XcopyProject

-1주차 스프린트 정리-

1. XcopyProject 개요

실제 존재하는 Xcom 게임의 멀티 시스템을 구현하는 것이 목표로 해당 기능을 구현하면서 실시간 멀티에 대한 이해 및 DB가 서버에 존재했을 때의 처리 방법 등을 중점을 두고 계획한 프로젝트이며 또한 unity 3d의 숙련도 향상을 목표로 하고 있다.

2. 1주차 스프린트 목표

-서버 구축(성공)

-Database 구축(성공)

-외부 리소스 로딩 기능(제외)

-채팅 서버 구축(성공)

3. 1주차 스프린트 개요

멀티 시스템을 구축하기 전 유니티와 서버 간의 데이터 전송 방법 및 서버의 구축이 필요하다 생각되었다. 따라서 1주차의 목표는 유니티와 서버의 관계에 집중하게 되었고, 이에 request과 response로 이루어지는 로그인 시스템. WebSocket을 통한 실시간 채팅을 목표로 하였다. 거기에 유저의 정보를 Database를 통하여 관리하기로 하였다.

4. 사용한 개발 도구

5. 프로젝트 관련 링크

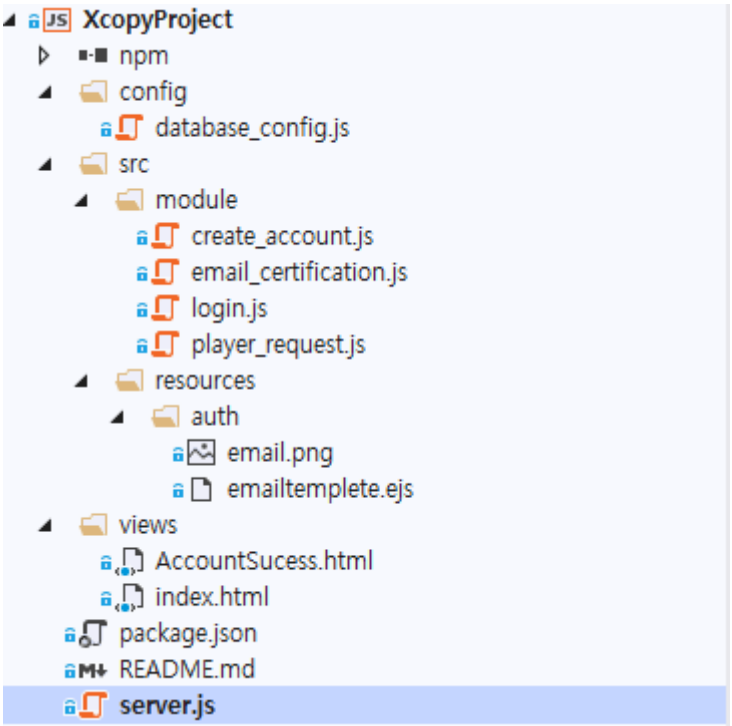
Git: https://github.com/irispon/XcomCopyProject

