



УНИВЕРЗИТЕТ У НОВОМ САДУ ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД

Департман за рачунарство и аутоматику

Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: Катарина Петровић

Број индекса: РА 231/2019

Тема рада: Имплементација репродукције декодованог аудио садржаја
у оквиру интернет прегледача

Ментор рада: др Илија Башичевић

Нови Сад, септембар 2023.



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР :	
Идентификациони број, ИБР :	
Тип документације, ТД :	Монографска документација
Тип записа, ТЗ :	Текстуални штампани материјал
Врста рада, ВР :	Завршни (Bachelor) рад
Аутор, АУ :	Катарина Петровић
Ментор, МН :	Др Илија Башичевић
Наслов рада, НР :	Имплементација репродукције декодованог аудио садржаја у оквиру интернет прегледача
Језик публикације, ЈП :	Српски / ћирилица
Језик извода, ЈИ :	Српски
Земља публикавања, ЗП :	Република Србија
Уже географско подручје, УГП :	Војводина
Година, ГО :	2023.
Издавач, ИЗ :	Ауторски репринт
Место и адреса, МА :	Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, ФО : (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/34/0/1/37/0/0
Научна област, НО :	Електротехника и рачунарство
Научна дисциплина, НД :	Рачунарска техника
Предметна одредница/Кључне речи, ПО :	Хромијум, прегледач, Веб звук технологија, плејер, фреквенција
УДК	
Чува се, ЧУ :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН :	
Извод, ИЗ :	У оквиру задатка потребно је прилагодити декодовани аудио сигнал за обраду и имплементирати додаток за модул, ослањајући се на Хромијум прегледач и већ постојећи Web Audio API. Додатно је потребно направити тестну страницу, која ће покрити све потребне функционалности које Web Audio API пружа. На тај начин ће сваки аудио садржај доћи до свог одредишта, репродуковати се и бити детектован од стране корисника.
Датум прихватања теме, ДП :	
Датум одбране, ДО :	
Чланови комисије, КО :	Председник:
	Члан:
	Члан, ментор:
	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	Monographic publication
Type of record, TR :	Textual printed material
Contents code, CC :	Bachelor Thesis
Author, AU :	Katarina Petrovic
Mentor, MN :	PhD. Ilija Basicovic
Title, TI :	Implementation of decoded audio reproduction content within the Internet browser
Language of text, LT :	Serbian
Language of abstract, LA :	Serbian
Country of publication, CP :	Republic of Serbia
Locality of publication, LP :	Vojvodina
Publication year, PY :	2023.
Publisher, PB :	Author's reprint
Publication place, PP :	Novi Sad, Dositeja Obradovica sq. 6
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/34/0/1/37/0/0
Scientific field, SF :	Electrical Engineering
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, S/KW :	Chromium, Browser, Web Audio, player, sample rate
UC	
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, N :	
Abstract, AB :	The task involves preparing the decoded audio signal for processing and implementing a plugin for the module, relying on the Chromium browser and the already existing Web Audio API. In addition, it is necessary to create a test page, which will cover all the necessary functionalities that the Web Audio API provides. In this way, each audio content will reach its destination, be played and be detected by the user.
Accepted by the Scientific Board on, ASB :	
Defended on, DE :	
Defended Board, DB :	President:
	Member:
	Member, Mentor:
	Mentor's sign

Захвалност

Захваљујем се институту РТ-РК на пруженој шанси и могућности за реализацију овог пројекта.

Посебно желим да се захвалим техничком ментору Дарку Лулићу на посвећености, подршци и помоћи током израде овог завршног рада.

Желим да се захвалим и академском ментору професору др Илији Башичевићу на вођењу и стручним саветима током израде рада.

САДРЖАЈ

1. Увод	1
2. Теоријске основе	2
2.1 Web Audio API	2
2.2 Chromium	5
3. Концепт решења	8
3.1 Teatro TV Browser	8
3.2 MM-LIB	9
3.3 Додатак за модул плејера	9
3.4 HTML/JavaScript апликација	10
4. Програмско решење	11
4.1 Спрега између прегледача и плејера	11
4.2 Интернет страница која имплементира Web Audio технологију	26
5. Резултати	29
6. Закључак	33
7. Литература	34

СПИСАК СЛИКА

Слика 1	График почетног стања аудио чворова	2
Слика 2	Приказ структуре stereo аудио бафера	3
Слика 3	Приказ преплетеног типа складиштења података	4
Слика 4	Приказ архитектуре прегледача [6]	6
Слика 5	Графички приказ комуникације плејера и прегледача	8
Слика 6	Структура за чување аудио података	11
Слика 7	Функција за иницијализацију	12
Слика 8	Функција <code>on_state_changed</code> у оквиру <code>Wa_callback_listener</code>	13
Слика 9	Остале функције у оквиру <code>Wa_callback_listener</code>	13
Слика 10	Функција <code>wa_dispatch_callback</code> која позива функцију повратног позива са <code>ENDED</code> статусом	14
Слика 11	Функција за регистровање догађаја, коју позива прегледач	14
Слика 12	Иницијализација параметара структуре унутар <code>waudio_initialize</code>	14
Слика 13	Функција <code>waudio_deinitilize</code> , супротна од <code>waudio_initilize</code>	15
Слика 14	Функција <code>WebAudio_Open</code> из <code>medialib.c</code> датотеке	16
Слика 15	Функција <code>WebAudio_Close</code> из <code>medialib.c</code> датотеке	16
Слика 16	Функција <code>WebAudio_Start</code> из <code>medialib.c</code> датотеке	17
Слика 17	Један део функције <code>waudio_start</code> , који се односи на припрему плејера	17
Слика 18	Неке од основних функција у оквиру <code>Web_audio_source</code>	18
Слика 19	Приказ <code>enable</code> и <code>disable</code> функција унутар екстрактора	19
Слика 20	Функције везане за саму аудио траку	19
Слика 21	Приказ функције <code>pass_param</code> унутар <code>Web_audio_source</code>	20
Слика 22	Садржај датотеке <code>Web_audio_source_factory.cpp</code>	20
Слика 23	Приказ иницијализације помоћних променљивих и рачунања вредности <code>num_samples_to_skip_on_start</code> и <code>num_samples_to_skip_on_end</code>	21
Слика 24	Изглед <code>buffer-a</code> за податке	21
Слика 25	Изглед <code>do-while</code> петље	22
Слика 26	Функција која прави временску контролу	23

Слика 27	Изглед функција <code>timerHandler</code> и <code>deleteTimer</code>	24
Слика 28	Изглед функције <code>timer_set</code>	24
Слика 29	Функција <code>set_track_meta</code> за постављање параметара медија формата	25
Слика 30	Део <code>waudio_start</code> функције у оквиру <code>mm-lib</code> спреге	25
Слика 31	Функција <code>waudio_stop</code> у оквиру <code>mm-lib</code> спреге	26
Слика 32	Приказ функционалности дугмића	26
Слика 33	Изглед функције <code>playSound</code> када <code>stop()</code> метода нема параметара	27
Слика 34	Изглед функције <code>stopSound</code> и <code>pause</code>	28
Слика 35	Изглед тестне странице	29
Слика 36	Изглед аудио сигнала у Audacity окружењу	30
Слика 37	Аудио сигнал који свира аудио од 2. до 4. секунде у континуитету	30

СПИСАК ТАБЕЛА

Табела 1	Резултати верификације	32
----------	------------------------------	----

СКРАЋЕНИЦЕ

MM-LIB - Media management library, организован скуп свих медија датотека

HTML - Hyper Text Markup Language, језик намењен опису веб страница

API - Application programming interface, програмска спрега

URI - Uniform resource identifier, идентификатор ресурса

1. Увод

Задатак је имплементирати репродукцију декодованог аудио сигнала у оквиру плејер модула. Овај задатак је сачињен од више целина, које је потребно испунити.

Прва целина се односи на само прилагођавање и обраду декодованог аудио сигнала према захтевима плејер модула за репродукцију звука на основу доступних Хромијум алата и модула.

Друга целина се односи на креирање додатака за модул који ће имплементирати команде и параметре потребне за аудио репродукцију.

Трећи део је припрема HTML/JavaScript апликације која ће обухватити све команде предвиђене Web Audio стандардом. Уз помоћ ње ће се вршити испитивања за различите сценарије и различите аудио сигнале, што је заправо и последња целина овог задатка.

Рад је реализован коришћењем следећих програмских језика: C, C++, HTML/JavaScript. У једном делу задатка прелази се на Mediatek AOW развојну плочу.

Први нацрт за Web Audio API се појавио током 2011. године. Иако је аудио на веб страницама подржан већ дуго времена, исправан начин генерисања звука из веб прегледача није био доступан све до недавно. Веб прегледачи су данас још једно средство за уметничко изражавање, а видео и аудио у веб прегледачу играју велику улогу у томе. Иако је и даље у развоју, постоји велики број библиотека које олакшавају његово коришћење. Постао је поуздан и широко применљив јер елиминише потребу за инсталирањем додатака и преузимањем засебних апликација.

