






PORTFOLIO

이우상

lus0506@gmail.com

<https://velog.io/@lus0505>

```
void MyProjcets::introduce( )  
{
```

- 프로젝트 이름 : CESP(Crypto Exchange Strategy Platform)
- 설명 :
 - * 시세 서버 : 클라이언트의 구독 관리 및 각 거래소의 종목 시세를 보내줌
 - * 모니터링 클라이언트 : 시세 서버로 부터 각 거래소의 종목을 구독하여 시세를 받고, 주문 서버로 종목의 주문, 손익 표시
- 역할 : 서버, 클라이언트
- 개발 언어 및 라이브러리 : C++, Qt
- 개발 환경 :  Visual Studio 2019  Qt Creator  SourceTree

```
}
```

```
void MyProjcets::tech( )
{
```

TR헤더 구조(주문 시세 공통 헤더)

전체 TR 구조				
<----- TR헤더(6Byte) ----->		<----- 데이터영역(Length) ----->		
		<--- 데이터헤더--->		<----- 데이터 ----->
Type(1)	Check Byte(1)	Length(4)	길이가변(아래 데이터헤더구조 참조)	
Q : RQ/RP		Max 4096	Data[Length-데이터헤더]	
R : RealTime(Push)				

데이터헤더 구조

Request Header				
TR Code(5)	ReqNo(7)			12byte
Response Header				
TR Code(5)	ReqNo(7)	ErrorNo(5)		19byte
Realtime Header				
TR Code(5)	SeqNo(9)	Server Time(10)		24byte
		'HHMMSSMMSS'		
TR 정의				
SI001	로그인			
SI002	종목구독			
SI901	실시간현재가			
SI902	실시간호가			
SI903	실시간거래정보			

시세 서버 – 모니터링 클라이언트 패킷 설계

모니터 클라이언트 - ver 1.4.5

모니터 주문 시세 잔고/손익

메인 클래스 ALL 서버 클래스 ALL 로그 레벨 ALL OMS 10:49:43 IMS 10:49:43

메인 클래스	서브 클래스	로그 레벨	로그 시각	로그 내용
IMS	IMS	INFO	2021-01-07 10:49:43	jykim - Client ProcessLog Subscribe
OMS	OMS	INFO	2021-01-07 10:49:42	전략 구독
OMS	OMS	DEBUG	2021-01-07 10:49:42	cli Packet : Q10085({"trCode":"OS002","reqNo":"00002","data":...
OMS	OMS	DEBUG	2021-01-07 10:49:42	balance recovery count tot : 15 page : 2
OMS	OMS	INFO	2021-01-07 10:49:41	전략별 잔고 조회
OMS	OMS	DEBUG	2021-01-07 10:49:41	cli Packet : Q10057({"trCode":"OS203","reqNo":"00002","data":{"stgID":"ALL"}})
OMS	OMS	DEBUG	2021-01-07 10:49:41	pos recovery count tot : 15 page : 2
OMS	OMS	INFO	2021-01-07 10:49:41	전략별 포지션 조회
OMS	OMS	DEBUG	2021-01-07 10:49:41	cli Packet : Q10057({"trCode":"OS202","reqNo":"00002","data":{"stgID":"ALL"}})
OMS	OMS	INFO	2021-01-07 10:49:41	전략별 주문리스트 조회
OMS	OMS	DEBUG	2021-01-07 10:49:41	cli Packet : Q10068({"trCode":"OS201","reqNo":"00002","data":...
OMS	OMS	INFO	2021-01-07 10:49:41	마스터 계좌 조회
OMS	OMS	DEBUG	2021-01-07 10:49:41	cli Packet : Q10034({"trCode":"OS104","reqNo":"00002"})
OMS	OMS	INFO	2021-01-07 10:49:41	전략 리스트 조회
OMS	OMS	DEBUG	2021-01-07 10:49:41	cli Packet : Q10034({"trCode":"OS103","reqNo":"00002"})
IMS	IMS	INFO	2021-01-07 10:49:41	jykim - Client ProcessLogList Request
IMS	IMS	INFO	2021-01-07 10:49:41	jykim - Client Login
OMS	OMS	INFO	2021-01-07 10:49:41	종목 리스트 조회
OMS	OMS	DEBUG	2021-01-07 10:49:41	cli Packet : Q10060({"trCode":"OS102","reqNo":"00002","data":{"exchange":"ALL"}})
OMS	OMS	INFO	2021-01-07 10:49:41	거래소 리스트 조회
IMS	IMS	INFO	2021-01-07 10:49:41	SocketID : 2008 - Client Connect
OMS	OMS	DEBUG	2021-01-07 10:49:41	cli Packet : Q10034({"trCode":"OS101","reqNo":"00001"})

시세그리드

거래소 ALL 마켓 ALL

구독	거래소	마켓	종목	현재가	등락	등락률(%)	매도호가	매수호가	거래량	시가	고가	저가	Exchange	Exif	InfoServer	Local
구독취소	HUOB	F	ETH210625	1320.975	190.415	0.17%	1321.040	1320.746	10	1194.225	1339.820	1146.673	10:55:12	10:55:12.140	10:55:12.141	10:55:13
구독취소	HUOB	F	ETH210108	1206.161	160.068	0.15%	1206.100	1205.921	10	1107.495	1229.615	1061.250	10:55:12	10:55:11.977	10:55:11.977	10:55:13
구독취소	HUOB	F	BTC210108	37012.14	4890.60	0.15%	37012.42	37012.13	4	34161.37	37518.00	33468.01	10:55:13	10:55:12.695	10:55:12.696	10:55:14
구독취소	HUOB	F	BTC210326	38739.63	5329.50	0.16%	38728.73	38728.72	10	35565.99	39233.42	34813.39	10:55:12	10:55:12.016	10:55:12.017	10:55:13
구독취소	HUOB	F	ETH210326	1267.475	178.487	0.16%	1267.475	1267.474	10	1150.469	1289.709	1105.483	10:55:13	10:55:12.452	10:55:12.452	10:55:14
구독취소	HUOB	F	BTC210625	40178.14	5676.68	0.16%	40170.37	40164.86	16	36776.68	40666.00	36025.86	10:55:11	10:55:11.276	10:55:11.276	10:55:13
구독취소	HUOB	F	ETH210115	1216.156	166.826	0.16%	1216.091	1215.313	160	1049.330	1143.447	982.557				10:49:43
구독취소	HUOB	F	BTC210115	37236.67	5010.68	0.16%	37232.12	37227.04	12	34303.05	37729.74	33607.41	10:55:12	10:55:11.697	10:55:11.698	10:55:13
구독취소	HUOB	W	ETHUSD_SWAP	1204.98	162.14	0.16%	1204.97	1204.96	200	1106.00	1227.81	1060.00	10:55:12	10:55:12.329	10:55:12.330	10:55:14
구독취소	HUOB	W	BTCUSD_SWAP	36983.7	4929.5	0.15%	36983.8	36983.7	2	34120.8	37476.5	33416.4	10:55:13	10:55:12.980	10:55:12.980	10:55:14
구독취소	HUOB	S	btcsdtd	36806.40	4815.23	0.15%	36804.55	36803.09	0.00098	33959.58	37293.89	33300.00	10:55:13	10:55:12.452	10:55:12.453	10:55:14
구독취소	HUOB	S	ethusdt	1199.45	157.65	0.15%	1199.45	1199.44	0.0103	1100.20	1220.79	1057.55	10:55:13	10:55:13.001	10:55:13.001	10:55:14
구독취소	HUOB	S	ethbtc	0.032589	0.000034	0.00%	0.032587	0.032586	0.0218	0.032407	0.034500	0.031321	10:55:13	10:55:12.770	10:55:12.770	10:55:14
구독취소	BINAN	F	ETHUSD_210625	684.78	-0.00	-0.00%	0.00	0.00	0	684.78	684.78	684.78				10:49:43
구독취소	BINAN	F	ETHUSD_210326	1204.36	158.83	0.15%	1204.38	1204.20	31	1108.28	1224.33	1064.69	10:55:13	10:55:12.903	10:55:12.903	10:55:14
구독취소	BINAN	F	ETHUSD_PERP	596.69	-0.00	-0.00%	0.00	596.69	0	596.69	596.69	596.69				10:49:43
구독취소	BINAN	F	ETHUSD_201225	589.04	0.00	0.00%	589.27	589.04	0	589.04	589.04	589.04				10:49:43
구독취소	BINAN	F	ETHUSD_201225	30000.0	6040.0	0.25%	30000.0	23960.0	1	23960.0	23960.0	23960.0				10:49:43
구독취소	BINAN	F	BTCUSD_210326	39902.1	5432.3	0.16%	39896.6	39893.8	2	37081.5	99888.0	36439.4	10:55:13	10:55:12.924	10:55:12.925	10:55:14
구독취소	BINAN	F	BTCUSD_PERP	36945.3	4915.1	0.15%	36941.0	36937.1	6	34043.4	98765.0	10900.0	10:55:13	10:55:12.996	10:55:12.996	10:55:14
구독취소	BINAN	W	ETHUSD_SWAP	1200.72	158.92	0.15%	1201.00	1200.63	10.682	1099.57	1800.55	1.90	10:55:12	10:55:11.327	10:55:11.327	10:55:13
구독취소	BINAN	W	BTCUSD_SWAP	36848.13	4829.40	0.15%	36846.97	36844.74	18.954	33947.99	37286.75	33338.06	10:55:12	10:55:11.583	10:55:11.583	10:55:13
구독취소	BINAN	S	ETHBTC	0.030000	-0.001391	-0.04%	0.000000	0.000000	1	0.031391	0.033500	0.010000				10:49:43
구독취소	BINAN	S	ETHUSD	200.00	-400.00	-0.67%	0.00	0.00	0.65621	600.00	346.47					10:49:43
구독취소	BINAN	S	BTCUSD	39379.00	23924.50	1.55%	0.00	0.00	0.02	15454.50	34301.00	4079.25				10:49:43

