거제시농업개발원 디지털체험콘텐츠 관리운영시스템 개발 결과 보고

2022. 11.

(주)프리모엠

Ⅰ 사업개요

□ 사 업 명 : 거제시농업개발원 디지털체험콘텐츠 관리운영시스템 개발

□ 사업목적

가. 4차 산업혁명 시대를 선도하는 '창의인재' 양성을 위해 정보통신기술(ICT)을 활용한 새로운 형태의 디지털농업체험콘텐츠 개발 필요

나. 코로나19 이후 체험 학습 콘텐츠가 디지털로 전환되고 있으며, 이를 관리하는 시스템이 필요

다. 가족 단위 개인 및 학교 교실 단위 단체 관광객을 대상으로 한 최적화 된 디지털체험콘텐츠 관리 시스템 개발 필요

□ 과업진행기간 : 2022년 12월 9일

□ 과업내용 : 거제시농업개발원 디지털체험콘텐츠 관리운영시스템 개발

- 거제시농업개발원 일대를 기반으로 제작한 디지털체험콘텐츠를 관리
- 가족 단위 및 학교 단위 디지털체험콘텐츠 관리하는 시스템 개발
- 디지털체험콘텐츠 연동되는 관리 시스템으로 체험객의 결과 분석
- 디지털체험콘텐츠별 체험객의 체험 시간 및 체험 서비스 품질에 대한 데이터 분석

Ⅱ 사업 내용

□ 사업내용

○ 거제시 농업개발원 디지털 체험 콘텐츠 관리 운영 시스템 개발

□ 서비스 운영을 위한 시스템 구성

○ 사용자 요구 분석을 통한 그룹별 로그인 시스템 관리
- 서비스 운영자 : 전체 서비스를 운영 관리할 수 있는 계정으로 서비스 운영 할 수 있도록 구성
- 그룹 로그인 : 농업 기술센터 체험에서 단체 체험객(학교 체험학습)을 위한 그룹 단위 사용자
- 개별 로그인 : 농업 기술센터 체험하고자 하는 개별 사용자

○ 전체 서비스 구성
- 사용자들은 Web서비스를 통해 접속 및 콘텐츠의 패킷 처리 작업 진행
- 보안을 위해 사용자는 Web서버를 통해서 제공되는 콘텐츠 진행
- 웹 서비스를 위한 웹 서버 구성
- 사용자 인증을 위한 사용자 인증 서버
- 사용자 정보를 저장관리하기 위한 DB 사용
- 콘텐츠의 정보를 저장관리하기 위한 DB 사용

□ 서비스 운영을 위한 운영 프로세스 기획

○ 서비스 운영자의 서비스 관리를 편리하게 관리
- 사용자가 콘텐츠 소비 진행에 따른 운영자가 고려해야 하는 내용들을 정리
- 사용자의 콘텐츠의 진행 상태를 서버에 저장하고, 사용자의 다시하기 상태를 고려하여 진행 할 수 있도록 구성

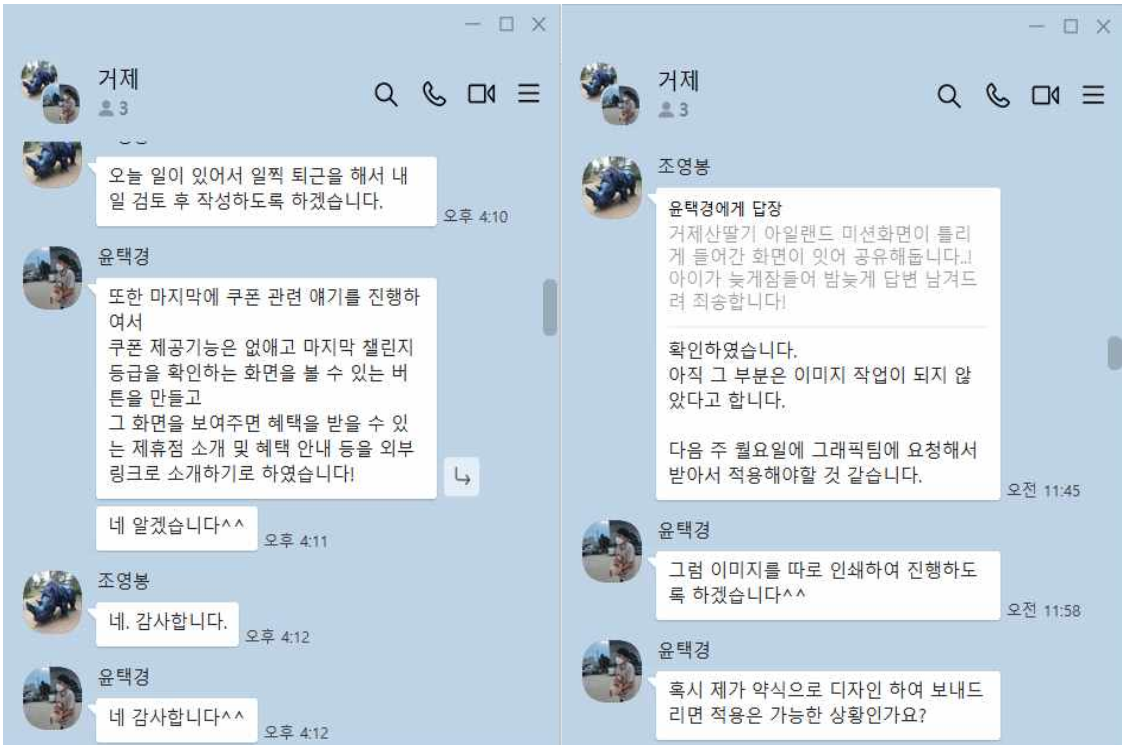
□ 클라이언트 통신을 위한 API 작성

○ 클라이언트 앱과 콘텐츠 진행을 관리하기 위한 API 작성

- 진행하고 있는 콘텐츠의 번호 및 콘텐츠의 진행상태를 저장하는 API 작성
- 사용자 클라이언트에서 요청하는 API를 처리하여 DB에 저장하고 관리하는 형식으로 제작

○ 클라이언트 프로그램에서 필요한 내용 정리를 위한 비대면 미팅

- 기존에 사용자가 콘텐츠 미션을 수행하고 수행한 결과에 따라 쿠폰을 지급하던 방식에서, 결과 화면을 저장하여 혜택을 지급받을 수 있는 형태로 변경



