Участники проекта:

* Абраменко Алексей, 751003
* Гринчик Всеволод, 751003

**ТЗ**

**«Меломан»**

«Меломан» - это музыкальная социальная сеть, с возможность редактирования и наложения эффектов на музыкальные файлы онлайн.

Функционал:

* добавление/удаление музыкальных файлов;
* прослушивание музыки;
* визуализация музыки;
* постинг музыки (а-ля Instagram);
* регистрация/авторизация профиля;
* возможность редактирования профиля;
* возможность лайкать посты;
* возможность комментировать посты;
* возможность скачивания музыки;
* поиск по профилю;
* просмотр чужих профилей;
* поиск музыки;
* обработка музыки:

o   запись и монтаж фрагментов;

o   амплитудные преобразования;

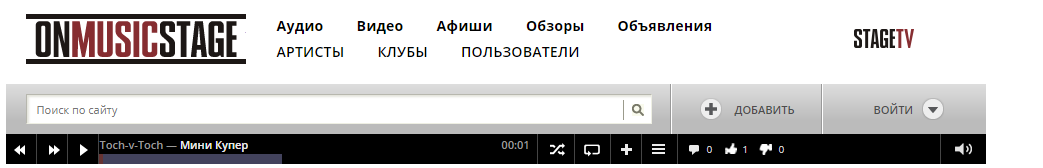
o   частотные преобразование;

o   временные преобразования.

**Обзор аналогов**

К аналогам нашего приложения можно отнести:

### **O**nmusicstage.com/– наиболее релевантная музыкальная соцсеть в google.



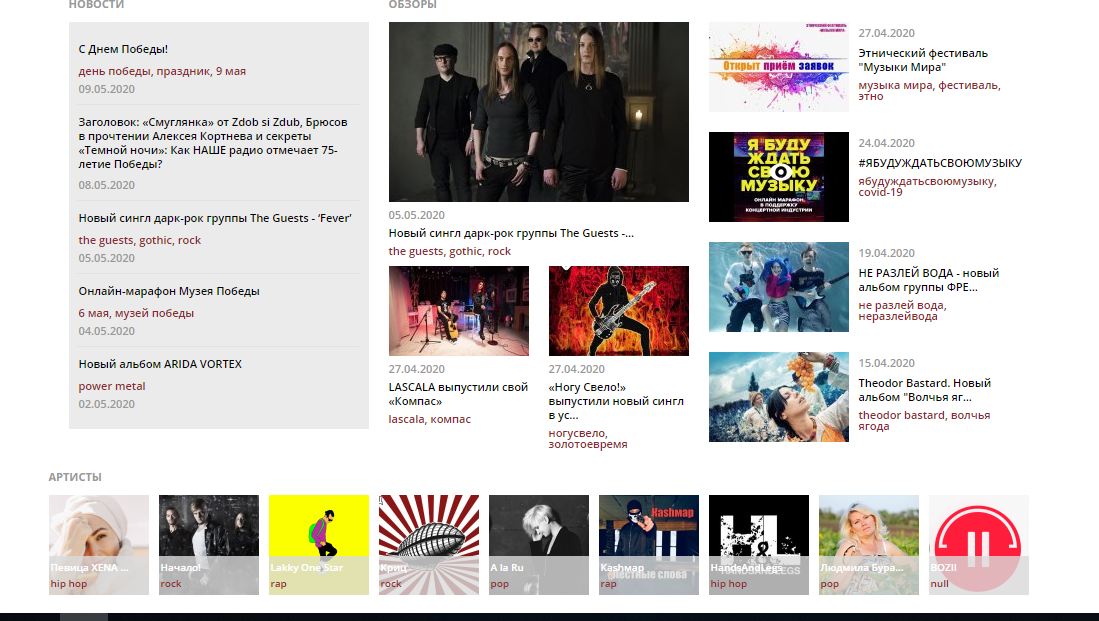


Рисунок 1.1.1– Внешний вид **O**nmusicstage

**O**nmusicstage – предоставляет возможность добавления и прослушивания музыки, а также поиск музыки и пользователей по параметрам. Кроме того на сайте могут регистрироватся представители клубов и оставлять афиши.

**Плюсы:**

Добавление музыки в плейлист;

Поиск по жанрам;

Просмотр афиш.

**Минусы:**

* Реклама;
* Невозможно обрабатывать музыку;
* Невозможно скачать музыку.