▶ 설명

- 모니터링 클라이언트의 메인화면 => 각 서버, 모듈의 로그 표시
- 메뉴를 선택하여 각 화면 오픈

▶ 기술

- 시세 서버로부터 로그 기록을 TCP로 받고, 실시간 로그는 UDP로 처리
- 싱글톤 패턴을 사용하여 각 화면을 관리

- 각 거래소의 거래방식, 종목별로 시세를 표시
- 구독 버튼을 클릭하여 시세 서버로 종목 시세를 구독 요청
- 호가 시세 더블 클릭 시 호가창 오픈

- 구독 버튼을 클릭하여, 시세 서버로 해당하는 거래소의 종목을 구독/구독 취소 TCP 전송
- 실시간 시세 데이터는 UDP(유니캐스트)로 받음

qt 신규주문창

전략 901

거래소 BINAN

종목 F ETHUSD_210326

방향 ☒ 매수 ☐ 매도

주문유형 LIMIT

수량 1,00000000 0,000

가격 1371,64000000 조회

주문전송 닫기

- 주문 서버로 조회를 통해 주문 가능한 수량을 얻어 오고, 거래소의 종목을 주문한다.

- 주문 서버로 조회를 통해 주문 가능한 수량을 얻어 오고, 거래소의 종목을 주문한다.

qt 전략주문제결항

전략ID ALL 거래소 ALL 마켓 ALL 상태 ALL ALL

전략ID	주문SET ID	거래소	마켓	종목	방향	주문유형	가격	현재가	주문번호	주문수량	제결수량	제결 평균가격	상태	접수시간	사유 코드	UserID
902	0	BINAN	F	BTCUSD_201225	매수	LIMIT	30000.0	28960.0	56072312	1	1	28960.0	전항제결	2021-01-07 12:26:21.414	0	jykim
902	0	BINAN	F	BTCUSD_201225	매도	LIMIT	28960.0	28960.0	56071909	3	1	28960.0	접수03	2021-01-07 11:42:46.946	0	hkkim
902	0	BINAN	F	BTCUSD_201225	매도	LIMIT	23960.0	28960.0	0	3	0	0.0	거부03		94024	hkkim
902	0	BINAN	F	BTCUSD_201225	매도	LIMIT	23960.0	28960.0	0	3	0	0.0	거부03		94024	hkkim
902	0	BINAN	F	BTCUSD_PERP	매수	LIMIT	36910.4	37661.7	50851272	1	1	36910.4	전항제결	2021-01-07 11:40:56.721	0	hkkim
902	0	BINAN	F	BTCUSD_PERP	매도	LIMIT	36938.6	37661.7	50850201	1	1	36938.6	전항제결	2021-01-07 11:38:17.570	0	hkkim
901	0	BINAN	F	BTCUSD_201225	매도	LIMIT	30000.0	28960.0	56071798	2	0	0.0	접수03	2021-01-07 11:30:46.279	0	wslee
901	0	BINAN	F	BTCUSD_210326	매도	LIMIT	39670.0	40623.9	10262546	5	5	39670.0	전항제결	2021-01-07 11:30:43.296	0	wslee
902	0	BINAN	F	BTCUSD_210326	매도	LIMIT	39659.6	40623.9	10262513	1	1	39663.9	전항제결	2021-01-07 11:30:38.225	0	wslee
902	0	BINAN	F	BTCUSD_201225	매도	LIMIT	30000.0	28960.0	56071795	2	0	0.0	접수03	2021-01-07 11:30:34.906	0	wslee
901	0	BINAN	W	ETHUSD_SWAP	매도	LIMIT	1185.82	1215.27	711179044	3	3	1185.82	전항제결	2021-01-07 11:30:30.651	0	wslee
903	0	BINAN	F	BTCUSD_PERP	매도	LIMIT	36756.7	37661.7	50845727	1	1	36764.4	전항제결	2021-01-07 11:27:36.137	0	wslee
903	0	BINAN	F	BTCUSD_PERP	매수	LIMIT	36756.7	37661.7	50845713	1	1	36756.7	전항제결	2021-01-07 11:27:33.560	0	wslee
903	0	BINAN	F	BTCUSD_PERP	매수	LIMIT	36739.2	37661.7	0	1.5	0	0.0	거부03		91111	wslee
901	0	BINAN	F	BTCUSD_PERP	매도	LIMIT	36750.0	37661.7	0	1.5	0	0.0	거부03		91111	wslee
901	0	BINAN	F	BTCUSD_PERP	매수	LIMIT	36749.6	37661.7	0	1.5	0	0.0	거부03		91111	wslee
901	0	BINAN	W	BTCUSD_SWAP	매도	LIMIT	36636.22	37564.24	2606561440	1.5	1.5	36636.22	전항제결	2021-01-07 11:26:00.425	0	wslee
901	0	BINAN	W	BTCUSD_SWAP	매수	LIMIT	36622.70	37564.24	2606561406	1.5	1.5	36622.70	전항제결	2021-01-07 11:25:56.393	0	wslee
901	0	BINAN	W	ETHUSD_SWAP	매도	LIMIT	1186.67	1215.27	711177979	1	1	1186.67	전항제결	2021-01-07 11:25:49.791	0	wslee
901	0	BINAN	W	ETHUSD_SWAP	매수	LIMIT	1186.67	1215.27	711177962	1	1	1186.67	전항제결	2021-01-07 11:25:46.406	0	wslee
901	0	BINAN	F	BTCUSD_210326	매수	LIMIT	39713.1	40623.9	10260248	5	5	39708.4	전항제결	2021-01-07 11:25:12.332	0	wslee
901	0	BINAN	F	BTCUSD_210326	매도	LIMIT	39703.1	40623.9	10259998	5	5	39703.1	전항제결	2021-01-07 11:24:36.930	0	wslee
901	0	BINAN	F	BTCUSD_210326	매수	LIMIT	39694.1	40623.9	10259964	5	5	39694.1	전항제결	2021-01-07 11:24:32.468	0	wslee
901	0	BINAN	F	BTCUSD_201225	매도	LIMIT	23960.0	28960.0	0	2	0	0.0	거부03		94024	wslee
902	0	BINAN	W	BTCUSD_SWAP	매도	LIMIT	36618.60	37564.24	2606560672	0.1	0.1	36618.60	전항제결	2021-01-07 11:24:21.431	0	hkkim
901	0	BINAN	F	BTCUSD_201225	매수	LIMIT	30000.0	28960.0	56071735	2	2	30000.0	전항제결	2021-01-07 11:24:11.412	0	wslee