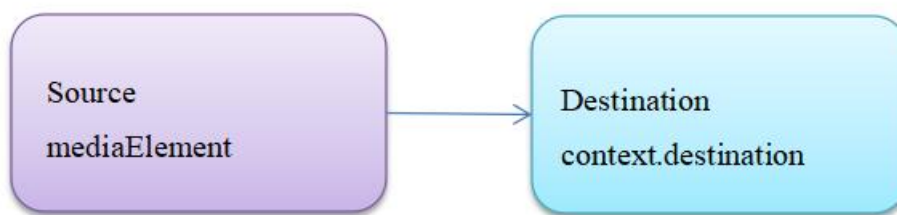
2. Теоријске основе

2.1 Web Audio API

Web Audio API пружа моћан и свестран систем за контролу звука на вебу, омогућавајући програмерима да бирају изворе звука, додају различите ефекте звуку, креирају аудио визуализације, примењују просторне ефекте и још много тога.

Укључује руковање операцијама унутар аудио контекста и дизајниран је да омогући модуларно рутирање. Основне операције изводе се над такозваним аудио чворовима, који су повезани заједно тако да формирају једну целину. Неколико извора, са различитим типовима распореда канала, је подржано унутар једног контекста. Овакав модуларни дизајн пружа флексибилност за креирање сложених аудио функција са динамичким ефектима.

Једноставан ток имплементације за Web Audio би се састојао од неколико кључних целина као што су креирање аудио контекста, креирање самог извора, ефектних чворова и на крају одабир одредишта звука. О самој имплементацији биће речи касније.



Слика 1 График почетног стања аудио чворова

График изнад ће се мењати у складу са потребним функционалностима и чворовима које се додатно уводе.

Web Audio нуди велики број интерфејса који су доступни за коришћење. Неки од основних су: `AudioBuffer`, `AudioContext`, `BaseAudioContext`, `AudioNode`, `GainNode`...

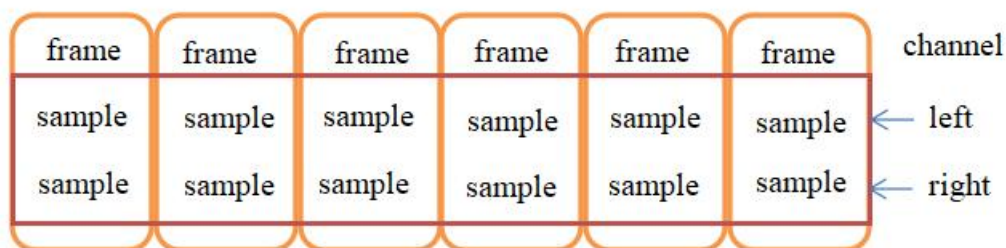
`AudioBuffer` је дефинисан са следећим параметрима:

- учестаност одабирања сигнала (енг. *sample rate*)
- број канала
- дужина

Учестаност одабирања сигнала је број узорака (енг. *sample*) који ће се репродуковати у једној секунди, мерено у Hz (херц). Што је већа учестаност, бољи је квалитет звука.

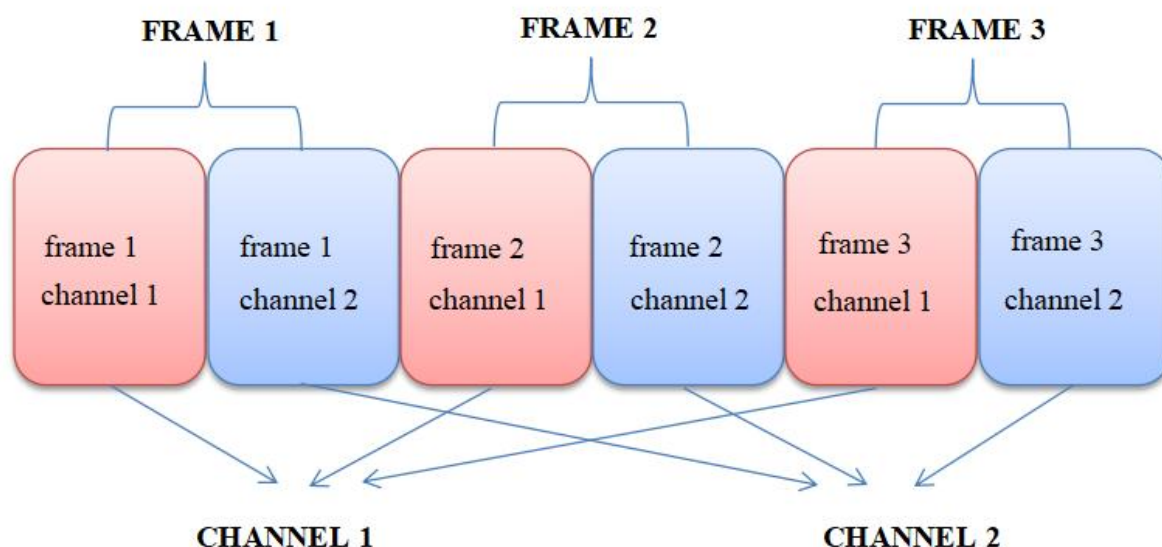
Када говоримо о броју канала, најчешћа је појава *mono* и *stereo* аудио бафера.

Уколико претпоставимо да је учестаност 44100 Hz, *mono* бафер ће имати 44100 узорака и исто толико оквира. Ако имамо *stereo* бафер, он ће имати 88200 узорака али и даље 44100 оквира.



Слика 2 Приказ структуре stereo аудио бафера

Web Audio користи планарни формат бафера (енг. *planar*). Преплетен формат је алтернативан избор и он преовлађује код репродукције звука без много обраде, на пример *.wav* датотека или декодовани *mp3* аудио.



Слика 3 Приказ преплетеног типа складиштења података

Када се подеси контекст и чворови, може се започети са JavaScript кодовањем за манипулацију аудио сигналом у реалном времену.

Web Audio има значајну улогу у свакодневном животу на интернету на следеће начине:

- Репродукција звука на веб страницама: Web Audio API омогућава веб страницама репродукцију звука као што је музика, звучни ефекти или говорна порука. Ово је веома корисно за музичке странице, платформе за репродукцију музичког тока, игре или апликације које захтевају аудио садржај.
- Видео репродукција: Web Audio API се најчешће користи заједно са WebRTC и HTML5 видео елементом како би се омогућила синхронизована репродукција звука са видео садржајем на веб страницама. То је кључно за гледање филмова, видео записа и других видео материјала на мрежи.
- Аудио ефекти и обрада звука: Web Audio API пружа велики спектар могућности за манипулацију и обраду звука. Може се користити за примену аудио ефеката попут еквилајзера, кашњења или промене брзине репродукције звука. Ово је веома корисно за аудио продукцију, музичке ефекте, аудио визуалне перформансе и друге креативне апликације.

- Музичка синтеза: Web Audio API омогућава програмирање звука у реалном времену, што је посебно корисно за музичке апликације и игре. Омогућава синтезу звука користећи различите осцилаторе, филтере и модулације.
- Аудио визуализација: Web Audio API се често користи за визуализацију звука на веб страницама. Може се користити за стварање спектралних анализатора, осцилограма, анимација у складу са музиком и других визуелних ефеката који се синхронизују са звучним садржајем.

Све ове могућности Web Audio API-ја отварају врата за креативност, интерактивност и богатије корисничко искуство на вебу, чинећи звук делом свакодневног живота на интернету.

2.2 Chromium

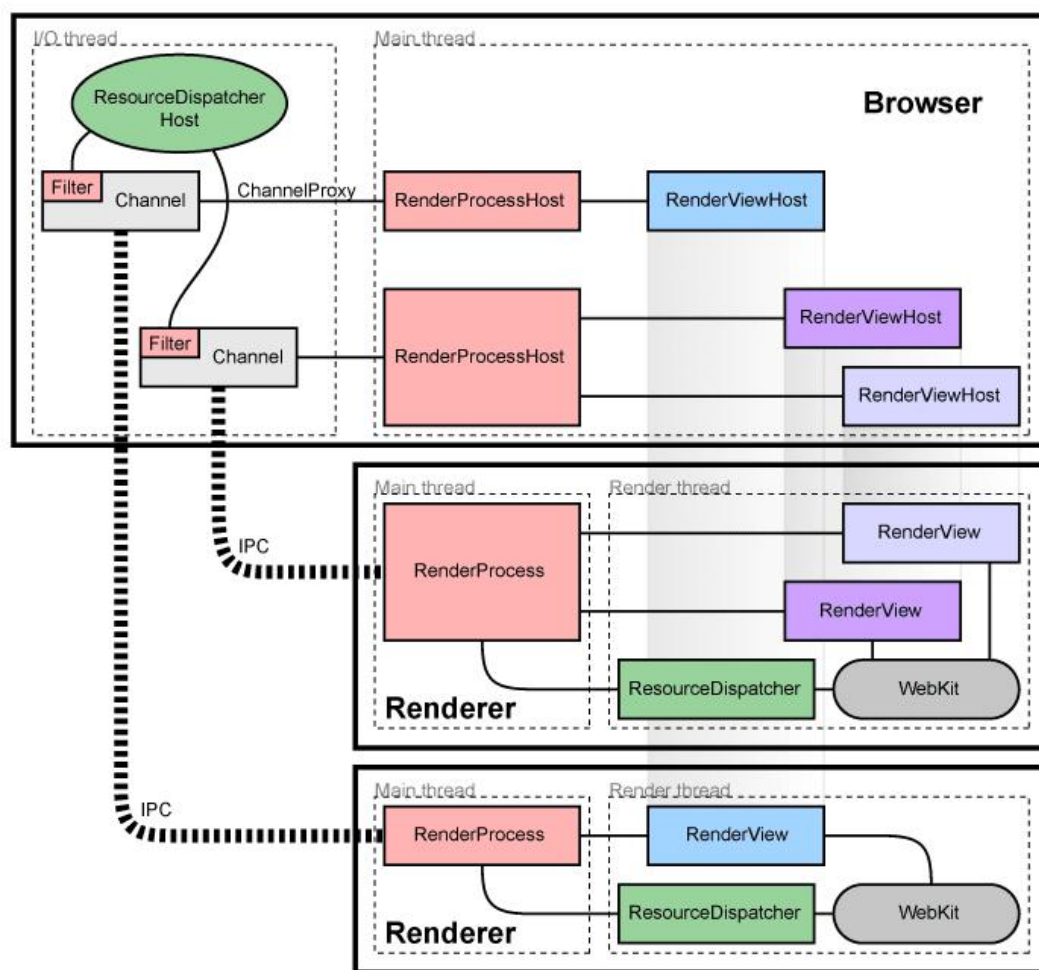
Chromium веб прегледач је пројекат отвореног кода (енг. *open source project*) који подржава *Google Chrome* прегледач. Будући да је пројекат отвореног кода, *Google* као и многи други могу да користе *Chromium* изворни код, што је и главни разлог зашто је толико популаран. Свако може да га преузме бесплатно, компајлира и мења у складу са сопственим потребама.