В целом, я считаю, что это неплохой сайт для обмена и постинга музыки, однако отсутствие обработки музыки и монтажа делает сайт не конкурентно способным. Кроме того в последнее время на сайте перестала работать функция регистрации.

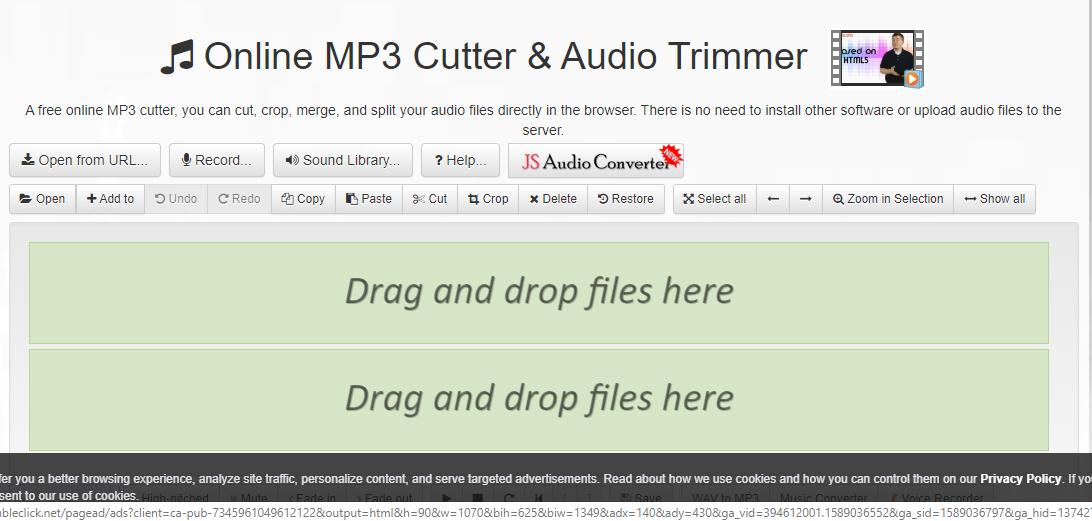
**1.1.2 Bear Audio Tool** – онлайн аудио редактор. 

Рисунок 1.1.2 – Внешний вид **Bear Audio Tool**

**Плюсы:**

Доступность;

Возможность монтажа музыки.

Возможность обработки аудио.

**Минусы:**

* Нет функционала соцсети;
* Отсутствие тонкой настройки эквалайзераж
* Нельзя сохранить файлы на стороне сервера.

Хороши аудио редактор, с полным набором необходимых функций. Однако может считаться лишь частичным аналогом нашего приложения.

Блок схемы



Рисунок 2.1.1 –Авторизация пользователя

Рисунок 2.1.2 –Получение списка треков (часть 1)



Рисунок 2.1.3 –Получение списка треков (часть 2)



Рисунок 2.1.4 –Получение данных авторизированного пользователя



Рисунок 2.1.5 –Получение треков, добавленных пользователем



Рисунок 2.2.1 –Вывод списка треков на страницу (часть 1)



Рисунок 2.2.2 –Вывод списка треков на страницу (часть 2)



Рисунок 2.2.3 –Вывод списка треков на страницу (часть 2)



Рисунок 2.3.1 –Схема обработки сигнала для получения эффекта множественного затухающего эха

