
GameBaseServer

N A M E : 권 성 호

T E L : 010-8874-6452

P A R T : PROGRAMING

Efforts make all doors open



목 차



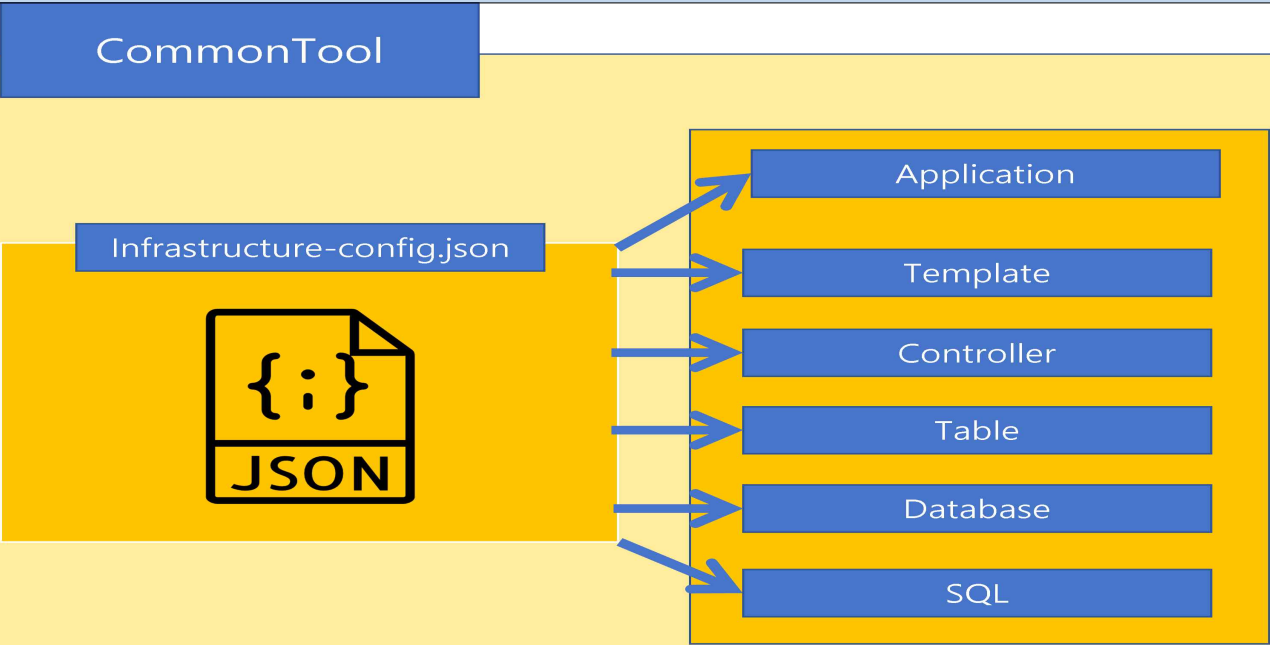
I . 포트폴리오 개요	1
1. GameBaseServer	1
2. CommonTool	2
3. TestProject	3
II . 기술 요약 및 UML	4
1. GameBaseServer	4
2. CommonTool	23
3. TestProject	27
III . 별첨	44

I . 포트폴리오 개요

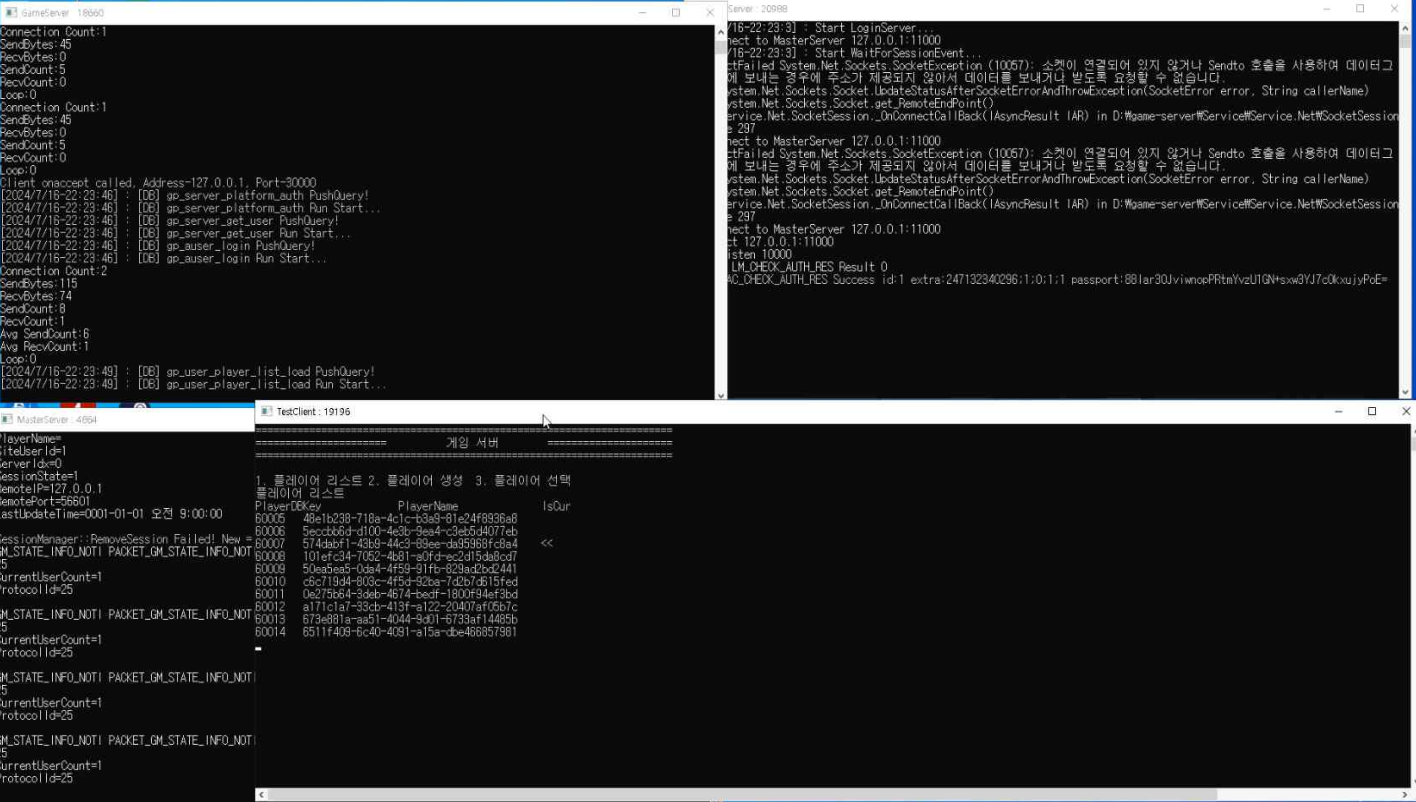
1. GameBaseServer

프로그램 명	GameBaseServer
플 랫 폼	크로스 플랫폼
프로그램 개요	
프로그램 소개	게임 서버의 공통적인 부분을 미리 개발하여 한 게임에서 만에서의 콘텐츠 개발이 아닌 여러 게임에서 공용으로 사용할 수 있는 콘텐츠 및 기능을 개발, 유지 할 수 있도록 구조화하였다.
개발 기간	~ 2024.7
개발 환경	VS2022 + .NetCore 3.1
설명	캐주얼 게임 개발 시 공통 적으로 사용되는 패키지, DB 테이블, 쿼리, 데이터 테이블 구조, 콘텐츠를 공용 적으로 설계 및 개발하여 다양한 프로젝트에 공통으로 빠르게 적용할 수 있게 서버를 설계하여 작업의 효율을 증대할 수 있게 하였다.
GitHub URL	https://github.com/AshOne91/game-server.git

2. CommonTool

프로그램 명	CommonTool
플 랫 폼	Windows
프로그램 개요	
	
프로그램 소개	GameBaseServer 제작에 필요한 기능을 생성하기 위한 공용 툴
개발 환경	VS2022 + .NetCore 3.1
설명	<ol style="list-style-type: none"> 1. Infrastructure-config.json JSON 문서를 정해진 규칙에 따라 파악하여 각 항목에 맞는 기능을 분리, 생성, 실행한다. 2. Application 생성 명시된 라이브러리와 Template를 사용하여 게임 서버 제작에 필요한 프로젝트, 프레임워크를 생성해 준다. 2. Template 생성 게임 콘텐츠를 기준으로 구분 생성한다. Template는 콘텐츠 개발에 필요한 기본적인 SQL 실행문, 패키지 생성 및 로직, 게임 로직을 생성하며 가이드라인을 설정하여 모든 게임에 공통으로 적용될 수 있게 해야 한다. 3. Controller 생성 실제 패키지를 구분 실행하는, Template 당 하나의 Controller 문이 생성된다. Template 생성 시 기본 생성되며, 클라이언트 패키지 공유를 위해 따로 구분되어 있다. 4. Table 생성 데이터 주도적 프로그래밍에 필요한 데이터 테이블을 생성한다. 5. Database 기능 생성 명세서를 기준으로 해당 프로젝트의 프레임워크에 맞춰 테이블, 스토어드프로시저를 실행하기 위한 코드를 생성한다. 6. SQL 문 생성 및 실행 명세서를 기준으로 테이블, 스토어드 프로시저 생성 SQL 문을 생성하고, 데이터베이스에 접속하여 SQL 문을 실행한다.
GitHub URL	https://github.com/AshOne91/common-tool.git

3. TestProject

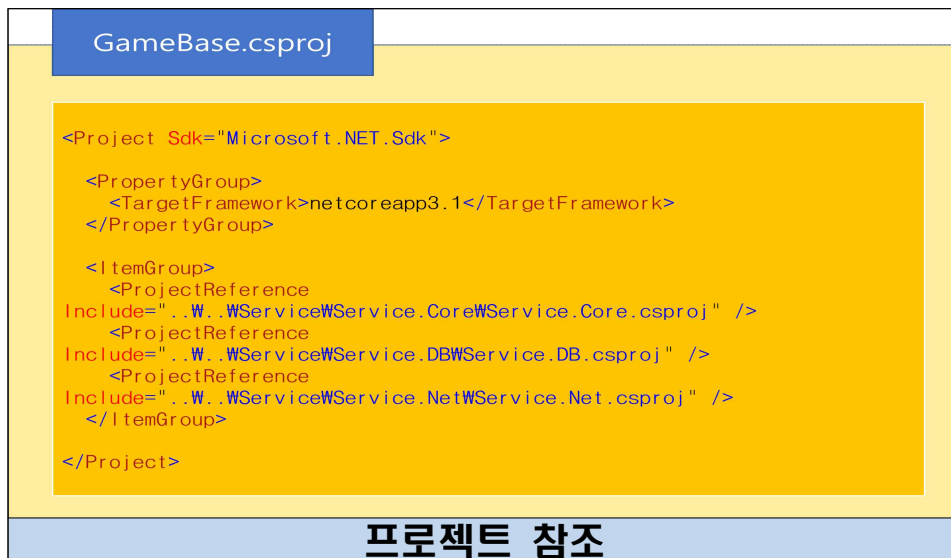
프로그램 명	TestProject
플랫폼	Windows
프로그램 개요	
	
프로그램 소개	GameBaseServer, CommonTool을 테스트하기 위한 테스트프로젝트
개발 환경	VS2022 + .NetCore 3.1
설명	<p>GameBaseServer와 CommonTool을 이용하여 해당 기능이 정상적으로 동작하는지와 공용 로직이 정상 동작하는지 테스트하기 위한 프로젝트이다. 게임 서버 Application, Login 서버 Application, Game 서버 Application, 테스트 클라이언트, MySQL로 구성되어 있다. 로그인 절차, 아이템 콘텐츠, 상점 콘텐츠가 Template, Service, CommonTool을 사용하여 제작되었다.</p> <ol style="list-style-type: none"> 1. Login 서버 클라이언트의 인증 절차를 수행하고 Game 서버의 IP와 Port, 접속 절차에 사용되는 Passport를 발급한다. 2. Game 서버 클라이언트의 인증 절차를 수행하고 게임의 콘텐츠를 담당한다. 3. Master 서버 서버의 상태 체크, 클라이언트 세션 상태 체크를 담당한다. 4. 테스트 클라이언트 게임 서버의 테스트를 담당한다. 계정 생성, 사용자 생성, 플레이어 생성, 인벤토리 요청, 아이템 구매를 테스트한다.
동영상 링크	https://youtu.be/NMV_BbQf7tM?si=7bOAtM8G9e-syxn0
GitHub URL	https://github.com/AshOne91/game-server.git

II. 기술요약 및 UML

1. GameBaseServer

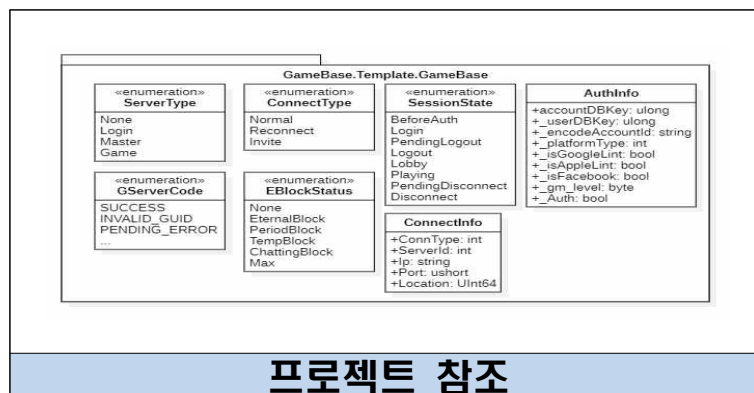
게임 개발 시 모든 게임에서 아이템, 상점, 길드, 채팅, 업적 등 공통 적으로 제작되는 콘텐츠가 있다. 해당 콘텐츠를 Template 프로젝트로 모듈화하여 미리 개발된 Template를 Application에 추가하여 빠르게 게임 개발이 가능하도록 구조화하였다. 또한 데이터 테이블, 데이터베이스 모델 또한 공통화하여 하나의 데이터베이스에서 여러 게임이 공통으로 사용될 수 있도록 일원화하였다.

1.1 GameBase.csproj



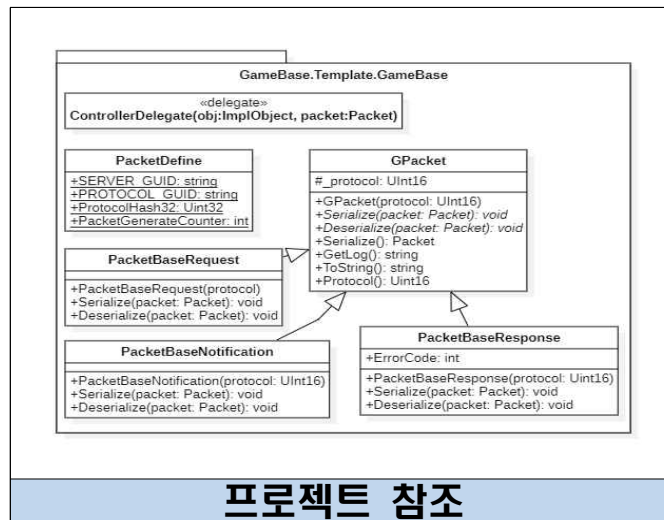
GameBase은 Template가 참조하는 기본 프로젝트로 Template의 공통 적으로 사용되는 패키지, 테이블, DB 처리에 관한 기본 구조를 정의한다.

- GameBaseDefine.cs



각 Template에서 공용으로 사용되는 구조체와 열거형을 정의한다.

- GameBasePacketStruct.cs



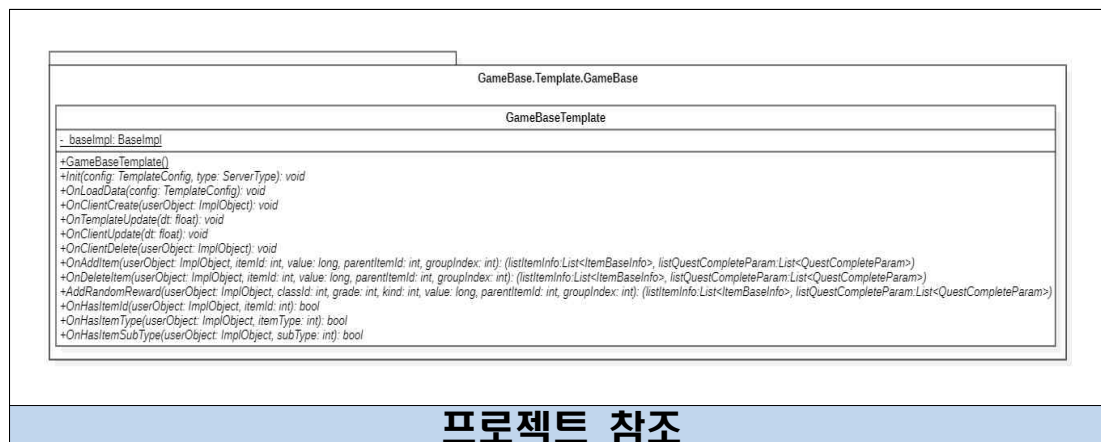
공용 적으로 사용되는 패킷의 요소를 정의, 또한 각 Template의 패킷 생성 시 재정의되는 인터페이스를 선언

- GameBaseSessionManager.cs



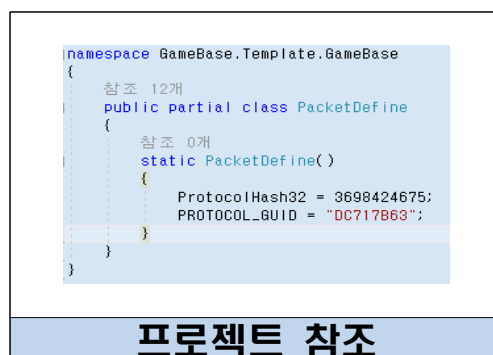
세션의 정보를 저장하고, 관리하는 매니저 클래스. MasterServer에서 세션의 상태를 저장, 삭제, 업데이트에 이용된다.