□ 웹 서버 보안을 위한 시스템 구성

○ 사용자 인증

- 웹 서버에 접속하기 위해서는 이메일을 통한 가입 절차 필요
- 가입을 진행한 이메일을 통하여 웹서버에 로그인
- 가입 완료된 계정은 운영팀에 관리자 권한을 요청하여 관리자 권한을 부여받아야 웹 서버 기능을 모두 활용할 수 있도록 제작

○ 데이터 Storage 접근 권한

- 웹 서버에 가입하여 로그인한 사용자만 데이터 읽기 권한을 가짐
- 웹 서버에 가입하여 로그인 후, 운영팀으로부터 관리자 권한을 부여받은 사용자만 데이터 쓰기 및 편집 권한을 가짐
- 5mb 이하의 크기를 가진 파일만 업로드 가능
- 업로드 가능한 파일은 이미지 형태의 포맷만 지원

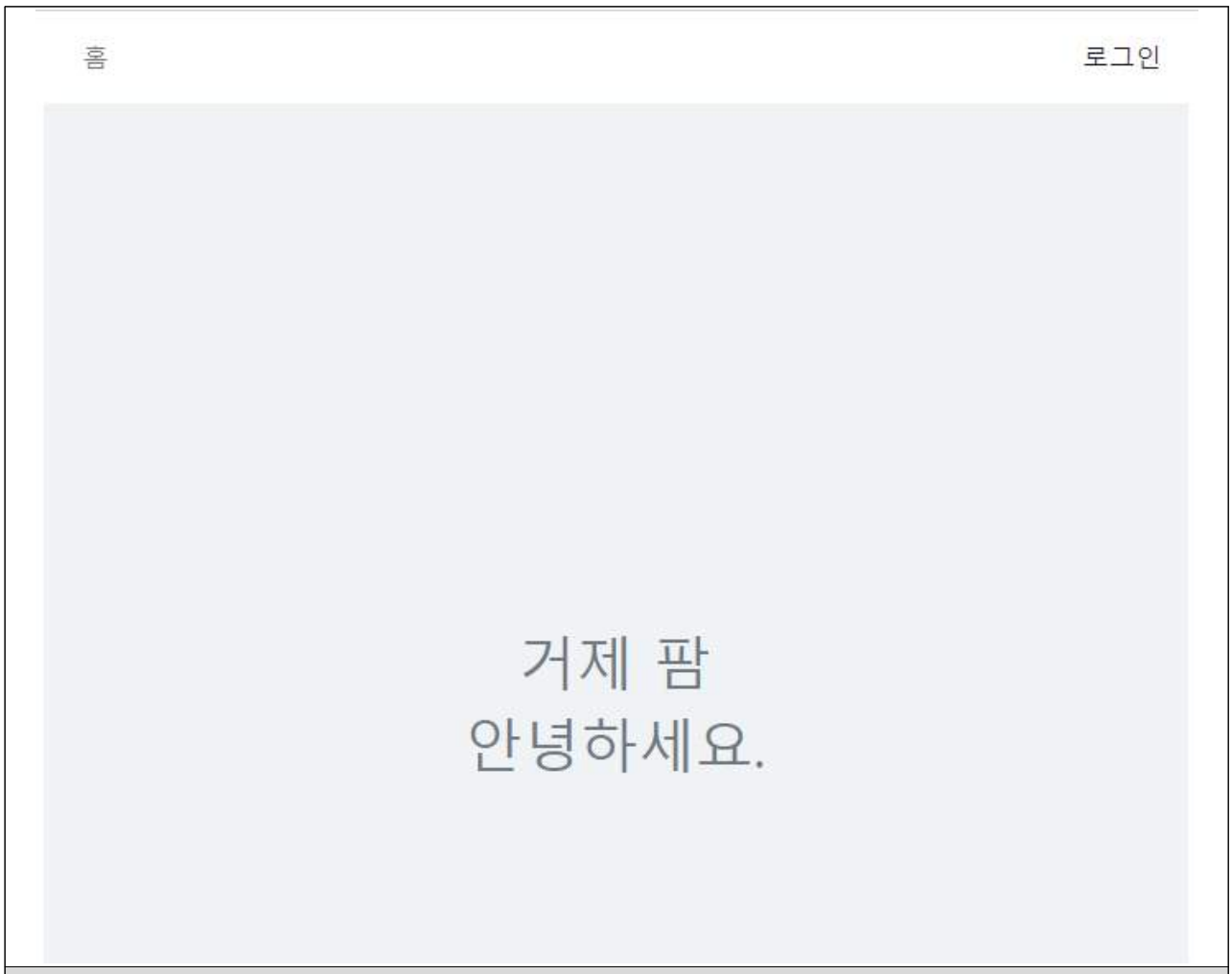
○ Firestore DB에 대한 접근 권한

- 웹 서버에 가입하여 로그인 후, 운영팀으로부터 관리자 권한을 부여받은 사용자만 모든 항목에 대한 쓰기 및 편집 권한을 가짐
- QRCode
  - 읽기: 모든 사용자가 접근 가능
  - 쓰기: 멀티 QRCode는 모든 사용자가 접근 가능, 싱글 QRCode는 폰 번호가 공백일 경우에만 가능
- 콘텐츠
  - 읽기: 모든 사용자가 접근 가능
  - 쓰기: 모든 사용자에 대한 접근 거절
- 플레이어
  - 읽기: 멀티 QRCode는 QRCode만 일치하면 접근 가능, 싱글 QRCode는 QRCode와 폰 번호가 전부 일치하면 가능
  - 쓰기: 멀티 QRCode는 QRCode만 일치하면 접근 가능, 싱글 QRCode는 QRCode와 폰 번호가 전부 일치하면 가능

□ 서비스 운영을 위한 시스템 구성

○ 사용자 요구 분석을 통한 그룹별 로그인 시스템 관리
- 진행하고 있는 콘텐츠의 번호 및 콘텐츠의 진행상태를 저장하는 API 작성
- 사용자 클라이언트에서 요청하는 API를 처리하여 Firestore DB에 저장하고 관리하는 형식으로 제작