Схема базы данных

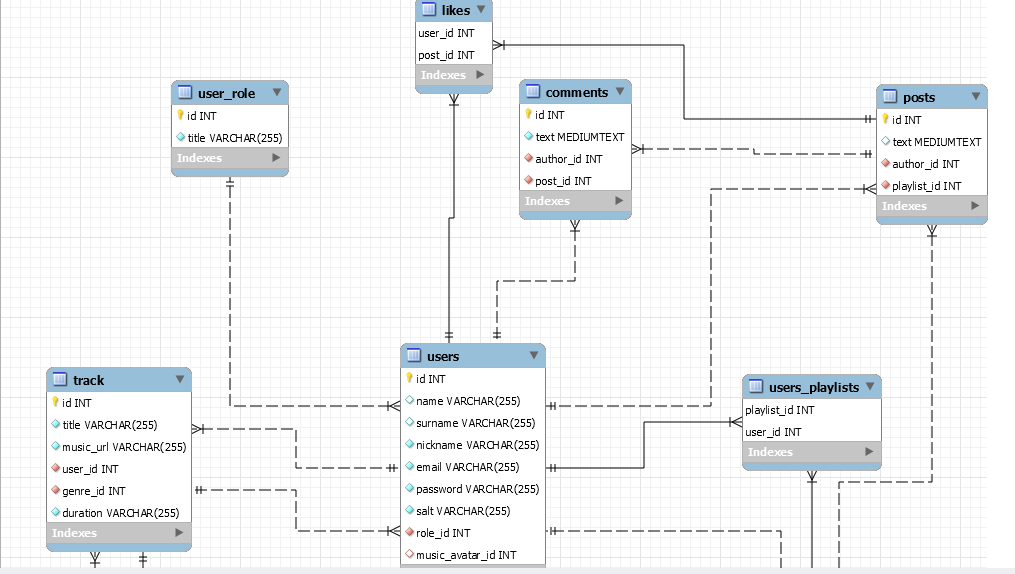


Рисунок 3.1.1 –Схема базы данных (часть1)

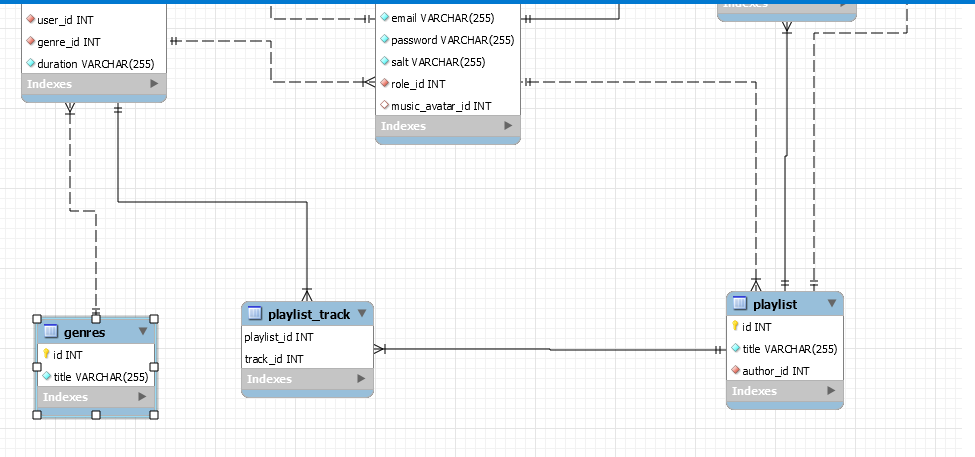


Рисунок 3.1.2 –Схема базы данных (часть1)

**Приложение А**

**Фрагменты кода**

**Назначение маршрутов API**

var express = require('express');  
var ***router*** = express.Router();  
  
var ***usersRout*** = require('./usersRout');  
***router***.use('/user', ***usersRout***);  
  
var ***trackRout*** = require('./trackRout');  
***router***.use('/track', ***trackRout***);  
  
var ***musicRout*** = require('./musicRout');  
***router***.use('/music', ***musicRout***);  
  
var ***playlistRout*** = require('./playlistRout');  
***router***.use('/playlist', ***playlistRout***);  
  
var ***postRout*** = require('./postRout');  
***router***.use('/post', ***postRout***);  
  
var ***postRout*** = require('./commentRout');  
***router***.use('/comment', ***postRout***);  
  
var ***postRout*** = require('./likeRout');  
***router***.use('/like\_in\_post', ***postRout***);  
  
var ***genreRout*** = require('./genreRout');  
***router***.use('/genre', ***genreRout***);  
  
module.exports = ***router***;

**Назначение обработчиков для запросов API/user/**

var express = require('express');  
const userController = require("../../controllers/usersController.js");  
var ***userRouter*** = express.Router();  
const authHelper = require("../../helpers/authHelper");  
  
  
***userRouter***.get('/', userController.getUsersByQuery);  
  
***userRouter***.post('/register', userController.registerUser);  
  
***userRouter***.post('/authenticate', userController.authenticateUser);  
  
***userRouter***.get("/me", authHelper.checkAuth, userController.getUserMe);  
  
***userRouter***.put('/', authHelper.checkAuth, userController.updateUser);  
  
***userRouter***.delete('/:id', authHelper.checkAuth, authHelper.checkAdmin, userController.deleteUser);  
  
module.exports = ***userRouter***;