YouTube : https://youtu.be/k_YKfKhirEI

6. 1주차 프로젝트 설명

1. 서버

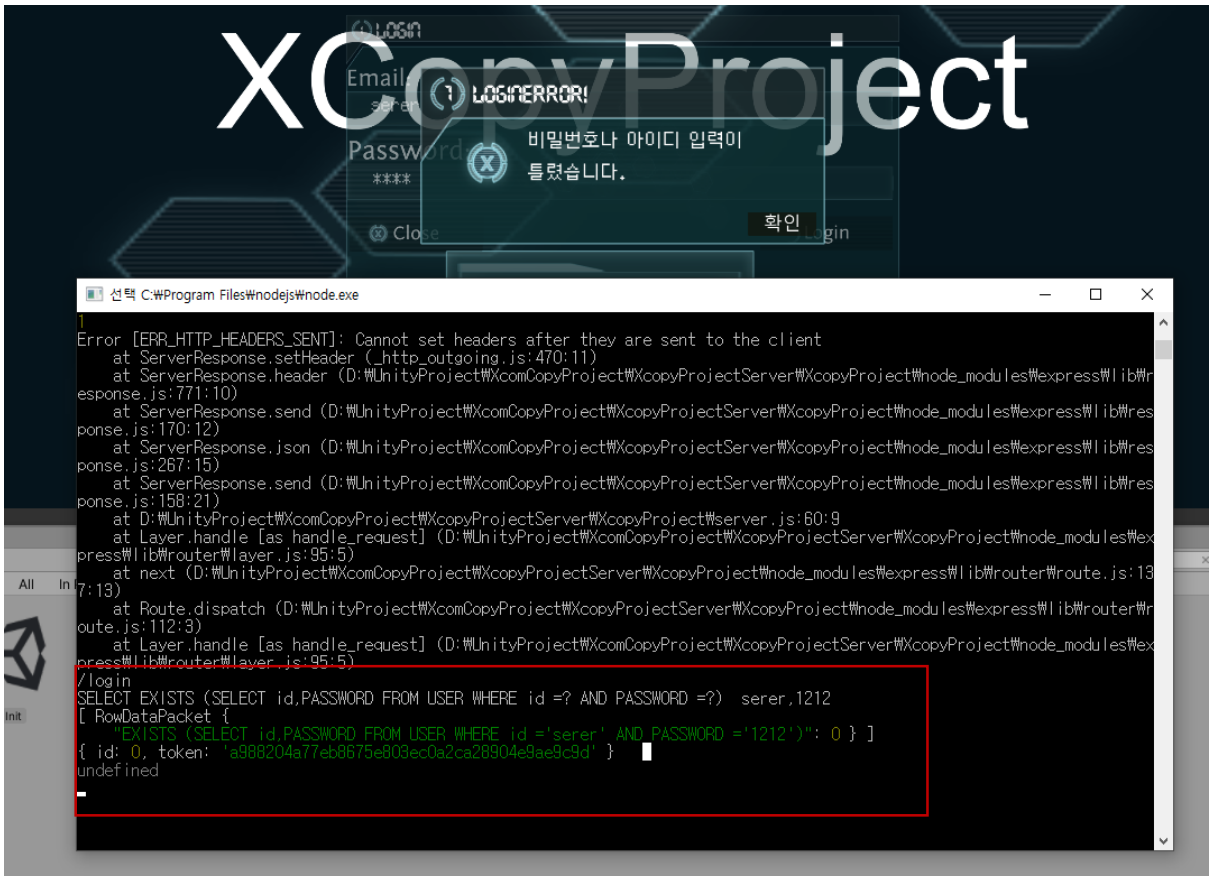
서버는 Node.js를 활용하였다. Node.js를 이용하게 된 이유로는 간단한 서버 구축, 다양하고 쉽게 적용할 수 있는 모듈들을 이유로 채택하게 되었다.



1주차 서버는 실시간 통신을 위한 WebSocket, request와 response를 위한 express 모듈을 중점으로 구현하였고. 여기에 DB에 접근하기 위하여 mysql 관련 모듈, 인증 시스템 및 보안성 향상을 위해 nodemailer와 crypto를 모듈을 사용하였다.

2. 로그인 시스템

로그인 시스템은 Client에서 서버를 향한 Post요청을 통해 이루어지게 된다. 포스트 요청은 Task와 Await를 통해 비동기 처리가 이루어지고 서버는 Post에 담긴 데이터를 Query로 바꿔 데이터 베이스에 접근한 뒤 결과를 Client에게 반환하게 된다. 이후 클라이언트는 결과에 따라 Callback을 받고 결과에 따른 처리를 하게 된다.