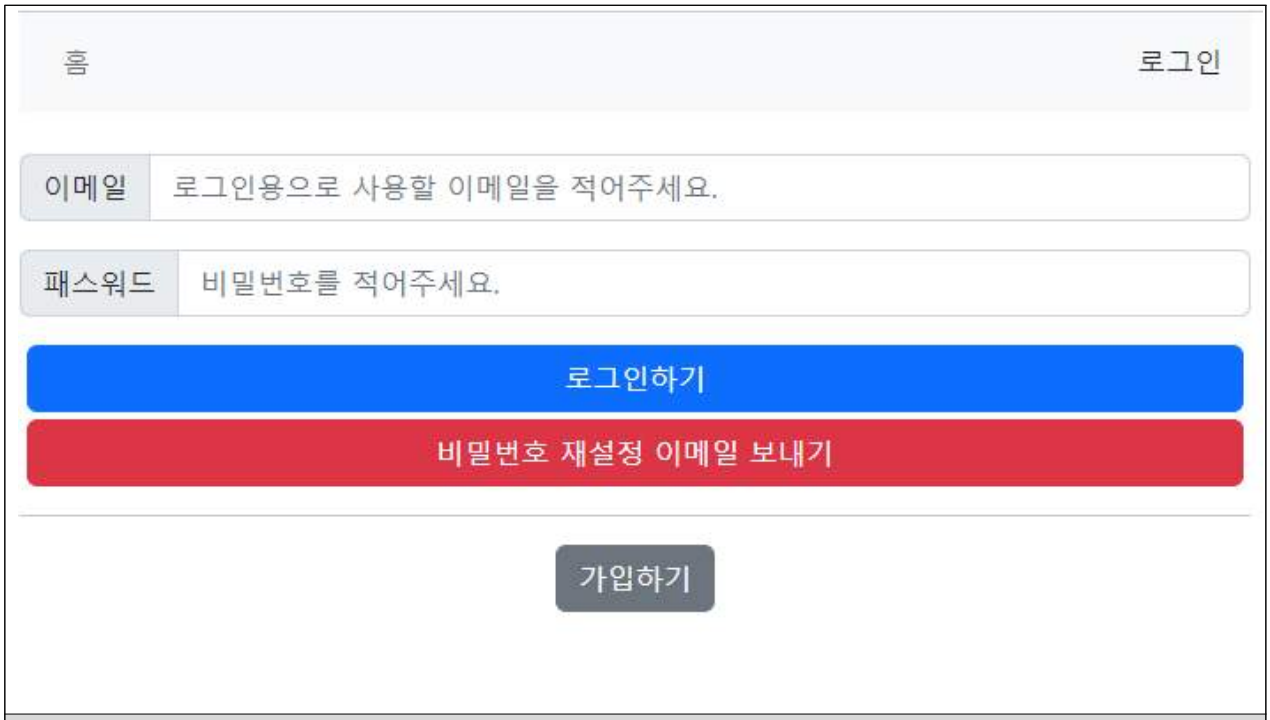
○ 사용자 데이터 관리를 위한 웹 관리자 시스템 구축
- 사용자 클라이언트에서 요청하는 API를 처리하여 Firestore DB에 저장하고 관리하는 형식으로 제작
- 플레이어가 클라이언트에서 입력한 플레이어 데이터를 DB에 저장
- 웹 관리자 권한을 가진 계정으로 로그인하여 DB를 관리할 수 있도록 제작
- 웹에서 콘텐츠 데이터를 수정하여 업로드하면 클라이언트에서 이를 업데이트하여 적용할 수 있는 기능 구현



웹 관리자 초기 화면

○ 웹 계정 로그인 기능

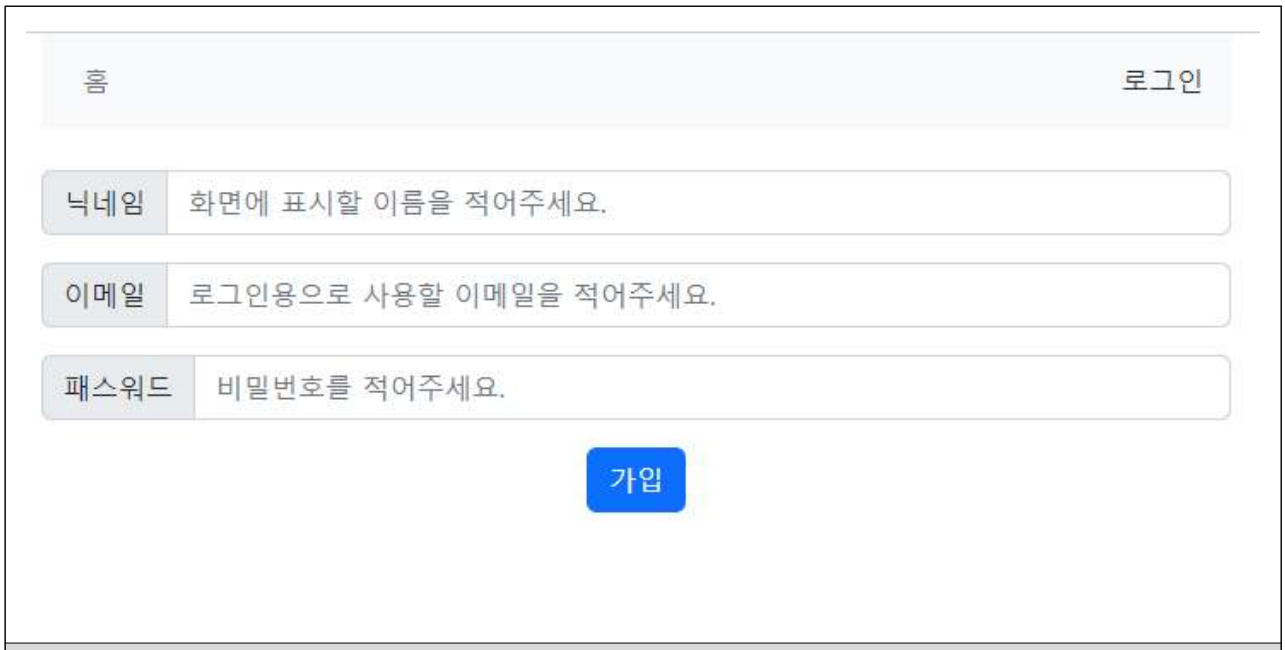
- 웹 기능을 사용하기 위해 필요한 웹 로그인 시스템 구축
- 가입 시 입력한 이메일 데이터와 패스워드 데이터 DB에 저장
- DB에 저장된 데이터와 사용자가 입력한 이메일과 패스워드를 확인하여 로그인 기능을 수행
- 가입 시 입력한 이메일에 비밀 번호 재설정을 진행할 수 있는 코드를 전송