- 거래 상태에 따라서 각 주문의 상태를 확인하고, 추가 주문/ 주문취소 처리

- 콤보 박스 UI를 사용하여, 내가 보고 싶은 정보만 표시할 수 있도록 필터링

- 클라이언트에서 콤보박스를 빠르게 처리하기 위해 데이터를 트리 형태로 구성한다.
Ex) 거래소마다 마켓이 여러 개 있고, 마켓에 해당하는 종목이 여러 개 존재

qt 대표계좌 포지션 대사											
거래소		ALL	마켓		ALL	Update					
거래소	마켓	종목	계좌	방향	평균단가	현재가	수량	수익률(%)	평가손익	실현손익	수수료
BINAN	F	BTCUSD_201225	대표	매수	29786.1	28960.0	4	-100.00%	0.001628	0.000000	0.000000
BINAN	F	BTCUSD_210326	대표		0.0	40624.5	0	0.00%	0.000000	0.000000	0.000000
BINAN	F	BTCUSD_PERP	대표		0.0	37672.9	0	0.00%	0.000000	0.000000	0.000000
BINAN	W	BTCUSD_T_SWAP	대표		0.00	37565.02	0	0.00%	0.000000	0.000000	0.000000
BINAN	W	ETHUSD_T_SWAP	대표		0.00	1216.00	0	0.00%	0.000000	0.000000	0.000000
거래소	마켓	종목	계좌	방향	평균단가	현재가	수량	수익률(%)	평가손익	실현손익	수수료
⊕ BINAN	F	BTCUSD_201225	대표	매수	59645.1	28960.0	4	test	0.001616	-0.000080	0.000007
⊕ BINAN	F	BTCUSD_210326	대표		0	40641.6	0	test	0.000000	-0.000032	0.000017
⊕ BINAN	F	BTCUSD_PERP	대표		0	37682.3	0	test	0.000000	0.000003	0.000002
⊕ BINAN	W	BTCUSD_T_SWAP	대표		0	37567.93	0	test	0.000000	-19.895000	45.543413
⊕ BINAN	W	ETHUSD_T_SWAP	대표		0	1216.96	0	test	0.000000	-51.258680	3.782616

qt 대표계좌 잔고 대사						
거래소		ALL	마켓		ALL	Update
거래소	마켓	결제통화	계좌	잔고	주문가능	Locked
BINAN	F	BTC	대표	9.993818	9.968755	0.000000
BINAN	S	BTC	대표	1.000000	1.000000	0.000000
BINAN	S	ETH	대표	99.290330	99.290330	0.000000
BINAN	S	USDT	대표	10446.769000	10446.769000	0.000000
BINAN	W	USDT	대표	10000.000000	0.000000	0.000000
거래소	마켓	결제통화	계좌	잔고	주문가능	Locked
⊕ BINAN	F	BTC	대표	9.993818	10.006805	0.094827
⊕ BINAN	S	BTC	대표	1.000000	1.000000	0.000000
⊕ BINAN	S	ETH	대표	99.290330	99.290330	0.000000
⊕ BINAN	S	USDT	대표	10446.769000	10446.769000	0.000000
⊕ BINAN	W	USDT	대표	9879.520291	9879.520291	0.000000

▶ 설명

- 주문한 종목들의 손익 표시
- 종목별로 실시간 대표계좌에 연결된 각 부계좌 잔액, 손익 디테일 표시

▶ 기술

- 트리형 그리드를 사용하여 부모 - 자식관계로 연결하면 같은 아이템(거래소 - 마켓 - 종목 - 계좌)끼리 묶어서 화면에 표현

void MyProjcets::source()

{

- 시세 서버

```
void MonitorClient::AddSubExCodeInfo(const std::string& symbolName, const std::string& subsType)
{
    mSubCoinInfoListMtx.lock();
    auto iter = mSubCoinInfoList.find(symbolName);
    // 구독한 종목일 때는 서비스만 추가
    if (iter != mSubCoinInfoList.end())
    {
        if (subsType.compare("All") == 0)
        {
            (*iter).second.SubsType.emplace("Price");
            (*iter).second.SubsType.emplace("BidAsk");
            (*iter).second.SubsType.emplace("TradeInfo");
        }
        else
        {
            (*iter).second.SubsType.emplace(subsType);
        }
    }
}
```

```
std::unordered_map<int, int> clientIDList = mClientIDList;
for (auto client : clientIDList)
{
    int clientID = client.second;

    if (mClientList[clientID]->GetIsConnected() == false) { ... }
    if (mClientList[clientID]->GetIsLoggedIn() == false) { ... }
    // 클라이언트가 종목을 구독을 했는지 검사
    if (mClientList[clientID]->FindSubCoinInfoBySymbolName(symbolName, TICK) == false) { ... }

    for (auto& tickData : dataList)
    {
        // Rapidjson을 사용하여 패킷을 json 포맷으로 변환
        std::string packet = GET_INSTANCE(JsonFormat)->GetSendRealTimeTck(exCode, market, coin, &tickData,
        if (checkDataSize(packet.size()) == false) { ... }
        mServerModule.SendPacket(ePROTOCOL::UsingUdp, client.first, packet.c_str(), packet.length());
    }
}
```

- 클라이언트의 종목 구독 정보를 저장한다.
- 클라이언트의 구독정보를 빠르게 찾기위해 std::unordered_map을 사용
- 구독/구독취소를 처리할 때 mutex를 사용하여 동기화

- 실시간 종목의 시세를 구독한 클라이언트에게만 전송
- 실시간 시세 데이터는 유니캐스팅을 사용하여 전송



- 모니터 클라이언트

```
product->ExCode = exCode;
product->Market = market;
product->Coin = coin;
product->State = doc["data"][i]["state"].GetString();
product->PrevClose = doc["data"][i]["prevClose"].GetDouble();
product->ContractSize = doc["data"][i]["contractSize"].GetDouble();

std::string localTime = QTime::currentTime().toString().toStdString();
product->TickDataInfoList.emplace_front(TickDataInfo(0, doc["data"][i]["price", "", "", 0, localTime]));

product->AskList[0].first = doc["data"][i]["askPrice"].GetDouble();
product->BidList[0].first = doc["data"][i]["bidPrice"].GetDouble();

product->PricePrecision = doc["data"][i]["pricePrecision"].GetDouble();
product->SettleCurrency = doc["data"][i]["settleCurrency"].GetString();

mProductList.emplace(symbolName, product);
```

- 실시간 시세데이터는 개수가 많고, 빈번한 삽입/삭제가 일어나므로 STL list를 사용하여 데이터를 관리
- 종목 정보를 빠르게 찾기위해 unordered_map을 사용



```
class WindowManager
{
    SINGLE_TONE(WindowManager)

public:
    bool Initialize();
    bool AddConWinodw(const std::string& symbolName, struct Product* product);
    void ShowLoginWindow();
    void ShowConWindow(struct Product* product);
    void CloseWindow();

    class Login* GetLoginWindow() {return mLogin;}
    class SelectMainWindow* GetMainWindow() {return mMain;}
    class SessionStatus* GetSessionStatus() {return mSessionStatus;}
    class PriceInfo* GetPriceInfoWindow() {return mPriceInfo;}
    class Order* GetOrderWindow() {return mOrder;}
    class OrderConclusion* GetOrderConclusionWindow() {return mOrderConclusion;}
    class ProfitAndLoss* GetProfitAndLossWindow() {return mProfitAndLoss;}
    class AccPositionWindow* GetAccPositionWindow() {return mAccPositionWindow;}
}
```

- 모니터 클라이언트의 많은 화면들을 관리하기 위해 WindowManager 클래스를 만듦
- 싱글톤 패턴을 사용하고, 각 화면의 포인터들을 갖고 있도록 함


```
void MyProjcets::introduce( )  
{
```

- 프로젝트 이름 : Wallet API 서버
- 설명 : Wallet 관리자 홈페이지에서 사용하는 REST API를 처리
- 역할 : Backend
- 개발 언어 및 라이브러리 : Node js, Express, WS 라이브러리, Rabbit MQ, MySQL
- 개발 환경 :  Visual Studio Code  SourceTree

```
}
```

```
void MyProjcets::tech( )
{
```