- GameBaseTemplate.c



각 Template의 공통적인 기능을 모아놓음. Template 간의 참조는 원칙적으로 불가능하므로 Application은 추상화된 기능을 호출하여 등록된 Template의 상황에 맞는 행위를 할 수 있게 한다. 각 Template는 OnFunction을 재정의해야 한다.

- GenPacketDefine.cs



패킷 변경 시 변경된 프로토콜의 해시코드와 GUID를 생성 저장한다. 클라이언트와 서버의 인증 절차에서 사용된다.

- TemplateConfig.cs



Application �행 시 불러오는 설정 파일 내용을 넣어준다. 해당 내용은 JSON 포맷의 설정 파일과 매칭이 되고, 등록된 Template의 초기화가 진행될 시 해당 내용으로 초기화가 진행된다.

- GameBaseTemplateContext.cs

GameBase.Template.GameBase	
«enumeration» ETemplateType	«static» GameBaseTemplateContext
None	<pre> - app: ServerApp - dbManager: GameBaseDBManager - objByUid: Dictionary<ulong, ImplObject> - templates: Dictionary<ETemplateType, GameBaseTemplate> - templateByUid: Dictionary<ulong, Dictionary<ETemplateType, GameBaseTemplate>> - objectTypeByUid: Dictionary<ulong, ulong> - uidByObjectType: Dictionary<ulong, HashSet<ulong>> - account: AccountTemplate - admin: AdminTemplate - advert: AdvertTemplate - attendance: AttendanceTemplate - auction: AuctionTemplate - battle: BattleTemplate - building: BuildingTemplate - character: CharacterTemplate - item: ItemTemplate - mailbox: MailBoxTemplate - matching: MatchingTemplate - notice: NoticeTemplate - quest: QuestTemplate - rank: RankTemplate - report: ReportTemplate - season: SeasonTemplate - shop: ShopTemplate +GameBaseTemplateContext() «property»+Account(): AccountTemplate «property»+Admin(): AdminTemplate «property»+Advert(): AdvertTemplate «property»+Attendance(): AttendanceTemplate «property»+Auction(): AuctionTemplate «property»+Battle(): BattleTemplate «property»+Building(): BuildingTemplate «property»+Character(): CharacterTemplate «property»+Item(): ItemTemplate «property»+MailBox(): MailBoxTemplate «property»+Matching(): MatchingTemplate «property»+Notice(): NoticeTemplate «property»+Quest(): QuestTemplate «property»+Rank(): RankTemplate «property»+Report(): ReportTemplate «property»+Season(): SeasonTemplate «property»+Shop(): ShopTemplate +Clear(): void +Remove(uid: ulong): void +RemoveAll(): void +SetApp(app: ServerApp): void +GetApp(): ServerApp +SetDBManager(dbMgr: GameBaseDBManager): void +GetDBManager(): GameBaseDBManager +AddTemplate<T>(userObject: T, key: ETemplateType, value: GameBaseTemplate): bool +AddTemplate(key: ETemplateType, value: GameBaseTemplate): bool +GetTemplate<T>(key: ETemplateType): T +GetTemplate<T>(uid: ulong, key: ETemplateType): T +RemoveTemplate(key: ETemplateType): void +RemoveTemplate(uid: ulong, key: ETemplateType): void +InitTemplate(config: TemplateConfig, type: ServerType): void +LoadDataTable(config: TemplateConfig): void +FindUserObj<T>(uid: ulong): T +FindUserObjFromTypes<T>(type: ulong): T +GetObjectCount(type: ulong): int +CreateClient(uid: ulong): void +UpdateTemplate(dt: float): void +UpdateClient(dt: float): void +DeleteClient(uid: ulong): void +AddItem(uid: ulong, itemId: int, value: long, parentItemId: int, groupIndex: int): (listItemInfo: List<ItemBaseInfo>, listQuestCompleteParam: List<QuestCompleteParam>) +DeleteItem(uid: ulong, itemId: int, value: long, parentItemId: int, groupIndex: int): (listItemInfo: List<ItemBaseInfo>, listQuestCompleteParam: List<QuestCompleteParam>) +HasItemId(uid: ulong, itemId: int): bool +HasItemType(uid: ulong, itemType: int): bool +HasItemSubType(uid: ulong, itemSubType: int): bool </pre>

프로젝트 참조

해당 클래스는 Template를 등록 및 관리를 담당하는 클래스이다. GameBaseTemplate의 기능을 호출하는 기능과 Template에서 다른 Template의 미리 정의된 함수를 통해 데이터에 접근 및 호출을 가능하게 해준다.

- instructure-config.json



Template에서 공통으로 생성하는 구조체(패킷에서 사용)를 넣어준다. 해당 구조체는 Serialize, Deserialize가 CommonTool에 의해 재정의 된다.

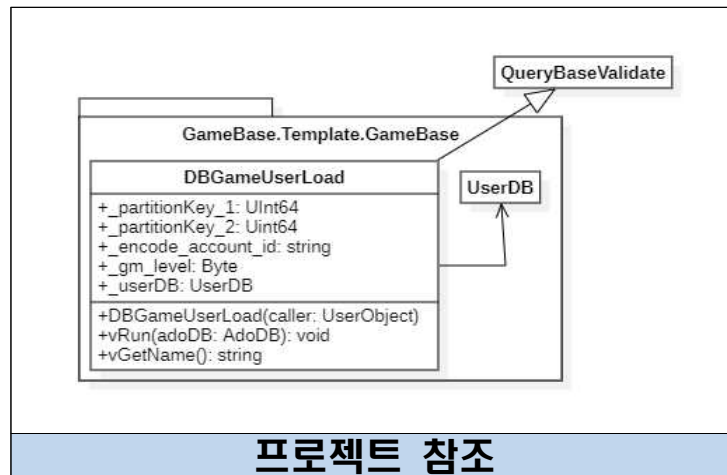
1. 1. 1 Base

각 Template의 공통으로 참조, 상속, 재정의하는 구조체, 클래스를 정의한다.

1. 1. 2. DB

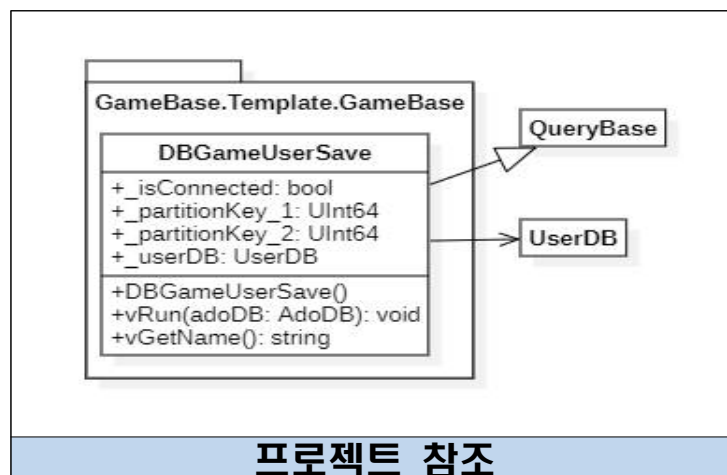
각 Template의 각각의 DB 테이블 정보를 Save, Load를 일원화하기 위하여 생성한 DB 관련 클래스이다.

- DBGameUserLoad.cs



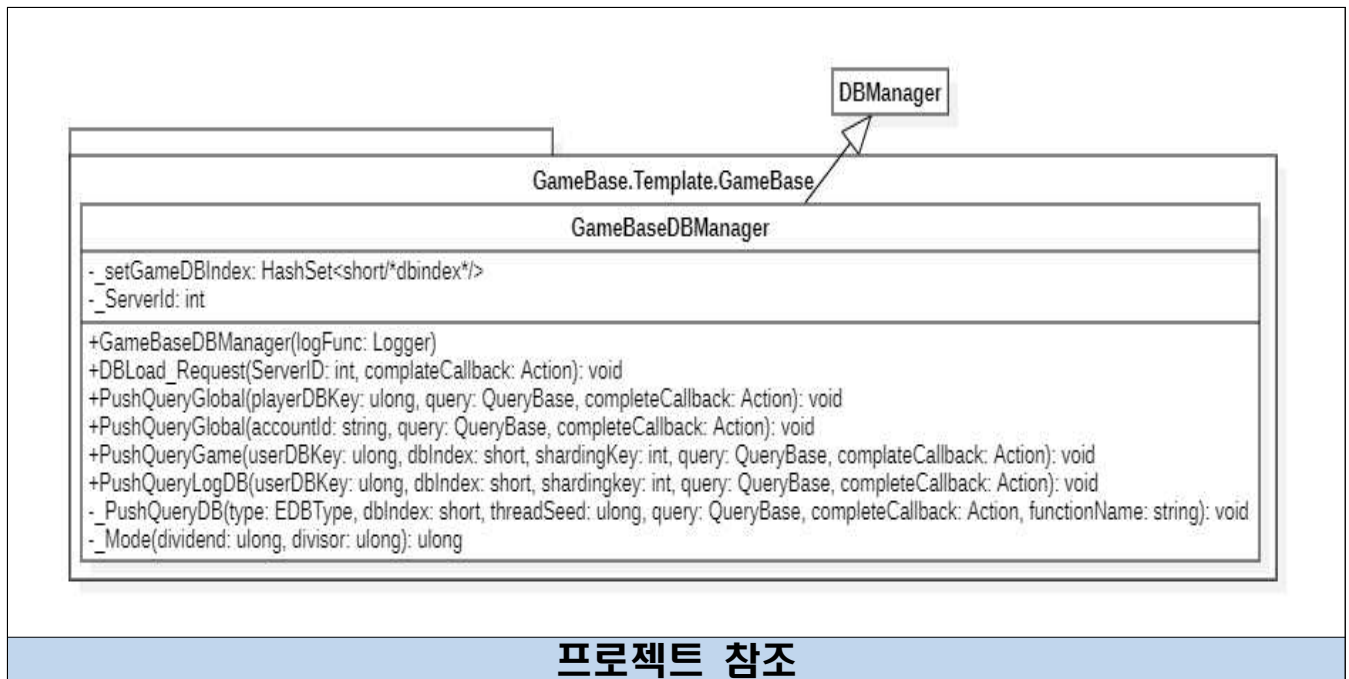
각 Template에서 생성한 DB 테이블 로드 스토어드 프로시저 쿼리를 호출하기 위한 테이블 로드 클래스

- DBGameUserSave.cs



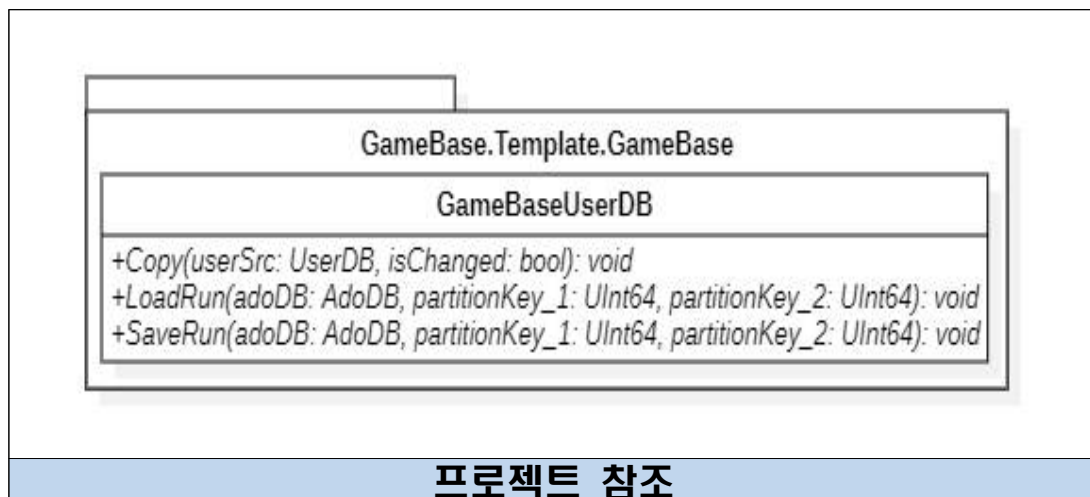
각 Template에서 생성한 DB 테이블을 저장하기 위한 스토어드 프로시저 쿼리를 호출하기 위한 테이블 세이브 클래스

- GameBaseDBManager.cs



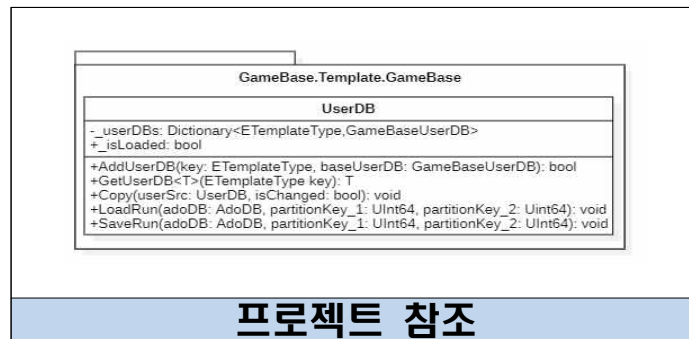
용도별 DB의 정보와 DB별 쓰레드에 쿼리를 실행하기 위한 클래스, GlobalDB에 저장된 GameDB, LogDB정보를 불러와 커넥트를 생성하고 유저 별 GameDB, LogDB샤딩과 올바른 쿼리를 호출하기 위한 DBManager 클래스

- GameBaseUserDB.cs



각 Template의 DB 정보를 로드, 저장, 카피하기 위한 추상화 클래스. 각 Template는 DB 명세서를 작성하고 CommonTool을 이용하여 해당 메소드를 재정의하게 된다.

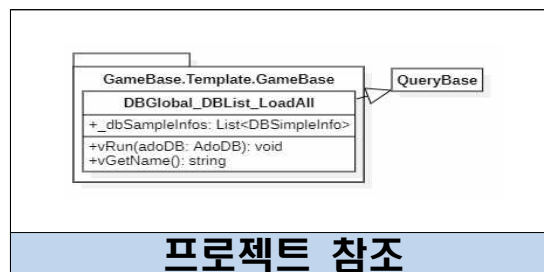
- UserDB.cs



Template별 GameBaseUserDB를 등록하여 각각의 Template의 DB 테이블을 로드, 저장, 카피를 호출하기 위한 클래스

1. 1. 2. 1 DBGGlobal

- DBGGlobal_DBList_LoadAll.cs



GlobalDB에서 GameDB와 LogDB의 정보를 얻기 위한 스토어드 프로시저 호출 클래스, 현재 GameBaseDBManager에서 GlobalDB에 접속 시 처음 호출하는 스토어드 프로시저 호출 클래스이다.

1. 1. 3. Template

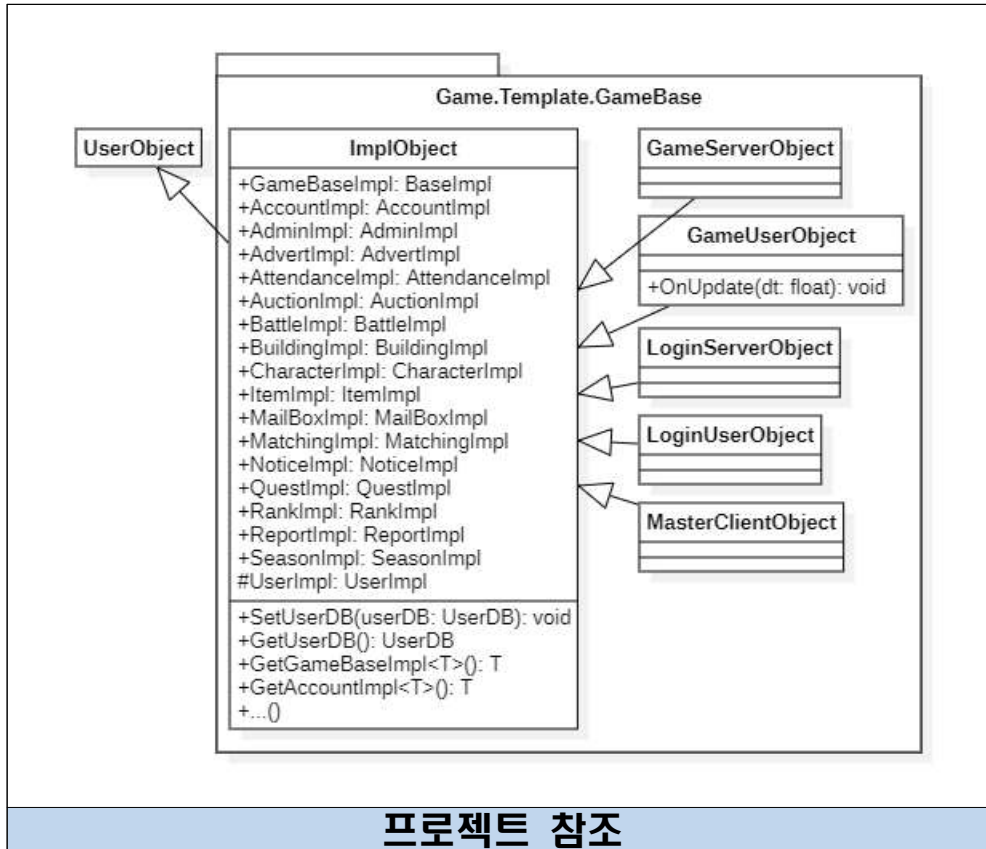
GameBaseTemplate를 상속받은 Template 클래스. Template와 Template는 서로 데이터를 공유받을 수 없기에 각 Template 간의 데이터 공유 및 함수 호출을 위한 메소드를 정의하기 위한 클래스. Template 별로 클래스를 분리해 놓음

- AccountTemplate.cs(예)



1. 1. 4. UserObject

하나의 세션을 담당하는 UserObject 서버의 접속한 서버 혹은 유저가 될 수 있다. 브릿지 패턴을 이용하여 구현 부를 분리하였다.