Једна од најчешће познатих чињеница о прегледачима заснованим на *Chromium*-у је да они имају много процеса, што је, пре свега, из безбедносних разлога. Ако покренемо цео код у истом процесу, успешан напад у оквиру једног дела може довести до извршавања кода у целом прегледачу. Теже је изоловати нити, тако да прегледач користи функције оперативног система и изолује процесе. У *Chromium*-у, модул за визуелизацију се не покреће у процесу главног прегледача. Различити сајтови ће радити у различитим модулима за визуелизацију који имају различите процесе. Такође, различите врсте процеса имају различите привилегије. Неки процеси су заштићени. Ово своди број успешних напада на минимум.

Главни процес који покреће кориснички интерфејс и управља модулом за визуелизацију и другим процесима је *browser process* или *browser*. Слично томе, процеси који рукују веб садржајем називају се *renderer processes* или *renderers*. Модули за визуелизацију користе *Blink* механизам отвореног кода за тумачење и постављање *HTML*-а. Имплементиран је на врху апстрактне платформе и стога се не може сам покренути.

Browser процес је главни процес одговоран за покретање Chromium веб прегледача, који служи као основа отвореног кода за многе прегледаче. Он управља различитим задацима, укључујући приказивање веб страница, извршавање JavaScript кода, руковање корисничким уносом и управљање екстензијама прегледача.

Када се покрене Chromium прегледач, он покреће један главни процес. Овај процес делује као родитељски процес и надгледа креирање и управљање другим подређеним процесима.



Слика 4 Приказ архитектуре прегледача [6]

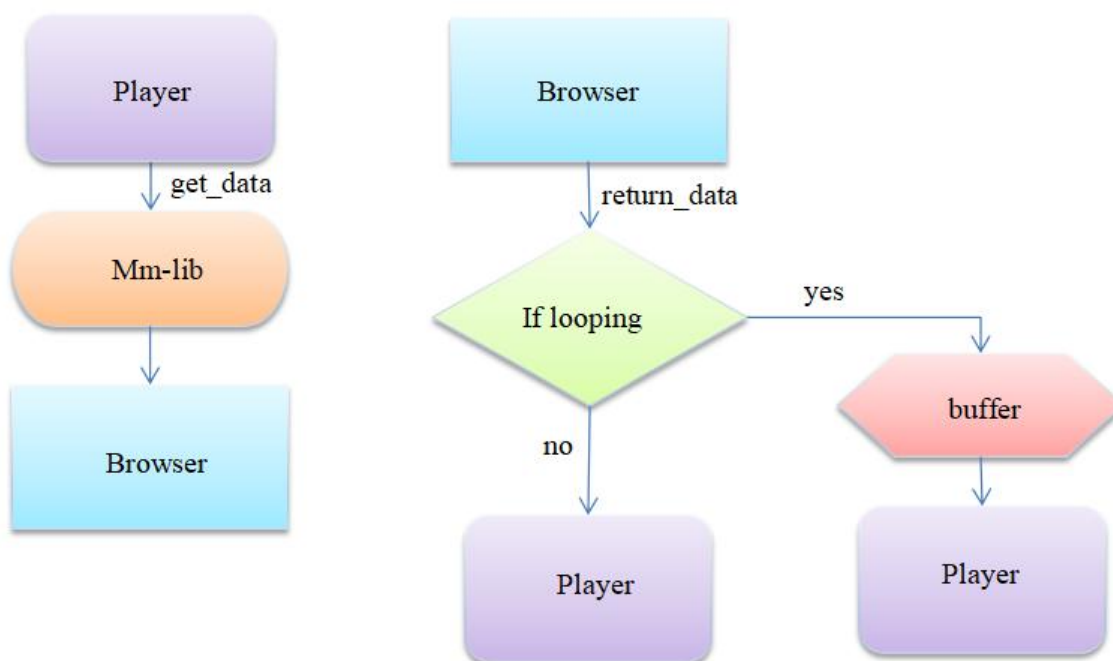
Након покретања browser процеса, отварање картица у Chromium прегледачу укључује следеће кораке:

- Корисник отвара нову картицу или кликне на постојећу картицу да је активира.
- Главни процес Chromium прегледача прима захтев за отварање нове картице.
- Главни процес креира renderer процес или проналази постојећи слободан који ће бити задужен за ту картицу.
- Renderer процес је изолован од главног процеса и других renderer процеса. Сваки renderer процес је одговоран за приказивање садржаја једне картице. Ово омогућава бољу сигурност и стабилност, јер уколико један падне или има проблема, остали процеси и картице ће и даље бити функционални.
- Renderer процес преузима информације о адреси веб странице која треба да се отвори у картици и шаље захтев за преузимање садржаја те странице.
- Главни процес прегледача управља мрежном комуникацијом и шаље захтев renderer процесу за преузимање података са веб сервера. Renderer процес прима одговор од сервера и прослеђује га даље на обраду.
- Renderer процес обрађује примљени одговор и извршава JavaScript код и рендерује HTML и CSS да би се приказала веб страница на екрану.
- Renderer процес шаље резултате свог рада главном процесу како би се приказала веб страница у одговарајућој картици на корисничком интерфејсу.

Овај процес се понавља за сваку нову картицу коју корисник отвори. Сваки renderer процес је одговоран само за једну картицу, чиме се постиже већа изолација и боља сигурност. Главни процес се бави координацијом између renderer процеса, управљањем ресурсима и интеракцијом са корисничким интерфејсом.

3. Концепт решења

Концепт решења се састоји из четири главне компоненте. Прва се односи на сами прегледач *Teatro TV Browser* који је покретач свега. Друга компонента је спрега између прегледача и плејера (*mm-lib*). Трећу компоненту чини додатак (енг. *plugin*) за модул плејера. Четврту *HTML/JavaScript* апликација која ће послужити као извор прегледачу и за испитивање свих функционалности плејера.



Слика 5 Графички приказ комуникације плејера и прегледача

3.1 Teatro TV Browser

Улога Teatro TV Browser-а у овом пројекту је комуникација са плејером преко спреге *mm-lib*-а. У оквиру њега се врши декодовање аудио сигнала. Такође, позивају се

одговарајуће функције из mm-lib-a и прослеђују се потребни параметри као што су учестаност одабирања сигнала, број канала, величина одбирака, време од када треба пустити аудио садржај, време након ког треба исти тај аудио стопирати и други параметри, о чему ће бити речи касније.

3.2 MM-LIB

MM-LIB, као што је претходно било речи, означава спрегу између веб прегледача и плејера. У овом делу се добављају све значајне вредности параметара аудио звука, као и функције повратног позива које служе за регистровање догађаја и добављање аудио података. Сви ти подаци су од великог значаја за сам плејер и функционалности које он треба да подржи.

Потребно је повезати овај део са самим плејером као и уградити додатак за модул плејера и осмислити начин тока функционалности и података.

3.3 Допатак за модул плејера

Треба обезбедити:

1. повратне информације за припрему сигнала (учестаност одабирања, број подржаних канала и величину одбирка)
2. припрему плејер модула за репродукцију према захтеваним параметрима
3. достављање аудио сигнала.
4. имплементацију команди за репродукцију(пусти, паузирај, заустави, итд)

Додатак за модул плејера треба да, при позивању play функције, тражи податке од прегледача, који је дужан да му те податке обезбеди. Уколико се захтева понављајућа репродукција аудио садржаја од плејера, подаци који дођу из прегледача ће се сместити у посебан бафер. Након тога ће плејер, сваког следећег пута, податке

узимати из бафера, а не од прегледача. Када се позове функција `stop`, плејер прекида репродукцију у зависности од прослеђеног параметра, који назначава после колико времена ће се то десити. Ове, а и остале функционалности, биће детаљно објашњене у поглављу програмског решења.