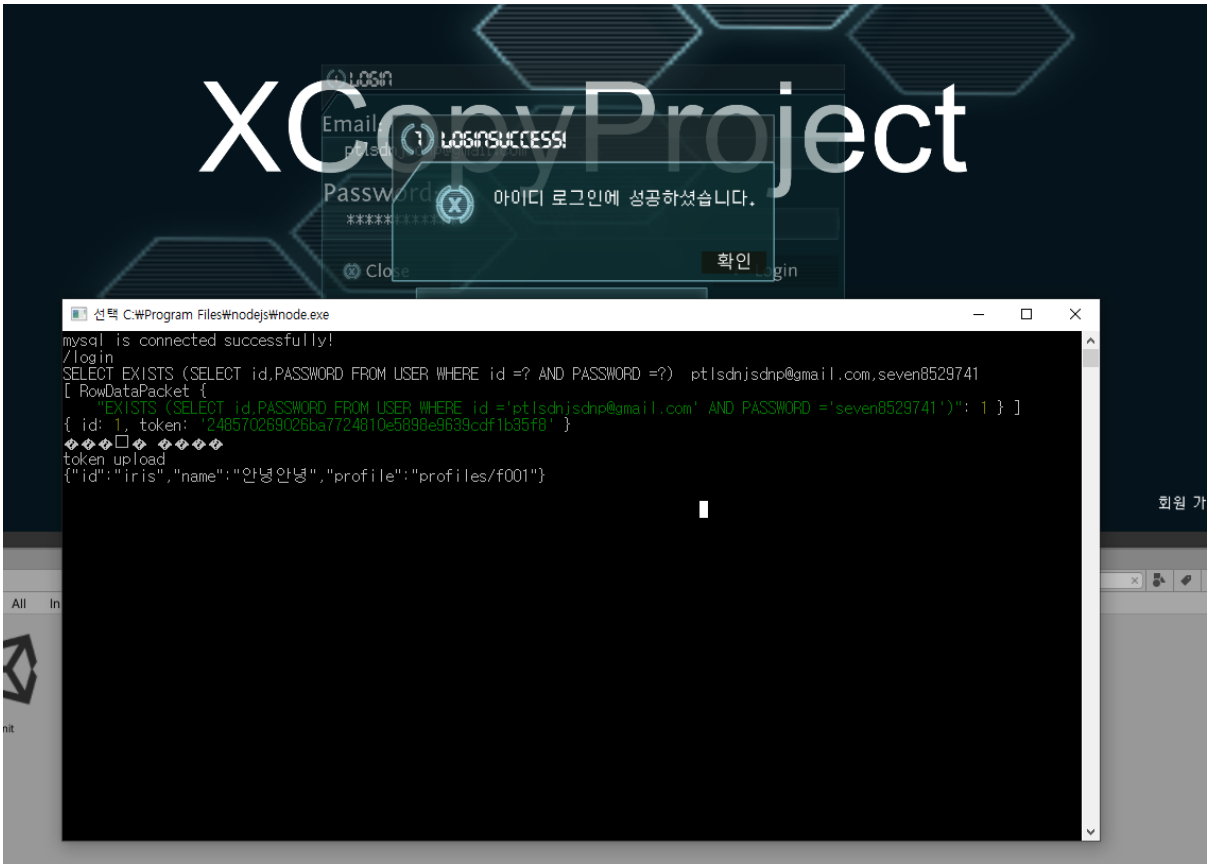
[그림 설명] Login 요청에 따른 Query문과 결과. id가 존재하지 않으면 0을 반환한다.

```
// Start is called before the first frame update
참조 5개
public static async Task<string> PostRequest(string uri, WWWForm form,ICallBack callBack =null)
{


    Debug.Log(uri);


    UnityWebRequest uwr = UnityWebRequest.Post(uri, form);

    uwr.SendWebRequest();

    while (!uwr.isDone)
    {
        await Task.Delay(100);
        //나중에 변경해야될 코드.(로그인이 안되는 경우)
    }



    if (uwr.isNetworkError)
    {
        Debug.Log("Error While Sending: " + uwr.error);
        if (callBack != null)
        callBack.Back(PostEvent.error.ToString());//나중에 enum으로 바꾸자
        return RequestMessage.serverError.ToString();
    }
    else
    {
        Debug.Log("Received: " + uwr.downloadHandler.text);
        if(callBack!=null)
        callBack.Back(PostEvent.success.ToString());
        return uwr.downloadHandler.text;

    }



}
```