웹 관리 시스템 로그인 화면

○ 웹 신규 계정 가입 기능

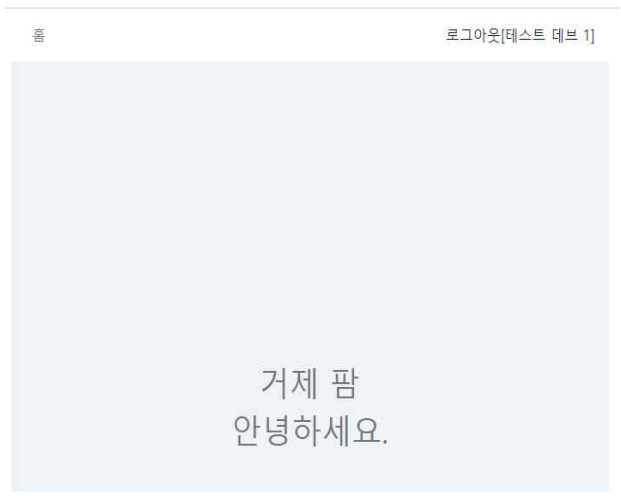
- 웹 로그인을 진행하기 위한 신규 가입 기능 API 작성
- 웹 사용자는 닉네임, 이메일, 패스워드를 입력하여 신규 가입 진행
- 사용자가 가입 시 입력한 닉네임, 이메일, 패스워드를 DB에 저장



웹 관리 시스템 신규 가입 화면

○ 보안을 위한 일반 계정 및 관리자 계정 분리

- 웹 관리 기능에 대한 추가적인 보안을 위해 신규 가입 계정에 대한 기능 제한을 적용하는 API 작성
- DB에 등록된 계정을 일반 계정과 관리자 계정으로 분리하여 관리
- 웹 관리자가 일반 로그인 계정에 관리자 권한을 부여하는 API 작성



일반 로그인 상태

관리자 로그인 상태

□ 클라이언트와 연계되는 서버 시스템 구성

○ Firestore 서비스를 이용한 서버 데이터 요청 기능

- 클라이언트에서 서버에 대한 데이터 접근을 요청하면, 서버 소스 코드는 클라이언트가 Firestore 서비스에서 제공하는 서버에 접속할 수 있도록 하는 형태로 클라이언트와 서버 간의 데이터 전송을 처리하도록 제작

```
// QRCode가 있는 경우에만 동작
match /players/{qrcodePhoneNumber} {
  function isQRCode(qrcode) {
    return exists(/databases/$(database)/documents/qrcodes/$(qrcode));
  }

  function isPhoneNumber(qrcode, phoneNumber) {
    return qrcode == "20220001"
      || get(/databases/$(database)/documents/qrcodes/$(qrcode)).data.PhoneNumber
  }

  function checkedQrcode(qrcodePhoneNumber) {
    let temp = qrcodePhoneNumber.split('-');
    return isQRCode(temp[0]) && isPhoneNumber(temp[0], temp[1]);
  }

  allow read, write: if checkedQrcode(qrcodePhoneNumber);
}

// qrcdode가 멀티이거나 PhoneNumber가 비어있는 경우에만 쓰기가 가능.
match /qrcodes/{qrcode} {
  allow read: if true;
  allow write: if qrcode == "20220001"
               || resource.data.PhoneNumber == "";
}


// 인증 받은 유저만 읽기/쓰기 가능
match  /users/{uid} {
  allow read, write: if request.auth != null;
}
```

클라이언트로부터 받은 데이터 접근 요청 사항을
Firestore DB 서버에 접속하여 이용할 수 있도록 하는 서버 소스 코드

○ QRCode 인증 및 저장

- 사용자가 콘텐츠 진행 중 QRCode 인증 기능을 사용하면, 사용자 클라이언트에서 해당 QRCode 데이터를 서버에 전송하는 API 작성
- 서버는 클라이언트로부터 전송된 데이터를 DB에 저장
- 웹 관리자가 DB에 저장된 데이터를 웹에서 확인 가능
- 저장된 데이터를 일정 QRCode 범위 내에서 필터링하여 확인 가능
- DB 데이터를 Excel 파일로 저장하여 내보낼 수 있는 기능 구현

- 클라이언트 입력 관련

사용자가 클라이언트에서 QR 코드 인증

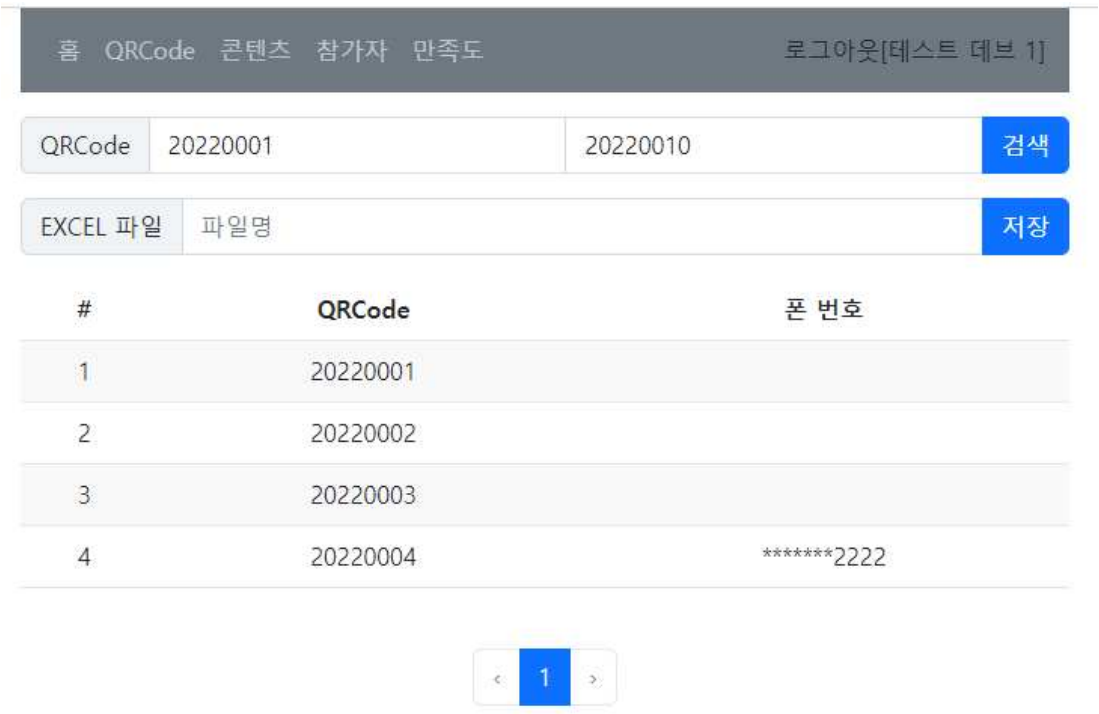
```
public async UniTask<(bool, QRData)> ReadQR(string doc)
{
    Debug.Log("FireStore에서 QR 데이터를 가져옵니다.");
    return await ReadAsync<QRData>("qrcodes", doc);
}
```