- GameServerObject.cs

마스터 서버에 접속한 게임 서버를 정의하기 위한 클래스

- GameUserObject.cs

게임 서버에 접속한 유저를 정의하기 위한 클래스

- ImplObject.cs

서버에 접속한 유저 혹은 서버가 각 Template의 기능을 구현하기 위해 Template 멤버 정의 사항을 멤버 변수로 가지고 있는 클래스

- LoginServerObject.cs

마스터 서버에 접속한 로그인 서버를 정의하기 위한 클래스

- LoginUserObject.cs

로그인 서버에 접속한 유저를 정의하기 위한 클래스

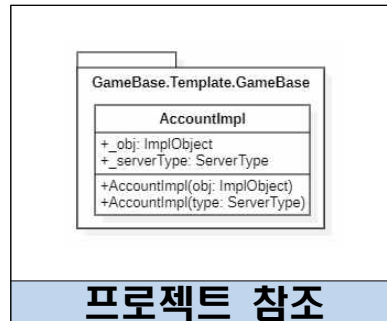
- MasterClientObject.cs

로그인, 게임 서버에 접속한 마스터 서버를 정의하기 위한 클래스

1. 1. 4. 1 Impl

각 Template에서 사용되는 서버 혹은 유저의 변수들을 정의 및 생성한다.

- AccountImpl.cs(예)



1.1.2. Common

CommonTool을 이용해 생성한 데이터 테이블, 패킷, 패킷 구현부, Template들이 공용으로 사용될 시 infrastructure-config.json에 기입 하여 생성한다.

- GameBaseModel.cs



infrastructure-config.json의 정의에 의해 생성된 클래스의 모음

- GameBasePacket.cs

infrastructure-config.json의 정의에 의해 생성된 패킷 클래스의 모음

- GameBaseProtocol.cs

infrastructure-config.json의 정의에 의해 생성된 패킷의 프로토콜의 등록과 콜백함수 등록

1.2. GameBaseShop

상점 콘텐츠를 담당하는 Template로 Template를 어떻게 생성하고 코드를 어떻게 작성해야 하는지 상점이라는 Template를 가지고 설명하고자 한다.

- infrastructure-config.json

```
"databases": [
  {
    "databaseType": "game",
    "tableType": "slot",
    "tableName": "DBShopTable",
    "partitionKey_1": "player_db_key",
    "partitionKey_2": "",
    "members": [
      {
        "type": "int",
        "name": "shop_index",
        "comment": "Shop인덱스"
      }
    ]
  }
],
"protocols": [
  {
    "id": 1001,
    "method": "react",
    "name": "CG_SHOP_INFO",
    "protocolType": "",
    "reqMembers": [],
    "resMembers": [
      {
        "type": "List<ShopInfo>",
        "name": "listShopInfo",
        "comment": "상점 정보"
      }
    ]
  }
],
"models": [
  {
    "name": "ShopInfo",
    "members": [
      {
        "type": "int",
        "name": "shopId",
        "comment": "ShopInfo 아이디"
      }
    ]
  }
]
```

프로젝트 참조

CommonTool이 Template 생성 시 읽어 들이는 명세서로 databases 항목은 DB 테이블 생성과 호출 SQL 생성과 호출을 담당, protocols는 패킷 생성과 프로토콜 생성, models는 패킷의 serialize, deserialize가 가능한 데이터 구조체를 생성한다.

- GameBaseShop_gamedb1_gameDB.sql
- GameBaseShop_gamedb2_gameDB.sql

```
USE gamedb1;
DROP TABLE if exists table_auto_dbshoptable;
CREATE TABLE table_auto_dbshoptable (
  idx BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  player_db_key BIGINT UNSIGNED NOT NULL DEFAULT 0,
  slot SMALLINT NOT NULL DEFAULT 0 COMMENT '슬롯 번호',
  deleted BIT NOT NULL DEFAULT b'0' COMMENT '삭제 여부',
  create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '생성 시간',
  update_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '수정 시간',
  shop_index INT NOT NULL DEFAULT 0 COMMENT 'Shop인덱스',
  shop_product_index INT NOT NULL DEFAULT 0 COMMENT 'ShopProductList인덱스',
  buy_count INT NOT NULL DEFAULT 0 COMMENT '구매 횟수',
  PRIMARY KEY(idx)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_general_ci;
ALTER TABLE table_auto_dbshoptable
ADD UNIQUE INDEX ix_dbshoptable_player_db_key_slot (player_db_key,slot);
DROP PROCEDURE if exists gp_player_dbshoptable_load;
DELIMITER $$

CREATE PROCEDURE gp_player_dbshoptable_load(...)
BEGIN...

END
$$
DELIMITER ; DROP PROCEDURE if exists gp_player_dbshoptable_save;
DELIMITER $$

CREATE PROCEDURE gp_player_dbshoptable_save(...)
)
BEGIN...

END
$$
DELIMITER ;
```

프로젝트 참조

CommonTool에 의해 생성된 SQL 파일로, 원격으로 DB에서 실행되어 databases 항목을 바탕으로 약속된 규격으로 만들어진 테이블 및 스토어드 프로시저를 생성한다.

- GameBaseShopDefine.cs

```
// TODO : 템플릿에서 사용할 열거형을 정의합니다.

참조 0개
public enum ShopStatusType...

참조 12개
public enum ShopBuyType...
```

프로젝트 참조

ShopTemplate에서 사용되는 열거형을 정의한다.

- GameBaseShopImpl.cs

```
참조 4개
public partial class GameBaseShopImpl : ShopImpl
{
    참조 1개
    public GameBaseShopImpl(ServerType type) : base(type){}
    // TODO : 서버에서 사용 될 변수 선언 및 함수 구현
}

참조 2개
public partial class GameBaseShopMasterImpl { ... }

참조 4개
public partial class GameBaseShopUserImpl { ... }

참조 2개
public partial class GameBaseShopLoginImpl { ... }

참조 2개
public partial class GameBaseShopGameImpl { ... }

참조 2개
public partial class GameBaseShopClientImpl { ... }
```

프로젝트 참조

UserObject에서 사용되는 변수 및 함수를 구현한다.

- GameBaseShopTemplate.cs

```
public partial class GameBaseShopTemplate : ShopTemplate
{
    ImplObject _obj = null;
    static GameBaseShopImpl _impl = null;

    참조 2개
    public override GameBaseUserDB CreateUserDB()
    {
        return new GameBaseShopUserDB();
    }

    참조 5개
    public override void Init(TemplateConfig config, ServerType type){...}

    참조 2개
    public override void OnLoadData(TemplateConfig config){...}

    참조 2개
    public override void OnClientCreate(ImplObject userObject){...}

    참조 2개
    public override void OnSetNewbie(ImplObject userObject)
    {
    }

    참조 2개
    public override bool OnPlayerSelectPrepare(ImplObject userObject)
    {
        Reset(userObject);
        foreach(var shopInfo in DataTable<int, ShopInfoTable>.Instance.Values)
        {
            // db에 있는지 체크
            var DBShopProductList = userObject.UserDB.GetReadUserDB<GameBaseShopUserDB>(ETemplateType.Shop)._dbSlotContainer_DBShopTable.FindAll(slot =>
            var productListTable = DataTable<int, ShopProductListTable>.Instance.Values.FindAll(productList => productList.shopId == shopInfo.id);

            foreach(var product in productListTable)
            {
                var dbShopProductTable = DBShopProductList.Find(dbProduct => dbProduct._DBData.shop_product_index == product.id);
                if (dbShopProductTable == null)
                {
                    dbShopProductTable = userObject.UserDB.GetWriteUserDB<GameBaseShopUserDB>(ETemplateType.Shop)._dbSlotContainer_DBShopTable.Insert(us
                    dbShopProductTable._DBData.shop_index = shopInfo.id;
                    dbShopProductTable._DBData.shop_product_index = product.id;
                }
            }
        }
    }
}
```

프로젝트 참조

실제 Template의 콘텐츠 제작을 해당 파일에서 진행한다. 클라이언트가 생성될 때, 처음 접속일 때, 플레이어 선택 전, 아이템을 얻었을 때 등 게임 진행 시 상점에서 할 사항들을 제작한다.

1.2.1 Common

위 사항에서 설명한 infrastructure-config.json를 파싱하여 생성된 파일, 클래스의 모임이다.

- GameBaseShopDBLoad.cs

```
public partial class GameBaseShopUserDB
{
    참조 1개
    private void _Run_LoadUser_DBShopTable(AdoDB adoDB, UInt64 user_db_key, UInt64 player_db_key)
    {
        try
        {
            QueryBuilder query = new QueryBuilder("call gp_player_dbshoptable_load(?)");
            query.SetInputParam("@p_player_db_key", player_db_key);

            adoDB.Execute(query);

            while (adoDB.RecordWhileNotEOF())
            {
                short nSlot = adoDB.RecordGetValue("slot");

                DBSlot_DBShopTable slot = _dbSlotContainer_DBShopTable.Insert(nSlot, false);

                slot._isDeleted = adoDB.RecordGetValue("deleted");
                slot._DBData.player_db_key = adoDB.RecordGetValue("player_db_key");
                slot._DBData.create_time = adoDB.RecordGetTimeValue("create_time");
                slot._DBData.update_time = adoDB.RecordGetTimeValue("update_time");

                slot._DBData.shop_index = adoDB.RecordGetValue("shop_index");
                slot._DBData.shop_product_index = adoDB.RecordGetValue("shop_product_index");
                slot._DBData.buy_count = adoDB.RecordGetValue("buy_count");
            }
            adoDB.RecordEnd();
        }
        catch (Exception e)
        {
            adoDB.RecordEnd();
            throw new Exception("[gp_player_dbshoptable_load] " + e.Message);
        }
    }
    참조 2개
    public override void LoadRun(AdoDB adoDB, UInt64 user_db_key, UInt64 player_db_key)
    {
        _Run_LoadUser_DBShopTable(adoDB, user_db_key, player_db_key);
    }
}
```

프로젝트 참조

databases 항목을 바탕으로 만들어진 테이블 로드 스토어드 프로시저를 호출하기 위한 Load 파일. DBGameUserLoad에서 LoadRun이 호출되어 진다. Load 된 DB 테이블은 해당 플레이어가 가지게 된다.

- GameBaseShopDBSave.cs

```
public partial class GameBaseShopUserDB
{
    참조 1개
    private void _Run_SaveUser_DBShopTable(AdoDB adoDB, UInt64 user_db_key, UInt64 player_db_key)
    {
        try
        {
            _dbSlotContainer_DBShopTable.ForEach((DBSlot_DBShopTable slot) =>
            {
                QueryBuilder query = new QueryBuilder("call gp_player_dbshoptable_save(?,?,?,?,?,?)");
                query.SetInputParam("@p_player_db_key", player_db_key);

                if (!slot._isDeleted)
                {
                    query.SetInputParam("@p_slot", slot._nSlot);
                    query.SetInputParam("@p_deleted", 0);
                    query.SetInputParam("@p_create_time", slot._DBData.create_time);
                    query.SetInputParam("@p_update_time", DateTime.UtcNow);

                    query.SetInputParam("@p_shop_index", slot._DBData.shop_index);
                    query.SetInputParam("@p_shop_product_index", slot._DBData.shop_product_index);
                    query.SetInputParam("@p_buy_count", slot._DBData.buy_count);
                }
                else
                {
                    query.SetInputParam("@p_slot", slot._nSlot);
                    query.SetInputParam("@p_deleted", 1);
                    query.SetInputParam("@p_create_time", DateTime.UtcNow);
                    query.SetInputParam("@p_update_time", DateTime.UtcNow);

                    query.SetInputParam("@p_shop_index", default(int));
                    query.SetInputParam("@p_shop_product_index", default(int));
                    query.SetInputParam("@p_buy_count", default(int));
                }
            });
            adoDB.ExecuteNoRecords(query);
        }, true);
        catch (Exception e)
        {
            throw new Exception("[gp_player_dbshoptable_save] + e.Message);
        }
    }
    참조 2개
    public override void SaveRun(AdoDB adoDB, UInt64 user_db_key, UInt64 player_db_key)
    {
        _Run_SaveUser_DBShopTable(adoDB, user_db_key, player_db_key);
    }
}
```

프로젝트 참조

databases 항목을 바탕으로 만들어진 테이블 세이브 스토어드 프로시저를 호출하기 위한 Save 파일. DBGameUserSave에서 SaveRun이 호출되어 진다. 플레이어가 가지고 있는 DB 테이블이 변경 혹은 추가 시 해당 사항을 DB에 저장 및 추가한다.

- GameBaseShopDBTable.cs

```
public class DBShopTable : BaseDBClass
{
    /// <summary>
    /// 파티션키_1
    /// </summary>
    public UInt64 player_db_key;
    /// <summary>
    /// 생성시간
    /// </summary>
    public DateTime create_time;
    /// <summary>
    /// 업데이트 시간
    /// </summary>
    public DateTime update_time;
    /// <summary>
    /// Shop인덱스
    /// </summary>
    public int shop_index;
    /// <summary>
    /// ShopProductList인덱스
    /// </summary>
    public int shop_product_index;
    /// <summary>
    /// 구매횟수
    /// </summary>
    public int buy_count;
    /// <summary>
    /// 초기 0개
    public DBShopTable() { Reset(); }
    /// <summary>
    /// 참조 0개
    ~DBShopTable() { Reset(); }
    /// <summary>
    /// 초기 5개
    public override void Reset(X...)
    /// <summary>
    /// 참조 2개
    public override void Copy(BaseDBClass srcDBDataX...)
}
/// <summary>
/// 참조 4개
public class DBSlot_DBShopTable : DBSlot<DBShopTable>{}
/// <summary>
/// 참조 2개
public class DBSlotContainer_DBShopTable : DBSlotContainer<DBSlot_DBShopTable, DBShopTable>{}
```

프로젝트 참조

databases 항목을 바탕으로 만들어진 테이블. 해당 테이블은 플레이어가 가지고 있으며 DB 테이블과 1:1 대응이 되어 사용된다.

- GameBaseShopModel.cs

```
public class ShopInfo : IPacketSerializable
{
    /// <summary>
    /// ShopInfo 아이디
    /// </summary>
    public int shopId = new int();
    /// <summary>
    /// 갱신 남은 초
    /// </summary>
    public int resetRemainType = new int();
    /// <summary>
    /// 상품 목록
    /// </summary>
    public List<ShopProductInfo> listProductInfo = new List<ShopProductInfo>();
    /// <summary>
    /// 재화 갱신 횟수
    /// </summary>
    public byte pointResetCount = new byte();
    /// <summary>
    /// 상품 상태 리스트
    /// </summary>
    public List<ProductStatus> ProductStatusList = new List<ProductStatus>();
    /// <summary>
    /// 참조 2개
    public void Serialize(Packet packet)
    {
        packet.Write(shopId);
        packet.Write(resetRemainType);
        int lengthlistProductInfo = (listProductInfo == null) ? 0 : listProductInfo.Count;
        packet.Write(lengthlistProductInfo);
        for (int i = 0; i < lengthlistProductInfo; ++i)
        {
            packet.Write(listProductInfo[i]);
        }
        packet.Write(pointResetCount);
        int lengthProductStatusList = (ProductStatusList == null) ? 0 : ProductStatusList.Count;
        packet.Write(lengthProductStatusList);
        for (int i = 0; i < lengthProductStatusList; ++i)
        {
            packet.Write(ProductStatusList[i]);
        }
    }
    /// <summary>
    /// 참조 2개
    public void Deserialize(Packet packet)
    {
        /// <summary>
        /// 참조 0개
        public string GetLog()
    }
}
/// <summary>
/// 참조 17개
public class ShopProductInfo
/// <summary>
/// 참조 6개
public class ProductStatus
```

프로젝트 참조

models 항목을 바탕으로 생성된 데이터 클래스. Packet에서 사용되며 패킷의 Serialize, Deserialize, Log가 지원된다.