3.4 HTML/JavaScript апликација

Последњи корак је испитивање, за које је потребно направити апликацију која игра улогу интернет странице. Она имплементира Web Audio технологију и покрива све токове у репродукцији сигнала.

4. Програмско решење

4.1 Спрега између прегледача и плејера

Први корак у имплементацији овог дела задатка је проширити `media-av.h` датотеку са главном структуром коју ћемо користити у току читавог пројекта. Направљена је структура `typedef _web_audio_entry WEB_AUDIO_ENTRY_t` у оквиру које се налазе функције повратног позива, које ће се позивати у оквиру `medialib.c` датотеке. Оне играју улогу посредника између ове две датотеке, будући да прослеђују параметре у `media-av.cpp`, где ће бити имплементирана даља логика задатка. Сем функција повратног позива, ту се налазе параметри које је потребно сачувати у оквиру идентификатора, као што су функције повратног позива за добављање аудио података и регистровање догађаја, `webaudio` параметри, време од почетка читања аудио сигнала, време када ће се слање података прекинути, `start_loop` и `end_loop`, `looping` и многи други параметри. Додате су и функције за иницијализацију и деиницијализацију и `typedef` за функцију показивача `StopPlayerCallback`.

```
typedef struct _web_audio_entry
{
    void* wa_player_handle;
    void* wa_event_listener_handle;
    WebAudioHandle web_audio_handle;
    void (*open)(WebAudio_GetDataCallback callback);
    uint8_t (*close)(WebAudioHandle handle);
    uint8_t (*start)(WebAudioHandle handle, const WebAudioParams wa_params, const double when, const double offset, const double duration);
    uint8_t (*pause)(WebAudioHandle handle);
    uint8_t (*resume)(WebAudioHandle handle);
    uint8_t (*stop)(WebAudioHandle handle, const double when);
    uint8_t (*set_loop_start)(WebAudioHandle handle, const double start);
    uint8_t (*set_loop_end)(WebAudioHandle handle, const double end);
    uint8_t (*loop)(WebAudioHandle handle, const uint8_t looping);

    WebAudio_EventCallback callback;
    WebAudio_GetDataCallback data_callback;
    WebAudioParams wa_params;
    double wa_start;
    double wa_end;
    double start_loop;
    double end_loop;
    int selected_audio_track_index;
    uint8_t looping;
    unsigned char *web_audio_buffer;
    timer_t timerID;
    WAPLAYER_DES_t waplayer_des;
    WASample wa_sample;
} WEB_AUDIO_ENTRY_t;

WEB_AUDIO_ENTRY_t *waudio_initialize();
void waudio_deinitialize(WebAudioHandle handle);
typedef void (*StopPlayerCallback)(WebAudioHandle handle);
```

Слика 6 Структура за чување аудио података

У оквиру функције `WEB_AUDIO_ENTRY_t *waudio_initialize()` потребно је извршити иницијализацију овог модула. Заузима се меморија за `WEB_AUDIO_ENTRY_t *entry`, након чега се додаје у листу `list<WEB_AUDIO_ENTRY_t *>wa_entries_list`. Иницијализују се `JavaVM* java_vm` и `jobject globalContext`, који ће добити вредности позивањем функције `TeatroSurfaceHandler_GetJavaEnviroment`. Тиме је омогућен приступ Java виртуелној машини и глобалном контексту саме апликације. Регистрован је и `surfaceListener` као `layout` слушац за руковање догађајима. Након тога, поставља се Java виртуелна машина у окружење које користи `iwu::jni`. Креира се `web_audio_source_factory` и `renderer_factory` објекат.

Потом је потребно креирати објекат `wa_player` коришћењем `instantiate` методе у оквиру `player::Default_player_factory`. Имплементирана је и регистрација слушаца за плејер који ће регистровати стања плејера.

```
WEB_AUDIO_ENTRY_t *waudio_initialize()
{
    WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t*)malloc(sizeof(WEB_AUDIO_ENTRY_t));
    wa_entries_list.push_back(entry);

    JavaVM* java_vm = NULL;
    jobject globalContext = 0;

    TeatroSurfaceHandler_GetJavaEnviroment(&java_vm, &globalContext);
    TeatroSurfaceHandler_RegisterLayoutListener(surfaceListener);
    iwu::jni::set_vm(java_vm);

    std::shared_ptr<player::android::api::IFactory> api_factory{
        player::android::Media_codec_utils::create_api_factory(
            player::android::Media_codec_utils::Api_type::java)};

    std::shared_ptr<player::android::Codec_factory> codec_factory{
        std::make_shared<player::android::Codec_factory>(api_factory, globalContext, iwu::jni::Reference_type::external)};

    std::shared_ptr<player::Web_audio_source_factory> web_audio_source_factory = std::make_shared<player::Web_audio_source_factory>();
    auto renderer_factory = std::make_shared<player::android::Default_renderer_factory>(globalContext, iwu::jni::Reference_type::external);

    #ifdef player_USE_TUNNEL_MODE
    long session_id = player::android::Media_codec_utils::generate_tunneling_session(globalContext);
    codec_factory->set_tunneling_session(session_id);
    renderer_factory->set_tunneling_session(session_id);
    #endif /*player_USE_TUNNEL_MODE*/

    auto wa_player = player::Default_player_factory{}.instantiate(
        codec_factory,
        renderer_factory,
        web_audio_source_factory,
        std::make_shared<player::Default_track_selector>((std::vector<player::Media_type>{})),
        nullptr).release();

    Wa_callback_listener *callback_listener = new (Wa_callback_listener);
    wa_player->register_listener(callback_listener, (void*)((WebAudioHandle)entry));
}
```

Слика 7 Функција за иницијализацију

```

struct Wa_callback_listener : public player::IPlayer_listener {
    void on_state_changed(const player::IPlayer_listener::State_changed_data& data) override {
        int video_tracks number;
        DBG_INFO("received Player state changed: %s -> %s",
            player::to_string(data.old_state).c_str(),
            player::to_string(data.new_state).c_str());

        WebAudioHandle webaudio_handle = (WebAudioHandle)data.user_data;
        WEB_AUDIO_ENTRY_t *entry = ((WEB_AUDIO_ENTRY_t*)webaudio_handle);

        switch(data.new_state)
        {
            case player::Player_state::prepared:
                video_tracks_number = get_video_tracks_number(webaudio_handle);
                break;

            case player::Player_state::paused:
                if(entry->waplayer_des.state == WA_PAUSED)
                    return;
                set_wa_state_t(WA_PAUSED, webaudio_handle);
                break;

            case player::Player_state::playing:
                if(entry->waplayer_des.state == WA_STARTED)
                    return;
                set_wa_state_t(WA_STARTED, webaudio_handle);
                break;

            case player::Player_state::stopped:
                wa_dispatch_callback(webaudio_handle, ENDED);
                set_wa_state_t(WA_STOPPED, webaudio_handle);
                break;
        }
    }
}

```

Слика 8 Функција on_state_changed у оквиру Wa_callback_listener

```

void on_stall_start(const player::IPlayer_listener::Stall_start_data& data) override {
    DBG_INFO("received on_stall_start");
}

void on_stall_stop(const Callback_data& data) override {
    DBG_INFO("received on_stall_stop");
}

void on_end_of_stream(const Callback_data& data) override {
    DBG_INFO("received on_end_of_stream");
    WebAudioHandle webaudio_handle = (WebAudioHandle)data.user_data;
    wa_dispatch_callback(webaudio_handle, ENDED);
}

```

Слика 9 Остале функције у оквиру Wa_callback_listener

Функција on_end_of_stream позива се када понестане аудио садржаја за репродукцију. У оквиру ње се позива wa_dispatch_callback, који ће прегледачу дати до знања да је плејер завршио са репродукцијом и проследити му одређени догађај.

```
static void wa_dispatch_callback(WebAudioHandle handle, WebAudioEvent event)
{
    DBG_INFO("WA-PLAYER: wa_callback dispatched ENDED handle %llu", handle);
    if(handle != 0)
    {
        if(((WEB_AUDIO_ENTRY_t*)handle)->callback != 0){
            ((WEB_AUDIO_ENTRY_t*)handle)->callback(handle, event);
        }
    }
}
```

Слика 10 Функција wa_dispatch_callback која позива функцију повратног позива са ENDED статусом

Овде се ослањамо на функцију повратног позива која је претходно сачувана на почетку у оквиру структуре.

```
void WebAudio_RegisterEventCallback(WebAudioHandle handle, WebAudio_EventCallback callback)
{
    DBG_INFO("WebAudio_RegisterEventCallback called");
    WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t *)handle;
    entry->callback = callback;
    return;
}
```