해당 QR이 올바른

```
public async UniTask<(bool, T)> ReadAsync<T>(string collection, string doc)
{
    var docRef = _db.Collection(collection).Document(doc);
    DocumentSnapshot snapshot = await docRef.GetSnapshotAsync();
    if (snapshot.Exists)
    {
        return (true, snapshot.ConvertTo<T>());
    }
    else
    {
        Debug.Log("Document " + snapshot.Id + " does not exist!");
        return (false, default(T));
    }
}
```

인증 데이터를 서버에 전송하는 소스 코드

- DB저장 내용



DB에 저장된 QR 인증 데이터를 웹에서 관리

○ 플레이어 현황 데이터 전송 및 저장

- 사용자가 콘텐츠 진행 중 플레이어 현황 제출하기 기능을 사용하면, 사용자 클라이언트에서 해당 플레이어 현황 데이터를 서버에 전송하는 API 작성
- 서버는 클라이언트로부터 전송된 데이터를 DB에 저장
- 웹 관리자가 DB에 저장된 데이터를 웹에서 확인 가능
- 저장된 데이터를 일정 QRCode 범위 내에서 필터링하여 확인 가능
- 데이터 통계를 웹에서 차트 형태로 표시하는 기능 적용
- DB 데이터를 Excel 파일로 저장하여 내보낼 수 있는 기능 구현

- 클라이언트 입력 관련

<table>
<tr>
<td> 
</td>
<td>

```
public async UniTask<(bool, PlayerData)> ReadPlayer(string doc)
{
    Debug.Log("FireStore에서 Player 데이터를 가져옵니다.");
    return await ReadAsync<PlayerData>("players", doc);
}
```

기존 사용자 O

```
public async UniTask<bool> AddPlayer(string doc, PlayerData playerData)
{
    Debug.Log($"FireStore에 현재 PlayerData Document을 생성합니다. ({doc})");
    var docRef = _db.Collection("players").Document(doc);
    await docRef.SetAsync(playerData);

    DocumentSnapshot snapshot = await docRef.GetSnapshotAsync();

    return snapshot.Exists;
}
```

신규

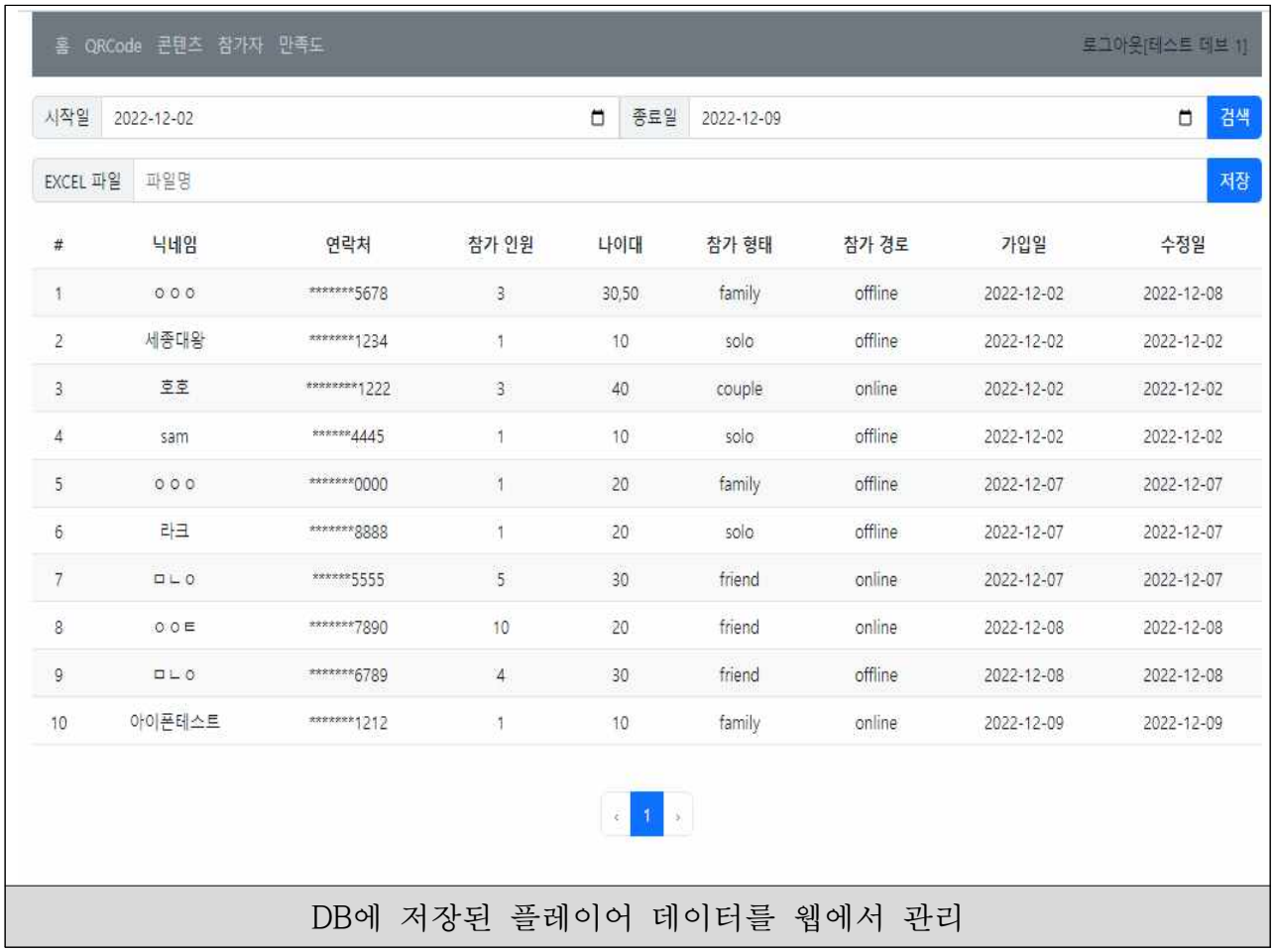
```
public async UniTask<(bool, T)> ReadAsync<T>(string collection, string doc)
{
    var docRef = _db.Collection(collection).Document(doc);
    DocumentSnapshot snapshot = await docRef.GetSnapshotAsync();
    if (snapshot.Exists)
    {
        return (true, snapshot.ConvertTo<T>());
    }
    else
    {
        Debug.Log("Document " + snapshot.Id + " does not exist!");
        return (false, default(T));
    }
}
```

```
public async UniTask UpdateQR(string doc, string phoneNumber)
{
    await Write("qrcodes", doc, "PhoneNumber", phoneNumber);
}
```

```
public async UniTask Write(string collection, string doc, string field, object value)
{
    var docRef = _db.Collection(collection).Document(doc);
    await docRef.UpdateAsync(field, value);
    Debug.Log($"컬렉션 {collection}의 문서 {doc}를 갱신합니다.");
}
```

