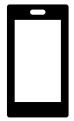
Portfolio

GAME DEVELOPER PORTFOLIO

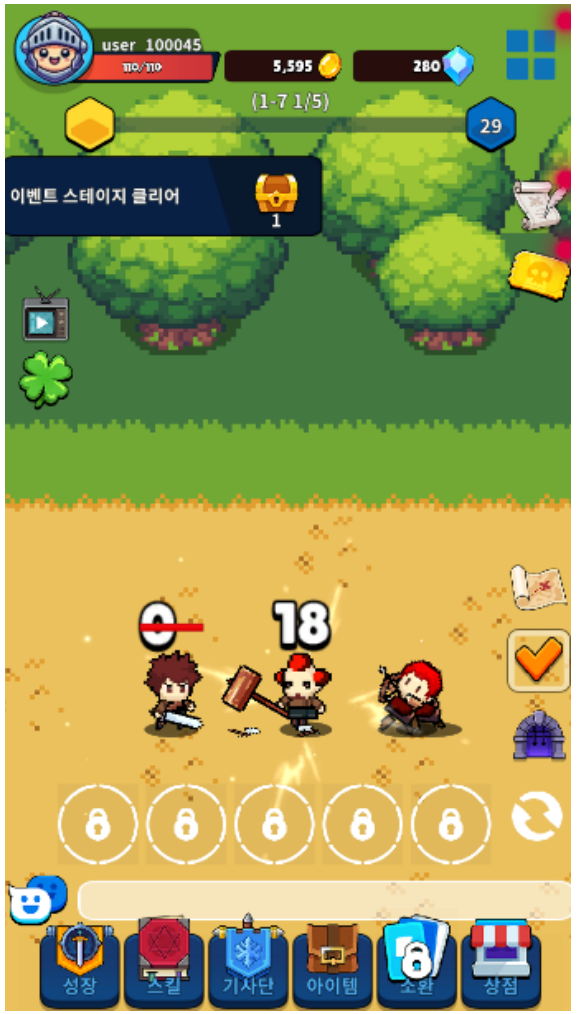
안 상 근



010 2434 8293

godyello123@gmail.com

Project K



프로젝트 구성

사용 언어 : C#
버전 관리 : SVN
데이터베이스 : Sql Server

게임 서버 개발

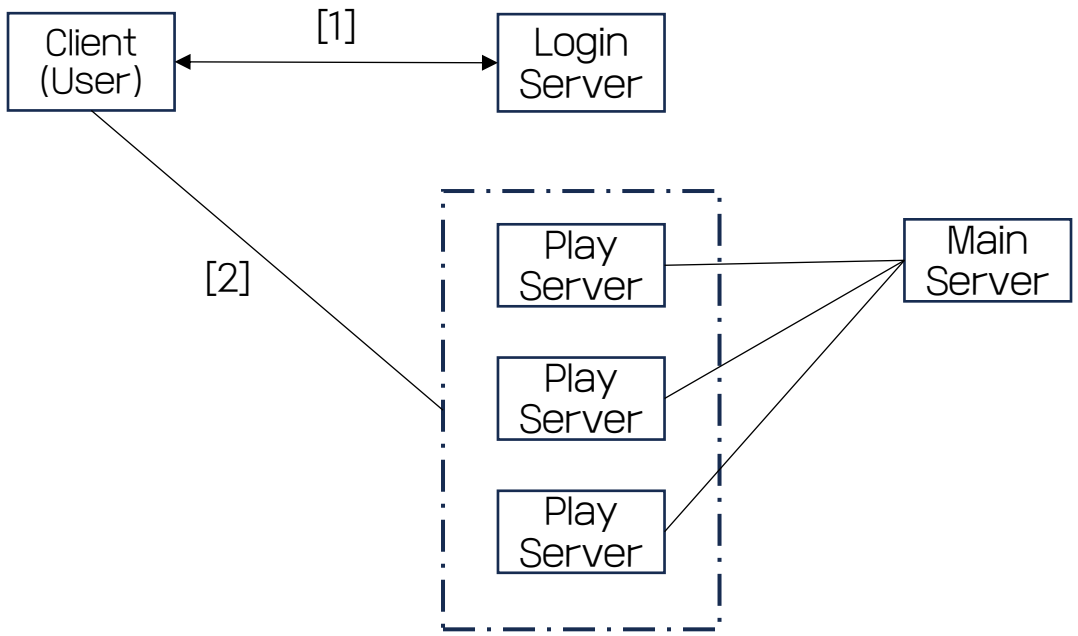
- 로그인 서버
- 유저
- 퀘스트
- 아이템
- 뽑기
- 우편
- 게임 서비스 로그