[그림 설명]Client에서 Post 요청은 Task를 통해 비동기로 처리된다.

```
router.post('/login',function (req, res) {

    console.log(req.url);



   // var sql = 'SELECT id,PASSWORD FROM USER WHERE id= ' + id + ' AND PASSWORD = ' +password
    var sql = 'SELECT EXISTS (SELECT id,PASSWORD FROM USER WHERE id =? AND PASSWORD =?)';
    var par = [req.body.id, req.body.password];
    console.log(sql+"   " +par);

    try {
        connection.query(sql, par, (error, rows, fields) => {
            if (error) throw error;
            console.log(rows);
            var string = JSON.stringify(rows);
            var json = JSON.parse(string);
            var firstKey = Object.keys(json[0])[0];
            const token = crypto.randomBytes(20).toString('hex');

            var request = { id: json[0][firstKey], token: token };
```

[그림 설명]Client에서 요청이 들어오면 해당 서버 코드를 통해 데이터베이스를 조회한다..



로그인에 성공하게 된다면 클라이언트는 토큰을 발급받게 된다. 해당 토큰은 로그인 시 바뀌는 고유 값으로 해당 토큰 값과 아이디가 DB와 일치해야 유저의 정보에 접근이 가능하게 된다. 토큰을 발급받은 클라이언트는 해당 토큰 값과 아이디로 유저의 정보를 받아오게 되고 대기화면으로 넘어가게 된다.

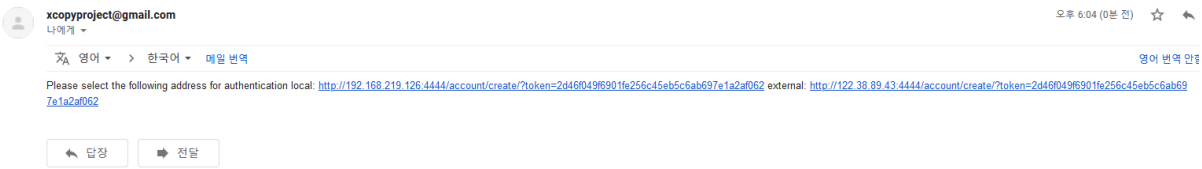
3. 회원가입 시스템

회원가입을 하게 된다면 간단한 아이디 중복 검사를 실행한다. 그 후 고유한 토큰을 생성하고 아이디와 패스워드를 임시적인 Database 공간에 저장한다.

id	password	token
qndrhl7@gmail.com	12341234	6072548ef0e69e674e79795f14a23ec65a6b8593
(NULL)	*(NULL)*	*(NULL)*