**Функции контроля доступа**

const ***logger*** = require('../configs/logger4jsInit')  
const PRIVATE\_KEY = require("../configs/token\_key").private\_key;  
var jwt = require('jsonwebtoken');  
  
  
exports.checkAuth = function(req, res, next) {  
  
 if (req.headers.authorization && req.headers.authorization.split(' ')[0] === 'Bearer') {  
 var token = req.headers.authorization.split(' ')[1];  
 }  
 if (!token) return res.status(401).send({ message: 'No token provided.' });  
 jwt.verify(token, PRIVATE\_KEY, function(err, decoded) {  
 if (err) return res.status(500).send({ message: 'Failed to authenticate token.' });  
  
 req.currentUser = {user\_id : decoded.user\_id, role\_id: decoded.user\_role};  
 ***logger***.debug(req.currentUser);  
 next();  
 });  
}  
  
exports.checkAdmin = function(req, res, next) {  
 ***logger***.debug("CheckAdmin");  
 if (req.currentUser.role\_id < 2) {  
 next();  
 }  
 else  
 {  
 return res.status(400).send({ message: 'Access denied.' });  
 }  
}

**Функции аутентификации**

exports.authenticateUser = function (request, response){  
  
 const email = request.body.email;  
 const password = request.body.password;  
 //logger.debug( id);  
 const connection = mysql.createConnection(mySqlConfig.config);  
 connection.query(User.GET\_USER\_BY\_EMAIL, [email], function(err, data) {  
 if(err) {  
 return response.status(400).send({message: err.message});  
 };  
  
 if(data.length == 0)  
 {  
 return response.status(400).send(  
 {message: "email or password incorrect!"});  
 }  
 else  
 {  
 const passwordEntity = bcrypt.hashSync(password, data[0].salt);  
  
 if(passwordEntity != data[0].password)  
 {  
 return response.status(400).send({message: "email or password incorrect!"});  
 }  
 else{  
 ***logger***.debug(data[0]);  
  
 const token = jwt.sign({  
 user\_id : data[0].id,  
 user\_role: data[0].role\_id }, PRIVATE\_KEY,  
 {  
 algorithm: 'HS256',  
 expiresIn: expirationTime  
 });  
 return response.status(200).send({token: token, exp: expirationTime});  
 }  
 }  
 });  
 connection.end();  
};

**Функции обращения к базе данных**

module.exports = ***Object***.freeze({  
 // USERS  
 ADD\_USER : 'INSERT INTO users (nickname, email, password, salt) VALUES (?,?,?,?)',  
 GET\_ALL\_USERS : 'SELECT id, name, surname, nickname, email, music\_avatar\_id FROM users',  
 DELETE\_USER\_BY\_ID : 'DELETE FROM users WHERE id=?',  
 UPDATE\_USER : 'UPDATE users SET name=?, surname=?, nickname=?, email=?, password=?, salt=?, music\_avatar\_id=? WHERE id=?',  
 GET\_USER\_BY\_ID :'SELECT \* FROM users WHERE id=?',  
 GET\_USER\_BY\_EMAIL :'SELECT \* FROM users WHERE email=?',  
 GET\_USER\_BY\_ROLE\_ID :'SELECT \* FROM users WHERE role\_id=?'  
  
});

**Получение отфильтрованного списка треков**

exports.getTracksByQuery = function (request, response){  
 var user\_id= request.query.user\_id  
 if(request.currentUser){  
 user\_id = request.currentUser.user\_id;  
 }  
 const id = request.query.id;  
 const title = request.query.title;  
 const genre\_id = request.query.genre\_id;  
  
 const connection = mysql.createConnection(mySqlConfig.config);  
  
  
 var sqlRequest = '';  
 if (user\_id) {  
 sqlRequest += ' user\_id=' + user\_id;  
 sqlRequest += ' AND';  
 }  
 if (genre\_id) {  
  
 sqlRequest += ' genre\_id=' + genre\_id;  
 sqlRequest += ' AND';  
 }  
 if (title) {  
 sqlRequest += ' title LIKE \'%'+title+'%\'';  
 }  
 if (sqlRequest.substr(sqlRequest.length-3, 3) == 'AND')  
 {  
 sqlRequest = sqlRequest.slice(0, -3);  
 }  
 if (sqlRequest != ''){  
 sqlRequest = ' WHERE' + sqlRequest;  
 }  
 ***logger***.debug(Track.GET\_ALL\_TRACKS+sqlRequest);  
 connection.query(Track.GET\_ALL\_TRACKS+sqlRequest, function (err, data) {  
 if(err) {  
 return response.status(400).send({message: err.message});  
 };  
 return response.status(200).send(data);  
 });  
  
 connection.end();  
};