Слика 11 Функција за регистровање догађаја, коју позива прегледач

Иницијализовани су и параметри унутар WEB_AUDIO_ENTRY_t структуре:

```
entry->wa_start = 0;
entry->wa_end = 0;
entry->start_loop = 0;
entry->end_loop = 0;
entry->looping = 0;
entry->selected_audio_track_index = -1;
entry->wa_player_handle = static_cast<void*> (wa_player);
entry->wa_event_listener_handle = static_cast<void*> (callback_listener);
entry->open = waudio_open;
entry->close = waudio_close;
entry->start = waudio_start;
entry->pause = waudio_pause;
entry->resume = waudio_resume;
entry->stop = waudio_stop;
entry->set_loop_start = waudio_set_loop_start;
entry->set_loop_end = waudio_set_loop_end;
entry->loop = waudio_loop;
entry->callback = nullptr;
entry->data_callback = nullptr;
```

Слика 12 Иницијализација параметара структуре унутар waudio_initialize

Имплементирана је и функција за `waudio_deinitilize`, у којој се празни листа, ослобађа меморија, брише плејер и параметри структуре се поништавају.

```
void waudio_deinitilize(WebAudioHandle handle)
{
    WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t*)handle;
    if(entry == nullptr)
    {
        return;
    }
    if(entry->wa_player_handle)
    {
        auto player = get_wa_player_from_handle(handle);
        delete(player);
        entry->wa_player_handle = nullptr;
    }
    if(entry->wa_event_listener_handle)
    {
        Wa_callback_listener *callback_listener = static_cast<Wa_callback_listener *> (entry->wa_event_listener_handle);
        delete(callback_listener);
        callback_listener = nullptr;
    }

    entry->wa_start = 0;
    entry->wa_end = 0;
    entry->start_loop = 0;
    entry->end_loop = 0;
    entry->looping = 0;
    entry->selected_audio_track_index = -1;
    entry->open = nullptr;
    entry->close = nullptr;
    entry->start = nullptr;
    entry->pause = nullptr;
    entry->resume = nullptr;
    entry->stop = nullptr;
    entry->set_loop_start = nullptr;
    entry->set_loop_end = nullptr;
    entry->loop = nullptr;
    entry->callback = nullptr;
    entry->data_callback = nullptr;

    wa_entries_list.remove(entry);
    free(entry);
}
```

Слика 13 Функција `waudio_deinitilize`, супротна од `waudio_initilize`

Када је тај део завршен, потребно је имплементирати све остале функције, које ће се позивати из `medialib.c` датотеке и повезати све.

Функције које су смештене у оквиру `medialib.c` датотеке су:

- `WebAudio_GetSupportedSampleType`
- `WebAudio_GetNearestSupportedSampleFrequency`
- `WebAudio_GetMaxChannels`
- `WebAudio_Open`
- `WebAudio_Close`
- `WebAudio_Start`
- `WebAudio_Pause`
- `WebAudio_Resume`
- `WebAudio_Stop`

- WebAudio_SetLoopStart
- WebAudio_SetLoopEnd
- WebAudio_SetLoop
- WebAudio_RegisterEventCallback

У оквиру WebAudio_Open функције потребно је позвати функцију за иницијализацију. Будући да повратна вредност функције за иницијализацију идентификатор entry, сада је омогућено чување параметара унутар ове структуре.

```
WebAudioHandle WebAudio_Open(WebAudio_GetDataCallback callback)
{
    DBG_INFO("WebAudio_Open called");
    WEB_AUDIO_ENTRY_t *entry = NULL;
    entry = waudio_initilize();
    entry->open(callback);
    entry->data_callback = callback;
    entry->web_audio_handle = (WebAudioHandle)entry;
    return (WebAudioHandle) entry;
}
```

Слика 14 Функција WebAudio_Open из medialib.c датотеке

Исто важи и за функцију waudio_deinitilize коју треба позвати у оквиру WebAudio_Close функције.

```
uint8_t WebAudio_Close(WebAudioHandle handle)
{
    if(handle != 0)
    {
        WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t *)handle;
        entry->close(handle);
    }
    waudio_deinitilize(handle);
    return 1;
}
```

Слика 15 Функција WebAudio_Close из medialib.c датотеке

Када се зада команда за репродукцију из прегледача позове WebAudio_Start функција.

```

void WebAudio_Start(WebAudioHandle handle,
                    const WebAudioParams wa_params,
                    const double when,
                    const double offset,
                    const double duration)
{
    if(handle != 0)
    {
        WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t *)handle;
        entry->wa_params = wa_params;
        entry->wa_start = when;
        entry->start(handle, wa_params, when, offset, duration);
    }
    return;
}

```

Слика 16 Функција WebAudio_Start из medialib.c датотеке

У оквиру ње се позива waudio_start функција из media-av.cpp. У овој функцији имплементирано је добављање плејера функцијом get_wa_player_from_handle, да би се касније позвала метода prepare, како би плејер прешао из стања стопиран (енг. stopped) у стање спреман (енг. prepared). Потребно је креирати инстанцу објекта media_descriptor, који припада player::Media_descriptor, и поставити параметар URI (Uniform Resource Identifier). Тек сада се над плејером позива prepare и прослеђује се media_descriptor.

```

WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t *)handle;
set_wa_state_t(WA_STARTED, handle);
auto wa_player = get_wa_player_from_handle(handle);
player::Media_descriptor media_descriptor;
char* val = "web_audio:Yodel_Sound_Effect.mp3";
media_descriptor.uri = val;

if(!val)
{
    DBG_ERROR("Invalid arguments");
    return 0;
}

wa_player->prepare(media_descriptor);

```

Слика 17 Један део функције waudio_start, који се односи на припрему плејера

Следећа етапа у имплементацији је прављење додатка за модул плејера. Он се састоји из три дела:

- Web_audio_source
- Web_audio_source_factory
- Web_audio_extractor

Web_audio_source представља посредника између екстрактора и mm-lib-a. У оквиру њега било је потребно одрадiti дефиницију већ постојећих чистих виртуелних функција и повезати их са екстрактором. Он треба да садржи већину функција као и Web_audio_source. Из функције enable позива се enable функција из екстрактора. Исто важи и за disable.

```
Web_audio_source::Web_audio_source(  
    const std::string& url, std::shared_ptr<IExtractor_factory> extractor_factory)  
    : wa_url{url}, wa_extractor_factory{extractor_factory} {}  
  
void Web_audio_source::open() {  
    DILOGI("TAG WEB AUDIO SOURCE %s ", __FUNCTION__);  
}  
  
void Web_audio_source::close() {  
    DILOGI("TAG WEB AUDIO SOURCE %s ", __FUNCTION__);  
}  
  
void Web_audio_source::enable(IExtractor* extractor) {  
    DILOGI("TAG WEB AUDIO SOURCE %s ", __FUNCTION__);  
    if(wa_audio_extractor)  
        wa_audio_extractor->enable();  
}  
  
void Web_audio_source::disable(IExtractor* extractor) {  
    DILOGI("TAG WEB AUDIO SOURCE %s ", __FUNCTION__);  
    if(wa_audio_extractor)  
        wa_audio_extractor->disable();  
}
```

Слика 18 Неке од основних функција у оквиру Web_audio_source

```

void WebAudioExtractor::enable() {
    DILOGI("TAG WebAudioExtractor %s ", __FUNCTION__);
    wa_disabled = false;
}

void WebAudioExtractor::disable() {
    DILOGI("TAG WebAudioExtractor %s ", __FUNCTION__);
    std::lock_guard<std::mutex> lock{m_mutex};
    if(wa_disabled)
        return;

    wa_disabled = true;
    if(wa_waiting_for_data){
        wa_waiting_for_data = false;
    }
}

```

Слика 19 Приказ enable и disable функција унутар екстрактора

Додате су функције за учитавање аудио траке и добављање информација о истој.

```

void Web_audio_source::load_track(
    gsl::index track_id,
    nonstd::optional<std::chrono::nanoseconds> approximate_position) {
    (void)track_id;
    (void)approximate_position;
}

Web_audio_source::Track_info Web_audio_source::get_track_info(
    gsl::index track_index) const {
    DILOGI("TAG WEB AUDIO SOURCE %s ", __FUNCTION__);
    Expects(track_index >= 0);

    const bool track_open{std::find(wa_open_track_indices.cbegin(),
                                    wa_open_track_indices.cend(), track_index)
                        != wa_open_track_indices.cend()};

    if (wa_audio_extractor) {
        DILOGI("TAG WEB AUDIO SOURCE return audio track meta");
        return Track_info{k_period_id, track_index, {wa_audio_extractor->get_track_meta(0)},
                        track_open};
    }
    return Track_info{k_period_id, 0, {Track_meta{}}}, false; //return empty track
}

std::vector<Web_audio_source::Track_info>
Web_audio_source::get_tracks_info() const {
    DILOGI("TAG WEB AUDIO SOURCE %s ", __FUNCTION__);

    int cumulative_index = 0;

    std::vector<Web_audio_source::Track_info> tracks_info;

    bool open = false;

    if (wa_audio_extractor) {
        for (gsl::index i{0}; i < wa_audio_extractor->get_num_tracks(); i++) {
            tracks_info.push_back(Track_info{k_period_id, cumulative_index + i, {wa_audio_extractor->get_track_meta(i)}, open});
            DILOGI("TAG WEB AUDIO SOURCE get_tracks_info push audio at index %d ", cumulative_index + i);
        }
    }

    return tracks_info;
}

