프로젝트 개발 문서

**201424522 장병엽**

Sole

1. **목차**
2. 개요 3p
3. 특징 및 차별점 3p
4. 개발 과정 4p
5. 결과 5p
6. 개선점 14p
7. **개요**

라이브 스트리밍으로 강의할 수 있는 웹 세미나 플랫폼

강의자에게는 다양한 종류의 강의를 웹 상에서 할 수 있는 환경을 제공하며, 학생에는 강의 뿐만이 아니라 학습에 더 효과적인 환경을 제공한다.

1. **특징 및 차별점**
2. 우리 플랫폼만의 특징
   1. 강의에 특화된 플랫폼을 제공한다. 이 방법으로는 동영상 뿐만이 아니라 ppt나 그림같은 강의자료들을 같은 페이지에서 화면에 표시해줌으로써 강의자는 강의에만 집중할 수 있도록 도와준다.
   2. 채팅서버를 사용하여 학생이 강의자에게도 의사를 전달할 수 있는 양방향 의사소통체계를 구축하여 강의자와 학생간에 더 나은 환경을 제공한다.
   3. 라이브 스트리밍이 종료되면 동영상 강의으로 전환되며. 네트워크 상황에 알맞게 적절한 화질로 끊김없는 동영상을 제공한다.
3. 유사시스템과의 차별 점
4. 기존의 스트리밍 플랫폼은 강의를 위한 플랫폼이 아니다 보니 강의자료와 동영상을 같이 볼 수 있는 환경이 없지만 우리 플랫폼에서는 강의자료를 같이 볼 수 있도록 도와준다.
5. 또한 동영상을 보는것은 제한이 없으므로 일반적인 온라인 강의보다 접근이 쉬우며, 온라인 강의의 경우 소수로 진행되는것에 반하여, 우리 플랫폼에서는 다수의 학생과도 소통이 가능하도록 환경을 제공한다.
6. **개발 과정**
7. 개발에 사용된 기술 및 프로그램
   1. HLS
      1. HLS(Http Live Streaming)는 기존에 자주 사용되던 영상 데이터를 Stream으로 전송하며 State를 가지는 RTSP(Real Time Streaming Protocol)와 다르게 영상 데이터를 정해진 단위 시간 별로 자른 Segment 들을 마치 Stream인 것처럼 연속적으로 실행시키는 프로토콜이다.
      2. 현재 설정된 값은 1초의 단위 시간을 가진 segment 3개가 하나의 리스트가 되는데, 실시간 스트리밍의 딜레이가 약 7.5초이다. 해당 딜레이는 설정 문제 이외에 현재 사용 중인 서버의 네트워크 가용성 문제로 추정된다.
      3. RTSP에 비해 서버 구축 비용이 낮고, HTTP에서 동작하기 때문에 접근이 편하다.
   2. FFMPEG
      1. 모든 멀티미디어 포맷들의 인코딩과 디코딩을 목표로 개발되고 있는 오픈소스 프로젝트이다.
      2. Windows 환경에서 DirectShow를 이용해 현재 사용 가능한 녹화 및 녹음 장치를 인식하여 input으로 하고, 해당 데이터들을 mpeg-2 포맷으로 인코딩하여 서버로 전송한다.
   3. VideoJS
      1. HLS는 최근 대부분의 웹 브라우저에서 지원하고 있지만, 몇몇(대표적으로 크롬) 브라우저에서는 지원하지 않는다. 따라서 VideoJS 플러그인을 이용하여 HLS가 지원되는 경우 그대로 재생하고, 아닌 경우 flash를 embedding 하여 재생한다.
   4. XAMPP
      1. Apache 서버 및 PHP와 MySQL을 쉽게 사용할 수 있도록 묶어놓은 프로그램
      2. 80번 포트로 기존의 http 연결로써 홈페이지에 접근하는 통신을 처리한다.
   5. Node.js
      1. JavaScript를 이용하여 백엔드 프로그래밍을 하기 위한 플랫폼이다.
      2. 80번 포트는 이미 apache 서버에 할당되어 있지만, 웹페이지 제공을 제외하고 http를 통한 통신이 필요했기 때문에 Node.js 를 통해 15000-15001 포트를 또 다른 http통신 용으로 열었다.
      3. 이 중 15000 포트는 개별 클라이언트가 인코딩 데이터를 송신할 때, 새로운 소켓을 열고 그것을 알려주는 일종의 listening 소켓이고, 15001 포트는 웹소켓의 listening 소켓이다.
   6. WebSocket
      1. Stateless한 http 통신에서 채팅을 비롯한 양방향 통신이 필요할 때 사용하는 프로토콜들을 지원 여부에 상관없이 통합적으로 사용하게 만든 모듈
      2. php에서는 WebSocket 모듈을 지원하지 않았기에 직접 4-way handshaking과 MD-5 Hashing과 Magic String을 이용한 인증 문제를 해결하였다.
      3. 여러가지 operation을 정의하여 채팅 뿐만이 아니라, Streaming의 시작 또는 끝을 알 수 있도록 하였다.