리소스 네이밍

		POST	GET	PUT
coin		코인 등록(private api)	코인 리스트 조회(public api)	코인 수정(private api)
	config			코인 입출금 정지(private api)
accounts	address	주소 발급(private api)	주소 유효성 체크(public api)	
	balance		코인 잔고 조회(private api)	
	transaction	코인 전송(private api)		
	config			계좌 입출금 정지(private api)
history	deposit-withdraw		입출금 내역 조회(private api)	
wallet	info			지갑 정보 수정(private api)

- CRUD에 맞추어 REST API 네이밍 설계

REST API (private api) - 코인 잔고 조회

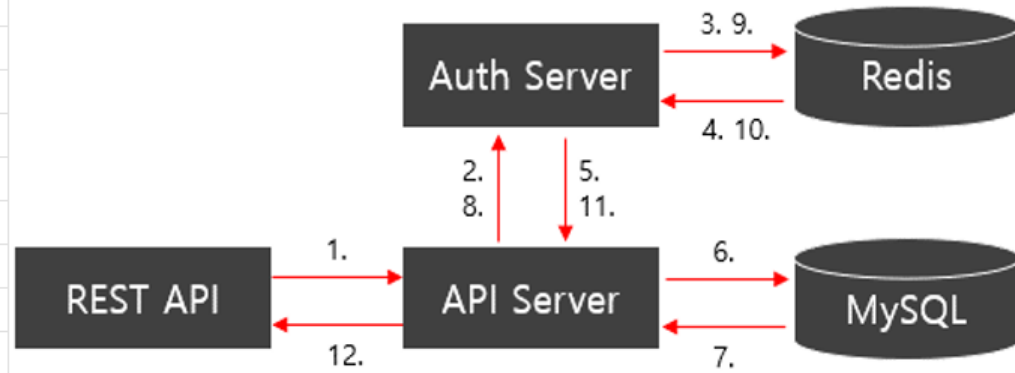
[Request] : **GET /wallet/v1/accounts/balance**

Name	Type	Mandatory	Description
serviceID	string	Yes	서비스ID
accountType1	string	Yes	"1" : 핫월렛계좌, "2" : 콜드월렛계좌, "3" : 고객계좌
accountType2	string	Yes	"1" : Storage, "2" : Custody, "3" : Bottorage, "4" : Total
accountNo	string	Yes	계좌 번호 (최대 15byte)
c_code	string	Yes	"BTC", "ETH", "EOS" 등

```
{
  serviceID : 'TEST001',
  accountType1 : '3',
  accountType2 : '1',
  accountNo : '0019990000000022',
  c_code : 'BTC'
}
```

- API 서버와 웹 서버간의 데이터 설계

REST API Process



순서	설명
1	API 키 전송
2	인증 서버로 API 키 확인 요청
3~4	Redis에서 API 키 존재 및 확인
5	API 서버로 API 키 응답 -> 만약 Redis에 API키가 있으면, 바로 12번으로 이동
6~7	MySQL DB에서 API 키 확인
8	인증 서버로 API 키 저장 요청
9~10	Redis에 API 키 저장
11	API 키 저장 응답
12	REST API 사용

▶ **설명** - API 서버로 REST API가 요청될 때마다 API 키를 체크하는 과정을 나타낸다.

▶ **기술** - REST API가 요청되면 API 키를 체크해야 한다. 그 과정에서 API 키를 매번 DB에 액세스하여 확인하는 절차를 거치면 REST API 처리 시간이 오래 걸리기 때문에 최초 요청시 API 키를 Redis에 타이머와 함께 저장한다. 그 이후부터는 Redis에 API 키가 남아있으면, Redis만 확인하면 되기 때문에 REST API 처리시간이 빨라진다.

서비스 BOS

메인넷코드 BTC

코인코드 BTC

조회

FROM

월렛종류 Hot

조회

번호	코인코드	계좌번호	월렛 잔고
1	BTC	HT000000001	100.00000000

송금항목

From계좌번호 HT000000001

송금수량 10.00000000

송금종류 폴드월렛보관

To계좌번호 CT000000004

To주소입력 0xaa90b4acc74cee41b004bc45e31a427406c4dca

주소검증 (검증됨)

그룹송금구성

코인코드	From	To	수량	담당자	주소	TXID	송금메모
BTC	HT000000001	CT000000001	10.00000000	김선아			폴드월렛보관
BTC	HT000000001	CT000000002	20.00000000	김선아			폴드월렛보관
BTC	HT000000001	CT000000003	25.00000000	김선아			폴드월렛보관

항목 추가

선택항목삭제

전체삭제

그룹송금실행

그룹 월렛건수 3

그룹송금 총수량 45.00000000

그룹송금메모 폴드월렛 대일리 송금 (20자 이내)

서비스 BOS

송금종류 전체

집금처리

외부송금

폴드월렛보관

내부이체

승인상태 전체

승인중

승인거부

승인완료

조회

(조회기준 2021.12.09 12:23:35)

No	서비스	담당자	승인자	트랜잭션	수량	역할그룹	등록일자	반려사유	승인상태
1	BOS	홍길동	홍길동, 김유지, 강소라	집금처리		Operators	2021-12-16		승인중(2/3)
2	BOS	김유지	김유지, 강소라	외부송금		Operators	2021-12-15	테스트 반려	승인거부(1/2)
3	BOS	강소라	강소라, 김유지	폴드월렛보관		Operators	2021-12-15		승인완료(2/2)
4									
5									

코인코드	From	To	수량	블록체인수수료	담당자	등록일시	송금목적
BTC	WT000001021	CT000000001	9999.12340000	0.00125719	홍길동	2021-12-16 11:35:11	집금
BTC	WT000001048	CT000000001	1000.00000000	0.00011111	홍길동	2021-12-16 11:35:11	집금
BTC	WT000001002	CT000000001	0.42665905	0.00000249	홍길동	2021-12-16 11:35:11	집금

담당자 홍길동

승인자 홍길동, 김유지, 강소라

트랜잭션 집금처리

승인상태 승인중(2/3)

등록번호 1234567

승인번호 -----

등록일시 2021-12-16 10:05:13

승인일시 -----

승인

반려

설명

- 코인전송 웹페이지에서 송금실행을 하면 다중승인 페이지에서 해당 서비스의 관리자가 승인 혹은 반려를 하는 절차를 거치고, 모두 승인시에 코인이 전송된다.
- 승인 관리자는 자신이 승인할 수 있는 송금종류만 출력된다.

기술

- 송금실행 REST API가 요청되면 API 서버는 DB 테이블에 저장한다. 각 승인이 될 때마다 테이블에서 해당하는 row를 업데이트 하며, 승인자가 모두 승인한 경우 API 서버는 해당 데이터를 Wallet 서버로 RabbitMQ를 통해 코인을 전송한다.

void MyProjcets::source() {

```
switch(api)
{
    case '주소발급':
    {
        const reqJsonData = RequestFormat.GetIssueAddressFormat(req.body);
        const event = await this.#mMQ.SendToWalletRQ(reqJsonData);

        await new Promise((resolve) => this.#mMQ.GetEvent().once(event, resolve))
            .then((msg) => { ResponseFormat.GetIssueAddressFormat(format, msg); });
        break;
    }

    case '코인등록':
    {
        const reqJsonData = RequestFormat.GetRegisterCoinFormat(req.body);
        const event = await this.#mMQ.SendToWalletRQ(reqJsonData);

        await new Promise((resolve) => this.#mMQ.GetEvent().once(event, resolve))
            .then((msg) => { ResponseFormat.GetRegisterCoinFormat(format, msg); });
        break;
    }
}
```

- API 서버에서 REST API의 처리 부분
- Request 발생시
 1. 지갑 서버의 데이터 포맷에 맞추어 변경
Rabbit MQ로 전송
 2. 지갑 서버의 응답을 기다리위해 await,
Promise - then을 사용한다. 왜냐하면, 지갑
서버의 응답이 오기전에 http 응답이 먼저
처리되기 때문이다.

```
CheckKey(apiType, obj, format)
{
    // api가 존재하는지
    if(this.#mTrList.has(apiType) == false)
    {
        return false;
    }

    // 깊은 복사를 해야한다. 얇은 복사시에 value가 원본도 바뀜
    const lodash = require('lodash');
    const newObj = lodash.cloneDeep(obj);
    // obj의 각 value의 타입을 값으로 가진 obj
    for(let key in newObj)
    {
        newObj[key] = typeof newObj[key];
    }



    const keyListSize = Object.keys(this.#mTrList.get(apiType).data).length;
    const newObjSize = Object.keys(newObj).length;

    if(keyListSize >= newObjSize)
    {
        // Tr리스트에 해당하는 모든 키가 있는지 확인
        for (let key in this.#mTrList.get(apiType).data)
        {
            if (newObj[key] == undefined)
            {
                format.errorCode = '00101';
                format.message = '[' + key + ']';
                return false;
            }
        }
    }
}
```