운영 툴 개발

- 유저 밴 추가/삭제
- 게임 서버 관리 기능
- 우편 발송
- 인게임 이벤트 관리
- 인게임 공지
- 게임 로그 조회

Project K

로그인 서버



[1] Client가 LoginServer에 접속 하여 유저의 ID인증
인증 완료 및 유저 데이터 유효성 검증 완료 시
PlayServer IP/DN, Port와 유저의 UID(고유ID)를
LoginServer에서 Clinet로 전달

[2] Client는 전달 받은 IP/DN, Port를 통해 PlayServer 접속
전달 받은 UID를 통해 DB에서 유저 데이터 유효성 검증 및 접속

유저

```
public class CUser : CActor
{
    private _UserData m_UserData;
    private CStageAgent m_StageAgent;
    private CPostAgent m_PostAgent;
    private CAssetAgent m_AssetAgent;
    private CLevelAgent m_LevelAgent;
    private CStatusAgent m_StatusAgent;
    private CQuestAgent m_QuestAgent;
    private CItemAgent m_ItemAgent;
    private CSkillAgent m_SkillAgent;
    private CCouponAgent m_CouponAgent;
    private CRelicAgent m_RelicAgent;
    private CPvpAgent m_PvpAgent;
    private CShopAgent m_ShopAgent;
    private CBuffAgent m_BuffAgent;
    private CEventAgent m_EventAgent;
```

▲게임 내 유저를 관리하는 핵심 클래스

게임 내 유저의 기능을 각각의 에이전트로 나누어 관리하는 구조

이를 통해 각 기능 별 에이전트는 별도로 관리되어 코드의 유지보수성을 높이고

새로운 컨텐츠가 추가 되거나 혹은 수정이 필요한 경우
해당 에이전트를 추가하거나 수정하는 식으로 확장성과 유연성을 확보

Project K

퀘스트

게임에서 일어나는 특정 이벤트를
퀘스트 시스템에 알리고 그에 따라 퀘스트를 업데이트


```
public class _QuestBoard : INetSerialize
{
    public CDefine.QuestType Type = CDefine.QuestType.Max;
    public string ID = string.Empty;
    public List<_Mission> Missions = new List<_Mission>();
    public DateTime ExpTime = DateTime.MinValue;
    public bool IsPassActivated = false;
```

▲퀘스트 보드 를 저장하는 클래스

Type : 퀘스트의 종류
다양한 퀘스트의 종류를 정의
해당 변수를 통해 퀘스트의 타입을 확인

ID : 퀘스트의 ID
기획데이터에서 퀘스트ID를 식별하는 필드

Missions : 퀘스트에 포함 된 여러 미션들
_Mission은 각각의 미션이며 한 퀘스트는 여러 개의 미션을 가질 수 있음

ExpTime : 퀘스트의 만료시간
특정 퀘스트가 제한시간동안 진행되는 퀘스트라면 사용되는 필드

IsPassActive : 패스(ex: 배틀패스 등) 가 활성화 값이 true 라면 퀘스트를 진행 할 수 있는 패스가 활성화 된 상태

```
public class CQuestSystemEx_Base
{
    protected CUser m_Owner;
    protected _QuestBoard m_Board = new _QuestBoard();
    protected Dictionary<string, List<_Mission>> m_Finder = new Dictionary<string, List<_Mission>>();
```

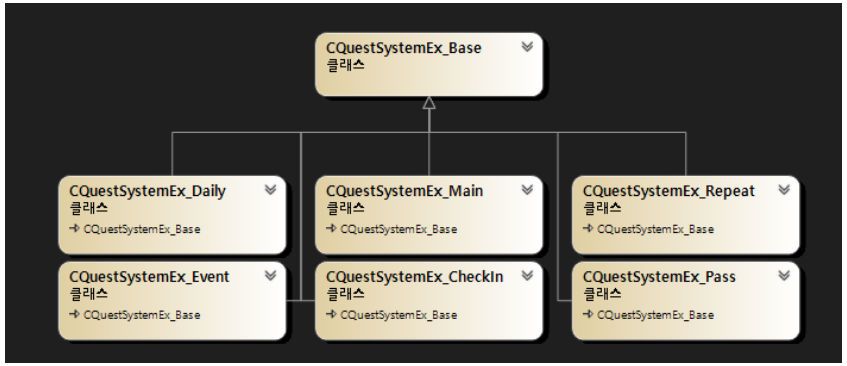
▲QuestSystem 다이어그램

m_Owner : 퀘스트 시스템이 적용 되는 주체 (유저)