- GameBaseShopPacket.cs

```
참조 8개
public sealed class PACKET_CG_SHOP_INFO_REQ { }
참조 8개
public sealed class PACKET_CG_SHOP_INFO_RES : PacketBaseResponse
{
    public static readonly ushort ProtocolId = 1002;
    /// <summary>
    /// 상점 정보
    /// </summary>
    public List<ShopInfo> listShopInfo = new List<ShopInfo>();
    참조 2개
    public PACKET_CG_SHOP_INFO_RES():base(ProtocolId){}
    참조 21개
    public override void Serialize(Packet packet)
    {
        base.Serialize(packet);
        int lengthlistShopInfo = (listShopInfo == null) ? 0 : listShopInfo.Count;
        packet.Write(lengthlistShopInfo);
        for (int i = 0; i < lengthlistShopInfo; ++i)
        {
            packet.Write(listShopInfo[i]);
        }
    }
    참조 21개
    public override void Deserialize(Packet packet)
    {
        base.Deserialize(packet);
        int lengthlistShopInfo = (listShopInfo == null) ? 0 : listShopInfo.Count;
        packet.Read(ref lengthlistShopInfo);
        for (int i = 0; i < lengthlistShopInfo; ++i)
        {
            ShopInfo element = new ShopInfo();
            packet.Read(element);
            listShopInfo.Add(element);
        }
    }
};
참조 8개
public sealed class PACKET_CG_SHOP_BUY_REQ { }
참조 10개
public sealed class PACKET_CG_SHOP_BUY_RES { }
```

프로젝트 참조

protocols 항목을 바탕으로 생성된 패킷. 정해진 규칙에 따라 포맷에 맞게 생성한다. Serialize, Deserialize를 규칙에 맞게 자동 생성한다.

- GameBaseShopUserDB.cs

```
public partial class GameBaseShopUserDB : GameBaseUserDB
{
    public DBSlotContainer_DBShopTable _dbSlotContainer_DBShopTable = new DBSlotContainer_DBShopTable();

    참조 2개
    public override void Copy(UserDB userSrc, bool isChanged)
    {
        GameBaseShopUserDB userDB = userSrc.GetReadUserDB<GameBaseShopUserDB>(ETemplateType.Shop);
        _dbSlotContainer_DBShopTable.Copy(userDB._dbSlotContainer_DBShopTable, isChanged);
    }
}
```

프로젝트 참조

DBLoad, DBSave에 사용되는 Slot형 DB 테이블이 선언되어 있다. Copy메소드는 테이블을 로드 또는 저장할 때 별도의 쓰레드에서 진행되기에 데이터의 오염 및 쓰레드 경합을 막기 위하여 데이터를 카피한 후 쿼리문을 별도의 DB쓰레드에서 실행한다.

- GameBaseShopProtocol.cs

```
public partial class GameBaseShopProtocol
{
    Dictionary<ushort, ControllerDelegate> MessageControllers = new Dictionary<ushort, ControllerDelegate>();

    함수 2개
    public GameBaseShopProtocol()
    {
        Init();
    }

    함수 1개
    void Init()
    {
        MessageControllers.Add(PACKET_CG_SHOP_INFO_REQ.ProtocolId, CG_SHOP_INFO_REQ_CONTROLLER);
        MessageControllers.Add(PACKET_CG_SHOP_INFO_RES.ProtocolId, CG_SHOP_INFO_RES_CONTROLLER);
        MessageControllers.Add(PACKET_CG_SHOP_BUY_REQ.ProtocolId, CG_SHOP_BUY_REQ_CONTROLLER);
        MessageControllers.Add(PACKET_CG_SHOP_BUY_RES.ProtocolId, CG_SHOP_BUY_RES_CONTROLLER);
    }

    함수 2개
    public virtual bool OnPacket(ImplObject userObject, ushort protocolId, Packet packet)
    {
        ControllerDelegate controllerCallback;
        if(MessageControllers.TryGetValue(protocolId, out controllerCallback) == false)
        {
            return false;
        }
        controllerCallback(userObject, packet);
        return true;
    }

    public delegate void CG_SHOP_INFO_REQ_CALLBACK(ImplObject userObject, PACKET_CG_SHOP_INFO_REQ packet);
    public CG_SHOP_INFO_REQ_CALLBACK ON_CG_SHOP_INFO_REQ_CALLBACK;
    함수 1개
    public void CG_SHOP_INFO_REQ_CONTROLLER(ImplObject obj, Packet packet)
    {
        PACKET_CG_SHOP_INFO_REQ recvPacket = new PACKET_CG_SHOP_INFO_REQ();
        recvPacket.Deserialize(packet);
        ON_CG_SHOP_INFO_REQ_CALLBACK(obj, recvPacket);
    }
    public delegate void CG_SHOP_INFO_RES_CALLBACK(ImplObject userObject, PACKET_CG_SHOP_INFO_RES packet);
    public CG_SHOP_INFO_RES_CALLBACK ON_CG_SHOP_INFO_RES_CALLBACK;
    함수 1개
    public void CG_SHOP_INFO_RES_CONTROLLER(ImplObject obj, Packet packet)
    {
    }
    public delegate void CG_SHOP_BUY_REQ_CALLBACK(ImplObject userObject, PACKET_CG_SHOP_BUY_REQ packet);
    public CG_SHOP_BUY_REQ_CALLBACK ON_CG_SHOP_BUY_REQ_CALLBACK;
    함수 1개
    public void CG_SHOP_BUY_REQ_CONTROLLER(ImplObject obj, Packet packet)
    {
    }
    public delegate void CG_SHOP_BUY_RES_CALLBACK(ImplObject userObject, PACKET_CG_SHOP_BUY_RES packet);
    public CG_SHOP_BUY_RES_CALLBACK ON_CG_SHOP_BUY_RES_CALLBACK;
    함수 1개
    public void CG_SHOP_BUY_RES_CONTROLLER(ImplObject obj, Packet packet)
    {
    }
}
```

프로젝트 참조

protocol 항목을 바탕으로 패킷 수신부를 생성한다. 패킷 수신 시 연결된 Controller 함수를 호출한다. 해당 함수는 Application 생성 시 정해진 규칙에 따라 사상 된다.

1.2.2 Controller

패킷 수신부에 해당하며 개발자는 만들어진 프레임에 콘텐츠 개발을 진행한다.

- CG_SHOP_BUYController.cs
- CG_SHOP_INFOController.cs

```
public void ON.CG.SHOP.INFO.REQ.CALLBACK(ImplObject userObject, PACKET.CG.SHOP.INFO.REQ packet)
{
    PACKET.CG.SHOP.INFO.RES sendData = new PACKET.CG.SHOP.INFO.RES();
    Dictionary<int, ShopInfo> shopInfoList = new Dictionary<int, ShopInfo>();
    userObject.UserDB.GetReadUserDB<GameBaseShopUserDB>(ETemplateType.Shop)._dbSlotContainer.DBShopTable.ForEach(slot =>
    {
        ShopInfo shopInfo = null;
        if (shopInfoList.TryGetValue(slot._DBData.shop_index, out shopInfo) == false)
        {
            shopInfo = new ShopInfo();
            shopInfo.shopId = slot._DBData.shop_index;
            shopInfo.resetRemainType = 0;
            shopInfo.listProductInfo = new List<ShopProductInfo>();
            shopInfo.pointResetCount = 0;
            shopInfo.ProductStatusList = new List<ProductStatus>();
            shopInfoList.Add(shopInfo.shopId, shopInfo);
        }
        ShopProductInfo productInfo = new ShopProductInfo();
        productInfo.shopProductId = slot._DBData.shop_product_index;
        productInfo.buyCount = (byte)slot._DBData.buy_count;
        shopInfo.listProductInfo.Add(productInfo);
    });
    foreach (var info in shopInfoList)
    {
        sendData.listShopInfo.Add(info.Value);
    }
    userObject.GetSession().SendPacket(sendData.Serialize());
}
참조 2개
public void ON.CG.SHOP.INFO.RES.CALLBACK(ImplObject userObject, PACKET.CG.SHOP.INFO.RES packet)...
```

프로젝트 참조

DB 테이블에 저장된 상점을 얻는 콘텐츠의 한 예이다.

1.2.3 Table

- ShopInfoTable.cs
- ShopProductListTable.cs

```
public class ShopProductListTable : ITableData
{
    public int id = new int();
    public string name = string.Empty;
    public int shopId = new int();
    public int buyType = new int();
    public int buyPrice = new int();
    public int itemId = new int();
    public int value = new int();

    참조 2개
    public void Serialize(Dictionary<string, string> data)...
```

프로젝트 참조

CSV 파일을 정해진 규칙에 따라 CommonTool에 의해 만들어진 데이터 테이블 클래스

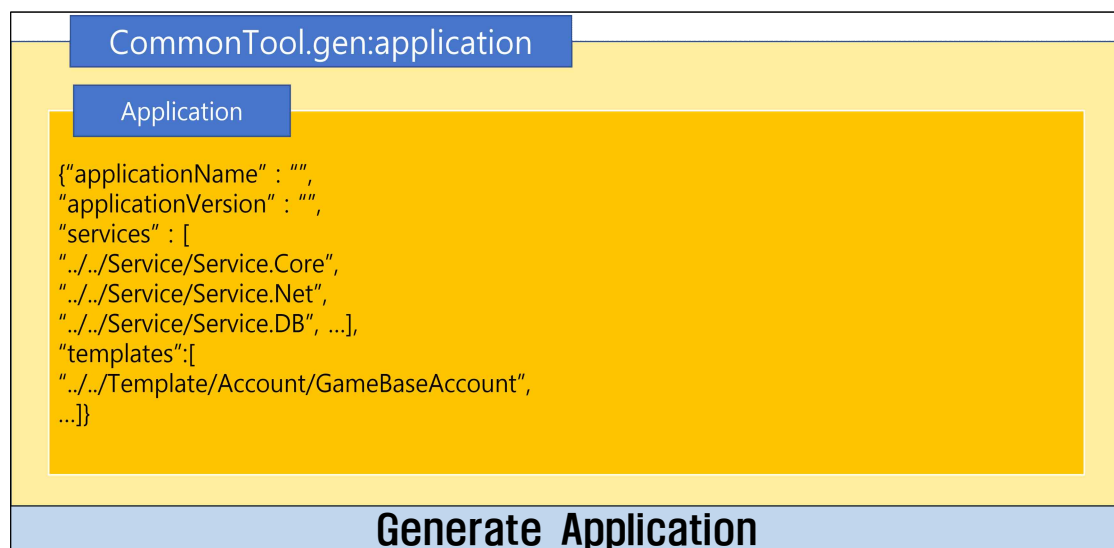
2. CommonTool

프로그래밍 업무의 일원화를 위해 코드, 프로젝트, DB 테이블, 스토어드 프로시저를 자동 생성해주는 공용툴. 중복된 작업을 줄이고 빠른 작업을 위해 제작 및 사용하였다.

2.1. Application

게임 서버의 기본 골격을 만들어 준다. 개발자는 게임 서버 제작 시 미리 제작된 service(라이브러리), template(콘텐츠)를 기재 하여 서버를 제작할 수 있다.

1) Generate Application



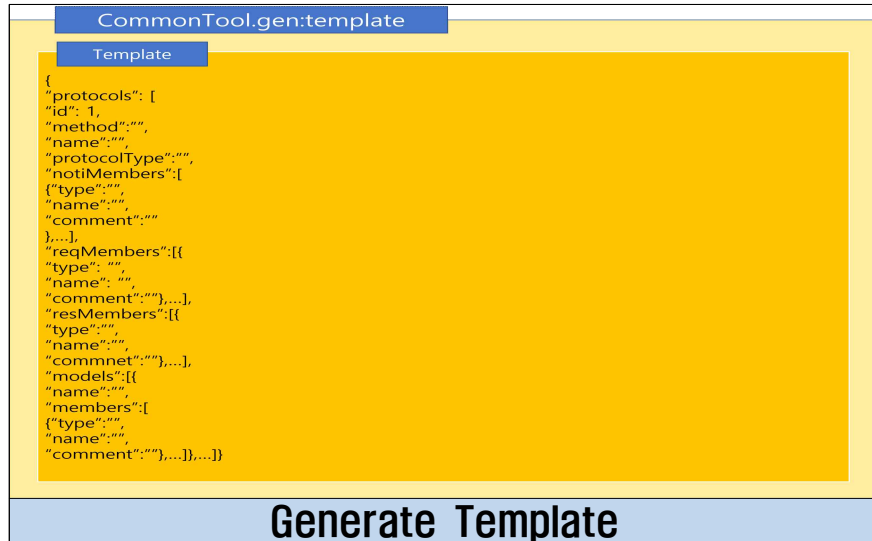
2) infrastructure-config.json 분석

항목	내용	비고
applicationName	해당 스트링을 프로젝트명으로 하여 닷넷을 이용하여 프로젝트를 생성한다.	
applicationVersion	application 버전을 기입한다.	
services	생성 된 .csproj파일에서 해당경로를 추가한다. 해당 service를 기본적으로 파일에 네임스페이스를 기입해 준다.	
templates	생성 된 .csproj파일에서 해당경로를 추가한다. 해당 template를 기본적으로 파일에 네임스페이스를 기입해 준다.	

2.2. Template

Template의 기본 프로젝트를 생성한다. 하나의 Template는 하나의 콘텐츠를 포함하며 패킷, DB, 데이터 테이블을 포함하고 있다.

1) Generate Template



2) infrastructure-config.json 분석

항목	내용	비고
protocols	<ul style="list-style-type: none"> - id 패킷 파일 생성 시 고유 아이디로 지정한다. - method "noti", "react"로 구분 된다. "noti"인 경우 접미어 "_NOTI"인 패킷과 "_NOTI_CONTROLLER"라는 컨트롤러가 생성된다. - name 패킷, 컨트롤러, 콜백 함수의 이름으로 사용되어 진다. - protocolType "admin"인 경우 서버에서만 생성 될 수 있도록 파일이 생성되어 진다. - notiMembers, reqMembers, resMembers method의 따라 분류 사용되어 진다. "noti"일 시 notiMembers가 사용되어 진다. "react"일 시 reqMembers, resMembers가 사용되어 진다. type: 패킷 문구 생성 시 중요한 부분으로 기본적으로 C# 데이터 타입을 따른다. 직렬화, 역직렬화를 지원하며, List, Dictionary또한 추가적으로 지원하고 있다. name: 패킷 변수의 이름을 기입한다. comment: 변수에 대한 주석을 기입할 수 있다. 	application, template 생성 시 패킷과 프로토콜을 연결 생성 한다.
models	<ul style="list-style-type: none"> - name Class의 명칭을 기입한다. - members 멤버변수의 정보를 기입한다. type : 패킷 문구 생성 시 중요한 부분으로 기본적으로 C# 데이터 타입을 따른다. 직렬화, 역직렬화를 지원하며, List, Dictionary또한 추가적으로 지원하고 있다. name : 멤버변수의 이름을 기입한다. comment : 멤버변수의 주석을 기입한다. 	application과 패킷에서 사용되는 Class를 기입한다. 패킷에서의 직렬화와 역직렬화, 로그출력을 지원한다.

2.3. Database, SQL

DB 테이블 생성 및 조작 코드를 Template와 데이터베이스에 추가해 준다. DB 테이블의 조작은 이벤트 방식이 아닌 업데이트 방식으로 Application에서 동작하고 있으며 서버에서 조작된 DB 데이터는 자동으로 DB에 저장되도록 설계되었다.

1) Generate Database, SQL

CommonTool.gen:database

CommonTool.gen:sql

DataBase

SQL

```

{
  "databases":[
    {
      "databaseType":"","/game, global, log
      "tableType": "",
      "tableName": "",
      "partitionKey_1":"user_db_key",
      "partitionKey_2":"player_db_key",
      "members":[
        {
          "type":"","
          "name":"","
          "comment":"","
        },...],...}]
      }
    }
  ]
}

```

Generate Database, SQL

2) infrastructure-config.json 분석

항목	내용		비고	
databases	<ul style="list-style-type: none">- databaseType 크게 단일DB로 되어 있는 “global” 데이터베이스, 유저의 고유값에 따라 분산 되어 있는 “global”, 로그정보를 저장하느 “Log”로 구분 되어 있다.- tableType 해당 key에 “slot”여러 레코드가 존재, “base” 단일 레코드만 존재(tableType에 따라 로직이 달라짐)- tableName 테이블 명을 기입. 데이터베이스에 테이블 명을 지정해 준다.- partitionKey_1 유저의 유일 키를 기입 일반적으로 global 데이터베이스의 유저키를 기입한다.- partitionKey_2 캐릭터의 유일 키를 기입, 일반적으로 global 데이터베이스의 player키(캐릭터)를 기입한다.		셋팅된 기본 데이터베이스를 기반으로 테이블의 생성, 코드 상 쿼리(로드, 세이브, 생성)를 생성한다.	
	- members	- type		일반 적으로 C#기준으로 변수의 타입을 입력한다. 테이블 생성, 쿼리 생성 시 크기에 맞는 타입으로 자동 변환 해준다.
		- name		필드의 명, 스토어드 프로시저의 파라미터의 이름으로 사용되어 진다.
		- comment		테이블의 주석과 테이블 클래스의 주석을 기입한다.