```

Слика 20 Функције везане за саму аудио траку

Да би могло да се настави са имплементацијом, потребно је проследити идентификатор из мм-либ спреге, како би били сигурни да у сваком тренутну радимо над једним истим. У те сврхе је имплементирана функција `pass_param`, чији су параметри функција повратног позива и идентификатор. Она ће позвати исту функцију у екстрактору, где ће се исти ти параметри сачувати у оквиру параметара класе.

```
void Web_audio_source::pass_param(WebAudio_GetDataCallback callback, WebAudioHandle handle) {
    DILOGI("TAG WEB AUDIO SOURCE %s ", __FUNCTION__);
    if(wa_audio_extractor)
        wa_audio_extractor->pass_param(callback, handle);
}
```

Слика 21 Приказ функције `pass_param` унутар `Web_audio_source`

У `Web_audio_source_factory` је дефинисан Web Audio протокол. У случају да постављен URI почиње тим протоколом, направиће се јединствени показивач на објекат типа `Web_audio_source`. У супротном позива статичку методу `instantiate` и враћа објекат типа `Default_media_source_factory`.

```
namespace {
/**
 * @brief File protocol
 */
constexpr auto web_audio_protocol{"web_audio:"};
}

namespace player {

std::unique_ptr<IMedia_source> Web_audio_source_factory::instantiate(const std::string& uri, const std::string& hint) {
    if (iwu::String_utils::starts_with(uri, web_audio_protocol) {
        return std::make_unique<Web_audio_source>{
            uri, nullptr};
    } else { // DEFAULT PROTOCOL*/
        const std::string hint{};
        return Default_media_source_factory::instantiate(uri, hint);
    }
}

} // namespace player
```

Слика 22 Садржај датотеке `Web_audio_source_factory.cpp`

Функција у оквиру које ће се читати подаци и слати плејеру је `read_sample`. Она прима референцу н објекат типа `IBuffer` као параметар.

Направљене су помоћне променљиве и иницијализоване на нулу. Динамички је алоцирана меморија за `wa_data_buffer` са бројем елемената дефинисаним са `DEFAULT_AUDIO_SAMPLE_SIZE`, чија је вредност 1024.

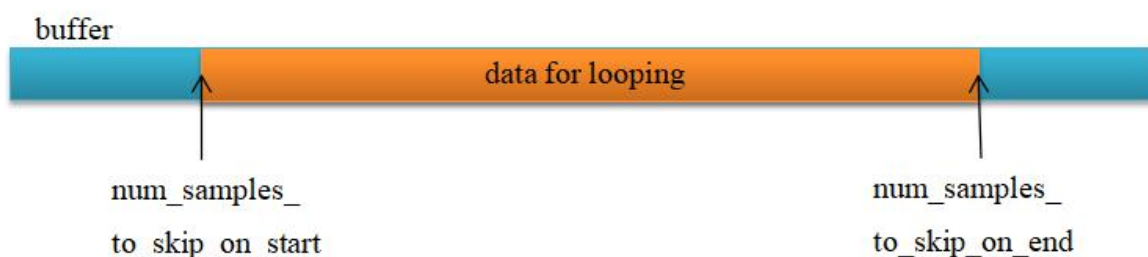
Следећи корак је израчунати број одбирака које треба прескочити у случају да су постављени параметри `loop_start` и `loop_end`. Број одбирака који се неће узети у обзир када се буду слали подаци плејеру рачуна се тако што се помножи фреквенција

узорковања, loop_start (или loop_end у зависности да ли се ради о прескакању одбирака са почетка или са краја), број канала и број бајтова по узорку.

```
Sample meta WebAudioExtractor::read_sample(IBuffer& buffer) {
    DLOGI("TAG KACA WebAudioExtractor: %s disabled %d", __FUNCTION__, wa_disabled);
    WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t *)wa_handle;
    WASample wa_sample = entry->wa_sample;
    uint32_t total_written = 0;
    uint32_t for_write = 0;
    uint8_t start_point = 0;
    uint32_t counter = 0;
    uint32_t total = 0;
    this->set_stop_flag(false);
    /******
    wa_data_buffer = new uint8_t[DEFAULT_AUDIO_SAMPLE_SIZE];
    /******
    entry->web_audio_buffer = (unsigned char*) malloc(1 * sizeof(unsigned char));
    uint8_t num_channels = entry->wa_params.number_of_channels;
    uint32_t num_samples_to_skip_on_start = entry->wa_params.sample_rate * (unsigned int)(entry->start_loop) * num_channels * 2;
    uint32_t num_samples_to_skip_on_end = entry->wa_params.sample_rate * (unsigned int)(entry->end_loop) * num_channels * 2;
```

Слика 23 Приказ иницијализације помоћних променљивих и рачунања вредности num_samples_to_skip_on_start и num_samples_to_skip_on_end

У оквиру петље имплементирано је добављање аудио података уз помоћ get_wa_data функције повратног позива, која је прослеђена преко већ поменуте pass_param функције. Параметар total_written се повећава при сваком учитавању података за 1024 одбирака. У колико је total_written већи или једнак броју одбирака који се прескачу на почетку и мањи или једнак броју одбирака који се прескачу на крају, шаљу се подаци како се и добијају. У колико то није случај, одбацује се одређени број одбирака.



Слика 24 Изглед buffer-а за податке

Само два пута ће бити потребно да се одбаци одређени део података, једном са почетка и једном са краја (када је разлика total_written и тренутно уписаних података већа од num_samples_to_skip_on_start и када total_written пређе број одбирака дефинисан са num_samples_to_skip_on_end).

```

do{
    if (get_wa_data && !wa_disabled) {
        wa_waiting_for_data = true;
        get_wa_data(wa_handle, &wa_data_buffer[0], DEFAULT_AUDIO_SAMPLE_SIZE, &wa_sample.data_size);
        DILOGI(" collecting data...");
        total_written += wa_sample.data_size;
        DILOGI(" total_written %d", total_written);

        if(entry->looping != 0 && total_written >= num_samples_to_skip_on_start && total_written <= num_samples_to_skip_on_end)
            if ((total_written - wa_sample.data_size) >= num_samples_to_skip_on_start){
                for_write = wa_sample.data_size;
                start_point = 0;
            }else {
                for_write = total_written - num_samples_to_skip_on_start;
                start_point = num_samples_to_skip_on_start - (total_written - wa_sample.data_size);
            }
        }

        if(entry->looping != 0 && total_written > num_samples_to_skip_on_end) {
            if((total_written - wa_sample.data_size) < num_samples_to_skip_on_end) {
                for_write = num_samples_to_skip_on_end - (total_written - wa_sample.data_size);
                start_point = 0;
            }else {
                break;
            }
        }

        if(entry->end_loop > 0 && entry->looping == 1) {
            total += for_write;
        }else {
            DILOGI(" not looping...");
            for_write = wa_sample.data_size;
            total += for_write;
        }

        if(total > 0) {
            if(entry->looping != 0){
                entry->web_audio_buffer = (unsigned char*)realloc(entry->web_audio_buffer, total * sizeof(unsigned char));
                for(unsigned int i = 0; i < for_write; i++) {
                    entry->web_audio_buffer[counter] = *(wa_data_buffer + start_point + i);
                    counter++;
                }
            }else{
                DILOGI(" *****sending*****");
                wa_sample_meta.size = wa_sample.data_size;
                if(wa_sample.data_size)
                    buffer.assign({wa_data_buffer, wa_sample_meta.size});
                DILOGI(" *****sent*****");
            }
        }
    }
}

```

Слика 25 Изглед do-while петље

Све док `looping` није постављен, плејер ће податке добијати одмах након њиховог учитавања. То је реализовано `assign` функцијом која, као параметре прима `buffer` и број података које шаље.

Када се `looping` постави, пре него што се плејеру доставе подаци смештају се у бафер. Из њега ће се сваки пут подаци слати плејеру, као што је већ напоменуто. Слање података прекинуће се оног тренутка када се позове функција за заустављање. У колико је њен параметар већи од нуле, стопирање слања дешава се након одређеног броја секунди одређених параметром. Како би се избегло успављивање процеса на одређени временски период, користе се временске контроле. У оквиру екстрактора имплементирана је функција `makeTimer` у којој се позива `timer_settime` који креира временску контролу, након чијег истека се позива `timerHandler`. У `makeTimer` функцији се чува инстанца класе у оквиру глобалне мапе `instance_map`. Као кључ мапе постављено је име инстанце које представља њену адресу заправо. Преко `sival_ptr` у

оквиру `sigevent` структуре прослеђује се име у `timerHandler` који се не налази унутар саме класе екстрактора. На основу имена инстанце додавиће саму инстанцу из мапе, након чега ће бити могуће постављање вредности `stop_flag` на `true`, како би се прекинуло слање података плејеру.