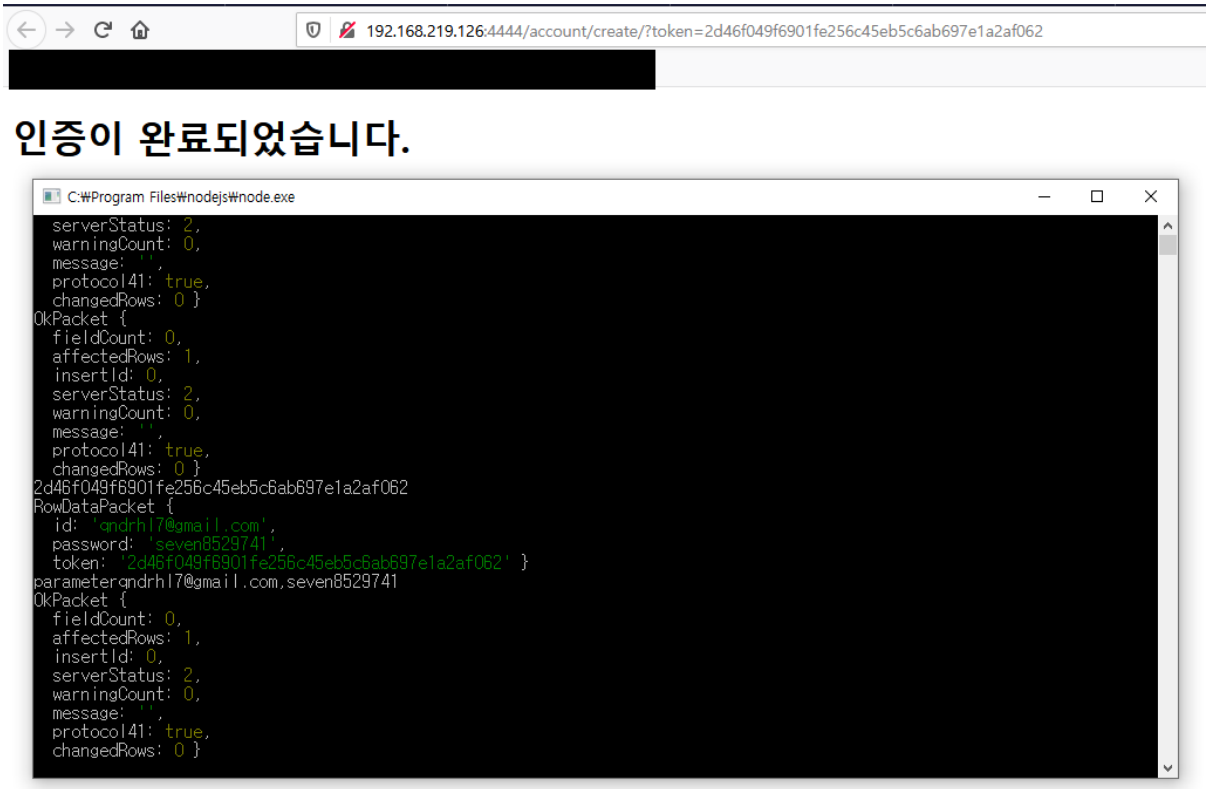
[그림 설명]임시적으로 저장된 데이터

이후 토큰이 담긴 링크를 NodeEmailer를 통해 유저의 이메일로 보내게 되고