**Добавление плейлиста**

exports.addPlayList= function(request, response){  
 const title = request.body.title;  
 var author\_id = request.currentUser.user\_id;  
 if((request.currentUser.role\_id < 2))  
 {  
 if(request.query.author\_id) {  
 author\_id = request.query.author\_id;  
 }  
 }  
 if (!request.body.tracks) {  
 return response.status(400).send({message: "field tracks must be not null"});  
 }  
  
 const connection = mysql.createConnection(mySqlConfig.config);  
 connection.query( PlayList.ADD\_PLAYLIST, [title, author\_id], function(err, data) {  
 if(err) {  
 connection.end();  
 return response.status(400).send({message: err.message});  
 };  
 //logger.debug( data);  
  
 var sqlParams = [];  
 const playlist\_id = data.insertId;  
 request.body.tracks.forEach(track\_id => sqlParams = sqlParams.concat([[playlist\_id, track\_id]]));  
 connection.query( PlayList\_Track.ADD\_PLAYLIST\_TRACK, [sqlParams], function(err, data) {  
 if(err) {  
  
 return response.status(400).send({message: err.message});  
 };  
 //logger.debug( data);  
 return response.status(201).send({insert\_id: playlist\_id});  
 });  
 connection.end();  
 });  
  
};

**Добавление треков в плейлист**

exports.addTrack= function(request, response){  
 const author\_id = request.currentUser.user\_id;  
 const id = request.params.id;  
 if (!request.body.tracks) {  
 return response.status(400).send({message: "field tracks must be not null"});  
 }  
  
 const connection = mysql.createConnection(mySqlConfig.config);  
 var sqlRequest = PlayList.GET\_ALL\_PLAYLISTS+' WHERE id='+id;  
 if(!(request.currentUser.role\_id < 2))  
 {  
 sqlRequest +=' AND author\_id='+author\_id;  
  
 }  
 ***logger***.debug(sqlRequest);  
 connection.query(sqlRequest, function (err, data) {  
 if(err) {  
 connection.end();  
 return response.status(400).send({message: err.message});  
 };  
  
 if(data.length == 0)  
 {  
 return response.status(400).send({ message: "playlist not found"});  
 }  
 var sqlParams = [];  
 const playlist\_id = data[0].id;  
 request.body.tracks.forEach(track\_id => sqlParams = sqlParams.concat([[playlist\_id, track\_id]]));  
 ***logger***.debug(data.id);  
 connection.query( PlayList\_Track.ADD\_PLAYLIST\_TRACK, [sqlParams], function(err, data) {  
 if(err) {  
  
 return response.status(400).send({message: err.message});  
 };  
 //logger.debug( data);  
 return response.sendStatus(201);  
 });  
 connection.end();  
 });  
  
};

**Функции обработки временных характеристик сигнала для получения эффекта множественного эха**

(function () {  
  
 var context = null,  
 //audio = null,  
 delayEchoNodes = [],  
 gainEchoNodes = [],  
  
  
 $ = ***document***.querySelector.bind(***document***),  
 createContext = function (param) {  
 context = param.context;  
 delayEchoNodes = [];  
 gainEchoNodes = [];  
 },  
 */\*\*  
 \* creates number input elements  
 \*/* createInput = function (container) {  
 var node, label;  
  
 node = ***document***.createElement('input');  
 node.id = "echoEffect";  
 node.type = "checkbox";  
 label = ***document***.createTextNode("Echo");  
 container.appendChild(node);  
 container.appendChild(label);  
 label = ***document***.createElement('br');  
 container.appendChild(label);  
 return node;  
 },  
  
 */\*\*  
 \* check param  
 \*/* validateParam = function (param) {  
 if (!param) {  
 throw new ***TypeError***('error equalizer must have audio and input container params');  
 }  
 return true;  
 },  
  
  
  
 */\*\*  
 \* bind input.change events to the filters  
 \*/* initEvent = function (input, source, output) {  
  
 input.addEventListener('change', function() {  
 if(this.checked) {  
  
 source.connect(gainEchoNodes[0]);  
 delayEchoNodes[0].connect(output);  
 }  
 else{  
 source.disconnect(gainEchoNodes[0]);  
 delayEchoNodes[0].disconnect(output);  
 }  
 });  
 },  
  
  
  