- Request를 처리하기 전에 데이터 양식이 맞는지
체크 과정

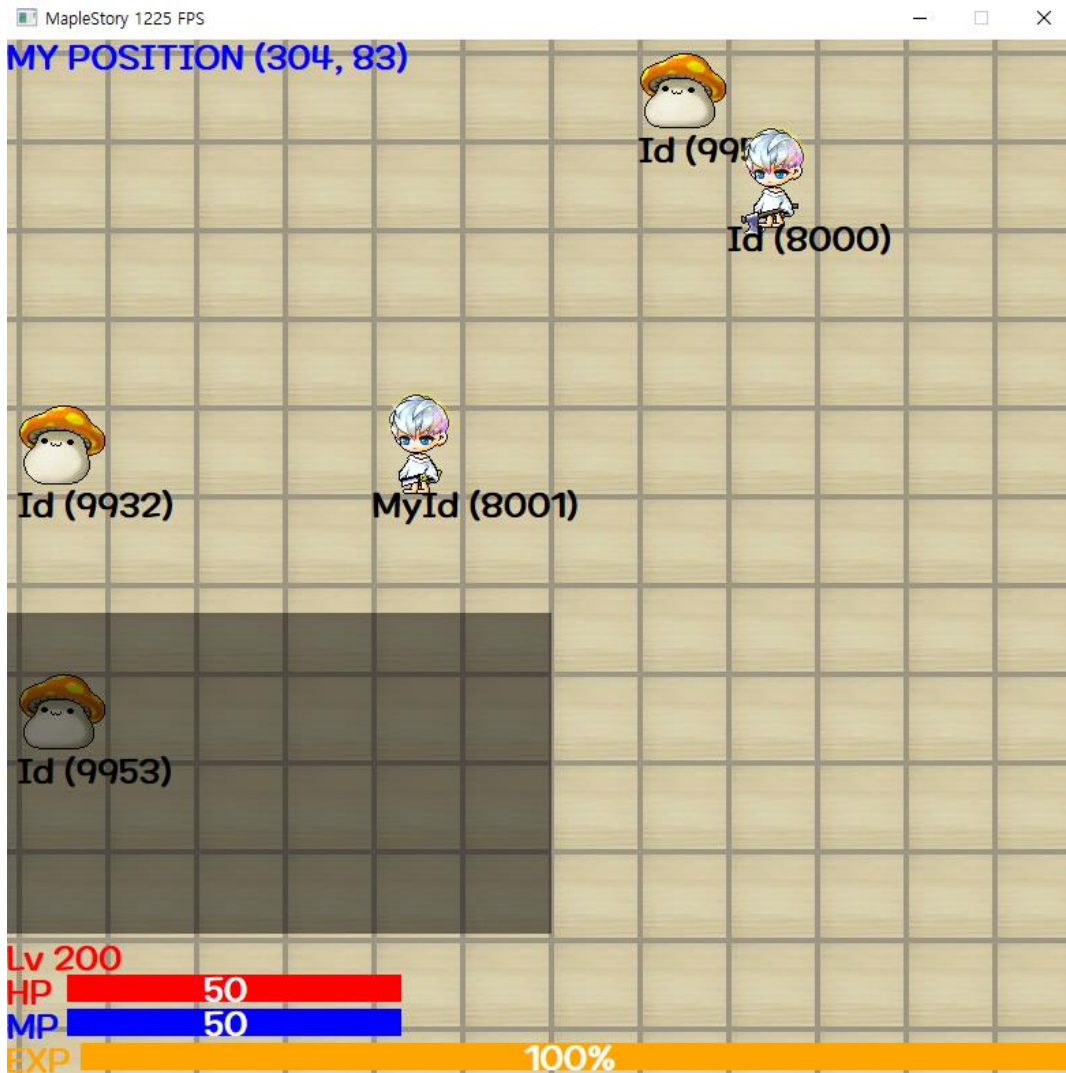


```
void MyProjcets::introduce( )  
{
```

- 프로젝트 이름 : 2D MMORPG
- 설명 : IOCP 게임서버, Direct2D 클라이언트
- 역할 : 서버, 클라이언트, 더미 클라이언트
- 개발 언어 및 라이브러리 : C++, IOCP, TBB, STL, Direct2D WIN32 API
- 개발 환경 :  Visual Studio 2022  SourceTree
- 저장소 : https://github.com/LeeWooSang/2D_MMORPG

```
}
```

```
void MyProjets::tech( )  
{
```



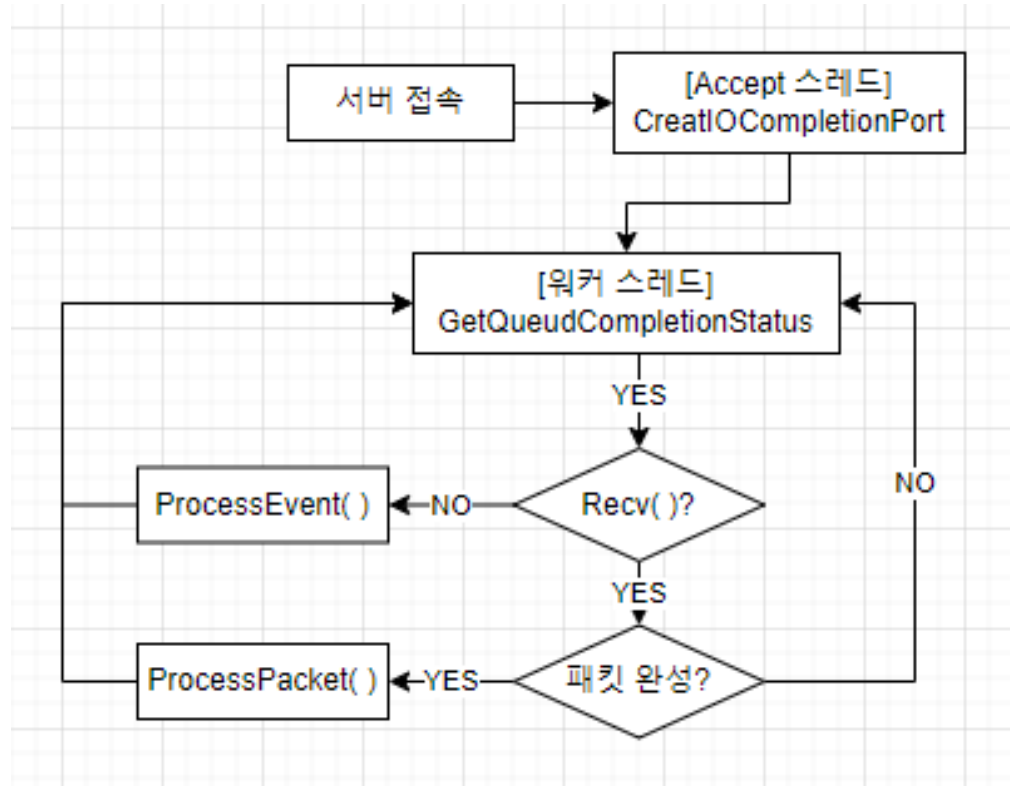
[게임 서버 스펙]