</td>
</tr>
<tr>
<td>사용자가 클라이언트에서 플레이어 현황 데이터를 입력하여 제출</td>
<td>입력된 플레이어 현황 데이터를 서버에 전송하는 소스 코드</td>
</tr>
</table>

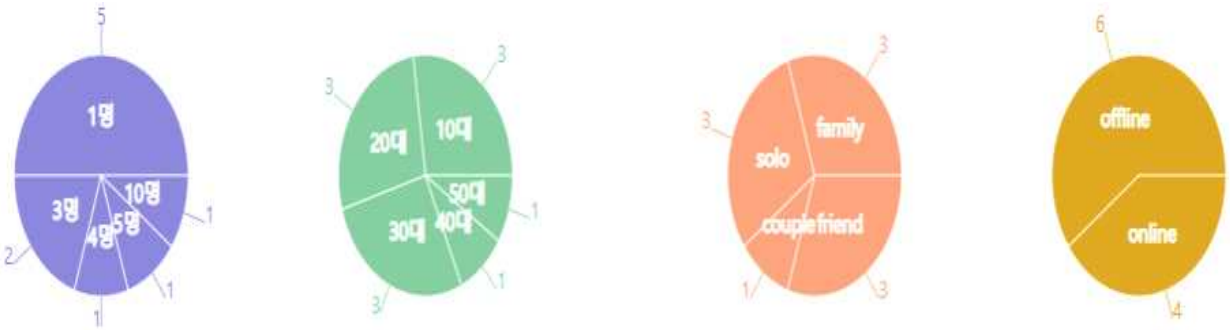
- DB저장 내용



#	닉네임	연락처	참가 인원	나이대	참가 형태	참가 경로	가입일	수정일
1	ㅇㅇㅇ	*******5678	3	30,50	family	offline	2022-12-02	2022-12-08
2	세종대왕	*******1234	1	10	solo	offline	2022-12-02	2022-12-02
3	호호	********1222	3	40	couple	online	2022-12-02	2022-12-02
4	sam	******4445	1	10	solo	offline	2022-12-02	2022-12-02
5	ㅇㅇㅇ	*******0000	1	20	family	offline	2022-12-07	2022-12-07
6	라크	*******8888	1	20	solo	offline	2022-12-07	2022-12-07
7	ㅁㄴㅇ	******5555	5	30	friend	online	2022-12-07	2022-12-07
8	ㅇㅇㅌ	*******7890	10	20	friend	online	2022-12-08	2022-12-08
9	ㅁㄴㅇ	*******6789	4	30	friend	offline	2022-12-08	2022-12-08
10	아이폰테스트	*******1212	1	10	family	online	2022-12-09	2022-12-09

‹ 1 ›

DB에 저장된 플레이어 데이터를 웹에서 관리



데이터 통계를 차트 형태로 표시

○ 사용자 데이터 로드

- 사용자가 플레이어 현황 데이터 입력을 완료하고 메인 화면에 진입하면, 서버로부터 플레이어 현황 데이터를 획득하여 클라이언트에 적용하는 API 작성
- 신규 사용자는 DB에 새로운 플레이어 데이터를 작성하여 이후 해당 DB에 진행 상황을 저장할 수 있도록 제작
- 기존 사용자는 DB에 저장된 데이터를 호출하여 기존의 플레이 데이터를 이어서 진행할 수 있도록 제작

- 클라이언트 입력 관련

<table>
<tr>
<td> 
</td>
<td>

```
public async UniTask<(bool, PlayerData)> ReadPlayer(string doc)
{
    Debug.Log("FireStore에서 Player 데이터를 가져옵니다.");
    return await ReadAsync<PlayerData>("players", doc);
}
```

사용자

```
public async UniTask<(bool, T)> ReadAsync<T>(string collection, string doc)
{
    var docRef = _db.Collection(collection).Document(doc);
    DocumentSnapshot snapshot = await docRef.GetSnapshotAsync();
    if (snapshot.Exists)
    {
        return (true, snapshot.ConvertTo<T>());
    }
    else
    {
        Debug.Log("Document " + snapshot.Id + " does not exist!");
        return (false, default(T));
    }
}
```

</td>
</tr>
<tr>
<td>사용자가 메인 화면에 진입할 때 서버로부터 진행 데이터를 획득</td>
<td>서버에 저장된 플레이어 현황 데이터로부터 클라이언트에 데이터를 가져오는 소스 코드</td>
</tr>
</table>

○ 사용자 아일랜드 진행 데이터 저장

- 사용자가 콘텐츠를 진행하여 플레이어 데이터 중 아일랜드 이동 데이터가 갱신될 때, 서버에 데이터 갱신 내용을 전송하는 API 작성
- 서버는 클라이언트로부터 전송받은 아일랜드 이동 데이터를 DB에 저장
- 이후 사용자가 클라이언트를 종료 후 재실행 시 서버로부터 해당 데이터를 불러와 아일랜드 진행 내용을 이어갈 수 있도록 클라이언트에 업데이트

- 클라이언트 입력 관련

<table>
<tr>
<td> 
</td>
<td>

```
public void SetEnterZoneId(int value)
{
    PlayerData.Save.EnteredZoneId = value;

    SavePlayerDataSaveField();
}
```

해당 아일랜드에 진입했다는 것을 저장함.