У оквиру `timerHandler` ће се позвати и брисање временске контроле (`deleteTimer`, који захтева идентификациони број временске контроле као параметар), како би били сигурни да се неће поновно покренути. У `deleteTimer` се позива функција `timer_delete` и временска контрола се брише.

```
void WebAudioExtractor::makeTimer(int intervalMS, WebAudioHandle handle)
{
    DILOGI("TAG WebAudioExtractor %s ", __FUNCTION__);
    struct sigevent      te;
    struct itimerspec     its;
    struct sigaction      sa;
    int                  sigNo = SIGRTMIN;
    // Set up signal handler
    sa.sa_flags = SA_SIGINFO;
    sa.sa_sigaction = timerHandler;
    sigemptyset(&sa.sa_mask);
    timer_t tid;
    if (sigaction(sigNo, &sa, NULL) == -1)
        DILOGI("WebAudioExtractor sigaction fail %s ", __FUNCTION__);

    // Set up timer
    te.sigev_notify = SIGEV_SIGNAL;
    te.sigev_signo = sigNo;
    std::string address = std::to_string((unsigned long long)(void*)this);
    te.sigev_value.sival_ptr = (void*)address.c_str();
    const char * name = address.c_str();
    instance_map[name] = this;
    if (timer_create(CLOCK_REALTIME, &te, &tid) == -1)
        DILOGI("WebAudioExtractor timer_create fail %s ", __FUNCTION__);

    its.it_value.tv_sec = intervalMS;
    its.it_value.tv_nsec = 0;
    its.it_interval.tv_sec = 0;
    its.it_interval.tv_nsec = 0;
    this->set_timer_id(tid);
    if (timer_settime(tid, 0, &its, NULL) == -1)
        DILOGI("WebAudioExtractor timer_settime fail %s ", __FUNCTION__);

    return;
}
```

Слика 26 Функција која прави временску контролу

```

void timerHandler(int sig, siginfo_t *si, void *uc)
{
    DBG_INFO("TAG WebAudioExtractor timer stopped!");
    char *name = (char *) si->si_value.sival_ptr;
    WebAudioExtractor *extractor = instance_map[name];
    extractor->set_stop_flag(true);
    StopPlayerCallback callback = extractor->get_callback();
    WebAudioHandle handle = extractor->get_handle();
    timer_t tid = extractor->get_timer_id();
    instance_map.erase(name);
    callback(handle);
    extractor->deleteTimer(tid);
}

void WebAudioExtractor::deleteTimer(timer_t timerID)
{
    if (timer_delete(timerID) == -1)
        DILOGI("WebAudioExtractor timer_delete fail %s ", __FUNCTION__);
    DILOGI("WebAudioExtractor timer_delete called %s ", __FUNCTION__);
}

```

Слика 27 Изглед функција timerHandler и deleteTimer

Направљена је и функција timer_set која се позива унутар истоимене функције у Web_audio_source, позване из media_av спреге. Њен параметар једнак је параметру функције за стопирање слања плејеру. У колико је тај параметар једнак нули, одмах се поставља stop_flag вредност на true. У супротном се позива већ поменути функција makeTimer.

```

void WebAudioExtractor::timer_set(double wa_end) {
    if (wa_end == 0){
        this->set_stop_flag(true);
    }else{
        WebAudioHandle handle = this->get_handle();
        this->makeTimer(wa_end, handle);
    }
}

```

Слика 28 Изглед функције timer_set

Да би све на крају функционисало, одрађена је имплементација функције set_track_meta у којој се подешавају параметри медија формата, као што је медија тип, тип кодека (енг. codec type), трајање, број канала, учестаност одабирања, језик и остало.

```

void WebAudioExtractor::set_track_meta(WebAudioHandle handle) {
    DILOGI("TAG WebAudioExtractor %s ", __FUNCTION__);
    Media_format format;
    WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t *)handle;
    format.set_media_type(Media_type::audio);
    format.set_codec_type(Codec_type::audio_aac);
    format.duration = std::chrono::milliseconds(100000);
    format.audio.channel_count = entry->wa_params.number_of_channels;
    format.audio.sample_rate = entry->wa_params.sample_rate;
    format.language = "eng";
    format.mime = "audio/mp4";
    wa_tracks.push_back(format);
}

```

Слика 29 Функција set_track_meta за постављање параметара медија формата

Тек сада се може завршити имплементација у оквиру mm-lib спреге. У функцији за покретање аудио садржаја биће добављен web_audio_source и позване одговарајуће функције како би уопште добили аудио садржај. Потребно је позвати и методу start() над плејером.

```

web_audio_source = static_cast<player::Web_audio_source*>(wa_player->get_media_source().get());
if(web_audio_source) {
    web_audio_source->set_track_meta(handle);
    web_audio_source->pass_param(entry->data_callback, handle); /* pass handle and callback to extractor */
    register_stop_player_callback(stop_player_callback, web_audio_source);
}
if(entry->selected_audio_track_index == -1)
    entry->selected_audio_track_index = select_audio_component(handle, 0);

try {
    wa_player->start();
    set_wa_state_t(WA_STARTED, handle);
} catch(const std::exception& e){
    std::cerr << "mmlib error" << e.what() << std::endl;
}

```

Слика 30 Део waudio_start функције у оквиру mm-lib спреге

Остала је waudio_stop функција где је потребно само позвати горе поменути timer_set функцију и проследити јој параметар wa_end. То ће покренути тајмер уколико је потребно или једноставно стопирати репродукцију аудио садржаја.

```
static uint8_t waudio_stop(WebAudioHandle handle, const double when)
{
    DBG_INFO("M-AV waudio_stop called");
    WEB_AUDIO_ENTRY_t *entry = (WEB_AUDIO_ENTRY_t *)handle;
    auto player = get_wa_player_from_handle(handle);
    player::Web_audio_source* wa_source = nullptr;

    wa_source = static_cast<player::Web_audio_source*>(player->get_media_source().get())

    if(wa_source)
        wa_source->timer_set(entry->wa_end);

    return 0;
}
```

Слика 31 Функција waudio_stop у оквиру mm-lib спреге

4.2 Интернет страница која имплементира Web Audio технологију

У оквиру странице постоје три дугмета. Свако од њих ће имати своју функционалност.

```
<div class="container">
  <h1>Web Audio Player</h1>
  <h2>Current Time: <span id="current-time"></span></h2>
  <button onclick="playSound()" class="button-container custom-button custom1-button">Play Sound</button>
  <button onclick="stopSound()" class="button-container custom-button custom2-button">Stop Sound</button>
  <button onclick="pause()" class="button-container custom-button white-button">Pause</button>
</div>
```

Слика 32 Приказ функционалности дугмића

Главну функционалност има дугме за пуштање аудио садржаја. Оно позива playSound функцију. У оквиру ње се постављају одређени атрибути дугмади. Поред тога, креира се нови аудио контекст за манипулацију и репродукцију аудио сигнала. Користи се и функција fetch а преузимање одређене аудио датотеке. Да би се преузети подаци сместили у бафер, користи се метода arrayBuffer(). Такве сирове податке је потребно прерадити у податке које је могуће репродуковати. У томе помаже decodeAudioData() метода.

Након што се то одради, позива се метода createBufferSource() која креира нови BufferSource објекат за репродукцију аудио података. На крају се успоставља веза између изворног објекта и излазног чвора (енг. destination) аудио контекста.

Након свега овог изворни објекат је спреман за репродукцију аудио података. Тако повезан са излазом аудио контекста омогућава репродукцију звука на одговарајућем излазном уређају, што омогућава даља испитивања.

Репродукција започиње методом `start()` која може имати највише три параметра: када да започне репродукцију, после колико секунди звука да свира и колико дуго да свира. У колико се не проследи ни један, аудио ће се репродуковати одмах након клика читавом дужином.

```
<script>
let source;
let audioContext;
var paused;
const startBtn = document.querySelector("button:nth-of-type(1)");
const stopBtn = document.querySelector("button:nth-of-type(2)");
const pauseBtn = document.querySelector("button:nth-of-type(3)");
const timeDisplay = document.querySelector("p");
pauseBtn.setAttribute("disabled", "disabled");
stopBtn.setAttribute("disabled", "disabled");
async function playSound() {
  startBtn.setAttribute("disabled", "disabled");
  pauseBtn.removeAttribute("disabled");
  stopBtn.removeAttribute("disabled");
  pauseBtn.textContent = "Pause";
  audioContext = new AudioContext();
  const response = await fetch("Kalimba.mp3");
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await audioContext.decodeAudioData(arrayBuffer);
  source = audioContext.createBufferSource();
  source.buffer = audioBuffer;
  source.connect(audioContext.destination);
  source.loopStart = 1;
  source.loopEnd = 3;
  source.loop = true;
  if(source.loop == false)
    startBtn.removeAttribute("disabled");
  var currentTime = audioContext.currentTime + 0.2;
  document.getElementById("current-time").textContent = currentTime.toFixed(2);
  paused = false;
  source.start();
  stopBtn.removeAttribute("disabled");

  return source;
}
```

Слика 33 Изглед функције `playSound` када `stop()` метода нема параметара

У оквиру `stopSound` функције се стопира репродукција аудио садржаја. Позива `stop()` методу која може бити без параметара, а може јој се проследити и за колико секунди треба репродукцију.

