available: https://github.com/Leonidas-from-XIV/node-xml2js. [Kasutatud 31. detsember 2019].

[16] fent, „node-ytdl-core,“ [Võrgumaterjal]. Available: https://github.com/fent/node-ytdl-core. [Kasutatud 7. oktoober 2019].

[17] B. Fitzpatrick ja B. Slatkin, „Authenticated Content Distribution,“ Google, [Võrgumaterjal]. Available: https://pubsubhubbub.github.io/PubSubHubbub/pubsubhubbub-core-0.4.html#authednotify. [Kasutatud 2. jaanuar 2020].

[18] B. Terlson, B. Farias ja J. Harband, „ECMAScript® 2019Language Specification,“ 06 2019. [Võrgumaterjal]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf. [Kasutatud 4. jaanuar 2020].

[19] „Subscribe to Push Notifications,“ YouTube, [Võrgumaterjal]. Available: https://developers.google.com/youtube/v3/guides/push_notifications. [Kasutatud 8. oktoober 2019].

[20] R. Cadenhead ja G. Smith, „The application/rss+xml Media Type,“ 22. mai 2006. [Võrgumaterjal]. Available: http://www.rssboard.org/rss-mime-type-application.txt. [Kasutatud 4. jaanuar 2020].

[21] M. K. Ned Freed, „Media Types,“ [Võrgumaterjal]. Available: https://www.iana.org/assignments/media-types/audio/mpeg. [Kasutatud 05 01 2020].

[22] R. Fielding, „HTTP Content-Length,“ IETF Trust, 06 2014. [Võrgumaterjal]. Available: https://tools.ietf.org/html/rfc7230#section-3.3.2. [Kasutatud 5. jaanuar 2020].

[23] „404 Not Found,“ Mozilla, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404. [Kasutatud 4. jaanuar 2020].

[24] „Silenceremove,“ FFmpeg team, [Võrgumaterjal]. Available: https://ffmpeg.org/ffmpeg-filters.html#silenceremove. [Kasutatud 16. november 2019].

[25] „Dynaudnorm,“ FFmpeg team, [Võrgumaterjal]. Available: https://ffmpeg.org/ffmpeg-filters.html#dynaudnorm. [Kasutatud 16. november 2019].

[26] „Podbean API,“ PodBean, [Võrgumaterjal]. Available: http://developers.podbean.com/podbean-api-docs/. [Kasutatud 4. jaanuar 2020].

[27] B. Fitzpatrick ja B. Slatkin, „Content Distribution,“ Google, [Võrgumaterjal]. Available: https://pubsubhubbub.github.io/PubSubHubbub/pubsubhubbub-core-0.4.html#contentdistribution. [Kasutatud 2. jaanuar 2020].

[28] „Bit rate,“ Wikipedia, The Free Encyclopedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Bit_rate. [Kasutatud 4. jaanuar 2020].

[29] „StreamCopy,“ FFmpeg team, [Võrgumaterjal]. Available: https://ffmpeg.org/ffmpeg.html#Stream-copy. [Kasutatud 16. november 2019].

[30] ID3Tags, *https://web.archive.org/web/20120620142716/http://www.unixgods.org/~tilo/ID3/docs/ID3_comparison2.html.*

[31] „206 Partial Content,“ Mozilla, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/206. [Kasutatud 5. jaanuar 2020].

[32] „User opt-in for installing unknown apps,“ Google, [Võrgumaterjal]. Available: https://developer.android.com/distribute/marketing-tools/alternative-distribution#unknown-sources. [Kasutatud 5. jaanuar 2020].