▶ 설명

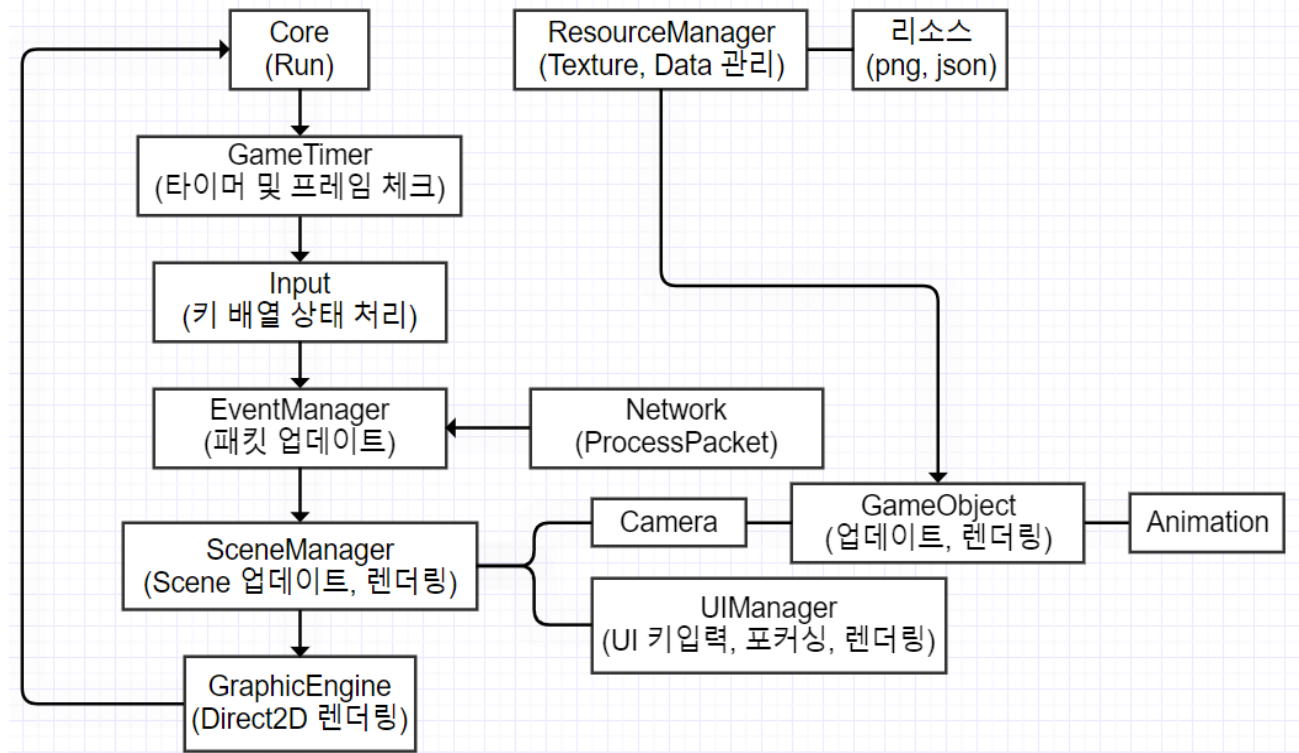
1. 맵 : 500m x 500m
2. 동접 유저 : 8,000명
3. 몬스터 수 : 90,000
4. 채널 : 30개

▶ 기술

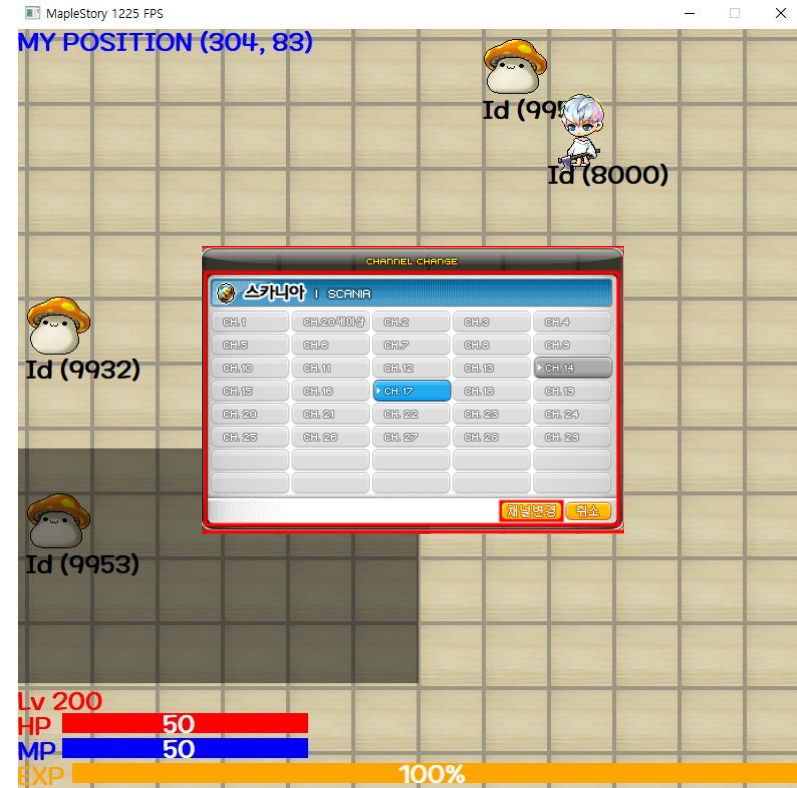
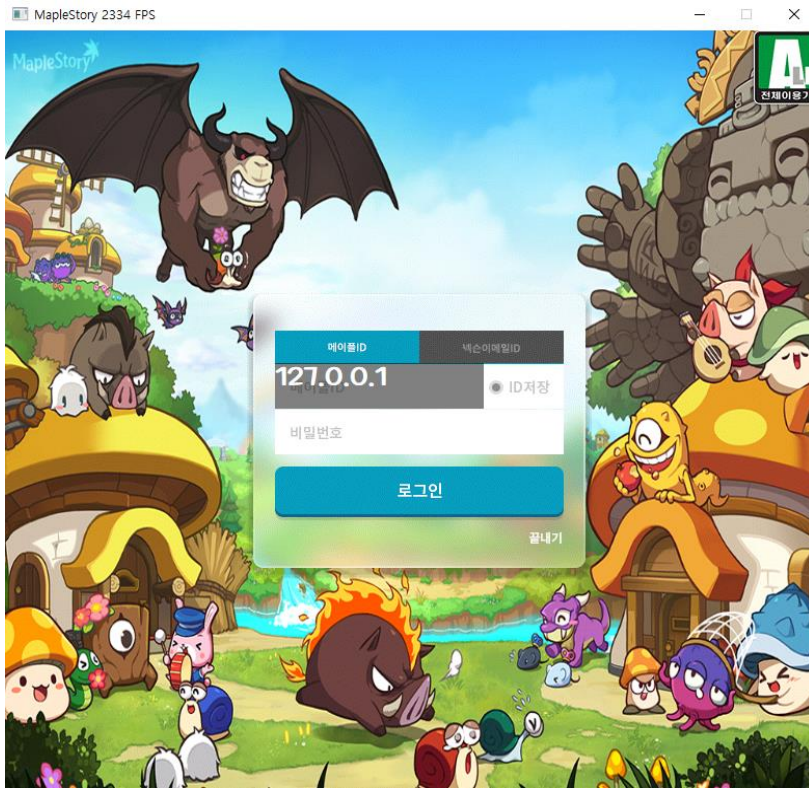
1. 다중 클라를 처리하기 위해 IOCP를 사용
2. 서버 부하를 줄이기 위해 Channel 추가 및 맵을 Sector로 분할
→ Channel : 30개
→ Sector : 50m x 50m
3. concurrent_hash_map을 사용하여 오브젝트 관리
→ 삽입, 삭제, 검색(thread safe container)
3. concurrent_priority_queue를 사용하여 몬스터 AI, 이벤트 처리
→ 타이머 관리(thread safe container)