m_Board : 유저의 퀘스트 보드
유저가 진행 중인 퀘스트 정보를 저장

m_Finder : 미션을 찾기 위한 필드
특정 키를 통해 미션을 빠르게 찾기 위한 필드

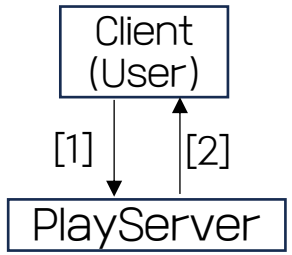
상속을 통해 기능 확장이 가능하도록 설계



▲상속을 통한 기능 확장

Project K

퀘스트



[1] 클라이언트로부터 특정 이벤트 발생

퀘스트 시스템에 이벤트에 따른 퀘스트를 갱신

[2] 갱신 결과를 클라이언트(유저)에게 알림

```
public void UpdateQuestBorad(CDefine.MissionCondition conditionType, string param, string target, long updateValue)
{
    foreach (var iter in m_QuestSystem)
    {
        var system = iter.Value;
        system.Update(ConditionRecord.Key(conditionType, param), target, updateValue);
    }
}
```

▲특정 이벤트에 따른 퀘스트를 갱신 하는 함수

클라이언트(유저)가 게임내에서 특정 이벤트를 수행 할때 그에 맞는 퀘스트의 상태를 갱신

조건과 파라미터를 기반으로 유연한 업데이트가 가능하도록 설계

```
public void Update(string key, string target, long updateValue)
{
    var foundMissions = Find(key);
    if (foundMissions == null)
        return;

    foreach (var mission in foundMissions)
    {
        if (!IsProcessing(mission))
            continue;

        var mRecord = MissionTable.Instance.Find(mission.ID);
        if (mRecord == null)
            continue;

        var condRecord = ConditionTable.Instance.Find(mRecord.ConditionID);
        if (condRecord == null)
            continue;

        if (!condRecord.IsValidTarget(target))
            continue;

        if (condRecord.IsSwapValue())
            mission.Val = updateValue;
        else
            mission.Val += updateValue;

        if (IsComplete(mission, mRecord))
            Complete(mission);
        else
            Excute(mission);
    }
}
```

▲퀘스트 시스템에서 특정 미션의 상태를 갱신 하는 함수

① 해당 키에 맞는 미션들을 찾음

②미션이 진행 중인지 여부 확인

③미션이 완료 되었는지 확인
완료되었다면 Complete를 호출하여 미션 완료 처리
완료되지 않은 경우 Excute를 호출하여 미션 진행 중 처리

Project K

아이템


```
public partial class _ItemData : INetSerialize
{
    public long ItemID = -1;
    public string TableID = string.Empty;
    public int Level = 0;
    public long Count = 0;
    public DateTime ExpireTime = DateTime.MinValue;
    public List<_RandomOption> RandomOption = new List<_RandomOption>();
```