1. 현재 개발 현황

|  |  |
| --- | --- |
|  |  |
| 1. 메인 화면   새로운 라이브 스트리밍이 시작될 때마다 하단에 해당 스트리머의 ID를 값으로 가진 버튼들이 생성된다.  해당 버튼을 클릭하면 스트리밍 페이지로 넘어간다.   1. 스트리밍 화면(공용 IP 서버에 PPT가 존재해야함)   좌측 상단에는 스트리머의 영상화면이 등장하고, 하단에는 해당 스트리머가 사용하기 위해 서버에 등록한 ppt가 보여지는데, 이 ppt는 스트리머의 화면에 맞추는 것이 아닌 시청자가 직접 페이지를 이동할 수 있다.  우측에는 스트리머, 그리고 시청자들이 소통할 수 있는 채팅창이 있다.   1. 시스템 순서도    1. 스트리머 SequenceDiagram |  |
| * 1. 시청자 SequenceDiagram |  |

1. **결과**
2. 기능 개발 내용

-Server

* 1. Init()
     1. Content 영역 로드
     2. Websocket callback 초기화
     3. Message Operation 동작 정의

function init(){

var content\_odd = document.getElementById('content\_odd');

var content\_even = document.getElementById('content\_even');

addPresentation('Stereo\_Vision\_Geometry');

socket.onopen = function () {

console.log("websocket connected");

}

socket.onmessage = function(message){

var data = JSON.parse(message.data);

switch(data.op){

case 'newStreaming':

connected++;

if(connected % 2 == 1){

content\_odd.appendChild(newStreaming(data.userID));

}else{

content\_even.appendChild(newStreaming(data.userID));

}

createVideo(data.userID);

break;

case 'endStreaming':

connected--;

player[data.userID].dispose();

player[data.userID] = null;

break;

case 'chat':

addChat(data.chatter, data.message)

break;

}

}

socket.onclose = function(){

}

socket.onerror = function (error) {

console.log('WebSocket error: ' + error);

};

}

* 1. createVideo(userID)
     1. Flash 지원 브라우저인지 확인 후 techOrder를 설정해 모든 브라우저에서 재생가능하게 설정
     2. 재생 동영상 인스턴스 생성

function createVideo(userID){

//Do Something for new streaming

if(hasFlash)

player[userID] = videojs('stream\_' + userID, {techOrder: ['flash']});

else

player[userID] = videojs('stream\_' + userID);

player[userID].ready(function(){

var mvPlayer = player[userID];

console.log('video is ready');

mvPlayer.play();

});

}

* 1. newStreaming(userID)
     1. video tag 생성 후 attribute와 style 설정

function newStreaming(userID){

var videoElement = document.createElement('video');

var sourceElement = document.createElement('source');

videoElement.setAttribute('id', 'stream\_' + userID);

videoElement.appendChild(sourceElement);

videoElement.setAttribute('class', 'video-js vjs-default-skin');

videoElement.setAttribute('width', '320');

videoElement.setAttribute('height', '240');

videoElement.autoplay=true;

videoElement.controls=true;

sourceElement.setAttribute('src', 'http://' + server\_host + '/Sole/mvSource/' + userID + '\_playlist.m3u8');

sourceElement.setAttribute('type', 'application/x-mpegURL');

return videoElement;

}

* 1. http.createServer 중 일부
     1. url method에 따라서 동작 정의
        1. getStreaming -> streaming 리스트를 전달한다.
        2. connectServer -> 새로운 스트리밍 연결을 생성한다.
        3. disconnectServer -> 기존 스트리밍 연결을 해제한다.

function getStreaming(res, requestQuery){

res.writeHead(200, {'Content-Type': 'text/plain', 'Access-Control-Allow-Origin': '\*'});

var cnt = 0;

var list = new Array();

for(var i=0; cnt<10 && i <= 55; i++){

if(streamers[i] != undefined){

list.push(streamers[i]);

cnt++;

}

}

res.write(JSON.stringify(list));

res.end();

}

function connectServer(res, requestQuery){

var idx;

for(var i=0; i<10; i++){

if(streamers[i] == undefined){

idx = i;

break;

}

}

var param = parseQuery(requestQuery);

var openPort = idx \* 100 + basePort;

var curStreamers = {

port: openPort,

op: 'newStreaming',

userID: param['userID']

};

processes[idx] = openFFMPEG(param['userID'], openPort);

var json = JSON.stringify(curStreamers);

streamers[idx] = curStreamers;

for(var i=0; i<clients.length; i++){

clients[i].send(json);

}

console.log('new connection at port ' + openPort + ' userID ' + param['userID']);

res.writeHead(200, {'Content-Type': 'text/plain'});

res.write(json);

res.end();

}

function disconnectServer(res, requestQuery){

var param = parseQuery(requestQuery);

console.log('disconnection at port ' + param['port']);

var user = findUser(param['port']);

if(user != undefined){

user.op = 'endStreaming';

var json = JSON.stringify(user);

for(var i=0; i < clients.length; i++){

clients[i].send(json);