 */\*\*  
 \* create filter for each frequency  
 \*/* createFilters = function (number) {  
  
 delayEchoNodes.push(context.createDelay());  
 gainEchoNodes.push(context.createGain());  
 delayEchoNodes[0].delayTime.value = 0.05;  
 gainEchoNodes[0].gain.value = 0.7;  
 gainEchoNodes[0].connect(delayEchoNodes[0]);  
  
 for(let i= 1; i<number; i++) {  
 delayEchoNodes.push(context.createDelay());  
 gainEchoNodes.push(context.createGain());  
 delayEchoNodes[i].delayTime.value = 0.05;  
 gainEchoNodes[i].gain.value = 0.6;  
 gainEchoNodes[i].connect(delayEchoNodes[i]);  
 gainEchoNodes[i-1].connect(gainEchoNodes[i]);  
 delayEchoNodes[i].connect(delayEchoNodes[i-1]);  
  
 }  
 },  
  
  
 */\*\*  
 \* main function  
 \*/* echo = function (param) {  
  
 if (validateParam(param)){  
 createContext(param);  
 var container = $(param.container);  
 let input =createInput(container);  
 ***console***.log(input.id);  
 createFilters(7);  
 initEvent(input, param.source, param.output);  
 return gainEchoNodes[0];  
  
 }  
 };  
  
 echo.context = context;  
  
 ***window***.echo = echo;  
  
}());

**Функция построения гистограммы частотных характеристик сигнала**

function drawGistograms(){  
  
 requestAnimationFrame(drawGistograms);  
 ***analyser***.getByteFrequencyData(***dataArray***);  
 ***canvas\_ctx***.fillStyle = 'rgb(0,0,0)';  
 ***canvas\_ctx***.fillRect(0,0, ***canva***.width, ***canva***.height);  
 var barWidth = (***canva***.width/ ***bufferLength***) \* 2.5;  
 var barHeight;  
 var x = 0;  
  
 for( var i=0; i< ***bufferLength***; i++){  
 barHeight = ***dataArray***[i]/2;  
 ***canvas\_ctx***.fillStyle = 'rgb(' + (barHeight+100)+',50,50)';  
 ***canvas\_ctx***.fillRect(x, ***canva***.height - barHeight, barWidth, barHeight);  
 x += barWidth +1;  
 }  
  
}

**Функция загрузки данных треков на страницу**

const ***selectGenreIdName*** = "choose-track-genre-list";  
  
function onAddTrackPageLoad(afterWhichElementId) {  
   
 var currUserToken = getCookie("userToken");   
   
 var requestOptions = {  
 method: 'GET',  
 headers: {  
 'Authorization':`Bearer ${currUserToken}`  
 },  
 redirect: 'follow'  
 };  
  
 fetch("http://localhost:3000/api/genre", requestOptions)  
 .then(async response => {  
 var result = await response.json();  
 if (response.ok)  
 {  
 let select\_choose\_track\_genre\_list = ***document***.createElement('select');  
 select\_choose\_track\_genre\_list.id = ***selectGenreIdName***;  
 select\_choose\_track\_genre\_list.className = "choose-track-genre-list"  
 select\_choose\_track\_genre\_list.setAttribute("required", "required");  
 select\_choose\_track\_genre\_list.setAttribute("size", "1");  
   
 for(let item of result)  
 {  
 var newOption = ***document***.createElement("option");  
 newOption.value = item.id;  
 newOption.innerHTML = item.title;  
  
 select\_choose\_track\_genre\_list.insertAdjacentElement('beforeend', newOption);  
 }  
   
  
 ***document***.getElementById(afterWhichElementId).insertAdjacentElement('afterend', select\_choose\_track\_genre\_list);  
 }  
 else  
 {  
 alert(result.message);  
 }  
 })  
 .catch(error => ***console***.log('error', error));   
}

**Функция добавления трека на сервер**

function onAddTrackClick(inputAuthorId, inputNameId, inputFileId, playerId) {  
   
 if (validateData(inputAuthorId, inputNameId, inputFileId) === false) {  
 return;  
 }  
   
 let targetFile = ***document***.getElementById(inputFileId).files[0];   
   
 const formData = new ***FormData***();   
 formData.append('file\_data', targetFile);  
  
 var currUserToken = getCookie("userToken");  
  
 fetch("http://localhost:3000/api/music", {  
 method: 'POST',  
 headers: {  
 'Authorization':`Bearer ${currUserToken}`  
 },  
 body: formData})  
 .then(async response => {  
 var result = await response.json();  
 if (response.ok)  
 {  
 ***console***.log('Успех:', ***JSON***.stringify(result));  
   
  
 var bufAudio = ***document***.createElement('audio');  
  