게임 서버



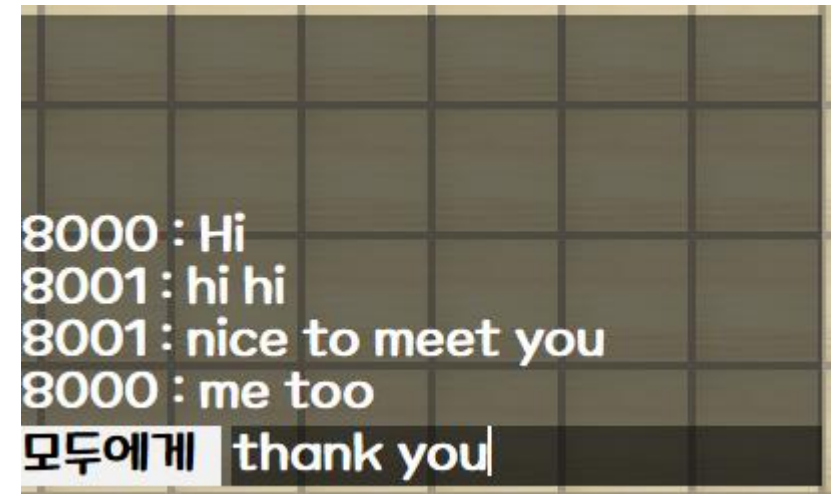
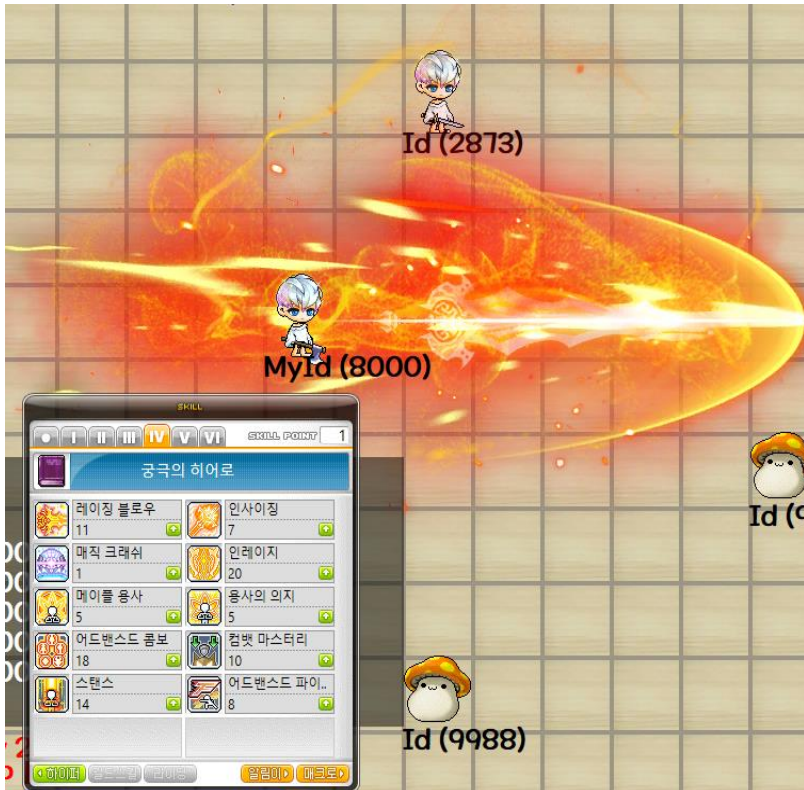
클라이언트



1. Scene

▶ 설명

- 1) 로그인 : 서버 IP 입력 (추후 아이디와 비밀번호 입력)
- 2) 서버 및 채널 선택 : 서버, 채널을 선택하고, 서버의 모든 채널 유저 비율을 표시
- 3) 인게임 : 채널 변경



1. 애니메이션 : 플레이어, 몬스터, 스킬 이펙트 등

▶ 설명

2. UI : 인벤토리(아이템, 슬롯 및 스크롤 구현), 장비창(아이템 장착 및 아바타 변경)

3. 채팅 : 전체 채팅(같은 채널 & 섹터), 특정 유저에게 귓속말(채널, 섹터 상관X)



1. 교환(1 : 1 거래)

▶ 설명

- 1) 다른 유저(좌측) – 상대방이 아이템을 슬롯에 올림
- 2) 나(우측) – 상대방에게 건내줄 돈을 올림
- 3) 교환이 완료되면, 나의 인벤토리에 상대방이 올린 아이템이 인벤토리 슬롯에 저장
- 4) 인벤토리에 있는 돈은 교환에 올린 돈만큼 차감

[클라이언트]

0. Graphic

- 1) Direct2D를 사용하여 Texture/Text 렌더링
- 2) 렌더 타겟 생성 및 관리

1. 키 입력

- 1) Input 클래스를 구현하여 키보드/마우스 입력시 상태를 체크하여 빠르게 처리
(키보드 배열의 상태를 저장)

2. 리소스 관리

- 1) ResourceManager를 구현하여 Texture 및 TextureData를 관리
- 2) rapidjson을 사용하여 json 데이터를 로드하고 TextureData 저장

3. Network

- 1) WSAAsyncSelect 모델, ProcessPacket, Send, Recv 처리
- 2) EventManager를 구현하고 EventQueue를 관리하여 서버로 부터의 패킷을 각 Scene에게 전달

4. Scene

- 1) SceneManager를 구현하여 Scene을 컨트롤
- 2) 각 Scene에서 사용하는 모든 게임 오브젝트 생성 및 업데이트, 렌더링

ex) 인게임

- a. STL – unordered_map을 사용하여 많은 플레이어와 몬스터를 빠르게 검색
- b. 보이는 오브젝트만 따로 관리하여 렌더링 (visibleObjects)

▶ 기술

5. Animation

- 1) Animation 클래스를 구현하여 각 애니메이션에 해당하는 Texture들과 Data를 저장
- 2) 애니메이션 타입에 따라 업데이트를 다르게 하여 전진 애니메이션, 반복 애니메이션 등 구현

6. GameObject

- 1) 플레이어는 머리, 몸통, 팔, 무기 등 다양한 부위로 나누어 관리
 - 장비 아이템을 장착했을 때, 아바타가 변경되어야 하기 때문에
 - 각 부위마다 애니메이션을 갖고 있어야 함

7. UI

- 1) UIManager을 구현하여 현재 Scene의 UI 오브젝트 키 입력 및 포커싱 처리
 - 하나의 UI는 부모UI와 여러 자식UI를 갖도록 구현
 - 키 입력이 발생하면, 자식 UI에게 키 입력 처리가 되도록 구현

ex) 인벤토리

- a. 부모UI : 인벤토리 box
- b. 자식UI : 슬롯, 각종 버튼, 스크롤, 아이콘 등

ex) 장비창

- a. 인벤토리에 있는 아이템 클릭시 장비창UI의 맞는 슬롯번호에 장착
- b. 장착된 아이템의 아바타로 변경되기 위해 textureId로 검색하여, 해당하는 텍스처의 모든 애니메이션 동작을 아바타 부위에 저장하고, 애니메이션 프레임, 시간 초기화

ex) 채팅

- a. Input 클래스에서 입력된 키를 체크하여, Direct2D DrawText로 렌더링 처리
- b. 캐럿 : 현재 입력된 문자열 레이아웃 사이즈를 체크하여, 캐럿의 위치를 계산

```
// 1. 플레이어의 각 부위를 생성
// 2. 플레이어의 애니메이션(IDLE, 걷기, 점프)을 추가
// ex) 플레이어가 무기를 장착했을 때, 무기의 애니메이션 동작들을 추가한다.
weapon = new AnimationCharacter;
{
    Animation* ani = new Animation;
    ani->Initialize();
    ani->IsRepeat();
    weapon->AddAnimation(ANIMATION_MOTION_TYPE::IDLE, ani);
}
{
    Animation* ani = new Animation;
    ani->Initialize();
    weapon->AddAnimation(ANIMATION_MOTION_TYPE::WALK, ani);
}
```

- 플레이어는 여러 부위로 나뉘어져 있다.
- Ex) 머리, 몸통, 팔, 겉옷, 무기 등
 - 모든 부위의 오브젝트를 생성하고, 애니메이션을 추가한다.
 - 각 부위들은 플레이어의 자식으로 구현

```
// SceneManager에서 현재 Scene에서 렌더링 되는 순서의 UI들을 갖고온다.
std::list<UI*> &sceneUls = GET_INSTANCE(SceneManager)->GetCurScene()->GetSceneUls();
std::list<UI*>::iterator targetIter = sceneUls.end();

for (auto iter = sceneUls.begin(); iter != sceneUls.end(); ++iter)
{
    if ((*iter)->IsVisible() == false) continue;

    // UI들을 루프돌면서, 현재 보이면서, 마우스와 충돌중인 UI를 찾는다.
    if ((*iter)->GetMouseOver() == true)
    {
        targetIter = iter;
    }
}

if (targetIter == sceneUls.end())
{
    return nullptr;
}

// 가장 마지막의 targetIter가 맨 위에 UI로 올라와야 함으로, 그것을 focusUI로 정함
// SceneManager의 UI집합중 targetIter를 맨 뒤로 넣는다. (맨 마지막에 그려져야 맨위에 올라오기 때문)
focusUI = (*targetIter);
sceneUls.erase(targetIter);
sceneUls.emplace_back(focusUI);
```