}

var idx = (parseInt(param['port']) - basePort) / 100;

streamers[idx] = undefined;

processes[idx].stdin.write('q');

processes[idx] = null;

exec('DEL /Q ' + mvSourceRoute\_BSlash + '\\' + user.userID + '\_\*.ts ' + mvSourceRoute\_BSlash + '\\' + user.userID + '\_\*.m3u8');

}

}

-Client

1. getDevices
   * 1. DirectX에서 제공하는 DSHOW를 이용하여 현재 사용할 수 있는 Record device들을 탐색한다.

void getDevices() {

char dummy[1024];

//List enable devices

string deviceList("ffmpeg -list\_devices true -f dshow -i dummy 2>&1");

FILE\* deviceList\_command = \_popen(deviceList.c\_str(), "r");

if (!deviceList\_command) throw runtime\_error("popen() failed!");

while (fgets(dummy, 1024, deviceList\_command) != 0) {

if (dummy[0] != '[')

continue;

DeviceCategory category = parseDeviceCategory(dummy);

fgets(dummy, 1024, deviceList\_command);

switch (category) {

case DeviceCategory::VIDEO:

//Friendly Name

parseDeviceName(devices\_friendlyName[0][videoDeviceNum], dummy);

//Alternative Name

fgets(dummy, 1024, deviceList\_command);

parseDeviceName(devices\_alternativeName[0][videoDeviceNum], dummy);

//cout << "[" << deviceNum << "]" << devices[0][deviceNum] << " - Video Device" << endl;

videoDeviceNum++;

break;

case DeviceCategory::AUDIO:

//Friendly Name

parseDeviceName(devices\_friendlyName[1][audioDeviceNum], dummy);

//Alternative Name

fgets(dummy, 1024, deviceList\_command);

parseDeviceName(devices\_alternativeName[1][audioDeviceNum], dummy);

//cout << "[" << deviceNum << "]" << devices[1][deviceNum] << " - Audio Device" << endl;

audioDeviceNum++;

break;

}

}

}

1. httpRequest
   * 1. C++에서는 HTTP Connection Library가 존재하지 않기 때문에 WinSock을 이용하여 기존 TCP 연결에 HTTP header를 추가하여 HTTP Request를 구현하고, parsing을 통해 HTTP Response를 받아올 수 있도록 하였다.

unsigned short httpRequest(string hostIP, string method, map<string, string> parameters, bool hasReturn) {

WSADATA wsaData;

SOCKET ConnectSocket = INVALID\_SOCKET;

struct addrinfo \* result = NULL, \*ptr = NULL, hints;

string sendbuf = "GET http://" + hostIP + ":15000/" + method + "?";

map<string, string>::iterator iter = parameters.begin();

while(true) {

sendbuf += iter->first + "=" + iter->second;

iter++;

if (iter != parameters.end())

sendbuf += "&";

else

break;

}

sendbuf += " HTTP/1.0\r\n\r\n";

cout << sendbuf << endl;

char recvbuf[DEFAULT\_BUFLEN];

int iResult;

int recvbuflen = DEFAULT\_BUFLEN;

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

printf("WSAStartup failed with error: %d\n", iResult);

return -1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

iResult = getaddrinfo(hostIP.data(), "15000", &hints, &result);

if (iResult != 0)

{

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

exit(1);

}

// Attempt to connect to an address until one succeeds

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

// Create a SOCKET for connecting to server

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype,

ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

WSACleanup();

exit(1);

}

// Connect to server.

iResult = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (iResult == SOCKET\_ERROR)

{

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

printf("The server is down... did not connect");

}

}

// no longer need address info for server

freeaddrinfo(result);

// if connection failed

if (ConnectSocket == INVALID\_SOCKET)

{

printf("Unable to connect to server!\n");

WSACleanup();

exit(1);

}

iResult = send(ConnectSocket, sendbuf.data(), (int)strlen(sendbuf.data()), 0);

if (iResult == SOCKET\_ERROR) {

printf("send failed with error: %d\n", WSAGetLastError());

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

if (hasReturn) {

iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);

if (iResult > 0) {

std::string str(recvbuf);

int dataStart = str.find('{') + 1;

int dataEnd = str.find('}');

str = str.substr(dataStart, dataEnd - dataStart);

map<string, string> params = parseParam(str, ',', ':');

iResult = atoi(params.at("\"port\"").data());

}

else if (iResult == 0) {

printf("Connection closed\n");

}

else {

printf("recv failed with error: %d\n", WSAGetLastError());

}

}

// cleanup

closesocket(ConnectSocket);

WSACleanup();

return iResult;

}

1. **개선점**
2. GUI를 통해 Streamer에게 편한 인터페이스를 제공해야할 필요성
3. 현재 3초 단위의 Segment를 2개까지 저장하여 재생하는 것으로 최소 6초의 딜레이가 존재하는데, 이를 줄이게 된다면 새 segment를 로딩하는 시간보다 재생시간이 빨라 에러가 발생할 수 있음. 적절한 시간 단위를 찾거나 네트워크 환경을 개선하는 작업이 필요.
4. Windows 환경에서 동작하는 WinSock과 WinAPI를 이용하였기 때문에 다른 운영체제에 대해서는 코드를 수정하여 재빌드 해야함.