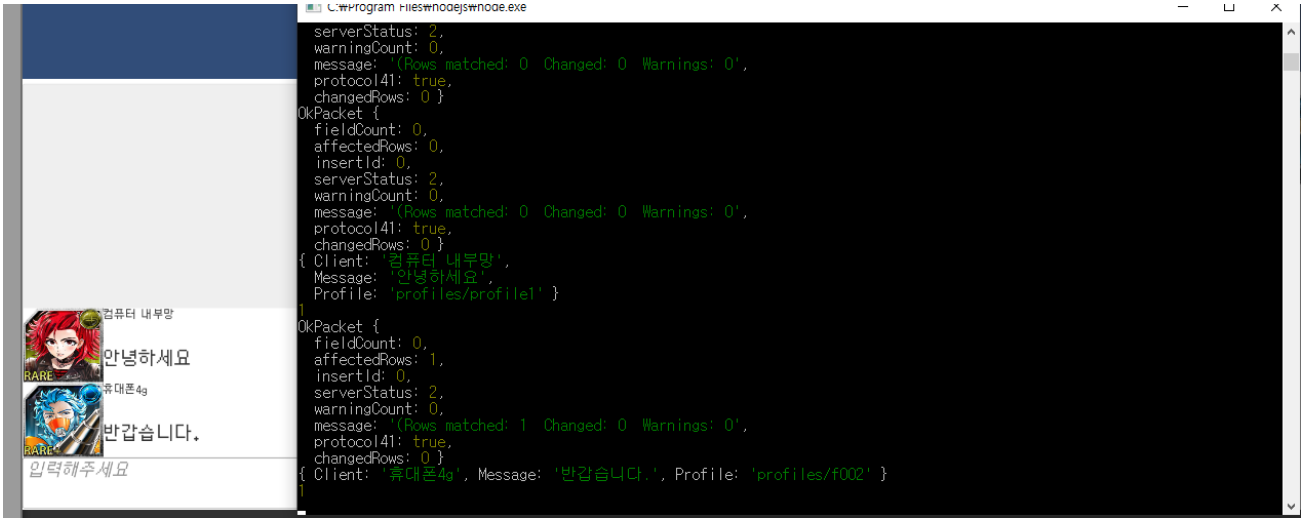


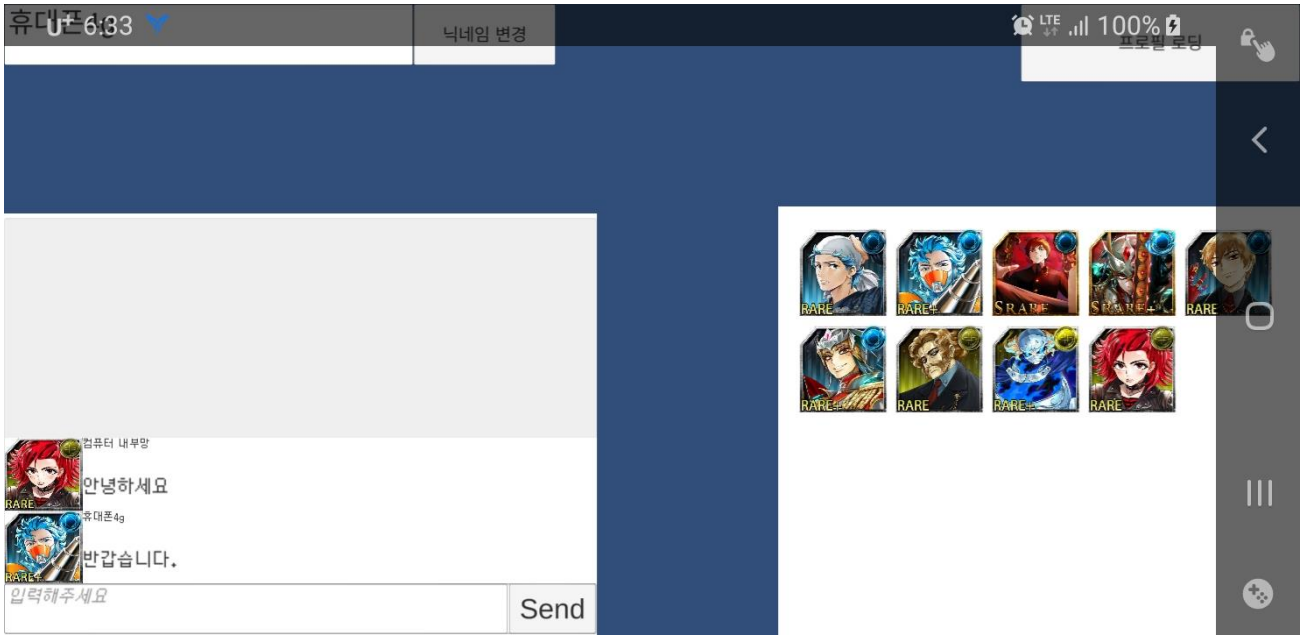
[그림 설명]공유기 루프백 지원이 되지 않아 내부 링크도 같이 첨부되었다.