- UIManager는 현재 씬에서 렌더링되는 UI들을 컨트롤 하기위해 구현
 - 마우스와 충돌중인 focusUI를 찾아서 키입력 이벤트를 focusUI에게 보내도록 하고, 마지막에 렌더링하여 제일 위에 그려지도록

```
case CS_PACKET_TYPE::CS_CHANGE_CHANNEL:
{
    CSChangeChannelPacket* packet = reinterpret_cast<CSChangeChannelPacket*>(buf);
    bool result = false;

    int oldChannel = mUsers[id].GetChannel();
    int newChannel = packet->channel;

    // 기존 채널과 새로운 채널이 같은지 먼저 확인
    if (oldChannel == newChannel) { ... }
    // 새로운 채널이 범위를 벗어났는지 확인
    if (newChannel >= MAX_CHANNEL) { ... }
    // 채널의 유저가 가득 찼는지 확인
    if (mChannels[newChannel].IsFull() == false)
    {
        int x = mUsers[id].GetX();
        int y = mUsers[id].GetY();
        int oldChannelIndex = mUsers[id].GetChannelIndex();

        // 플레이어 기존 채널, 섹터에서 제거
        mChannels[oldChannel].PopUser(oldChannelIndex);
        mChannels[oldChannel].PopSectorObject(x, y, id);
        // 새로운 채널, 섹터로 추가
        int newChannelIndex = mChannels[newChannel].PushUser(id);
        mChannels[newChannel].PushSectorObject(x, y, id);

        mUsers[id].SetChannelIndex(newChannelIndex);
        mUsers[id].SetChannel(newChannel);
        // 채널이동시 기존 채널의 오브젝트와 새로운 채널 오브젝트들의 버리스트들을 처리한다.
        mUsers[id].ProcessChangeChannelViewList(oldChannel, newChannel);
    }
}
```

- 플레이어가 채널을 변경하는 패킷 처리 과정
 - 플레이어가 채널, 섹터를 변경할 때, concurrent_hash_map을 사용하여, 안전하게 erase와 insert 처리

```
TimerEvent ev;
// 타이머 큐에서 이벤트가 있는지 확인
if (mTimerQueue.try_pop(ev) == false)
{
    break;
}

// 타이머 큐에서 해당 이벤트가 시간이 되었는지 체크한다.
if (ev.startTime > chrono::high_resolution_clock::now())
{
    mTimerQueue.push(TimerEvent(ev.startTime, ev.eventType, ev.myId, ev.channel, ev.sectorXId, ev.sectorYId));
    break;
}



// ...

// 이벤트가 몬스터AI인 경우
if (ev.eventType == SERVER_EVENT::MONSTER_MOVE)
{
    // 몬스터id와 채널과 섹터id 정보를 워커스레드에게 전달
    Over* over = new Over;
    over->eventType = ev.eventType;
    over->myId = ev.myId;
    over->channel = ev.channel;
    over->sectorXId = ev.sectorXId;
    over->sectorYId = ev.sectorYId;

    GET_INSTANCE(Core)->PushLeafWork(over);
    PostQueuedCompletionStatus(GET_INSTANCE(Core)->GetIOCP(), 1, ev.myId, &over->overlapped);
}
```

- 몬스터가 AI를 타이머 큐에서 관리한다.
 - concurrent_priority_queue를 사용하여, 안전하게 pop과 push를 처리한다.
 - pop된 정보는 워커스레드에게 넘겨 AI 처리


```
void MyProjcets::introduce( )  
{
```

- 프로젝트 이름 : MVP등급 유틸
- 설명 : 메이플에서 MVP등급을 올리거나 유지할 때 이득인 캐시 아이템을 계산
- 역할 : 클라이언트
- 개발 언어 및 라이브러리 : C++, Qt
- 개발 환경 :  Qt Creator  SourceTree
- 저장소 : <https://github.com/LeeWooSang/MapleUtility>
- 인벤 : <https://www.inven.co.kr/board/maple/2304/31458?name=subject&keyword=mvp%EC%9E%91>

```
}
```

```
void MyProjcets::tech( )
{
```



MVP작 유틸리티(ver1.53)

파일

1억당 비율(단위 원)

비율 : 2,700원 -----▶ 1원당 35,925 메소 수수료 ☐ 5% ☒ 3%

 캐시 총전 할인율(%)

아이템	캐시 가격	메소 가격	손익(메소)	손익(원)	1억당 비율	수수료(%)	마일리지 할인 여부
위습의 원더베리(11개)	50,220	2,695,000,000	809,996,500	22,546	2,700	3	X
로알 헤어쿠폰	3,580	178,000,000	44,048,500	1,226	2,700	3	O
플래티넘 카르마의 가위	3,840	187,000,000	43,438,000	1,209	2,700	3	O
레드 큐브(11개)	9,207	333,000,000	-7,751,475	-215	2,700	3	X
블랙 큐브(11개)	16,879	615,000,000	-9,828,075	-273	2,700	3	X
로알 헤어쿠폰	5,115	178,000,000	-11,096,375	-308	2,700	3	X
플래티넘 카르마의 가위	5,487	187,000,000	-15,730,475	-437	2,700	3	X
에디셔널 큐브(11개)	18,414	660,000,000	-21,322,950	-593	2,700	3	X
메이플 로알 스타일(20개)	40,920	1,460,000,000	-53,851,000	-1,498	2,700	3	X

아이템 등록

이름

캐시 가격

메소 가격

☒ 마일리지 할인(30%)

3. 메이플 로알 스타일(10개) : 22,000원
 4. 메이플 로알 스타일 : 2,200원
 5. 프리미엄 마스터피스(10개) : 19,000원
 6. 프리미엄 마스터피스 : 1,900원
 7. 위습의 원더베리(11개) : 54,000원
 8. 위습의 원더베리 : 5,400원
 9. 루나 크리스탈 : 3,900원
 10. 블랙 큐브(11개) : 18,150원
 11. 블랙 큐브 : 2,200원
 12. 에디셔널 큐브(11개) : 19,800원
 13. 에디셔널 큐브 : 2,400원
 14. 레드 큐브(11개) : 9,900원
 15. 레드 큐브 : 1,200원
 16. 플래티넘 카르마의 가위 : 5,900원
 17. 골드애플(100개) : 48,000원
 18. 골드애플(50개) : 25,000원
 19. 로알 헤어쿠폰 : 5,500원
 20. 로알 성형외과 쿠폰 : 3,500원

- ▶ 설명
- 1억당 비율에 가격 세팅 후 추천 품목에 캐시 아이템을 클릭하여 메소 가격을 입력한다. 그리고 정렬을 하면 등록된 아이템 손익별로 내림차순 정렬이 되고, 제일 이득인 것을 노란색으로 표시
 - 각 셀마다 가격을 다시 편집가능

- ▶ 기술
- json 파일로 캐시 아이템 품목을 저장
 - 손익 메소가 높은 순서로 내림차순 정렬
 - 파일로 저장시 이제까지 추가된 아이템 목록, 가격 등을 파일로 저장

```
void MyProjets::source( )  
{
```

```
    // 테이블 초기화  
    mAuctionTable->setRowCount(0);  
  
    std::list<std::pair<int, AuctionItem>> auctionItemList;  
    for(const auto& auctionItem : GET_INSTANCE(Auction)->GetAuctionItemList())  
    {  
        auctionItemList.emplace_back(auctionItem.first, auctionItem.second);  
    }  
  
    // 손익(메소) 오름차순 정렬  
    using tempType = std::pair<int, AuctionItem>;  
    auctionItemList.sort([](const tempType& a, const tempType& b) { return a.second.GetProfitAndLossOfMeso() < b.second.GetProfitAndLossOfMeso(); });  
  
    // 테이블 셋팅  
    for(const auto& item : auctionItemList)  
    {  
        AddAuctionTable(item.first, item.second);  
    }
```

- 정렬 버튼을 클릭하면, 테이블을 초기화 후 경매장 아이템 목록을 저장
- 아이템 리스트의 손익을 비교하면서 오름차순 정렬
- 정렬이 끝나면 화면 테이블에 순서대로 삽입



THANK YOU