 // Define the URL of the MP3 audio file  
 bufAudio.src = result.file\_path;  
  
 // Once the metadata has been loaded, display the duration in the console  
 bufAudio.addEventListener('loadedmetadata', function(){  
  
 let selectGenreElement = ***document***.getElementById(***selectGenreIdName***);  
  
 let trackAuthor = ***document***.getElementById(inputAuthorId).value;  
 let trackName = ***document***.getElementById(inputNameId).value;  
 let trackGenreId = selectGenreElement.options[selectGenreElement.selectedIndex].value;  
 let trackUrl = result.file\_path;  
 let trackDuration = ***Math***.trunc(bufAudio.duration);  
   
 let trackTitle = trackAuthor + " - " + trackName;  
   
 var raw = `{\"title\": \"${trackTitle}\",\"genre\_id\": ${trackGenreId},\"music\_url\": \"${trackUrl}\",\"duration\": \"${trackDuration}\"}`;  
   
 var requestOptions = {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json;charset=utf-8',  
 'Authorization':`Bearer ${currUserToken}`  
 },  
 body: raw,  
 redirect: 'follow'  
 };  
  
 fetch("http://localhost:3000/api/track", requestOptions)  
 .then(async response => {  
 var result = await response.json();  
 if (response.ok)  
 {  
 alert("Success");  
 }  
 else  
 {  
 alert(result.message);  
 }  
 })  
 .catch(error => ***console***.log('error', error));   
  
 },false);   
 }  
 else  
 {  
 alert(result.message);  
 }  
 })  
 .catch(error => ***console***.log('error', error));  
  
}

**Функции проигрывания музыки**

function playMusic(srcElementId, srcDataAttributeName, srcIsLastAttributeName, playerElementId, nextTargetId) {  
 var prevElementChildrens;  
 var prevChildrenChildrens;   
   
 var previousTrackId = ***document***.getElementById(playerElementId).getAttribute(***PREVIOUS\_TRACK\_ID\_DATA\_NAME***);  
 var isPause = ***document***.getElementById(playerElementId).getAttribute(***IS\_AUDIO\_PAUSE\_DATA\_NAME***);   
   
 if (previousTrackId != null) {  
   
 if (previousTrackId == srcElementId) {  
 if (isPause != null) {  
 ***console***.log("isPause exist");  
 ***console***.log(isPause);  
   
 if (isPause == "false") {  
 ***console***.log("isPause = false");  
 musicIconPause(previousTrackId);  
  
 ***document***.getElementById(playerElementId).pause();  
 ***document***.getElementById(playerElementId).setAttribute(***IS\_AUDIO\_PAUSE\_DATA\_NAME***, "true");   
 return;  
 }  
 else if (isPause == "true"){  
 ***console***.log("isPause = true");  
 musicIconPlay(previousTrackId);  
  
 ***document***.getElementById(playerElementId).play();  
 ***document***.getElementById(playerElementId).setAttribute(***IS\_AUDIO\_PAUSE\_DATA\_NAME***, "false");   
 return;   
 }  
 }   
 }   
 else {  
 musicIconDefault(previousTrackId);  
 }  
 }  
  
 ***document***.getElementById(playerElementId).src = ***document***.getElementById(srcElementId).getAttribute(srcDataAttributeName);  
 ***document***.getElementById(playerElementId).autoplay = "autoplay";   
 ***document***.getElementById(playerElementId).setAttribute(***IS\_AUDIO\_PAUSE\_DATA\_NAME***, "false");  
 ***document***.getElementById(playerElementId).setAttribute(***PREVIOUS\_TRACK\_ID\_DATA\_NAME***, srcElementId);  
   
 musicIconPlay(srcElementId);   
   
 var isLast = ***document***.getElementById(srcElementId).getAttribute(srcIsLastAttributeName);  
   
 if (isLast == "false") {  
 ***console***.log("not Last");  
 ***console***.log(nextTargetId);  
 ***document***.getElementById(playerElementId).onended = function() {  
 ***console***.log("not Last ended");  
 musicIconDefault(srcElementId);   
   
 onMusicEnded(nextTargetId);  
 }   
 }  
 else {  
 ***console***.log("is Last");  
 ***console***.log(nextTargetId);  
 ***document***.getElementById(playerElementId).onended = function() {  
 ***console***.log("is Last ended");  
 musicIconDefault(srcElementId);   
 }  
 }  
}  
  