앱을 재실행했을 시, 해당 아일랜드부터 시작

```
public void SavePlayerDataSaveField()
{
    Debug.Log("PlayerData - Save 저장.");
    Managers.Firebase.Write("players", PlayerData.Info.id, "Save", PlayerData.Save).Forget();
}
```

```
public async UniTask Write(string collection, string doc, string field, object value)
{
    var docRef = _db.Collection(collection).Document(doc);
    await docRef.UpdateAsync(field, value);
    Debug.Log($"컬렉션 {collection}의 문서 {doc}를 갱신합니다.");
}
```

</td>
</tr>
<tr>
<td>사용자가 클라이언트에서 아일랜드 이동 정보 갱신</td>
<td>아일랜드 이동 정보를 서버에 전송, 저장하는 소스 코드</td>
</tr>
</table>

○ 사용자 GPS 데이터 저장

- 사용자가 콘텐츠를 진행하여 플레이어 데이터 중 GPS 인증 데이터가 갱신될 때, 서버에 데이터 갱신 내용을 전송하는 API 작성
- 서버는 클라이언트로부터 전송받은 GPS 인증 데이터를 DB에 저장
- 이후 사용자가 클라이언트를 종료 후 재실행 시 서버로부터 해당 데이터를 불러와 GPS 인증 내용을 이어갈 수 있도록 클라이언트에 업데이트

- 클라이언트 입력 관련

<table>
<tr>
<td> 
</td>
<td>

```
public void SetDidGPS1(bool value)
{
    PlayerData.Save.DidGPS1 = value;
    SavePlayerDataSaveField();
}
```

GPS 인증을 완료했다는 것을 저장함.

앱을 재실행했을 시, GPS 화면을 Skip함.

```
public void SavePlayerDataSaveField()
{
    Debug.Log("PlayerData - Save 저장.");
    Managers.Firebase.Write("players", PlayerData.Info.id, "Save", PlayerData.Save).Forget();
}
```

```
public async UniTask Write(string collection, string doc, string field, object value)
{
    var docRef = _db.Collection(collection).Document(doc);
    await docRef.UpdateAsync(field, value);
    Debug.Log($"컬렉션 {collection}의 문서 {doc}를 갱신합니다.");
}
```

</td>
</tr>
<tr>
<td>사용자가 클라이언트에서 GPS 인증 갱신</td>
<td>GPS 인증 완료 정보를 서버에 전송, 저장하는 소스 코드</td>
</tr>
</table>

○ 미션 완료 데이터 저장

- 사용자가 콘텐츠를 진행하여 플레이어 데이터 중 미션 완료 데이터가 갱신 될 때, 서버에 미션 완료 데이터 갱신 내용을 전송하는 API 작성
- 서버는 클라이언트로부터 전송받은 미션 완료 데이터를 DB에 저장
- 이후 사용자가 클라이언트를 종료 후 재실행 시 서버로부터 해당 데이터를 불러와 미션 완료 내용을 이어갈 수 있도록 클라이언트에 업데이트

- 클라이언트 입력 관련

<table>
<tr>
<td> 
</td>
<td>

```
public void CorrectAnswer(int getPoint = 0)
{
    PlayerData.Save.CurrentMissionNumber++;
    PlayerData.Play.Point += getPoint;

    SavePlayerData();
}
```

현재 미션을 성공한 것을 저장함.

앱을 재실행했을 시, 다음 미션부터 진행함.

```
public void SavePlayerDataSaveField()
{
    Debug.Log("PlayerData - Save 저장.");
    Managers.Firebase.Write("players", PlayerData.Info.id, "Save", PlayerData.Save).Forget();
}
```

```
public async UniTask Write(string collection, string doc, string field, object value)
{
    var docRef = _db.Collection(collection).Document(doc);
    await docRef.UpdateAsync(field, value);
    Debug.Log($"컬렉션 {collection}의 문서 {doc}를 갱신합니다.");
}
```

</td>
</tr>
<tr>
<td>사용자가 클라이언트에서
미션을 완료</td>
<td>미션 완료 정보를
서버에 전송하는 소스 코드</td>
</tr>
</table>

○ 아일랜드 완료 데이터 저장

- 사용자가 콘텐츠를 진행하여 플레이어 데이터 중 아일랜드 완료 데이터가 갱신될 때, 서버에 아일랜드 완료 데이터 갱신 내용을 전송하는 API 작성
- 서버는 클라이언트로부터 전송받은 아일랜드 완료 데이터를 DB에 저장
- 이후 사용자가 클라이언트를 종료 후 재실행 시 서버로부터 해당 데이터를 불러와 아일랜드 완료 내용을 이어갈 수 있도록 클라이언트에 업데이트

- 클라이언트 입력 관련

<table>
<tr>
<td> 
</td>
<td>

```
public void CompleteZone(int zoneId)
{
    Inventory.Add(Managers.Data.ZoneDic[zoneId].SuccessItem);

    for (int i = 0; i < CompletedZoneIds.Length; i++)
    {
        if (CompletedZoneIds[i] == 0) // 마지막에 넣기.
        {
            CompletedZoneIds[i] = zoneId;
            break;
        }
    }

    // 게임 도중에 필요한 Save 데이터들은 다시 초기화.
    PlayerData.Save.Init();

    SavePlayerData();
}
참조 1개
public void CompleteLastZone(int zoneId)
{
    PlayerData.Play.CompleteLastIslandTime = DateTime.Now;
    CompleteZone(zoneId);
}
```

해당 아일랜드 완료를 저장

```
public void SavePlayerDataSaveField()
{
    Debug.Log("PlayerData - Save 저장.");
    Managers.Firebase.Write("players", PlayerData.Info.id, "Save", PlayerData.Save).Forget();
}
public void SavePlayerDataPlayField()
{
    Debug.Log("PlayerData - Play 저장.");
    Managers.Firebase.Write("players", PlayerData.Info.id, "Play", PlayerData.Play).Forget();
}
public async UniTask Write(string collection, string doc, string field, object value)
{
    var docRef = _db.Collection(collection).Document(doc);
    await docRef.UpdateAsync(field, value);
    Debug.Log($"컬렉션 {collection}의 문서 {doc}를 갱신합니다.");
}
```