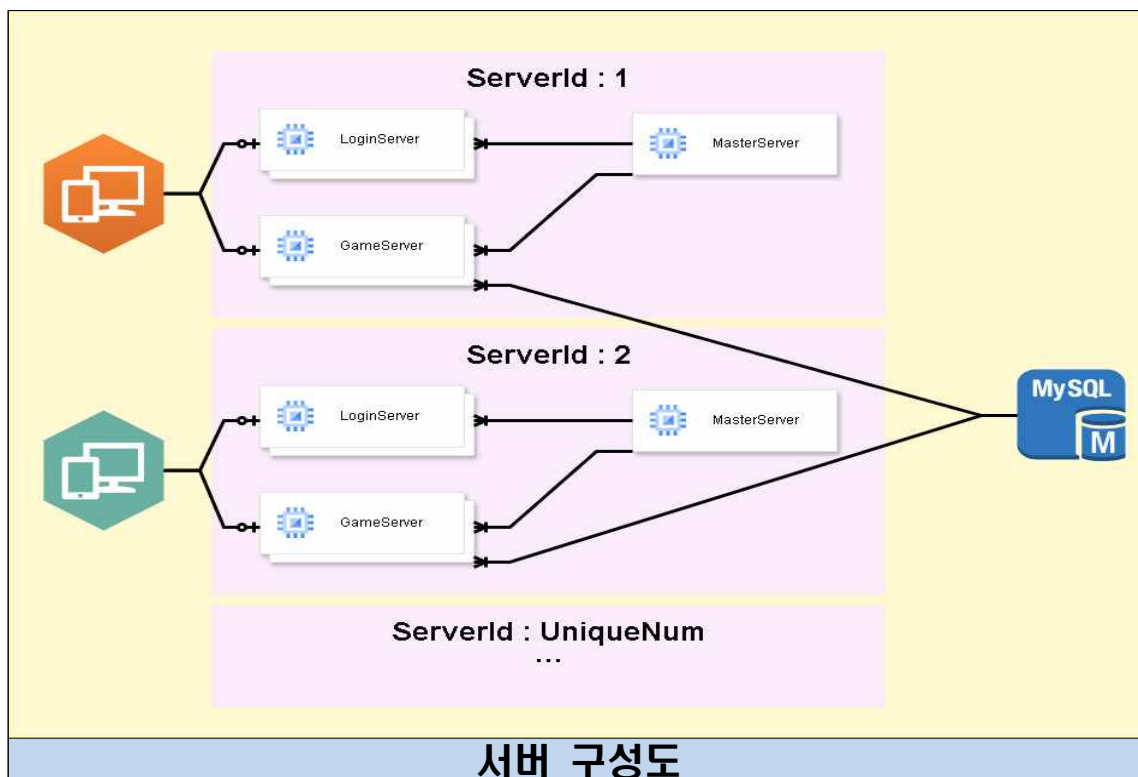
2.4. Table, Controller

CSV를 읽어 들여 데이터 테이블을 생성할 수 있는 기능이 있다. 또한 Template의 Controller와 Application에 연결할 수 있는 Controller파일 생성 기능이 있다. Application 생성 시 자동으로 생성되지만 Client또한 동일한 CSV와 Controller를 사용할 경우가 있을 것을 대비하여 별도의 기능으로 추가하였다.

3. TestProject

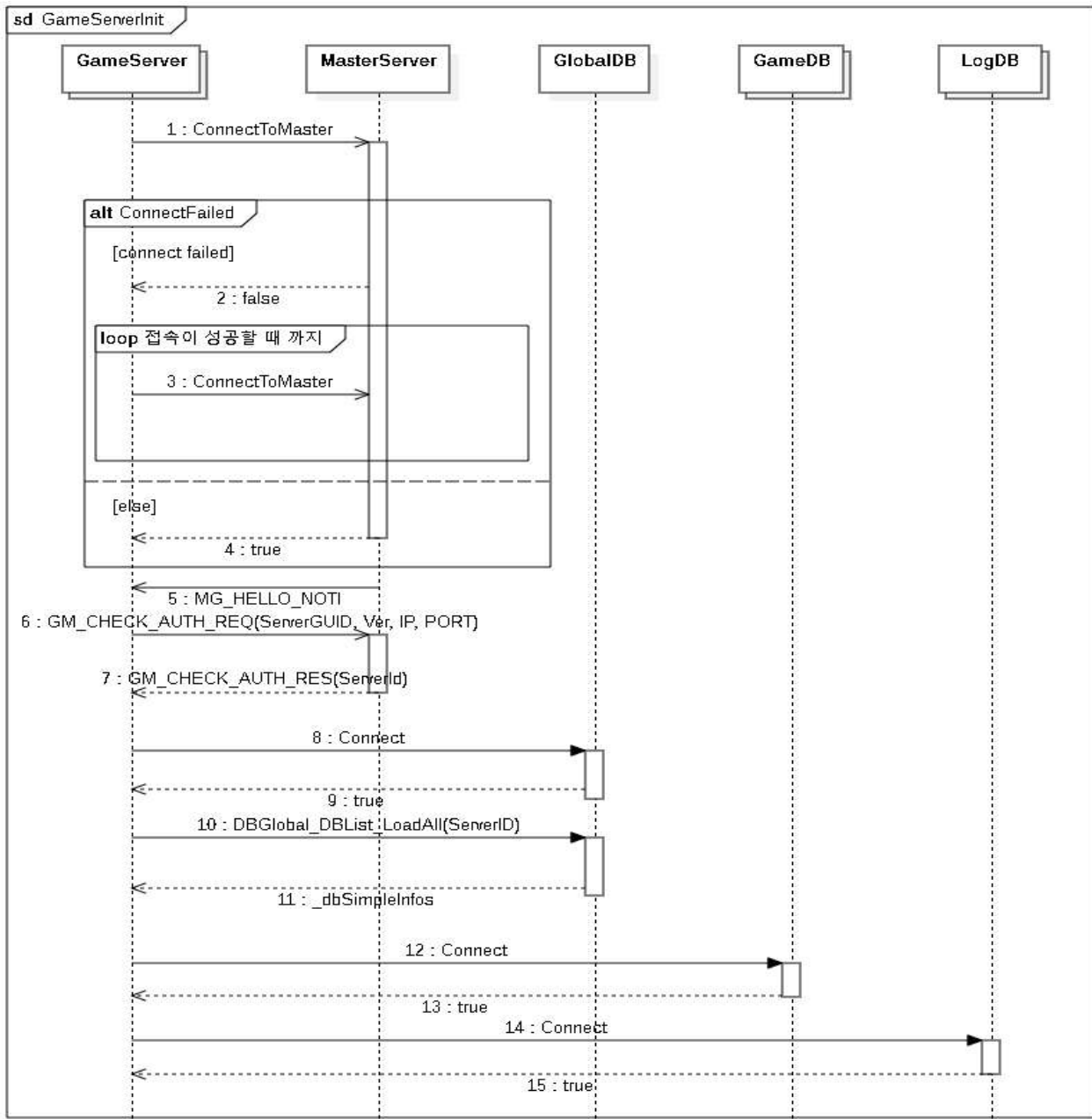
해당 프로젝트는 GameBaseServer와 CommonTool을 이용하여 공용 게임 서버 제작을 목표로 한다. CommonTool을 이용한 서버 제작의 기능 테스트와 디버깅을 위하여 해당 프로젝트를 제작하였다. ServerId는 하나의 게임을 나타내며 ServerId를 기준으로 플레이어, 유저가 구분되어진다. 동일한 데이터 테이블을 사용한다는 기준으로 개발을 목표로 제작하였으며 동일한 서버를 가지고 다양한 게임을 개발할 수 있는 것을 목표로 구성하였다.

1) 서버 구성도



현재 추구하는 서버 구성도이다. 하나의 원 DB로 여러 개의 게임을 동시에 처리할 수 있는 것을 목표로 구성하였다. 각 서버군은 하나의 게임을 담당하며 하나의 빌드로 여러 게임을 런칭할 수 있도록 데이터와 프로토콜, DB 테이블 등 각 기능을 일원화하여 제작하고자 하였다. 세션과 서버를 관리하는 MasterServer를 제외한 나머지 서버는 이중화하여 장애에 대비했다.

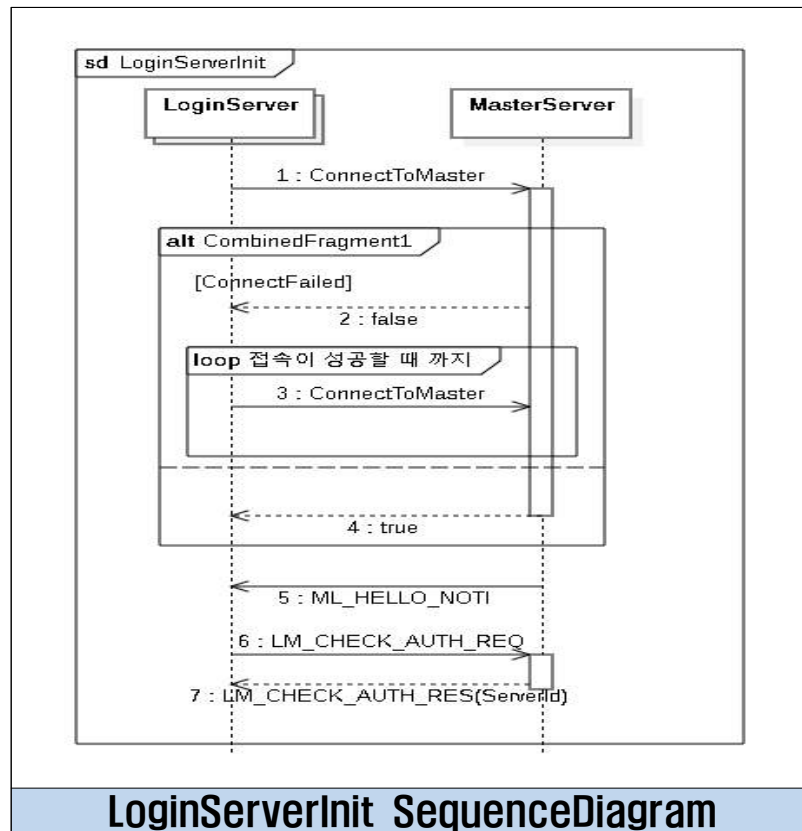
1) GameServerInit



GameServerInit SequenceDiagram

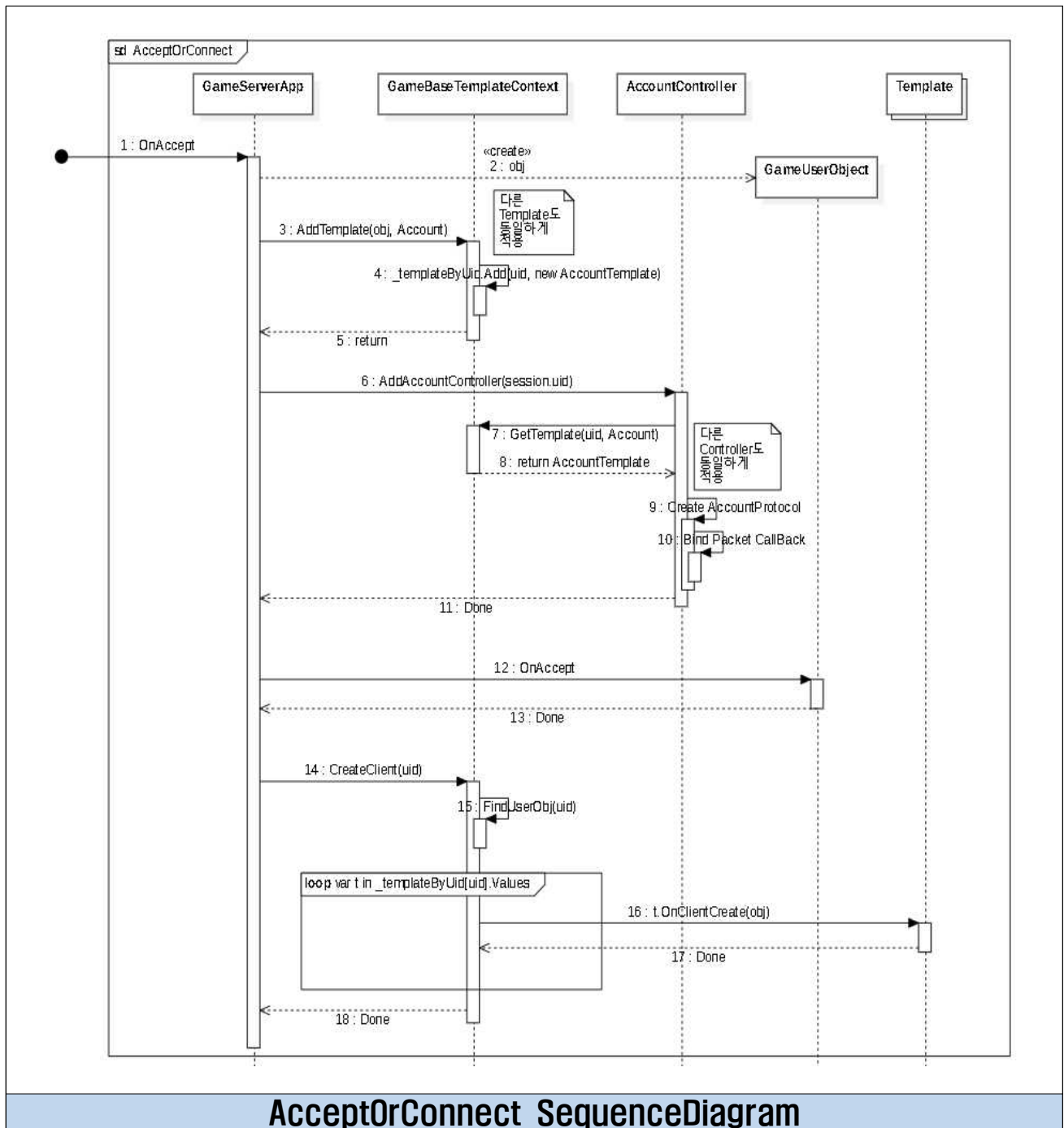
GameServer 구동 시 준비절차를 보여주고 있다. 게임 서버는 구동 시 중앙 원서버인 MasterServer에 인증 절차를 진행한다. MasterServer는 인증이 완료되면 Game Server에 고유번호를 발급해 준다. 이후 게임 서버는 GlobalDB에 접속하여 ServerID를 파라미터로 GlobalDB에 DB 리스트 테이블을 로드한다. 이후 DB 리스트의 DB에 접속을 수행하고 준비를 완료한다.

2) LoginServerInit



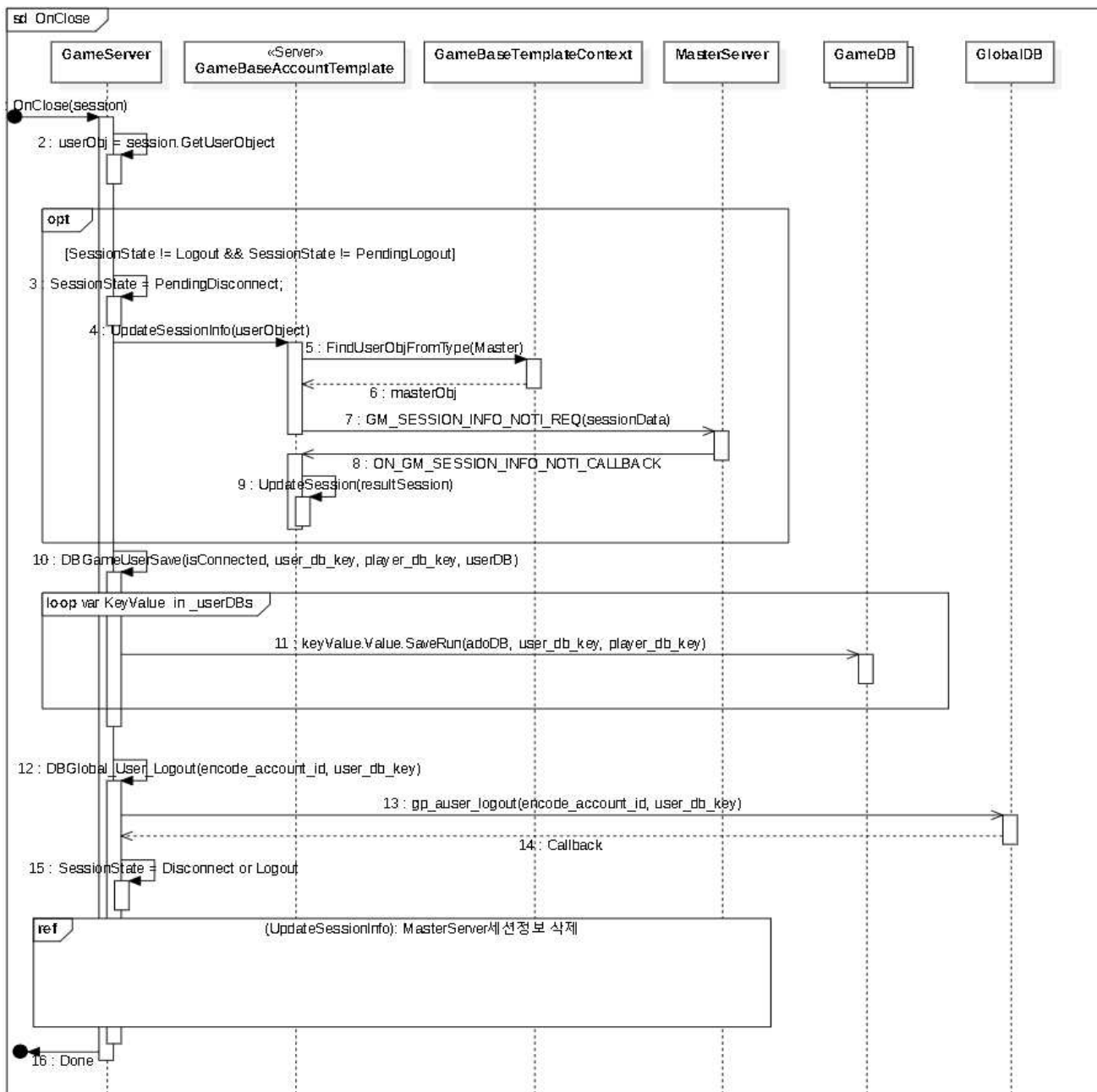
GameServer 구동 시 준비절차를 보여주고 있다. LoginServer는 MasterServer와 인증 절차를 진행 후 준비를 완료한다.