Функција `pause` има двоструку улогу. Она паузира учитавање аудио садржаја и наставља учитавање у зависности од тога шта је покренуто прво. Ако се учитавање садржаја заврши а позове се `pause` функција, након које ће уследити комплетно стопирање репродукције, као што то ради `stopSound`.

```
function stopSound() {  
  if (source) {  
    startBtn.removeAttribute("disabled");  
    pauseBtn.setAttribute("disabled", "disabled");  
    source.stop(2);  
    stopBtn.setAttribute("disabled", "disabled");  
    pauseBtn.textContent = "Pause";  
  }  
}  
  
function pause() {  
  startBtn.removeAttribute("disabled");  
  if (audioContext.state === "running") {  
    audioContext.suspend().then(() => {  
      pauseBtn.textContent = "Resume";  
    });  
  } else if (audioContext.state === "suspended") {  
    audioContext.resume().then(() => {  
      pauseBtn.textContent = "Pause";  
    });  
  }  
}
```

Слика 34 Изглед функције stopSound и pause

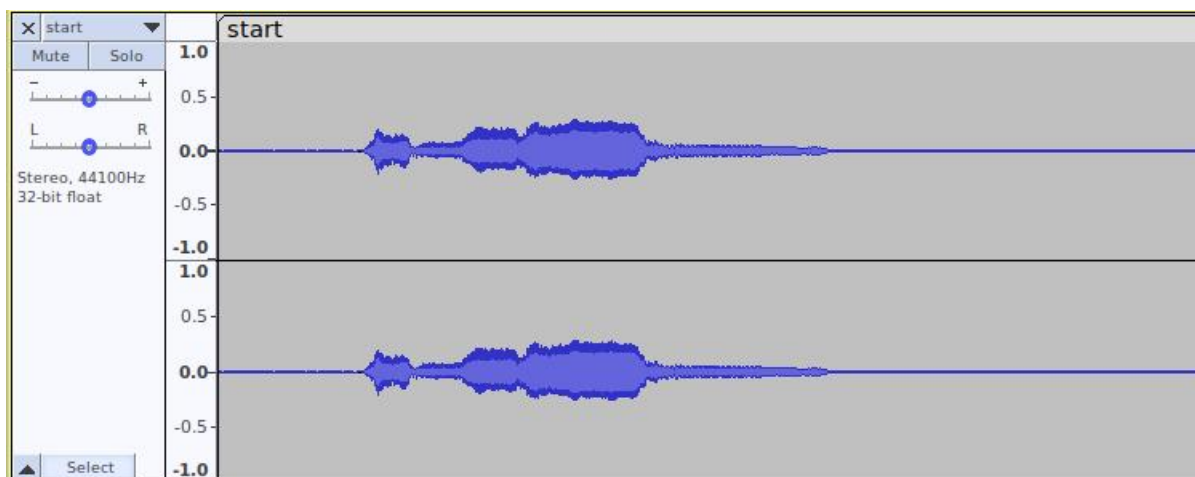
5. Резултати

У овом делу биће представљени начини испитивања и верификације. Испитивање је остварено на основу исписа у оквиру функција, где су проверена позивања свих функционалности.

Провера самог добављања аудио сигнала, као и његова репродукција, реализована је учитавањем снимљене датотеке у програмско окружење Audacity. У томе ће помоћи креирана интернет страница.

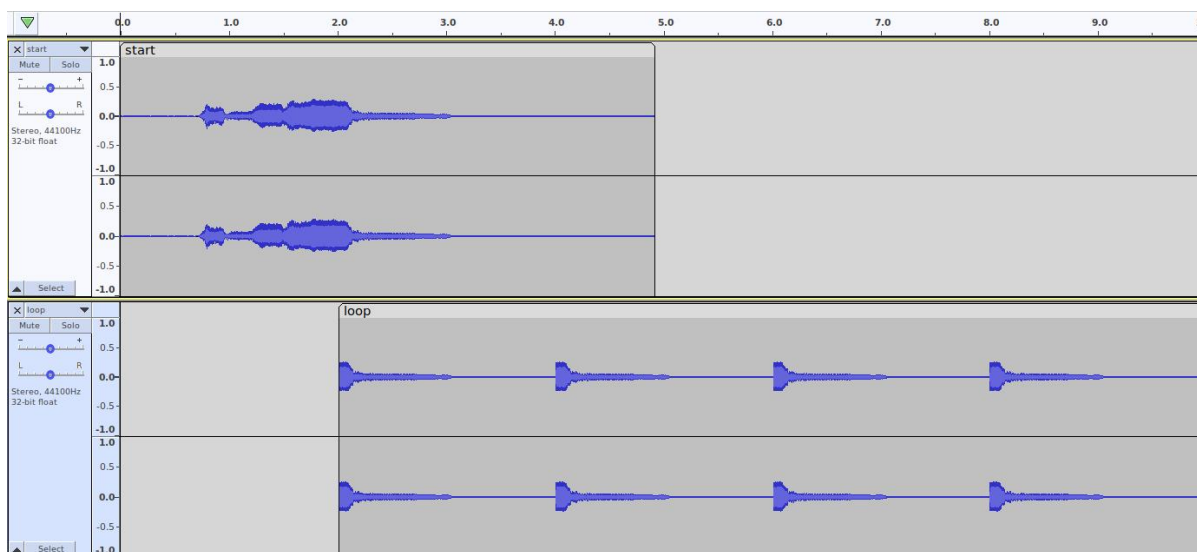


Слика 35 Изглед тестне странице



Слика 36 Изглед аудио сигнала у Audacity окружењу

Проверено је сечење сигнала и репродуковање у петљи тако што се изнова и изнова врши упис у датотеку и тако симулира континуално пуштање звука на плејер.



Слика 37 Аудио сигнал који свира аудио од 2. до 4. секунде у континуитету

Будући да прегледач TeatroTVBrowser већ одбацује делове аудио садржаја који му нису потребни на основу параметара start методе, потребно је само покупити тај део који остане и исписати у датотеку.

Што се тиче даљих испитивања, одрађено је испитивање на основу странице која у себи садржи испитне скрипте, које ће помоћи да се додатно испита Web Audio API.

Свих деветнаест испитних случајева је успешно прошло испитивање. За први испитни случај било је неопходно прикачити звучник за плочу и покренути испитивање. Када се покрене испитни случај, требало би да се чује одређени звук и тако све до краја испитног случаја. За остала испитивања није био неопходан звучник ни подешен плејер.

Тест идентификатор	Име теста	Број догађаја	Резултат
ACT-1174	Web Audio Manual Verification	9	успешан
ACT-1178	Biquad bandpass filter test	1	успешан
ACT-1168	Web Audio AnalyserNode Basic Test	1	успешан
ACT-1179	Biquad highshelf filter test	1	успешан
ACT-1173	Web AudioBuffer Test	4	успешан
ACT-1175	Web AudioSourceNode Test	2	успешан
ACT-1169	Web Audio DynamicCompressorNode Basic Test	1	успешан
ACT-1180	Biquad lowshelf filter test	1	успешан
ACT-1181	Biquad lowpass filter test	1	успешан
ACT-1185	Biquad peaking filter test	1	успешан
ACT-1170	Web Audio AnalisyserNode Test	121	успешан
ACT-1182	Web Audio Biquad Node Test	3	успешан
ACT-1186	Web Audio GainNode Test	1	успешан
ACT-1183	Biquad notch filter test	1	успешан
ACT-1171	Web AudioContext Test	1	успешан

ACT-1172	Web Audio DynamicCompressorNode Test	11	успешан
ACT-1177	Biquad highpass filter test	1	успешан
ACT-1184	Web Audio Delay Test	1	успешан
ACT-1176	Biquad allpass filter test	1	успешан

Табела 1 Резултати верификације

6. Закључак

У овом раду је реализовано прилагођавање аудио садржаја и његова обрада, као и реализација спреге која повезује *Chromium* прегледач и плејер. Обрађени су подаци добијени са *Chromiuma* и такви прослеђени даље, за додатну обраду сигнала. Омогућене су, и испитане, све функционалности које један плејер захтева, а које Web Audio API пружа.

Све функционалности су успешно испитане писањем података у датотеку, уместо слања плејеру директно. На тај начин је осигурано да ће плејер добити аудио који му се достави, био то комплетан аудио садржај или његов одређени део. Испитивања су реализована уз помоћ веб странице, направљене управо ради испитивања функционалности целокупног Web Audio API. За додатна испитивања имплементације коришћена је аутоматизована испитна страница на којој је остварена успешност од 100%. За све ово коришћена је Mediatek AOW развојна плоча.

Даља проширења и оптимизацију могуће је применити на саму апликацију и њен дизајн. Што се тиче осталих функционалности, оне су рађене на основу већ постојећег Web Audio API-ја, тако да њихова даља проширења зависе искључиво од развоја самог API-ја.

7. Литература

- [1] <https://www.lifewire.com/chromium-web-browser-4171288> посећено 6.6.2023.
- [2] <https://www.island.io/blog/chromium-internals-101> посећено 6.6.2023.
- [3] https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API посећено 7.6.2023.
- [4] <https://www.chromium.org/developers/design-documents/multi-process-architecture/> посећено 26.6.2023.
- [5] <https://www.computerworld.com/article/3261009/googles-chromium-browser-explained.html> посећено 6.6.2023.
- [6] <https://www.chromium.org/developers/design-documents/multi-process-architecture> посећено 6.6.2023.