</td>
</tr>
<tr>
<td>사용자가 클라이언트에서
아일랜드 이동 정보 갱신</td>
<td>아일랜드 이동 정보를
서버에 전송하는 소스 코드</td>
</tr>
</table>

○ 콘텐츠 데이터 관리

- 클라이언트에서 사용되는 콘텐츠 데이터는 DB에 저장되어 있는 데이터를 서버에서 가져와 클라이언트에 표시하는 형태로 제작
- 웹에서 클라이언트에 적용되어 있는 데이터를 확인할 수 있도록 콘텐츠 관리자 페이지를 제작
- 웹 관리자가 웹에서 콘텐츠 수정 기능을 사용하여 DB에 콘텐츠 내용을 업로드한 후 저장하면, 클라이언트가 해당 데이터를 서버로부터 전송받아 클라이언트에 적용

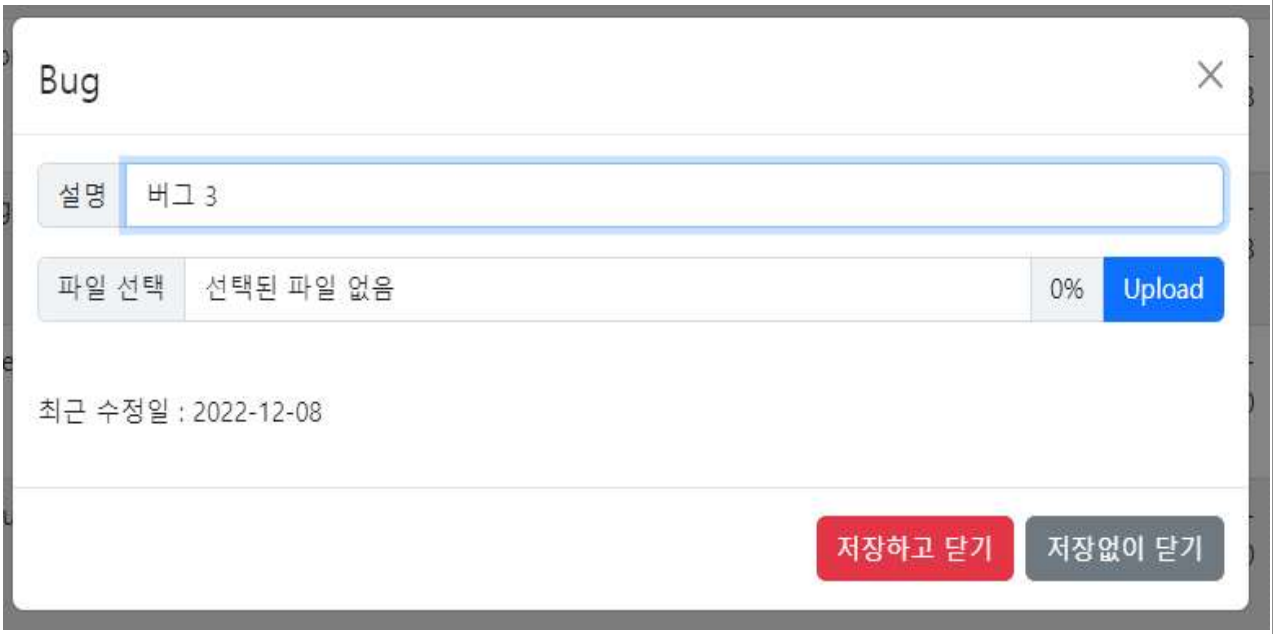
- DB 목록 확인

클라이언트에 적용되어 있는 콘텐츠 데이터를 웹에서 확인

○ 콘텐츠 데이터 수정 기능

- 데이터에 대한 설명문을 작성하여 수정 대상이 되는 데이터에 대한 정보를 확인할 수 있도록 제작
- 웹 서버 로그인 후 콘텐츠 수정을 진행할 파일을 PC에서 선택하여 업로드하는 것으로 콘텐츠 데이터 업데이트 수행
- 보안을 위해 콘텐츠 수정 시 업로드 가능한 파일은 5MB 이하의 이미지 포맷을 가진 파일만 선택할 수 있도록 구성
- 업로드 이전 최종 저장 확인 과정을 거쳐 데이터 업로드 시 발생할 수 있는 오류를 줄일 수 있도록 구성
- 데이터 업로드 시 최종 수정 날짜를 DB에 기록하여 원활한 데이터 관리를 수행할 수 있도록 함

- 콘텐츠 수정 기능 관련



웹에 파일을 업로드하여 콘텐츠 수정 진행

- 클라이언트 변경 관련 내용

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>데이터 수정 이전 클라이언트 화면</td>
<td>데이터 수정 이후 클라이언트 화면</td>
</tr>
</table>

데이터 버전을 확인하여 서버로부터 새로운 데이터를 읽어 오는 소스 코드

○ 사용자 만족도 데이터 저장

- 사용자가 콘텐츠 진행 중 사용자 만족도 조사 제출하기 기능을 사용하면, 사용자 클라이언트에서 해당 만족도 조사 데이터를 서버에 전송
- 서버는 클라이언트로부터 전송된 데이터를 DB에 저장
- 웹 관리자가 DB에 저장된 데이터를 웹에서 확인 가능
- 저장된 데이터를 일정 QRCode 범위 내에서 필터링하여 확인 가능

- 클라이언트 변경 관련

<table>
<tr>
<td> 
</td>
<td>

```
async UniTaskVoid Submit(PlayerSurveyData survey)
{
    Managers.GameState.SavePlayerPrefsPlayerData();

    if (Managers.GameState.IsConnectNetwork)
    {
        await Managers.Firebase.Write("players", Managers.GameState.PlayerData.Info.id, "Survey", survey);
    }
    else
    {
        Debug.Log("인터넷에 연결되어 있지 않아 바로 진행.");
    }
}
```

```
public async UniTask Write(string collection, string doc, string field, object value)
{
    var docRef = _db.Collection(collection).Document(doc);
    await docRef.UpdateAsync(field, value);
    Debug.Log($"컬렉션 {collection}의 문서 {doc}를 갱신합니다.");
}
```

</td>
</tr>
<tr>
<td>클라이언트에서 사용자 만족도 조사 내용 제출</td>
<td>DB에 저장된 데이터를 웹에서 관리</td>
</tr>
</table>

- DB저장 내용

DB에 저장된 사용자 만족도 데이터를 웹에서 관리

데이터 통계를 차트 형태로 표시