3) AcceptOrConnect



Application의 Accept, Connect 이벤트가 발생 시 Application이 어떻게 동작하는지 보여준다. 해당 과정에서 중요한 것은 Template, Controller의 생성 및 등록이다. 유저객체가 생성되면 해당 객체에 이용하는 Template(콘텐츠), Controller(패킷 처리)를 등록한다. 이후 유저가 생성되었다는 액션을 각 Template를 순회하면서 OnClientCreate를 호출한다.

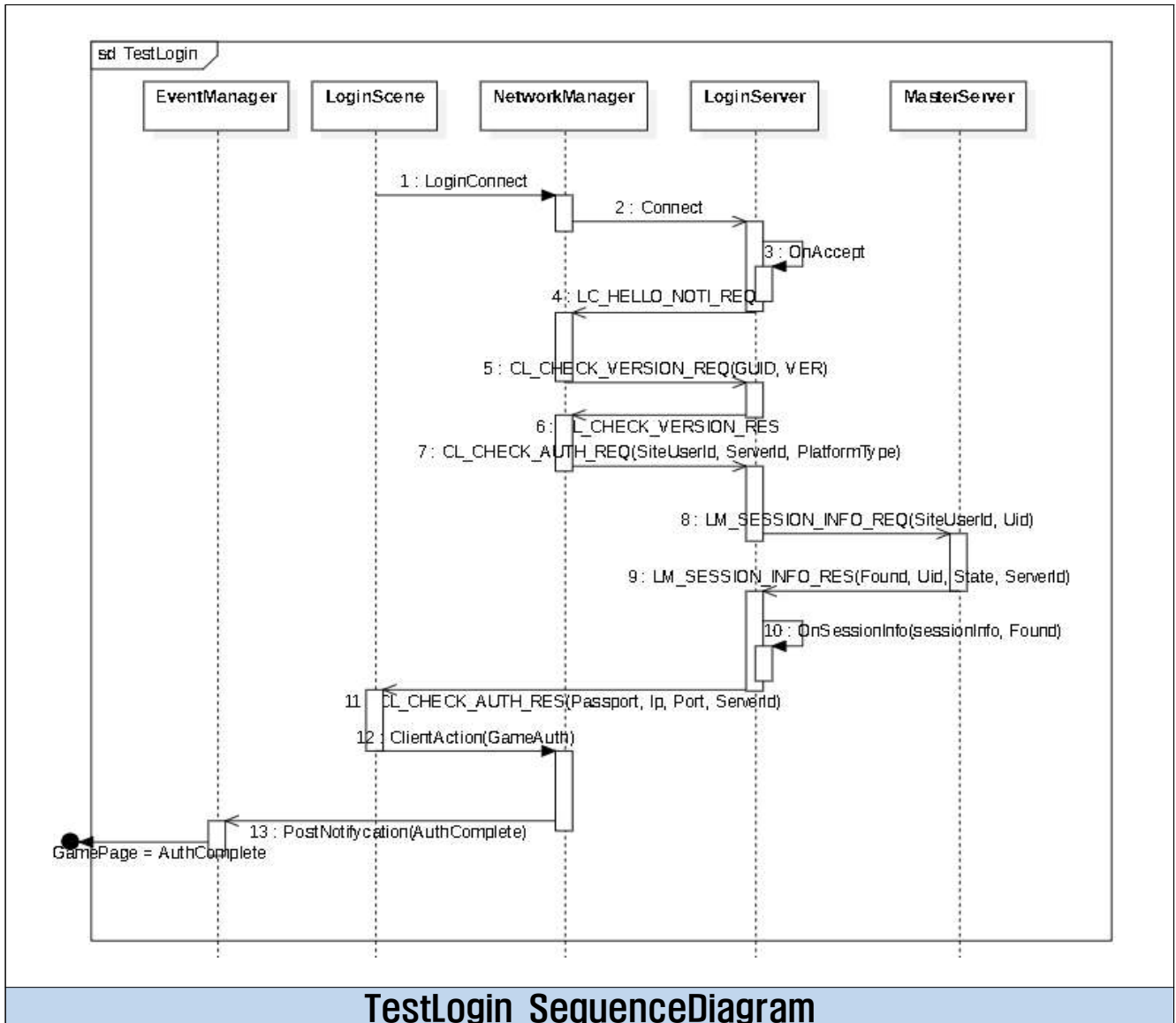
4) OnClose



OnClose SequenceDiagram

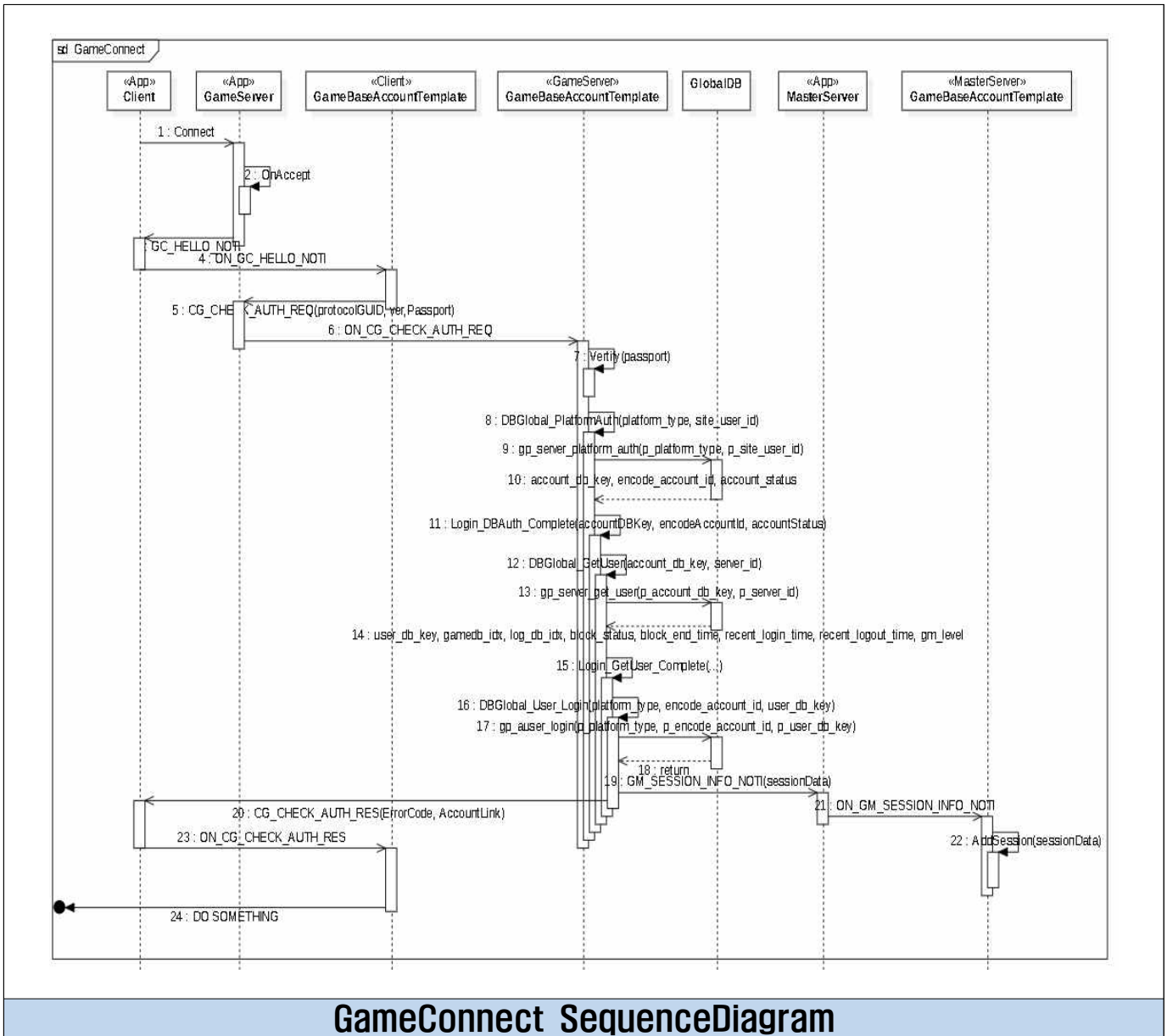
GameServer Application에서 접속종료 절차를 보여주고 있다. 유저가 접속 종료 시 접속 종료 대기 상태로 변경 후 세션 정보를 MasterServer에 업데이트한다. 이후 각 Template의 관리하는 GameDB테이블을 유저의 마지막 정보로 업데이트한다. 이후 GlobalDB에 로그아웃 상태로 테이블 업데이트한 후 해당 쿼리가 완료 처리가 되면 MasterServer에서 해당 정보를 삭제한다.

5) TestLogin



Client가 LoginServer와의 인증 절차를 보여주고 있다. 1. 패킷의 버전 체크를 진행한다. 서버와 클라이언트의 패킷 버전을 일치하기 위한 진행 상황이다. 2. 계정 고유키, 게임 고유 아이디, 플랫폼 타입을 로그인 서버에 전달한다. 3. LoginServer는 MasterServer에 세션의 정보를 넘겨준다. 초대, 재접속, 중복 로그인을 위한 절차로 세션의 정보를 업데이트한 후 결과와 접속해야 하는 GameServer의 정보를 LoginServer로 전달한다. LoginServer는 해당 결과를 Passport로 만든 후 Client로 전달한다.

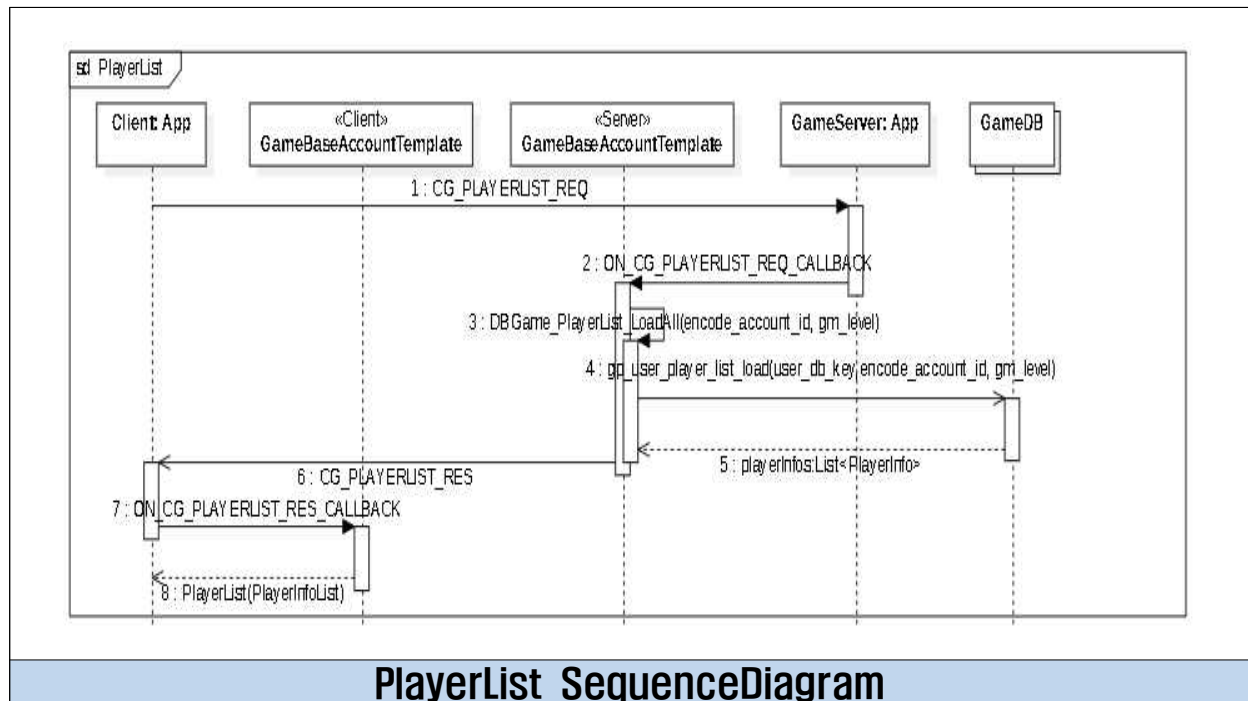
6) GameConnect



Client의 GameServer와의 인증 절차를 보여주고 있다.

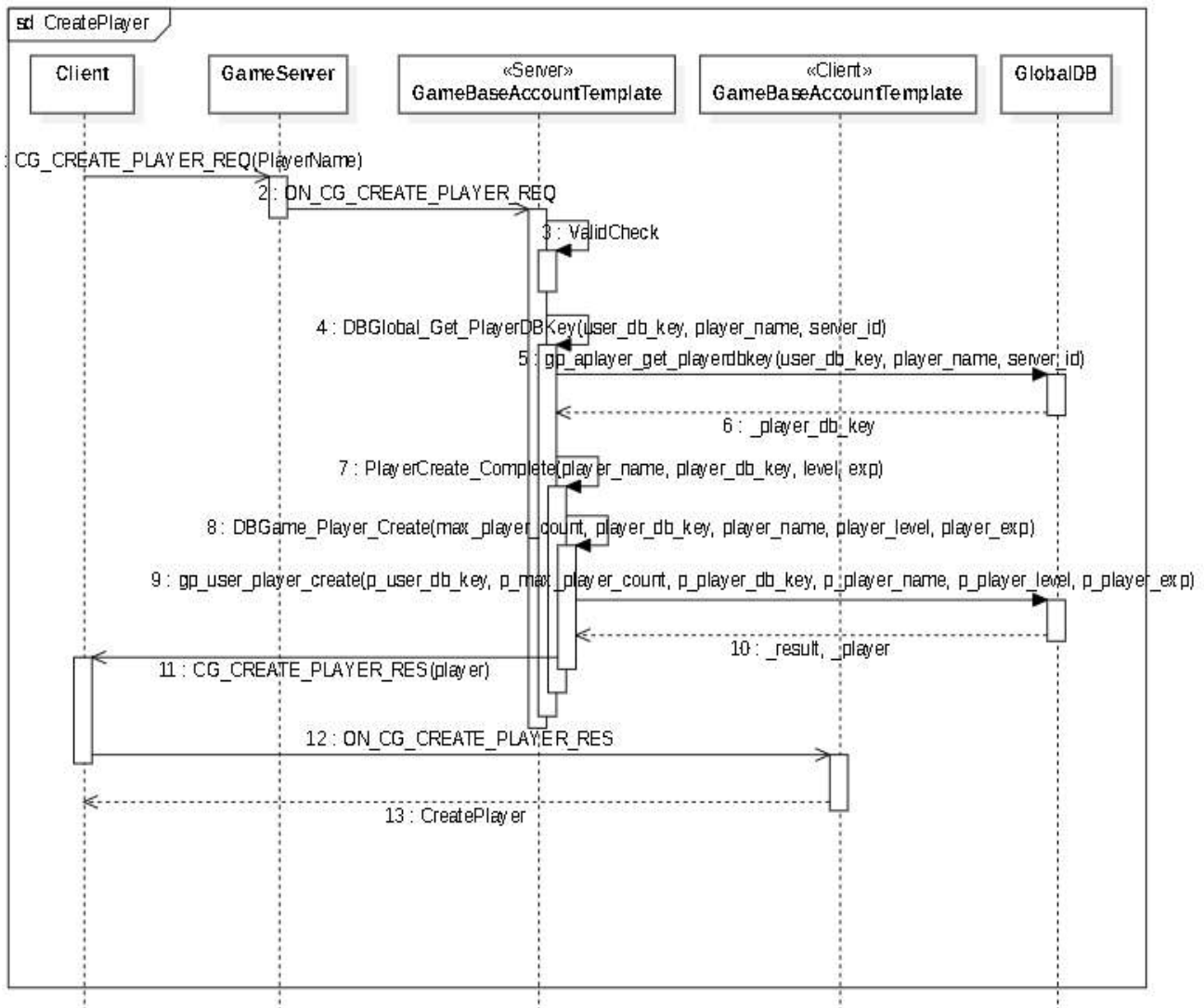
1. LoginServer에서 전달받은 IP, Port로 게임 서버에 접속한다.
2. Client는 Passport와 프로토콜 버전, 앱 버전을 서버에 전송한다.
3. 플랫폼별 SiteId에 대한 Account 테이블을 생성한다. 이를 바탕으로 User 테이블을 생성한다. 계정 연동 혹은 변경을 위해 Account와 User는 별도의 테이블로 서로 링크되어 연결되어 진다.
4. 로그인이 완료되면 MasterServer에 세션의 정보를 업데이트 혹은 생성한다.

7) PlayerList



유저 DB키를 기준으로 생성된 플레이어 리스트를 GlobalDB에서 불러온다.