▲아이템을 저장하는 클래스

ItemID : 아이템을 식별하는 고유ID
아이템을 식별하는 고유ID

TableID : 아이템 참조ID
기획데이터에서 아이템을 참조 할때 사용하는 ID

Level : 아이템의 Level
아이템 강화 시에 사용되는 필드

Count : 아이템의 보유 개수
아이템의 보유 개수를 나타내는 필드

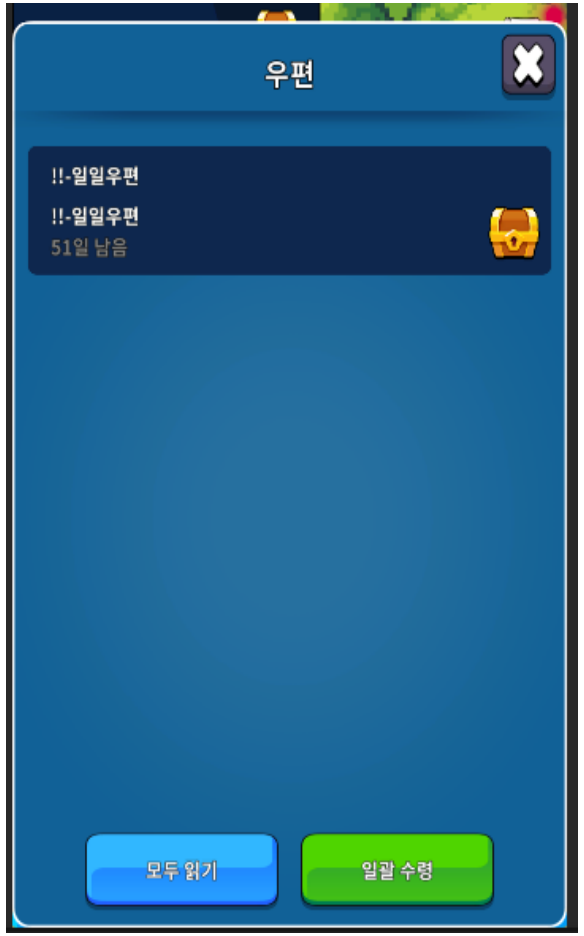
ExpireTime : 아이템의 제한 시간
특정 이벤트성 아이템이나 지정 된 기간동안 사용가능한
아이템을 관리하기 위한 필드

RandomOption : 아이템의 랜덤 옵션
아이템의 특정 랜덤 옵션을 부여하기 위한 필드

아이템 강화, 아이템 장착, 아이템 소모 구현

Project K

우편




```
1
insert into dbo.tb_character_post
(
    uid, id, type, title,
    msg, is_read, is_reward, begin_time, expire_time,
    reward, dw_update_time
)
select
    @uid, sp.id, sp.type, sp.title,
    sp.msg, 'FALSE', 'FALSE',
    sp.begin_time, sp.expire_time,
    sp.reward, sp.dw_update_time
from dbo.tb_system_post as sp
2
left outer join dbo.tb_character_post as cp
on cp.id = sp.id and cp.type = sp.type and cp.uid = @uid
where cp.id is null
and sp.begin_time <= @utc_time
and @utc_time < sp.expire_time;

select id, type, title, msg, is_read, is_reward,
begin_time, expire_time, reward
from dbo.tb_character_post
where uid = @uid
```

▲유저 우편 조회 쿼리

① 유저의 우편테이블에 새로운 우편을 삽입

② 시스템 우편을 Left outer join으로 사용자의 우편테이블과 연결하여, 존재하지 않는 우편 만 선택

유효한 우편만 삽입 되도록 begin_time과 expire_time 조건을 설정하여 현재시간 (@utc_time)에 따라 필터링

Project K

게임로그

MongoDB를 사용하여 게임 내 발생하는
로그를 수집 하는 시스템 개발

```
public class LogBson : BsonBase
{

    [BsonElement("deviceID")]
    참조 1개
    public string DeviceID { get; set; }

    [BsonElement("uid")]
    참조 1개
    public long UID { get; set; }

    [BsonElement("type")]
    참조 2개
    public ushort Type { get; set; }

    [BsonElement("log")]
    참조 3개
    public string LogStr { get; set; } = string.Empty;

    [BsonElement("coin")]
    참조 1개
    public string Update_Coins { get; set; } = string.Empty;

    [BsonElement("item")]
    참조 1개
    public string Update_Items { get; set; } = string.Empty;

    [BsonElement("cnt")]
    참조 1개
    public int Count { get; set; }
}
```

▲게임 로그 클래스

MongoDB의 BSON포맷으로 데이터를 직렬화 하기 위해
BsonElement를 사용하여 각 속성을 맵핑

이를 통해 로그를 MongoDB에 저장하고 필요 시 조회 가능

```
public class CMongoDBManager : SSingleton<CMongoDBManager>
{
    private SMongoDB m_DB;
    private bool m_Run = false;
```

▲MongoDB 와의 연결을 관리하는 클래스

```
public class SMongoDB
{
    private Thread m_Thread;

    public delegate void LogDelegate(string Log);
    private bool m_Run;
    private ConcurrentQueue<IMongoDBQuery> m_InputQueue = new ConcurrentQueue<IMongoDBQuery>();
    private ConcurrentQueue<IMongoDBQuery> m_OutputQueue = new ConcurrentQueue<IMongoDBQuery>();
```

▲실제 MongoDB와 연결되며, 쿼리를 처리하는 클래스

로그 적재를 위해 별도의 쓰레드를 사용하여 게임 로직에 영향을 주지 않도록 개발