  
  
function onAudioPlay(playerElementId) {  
 var previousTrackId = ***document***.getElementById(playerElementId).getAttribute(***PREVIOUS\_TRACK\_ID\_DATA\_NAME***);  
  
 if (previousTrackId == null) {  
 return;  
 }  
  
 musicIconPlay(previousTrackId);   
}  
  
  
function onAudioPause(playerElementId) {  
 var previousTrackId = ***document***.getElementById(playerElementId).getAttribute(***PREVIOUS\_TRACK\_ID\_DATA\_NAME***);  
  
 if (previousTrackId == null) {  
 return;  
 }  
  
 musicIconPause(previousTrackId);   
}  
  
  
  
function musicIconDefault(trackId) {  
 musicIconAction(trackId, ***DEFAULT\_TRACK\_ICON\_CLASSNAME\_ADD***, ***PLAYING\_TRACK\_ICON\_CLASSNAME\_ADD***, ***PAUSED\_TRACK\_ICON\_CLASSNAME\_ADD***);   
}  
  
function musicIconPlay(trackId) {  
 musicIconAction(trackId, ***PLAYING\_TRACK\_ICON\_CLASSNAME\_ADD***, ***DEFAULT\_TRACK\_ICON\_CLASSNAME\_ADD***, ***PAUSED\_TRACK\_ICON\_CLASSNAME\_ADD***);  
}  
  
function musicIconPause(trackId) {  
 musicIconAction(trackId, ***PAUSED\_TRACK\_ICON\_CLASSNAME\_ADD***, ***PLAYING\_TRACK\_ICON\_CLASSNAME\_ADD***, ***DEFAULT\_TRACK\_ICON\_CLASSNAME\_ADD***);  
}  
  
function musicIconAction(trackId, addAction, removeActionFirst, removeActionSecond) {  
 var elementChildrens = ***document***.getElementById(trackId).children;  
 var childrenChildrens = elementChildrens[0].children;  
 childrenChildrens[0].classList.add(addAction);   
 childrenChildrens[0].classList.remove(removeActionFirst);   
 childrenChildrens[0].classList.remove(removeActionSecond);   
}

**Источники**

[1]ЧАСТОТНО-ВРЕМЕННЫЕ ПРЕОБРАЗОВАНИЯ, ИСПОЛЬЗУЕМЫЕ В ЦИФРОВОЙ ОБРАБОТКЕ СИГНАЛОВ

© 2015 г. Ю.Е. Ульянова, Р.Г. Бабенко, А.В. Чернов

**[2]Преобразование звука и cпецэффекты (**<http://www.tiflocomp.ru/docs/sound/snd01.php>**)**

**[3] Методическое пособие Цифровая обработка сигнало**

**Алекцеев, Бранцевич, Костюк**

**Ссылки**

**ТЗ**

**https://docs.google.com/document/d/1gXPqgdb9LkeMJtLgCWaR8RN9D2LKbJFvtyBS81jeHbw/edit?usp=sharing**

**Промежуточный отчет**

<https://docs.google.com/document/d/1SW8_hipiiI05Mf6J3XQYcjqY3H6galnHzTXtGqoO3hs/edit?usp=sharing>

Github

<https://github.com/Cothn/Meloman>

Trello

<https://trello.com/b/uKRnW59Q/backend-v010>

<https://trello.com/b/RPTqQH13/frontend-002>

**Титры**

Проектировка структуры БД – Гринчик

Реализация REST API – Гринчик, Абраменко

CRUD (User, Post, Playlist) – Гринчик

CRUD(Like, Comment, Track) –Абраменко

Регистрация/авторизация(Beckend) - Гринчик

Регистрация/авторизация(Frontend) – Абраменко

Фильтрация пользователей и треков – Гринчик

Фильтрация плейлистов - Абраменко

Вывод и добавление треков - Гринчик

Вывод, добавление постов и плейлистов – Абраменко

Тестирование API – Гринчик, Абраменко

Дизайн страниц – Абраменко

Дизайн страницы обработки звука – Гринчик

Реализация стилей и шаблонов страниц – Абраменко

Обработка звука – Гринчик

Проигрывание плейлистов(основной плеер) – Абраменко

Дизайн и разработка главной страницы – Абраменко

Интеграция бэкенда и фронтенда – Абраменко, Гринчик

Обзор аналогов – Гринчик

Генерация идеи – Абраменко, Гринчик

Документация – Гринчик