8) CreatePlayer

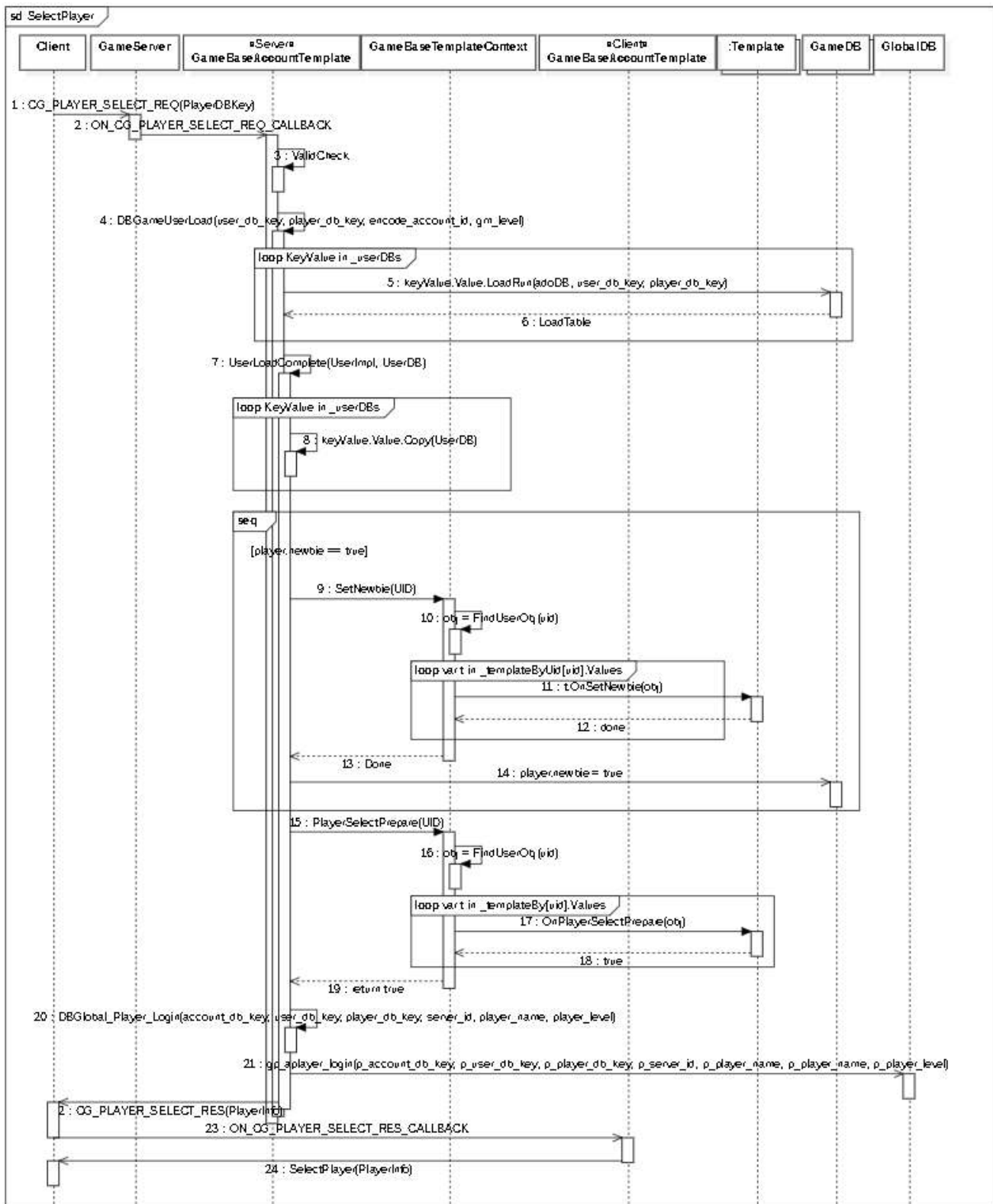


CreatePlayer SequenceDiagram

플레이어 생성 절차를 보여주고 있다. 플레이어는 유저별 1:N으로 생성될 수 있다.

1. 유저 DB키를 기준으로 playerDBKey의 Serial 번호를 유저 테이블에 생성 및 기록한다.
2. 생성된 playerDBKey로 Player를 생성한다.

9) SelectPlayer

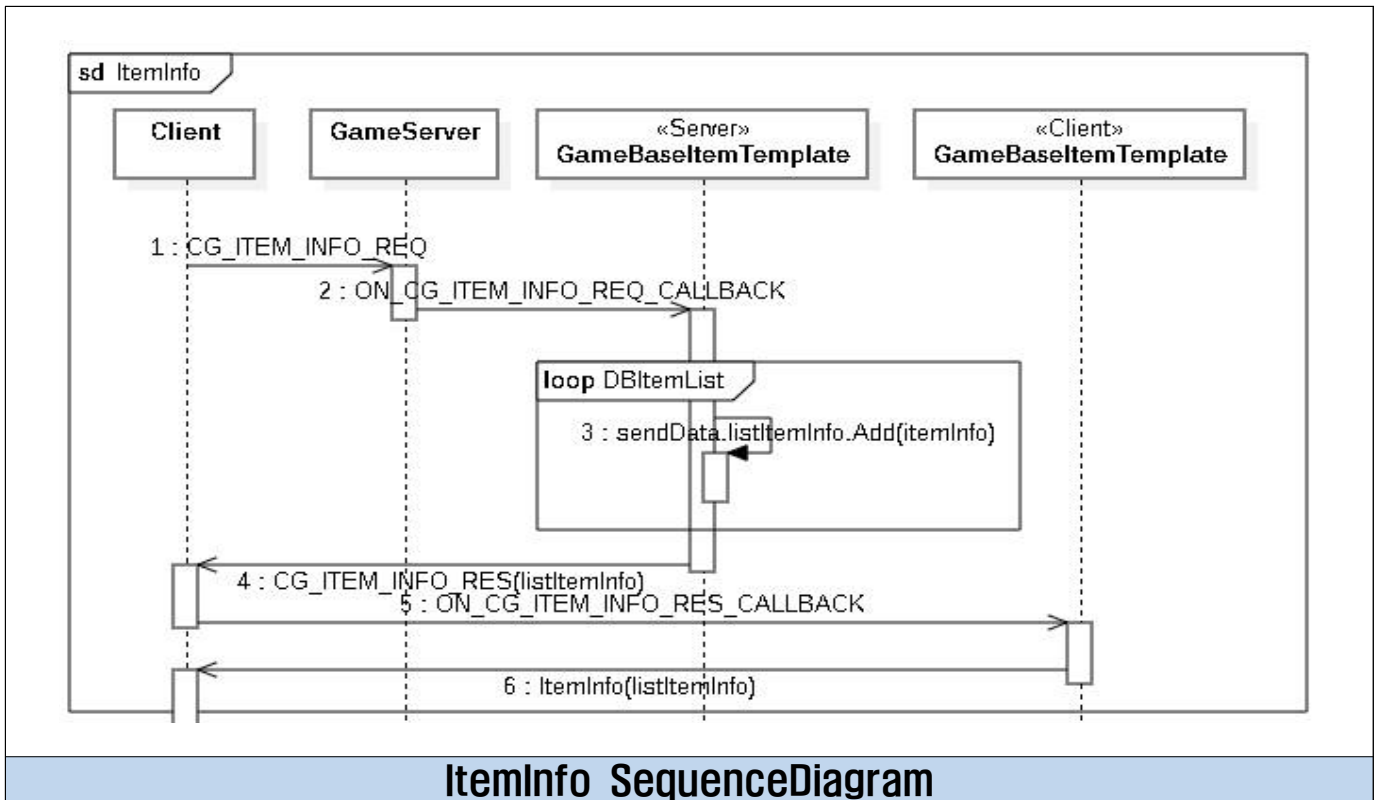


SelectPlayer SequenceDiagram

로드된 플레이어 리스트에서 플레이어 선택을 보여준다. 플레이어 기준으로 게임의 데이터가 준비되기에 플레이어 데이터 준비까지 다소 복잡하다.

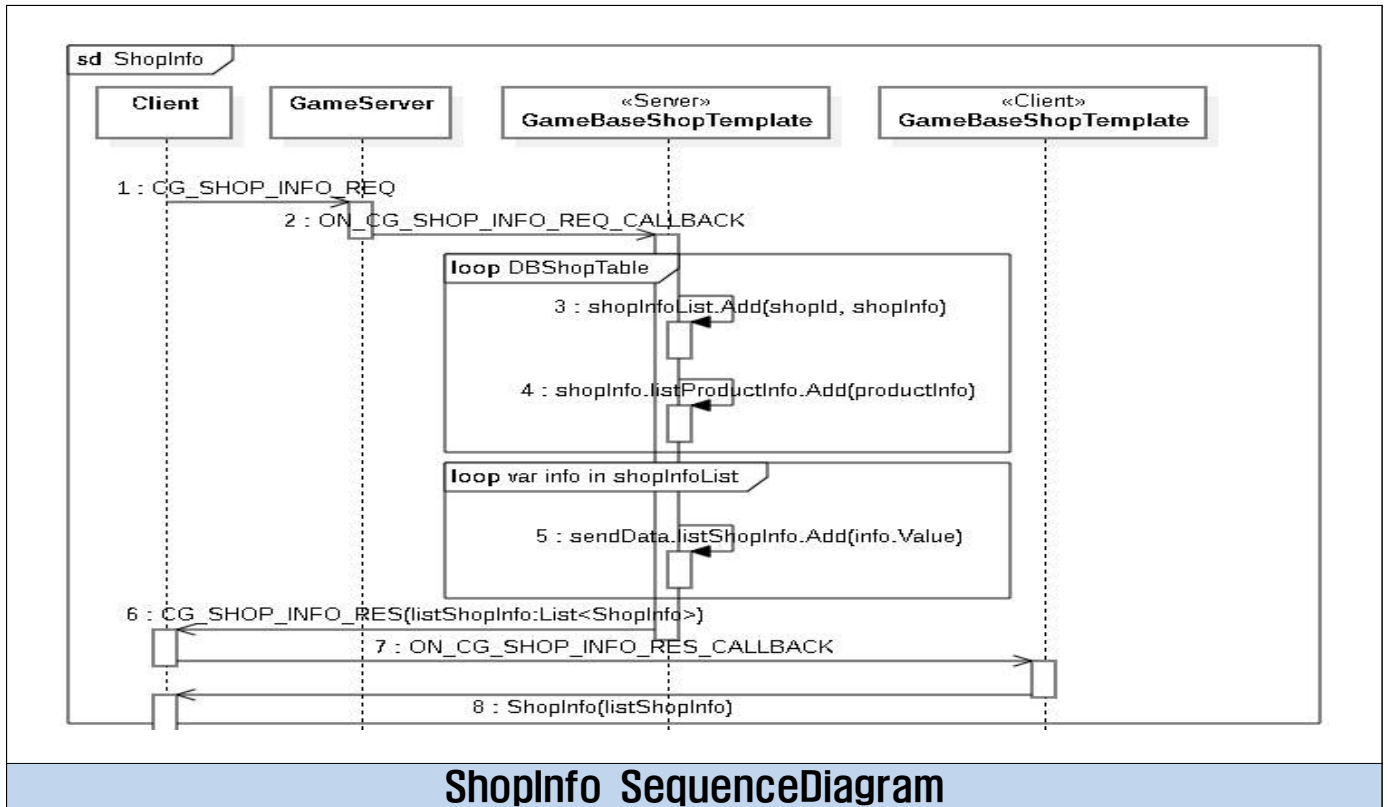
1. Client는 선택한 PlayerDBKey를 서버로 전송한다.
2. GameServer는 UserDBKey, PlayerDBKey를 기준으로 DB테이블 로드를 준비한다. 각 Template 당 생성된 UserDB의 쿼리 들을 실행 및 Template에 복사하여 DB 테이블을 각 Template에서 관리하에 둔다.
3. 처음 생성된 플레이어인지 체크를 한다. 만약 처음 생성된 플레이어라면 각 Template의 OnSetNewbie를 호출하여 처음 생성된 플레이어와 관련된 콘텐츠 작업을 할 수 있게 한다. 이후 PlayerDB테이블의 Newbie항목을 false로 변경해 준다.
4. Template의 OnPlayerPrepare를 호출한다. Player 선택 전 콘텐츠의 업데이트, 데이터 세팅 등 콘텐츠 성격에 맞는 작업을 게임 전 수행 할 수 있다.
5. 플레이어의 로그인 상태를 현재 시각 기준으로 DB에 기록한다.
6. 플레이어 선택이 완료되면 Client에 응답을 보낸다.

10) ItemInfo



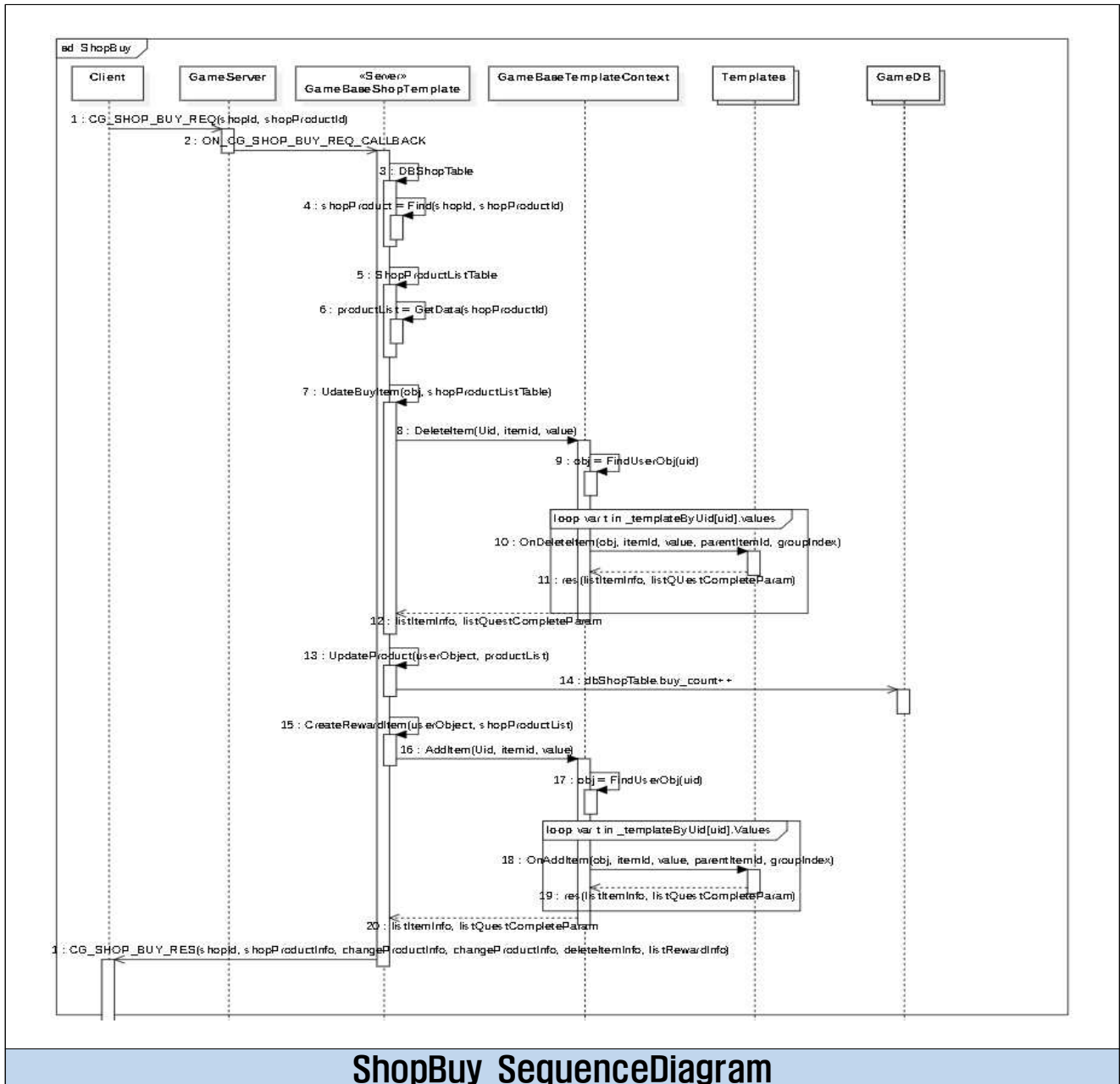
플레이어의 아이템 정보를 얻는 시퀀스이다. Client에 아이템 요청이 오면 서버의 ItemTemplate에 로드된 아이템 정보DB 리스트를 순회, 아이템 정보를 생성하여 Client에 응답을 한다.

11) ShopInfo



상점 정보를 로딩한다. 현재는 구현되어 있지 않지만, 갱신 상점 정보, 추가 상점 정보를 DB에 업데이트한 후 DB 테이블을 순회하며 상점 정보를 생성, Client에 전달한다.