이후 이메일 링크를 클릭하게 되면 Get을 통해 해당 Token값을 읽어오게 되고. 해당 토큰 값과 일치되는 임시 데이터를 유저 데이터를 저장하는 공간으로 Insert되고 임시 데이터는 지워지게 된다.



4. 채팅 시스템





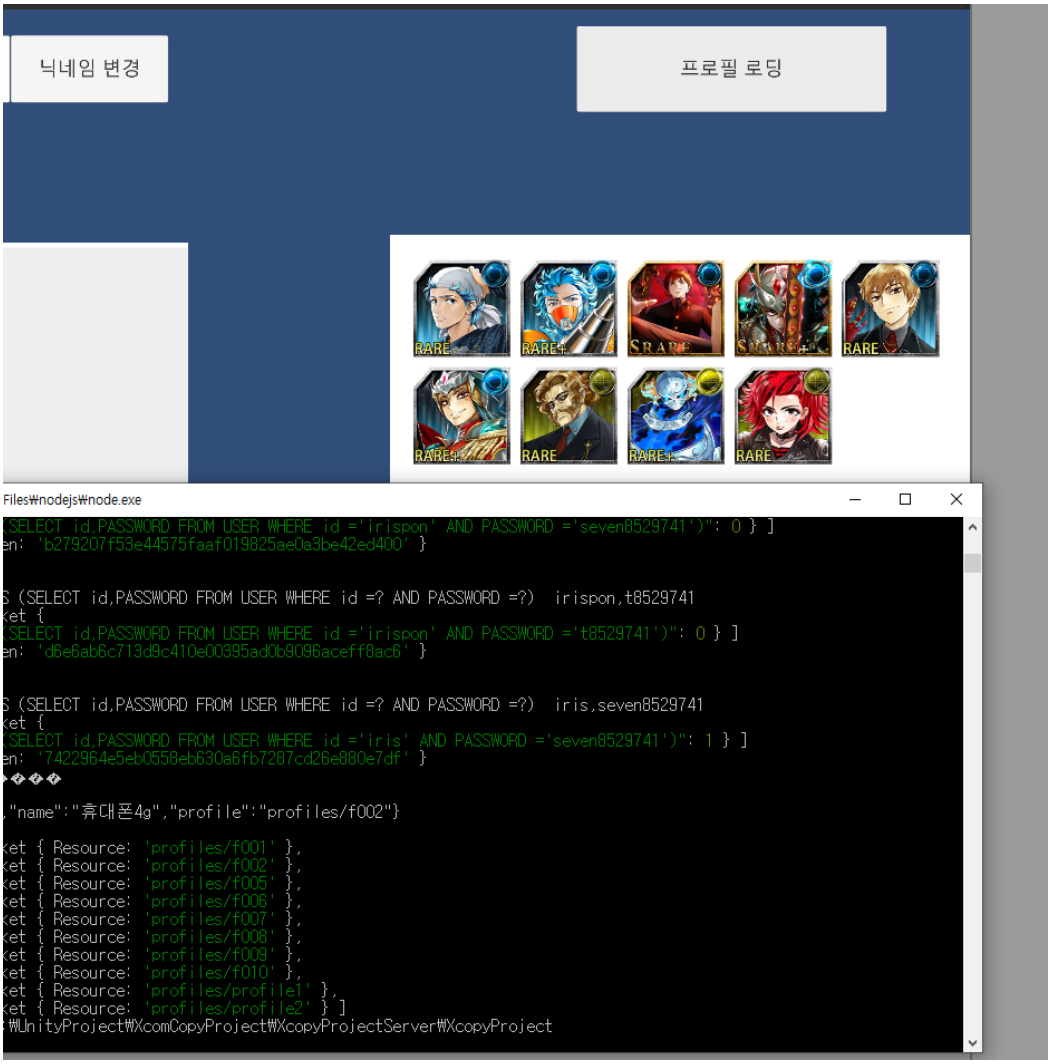
채팅 시스템은 웹 소켓 기술을 활용하였다. 채팅 시스템을 만든 목적은 웹 소켓을 통한 실시간 채팅을 먼저 구현해보고 이를 응용하여 멀티 플레이 게임을 만들어보고자 간단한 채팅 시스템을 먼저 구현하였다.

```
Socket chat;
ChatManager manager;
Action<string> messageTarget;
PlayerCache playerInfo;
// Start is called before the first frame update
Unity 메시지 | 참조 0개
void Start()
{
    messageTarget = s => ReciveMessage(s);
    manager = ChatManager.GetInstance();
    inputText = manager.inputText;
    chat = Socket.Connect(ServerHelper.SERVERPATH + "/"+SocketEvent.chat.ToString());
    playerInfo = PlayerCache.GetInstance();
    chat.On("connect", () => {
        Debug.Log("채팅 서버 접속"+ chat.IsConnected);
        chat.On(SocketEvent.chat.ToString(), messageTarget);

    });
```