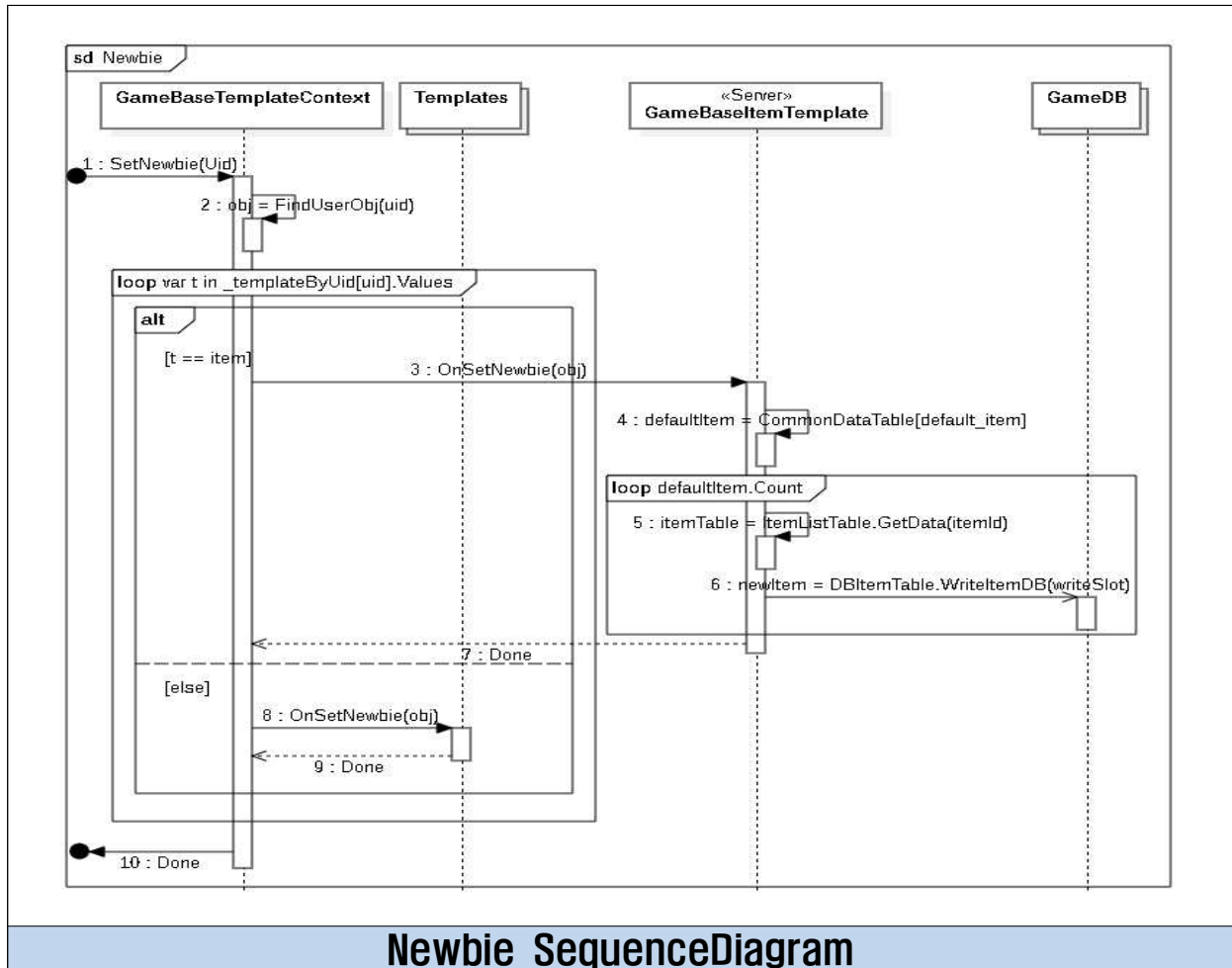
12) ShopBuy



상점 구매 로직을 보여주고 있다.

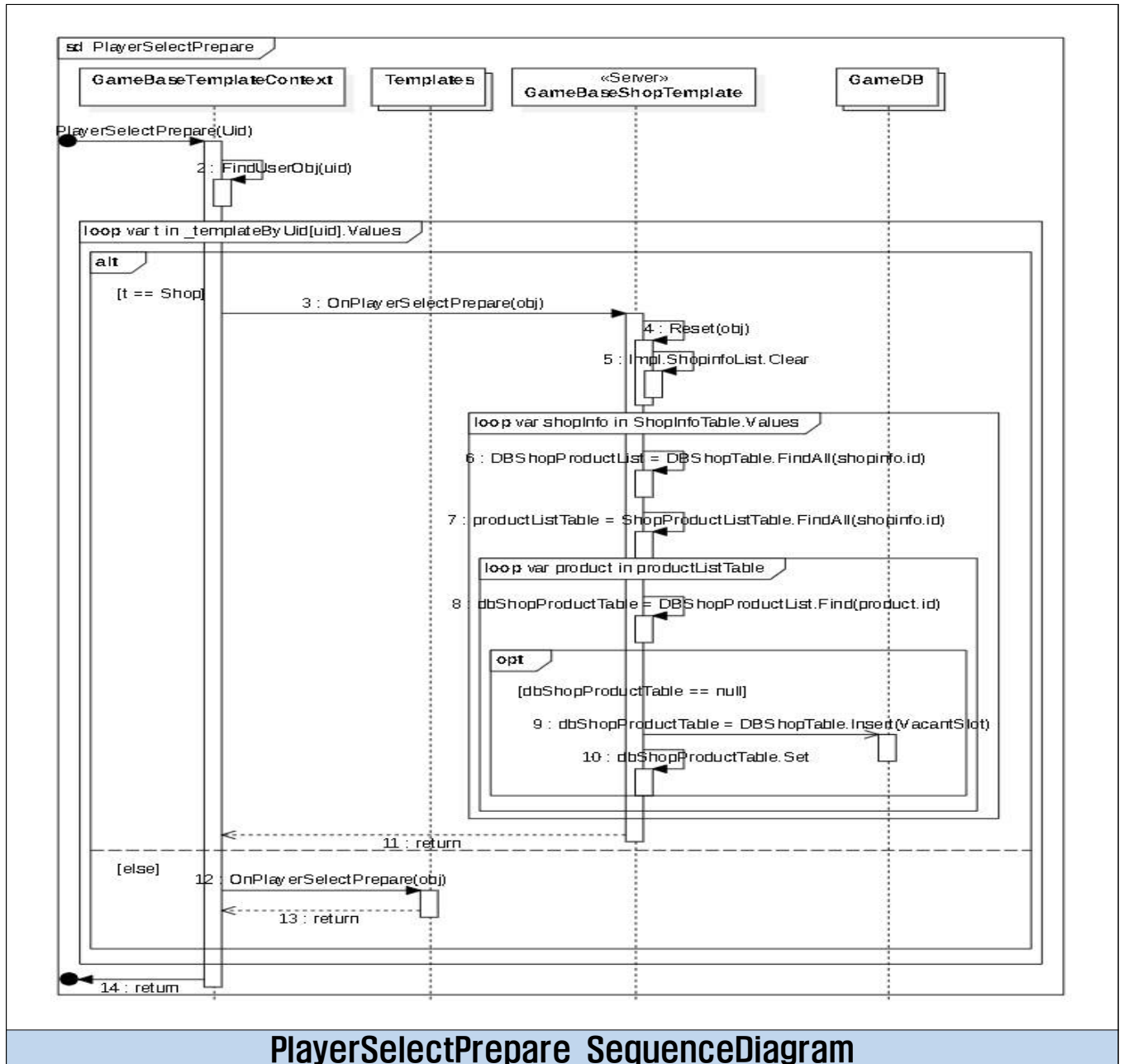
1. 상점DB테이블의 Client의 구매 요청 품목을 확인한다.
2. 구매 재화를 업데이트한다. 상점에서는 아이템 접근 권한이 없으므로 각 Template의 OnDeleteItem을 호출하여 결과를 얻는다.
3. 상점의 갱신 사항을 DB에 기록한다.
4. 보상 아이템을 업데이트한다. 상점에서는 아이템 접근 권한이 없으므로 각 Template의 OnDeleteItem을 호출하여 결과를 얻는다.
5. 보상 아이템, 변경 아이템, 상점 갱신 사항을 Client에 전달한다.

13) Newbie



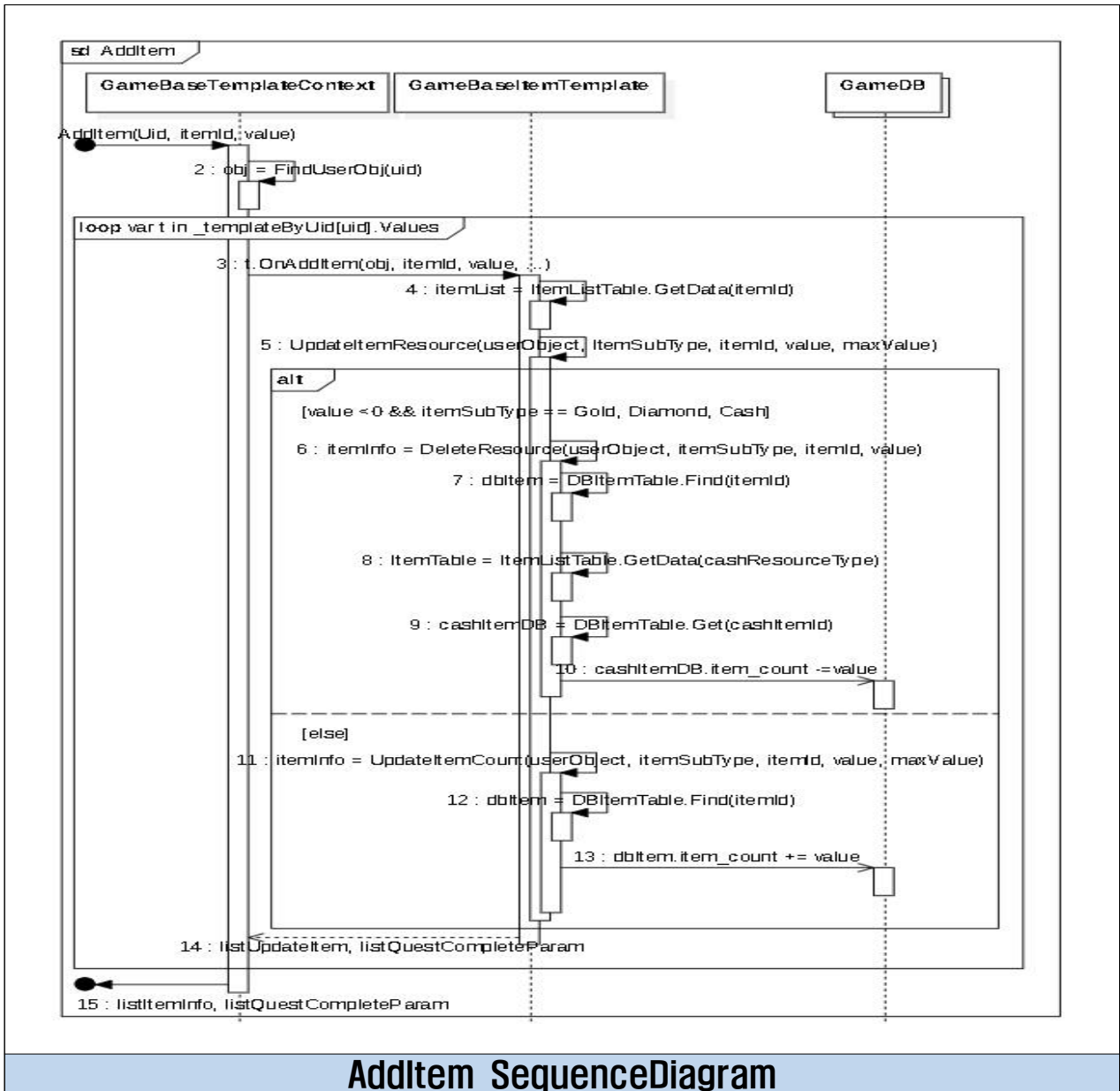
위에서 플레이어가 처음 생성 시 각 Templates에 OnSetNewbie 호출 시 한 예로 ItemTemplate에서 어떤 로직이 처리되는지 보여주고 있다. ItemTemplate은 아이템을 관리하는 Template로 플레이어 생성 시 기본 아이템 데이터 테이블을 참조하여 DB 테이블에 아이템 생성 쿼리를 호출한다.

14) PlayerSelectPrepare



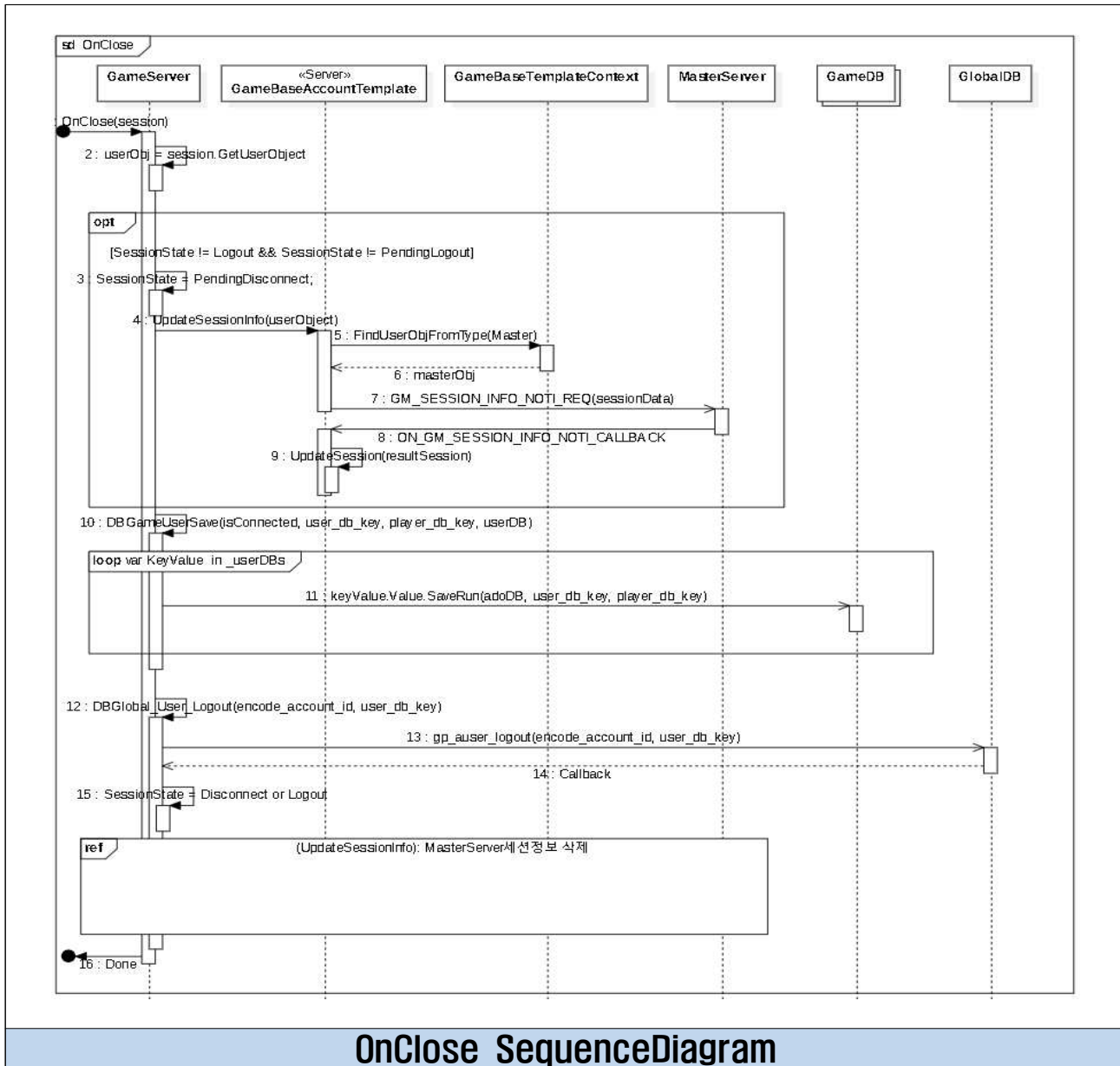
플레이어 선택 전 각 Template에서는 콘텐츠에 맞는 로직을 수행해야 한다. 한 예로 ShopTemplate은 기간제 상점을 초기화하고 추가 상점을 DB에 기록하는 초기화 로직을 수행하게 된다.

15) AddItem



ItemTemplate에서 OnAddItem 혹은 OnDeleteItem 호출 시 로직을 표현하였다. 재화 감소 처리인 경우 콘텐츠에 따라 다르겠지만 현재는 캐시 재화를 DB 테이블에 차감 후 업데이트한다. 이후 나머지 차감 분을 제거하는 순으로 진행되어 진다. OnAddItem인 경우 예외 처리 사항을 제외한 후 바로 아이템 DB 테이블에 추가분을 추가한다.

16) OnClose



유저가 게임을 종료했을 때의 로직을 보여준다.

1. GameServer는 OnClose가 호출되면 MasterServer에 세션의 상태를 알린다.
2. DBGameUserSave를 호출하여 각 Template의 SaveRun을 호출하여 Template의 관리하에 있는 DB 테이블을 저장한다.
3. 이후 유저의 로그아웃 정보를 저장하고 DB 저장 완료 통지가 오면 MasterServer에서 세션의 정보를 삭제한다.
4. DB를 저장하는 Thread가 유저 정보를 통해 해시 되고 지정되어 있어 순차적으로 안전하게 로그아웃 처리가 된다.

III. 별첨

1) GitHub

GameBaseServer : <https://github.com/AshOne91/game-server.git>

CommonTool : <https://github.com/AshOne91/common-tool.git>

2) 동영상

게임 서버 테스트 링크 : https://youtu.be/NMV_BbQf7tM?si=7bOAtM8G9e-syxn0