[그림 설명] 클라이언트의 채팅 접속 코드

```
// socket.io 스타트
var io = require('socket.io')(server);

// 채팅방//
var chat = io
    .of('/chat')
    .on('connection', function (socket) {
      // chat.emit('a message', { everyone: 'in', '/chat': 'will get' });
    //   socket.emit('a message', { that: 'only', '/chat': 'will get' });
        console.log("__dirname: " + __dirname);

        socket.on('chat', function (data) {

            try {

                console.debug(data);
                console.debug(chat.clients.length);
                chat.emit('chat', data);

            } catch (e) {
                console.log(e);
            }



        });
        socket.on("disconnect", function () {
         socket.disconnect(0);
        });

    });
// 채팅방//
```

[그림 설명] 서버의 채팅 응답 코드



또한 이미지와 관련된 데이터를 DB에 넣어두어 데이터와 클라이언트를 분리시키고자 하였다. 클라이언트는 처음 비동기로 프로필 사진 관련 데이터를 요청하게 되고 버튼을 누르게 되면 동기처리를 통해 플레이어가 선택 가능한 프로필이 나오게 된다. 추후 외부 리소스 로딩을 적용하게 된다면 이를 응용하여 MVC 패턴을 일부 적용할 수 있을 것이라 전망된다.

```
player_request.requests = function (app,connection) {

    app.use('/', router);

    router.post('/getinfo', function (req, res) {

        try {
            var sql = 'SELECT u.id,u.name,u.profile FROM USER AS u JOIN auth au ON u.id = au.id WHERE  au.id = ? AND au.token = ?';
            var par = [req.body.id, req.body.token];
            console.log(par);
            connection.query(sql, par, (error, rows, fields) => {
                console.log(sql);
                var string = JSON.stringify(rows[0]);
                console.log(string);
                res.send(string);
                res.end();

            });
        } catch (e) {

        }


    });
    router.post('/setinfo', function (req, res) {

        try {
            var sql = 'UPDATE USER u INNER JOIN auth au ON u.id = au.id SET u.name = ?,u.profile =? WHERE u.id = ? AND au.token = ?';
            var par = [req.body.name, req.body.profile, req.body.id, req.body.token];

            try {
                connection.query(sql, par, (error, rows, fields) => {
                    if (error)
                        res.send('error');
                    console.log(rows);
                    res.end();

                });


            } catch (e) {

            }
        } catch (e) {


        }


    });
```

또한 플레이어의 요청 (닉네임 변경, 프로필 사진 변경)은 사용자가 로그인 시에 발급받은 임시 토큰과 id 확인을 통해 처리하게되게 된다. 이를 통해 중복 접속을 방지할 수 있고 password 대신 token을 통해 데이터를 조회하게 되므로 약간의 보안을 고려하여 토큰을 확인하도록 하였다.

7. 추후 고려할 점

1. 외부 리소스 로딩
2. 구글 로그인
3. 서버 보안