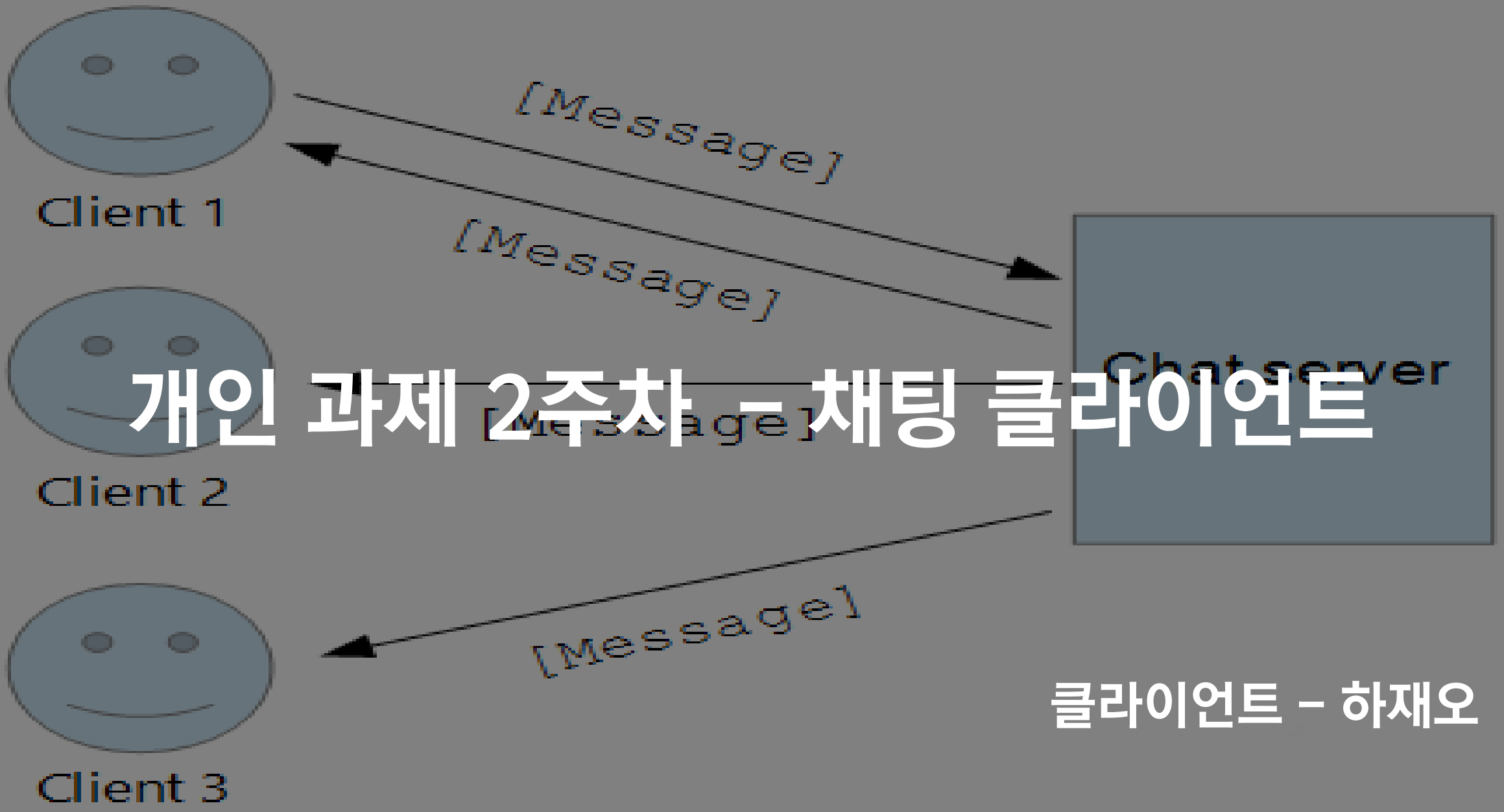


개인 과제 2주차 – 채팅 클라이언트

클라이언트 – 하재오

목차

개요

소켓

FSocket

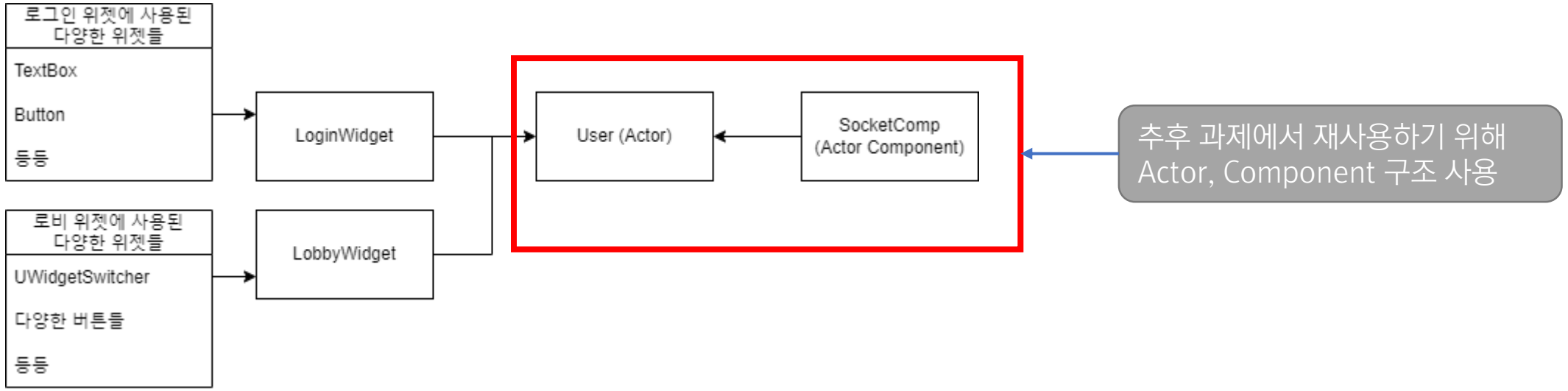
언어 & 엔진

C++, Unreal Engine 4

목표

- 블루프린트 사용을 최소화 하고 언리얼 엔진을 사용해 채팅 서버 클라이언트 제작

구조



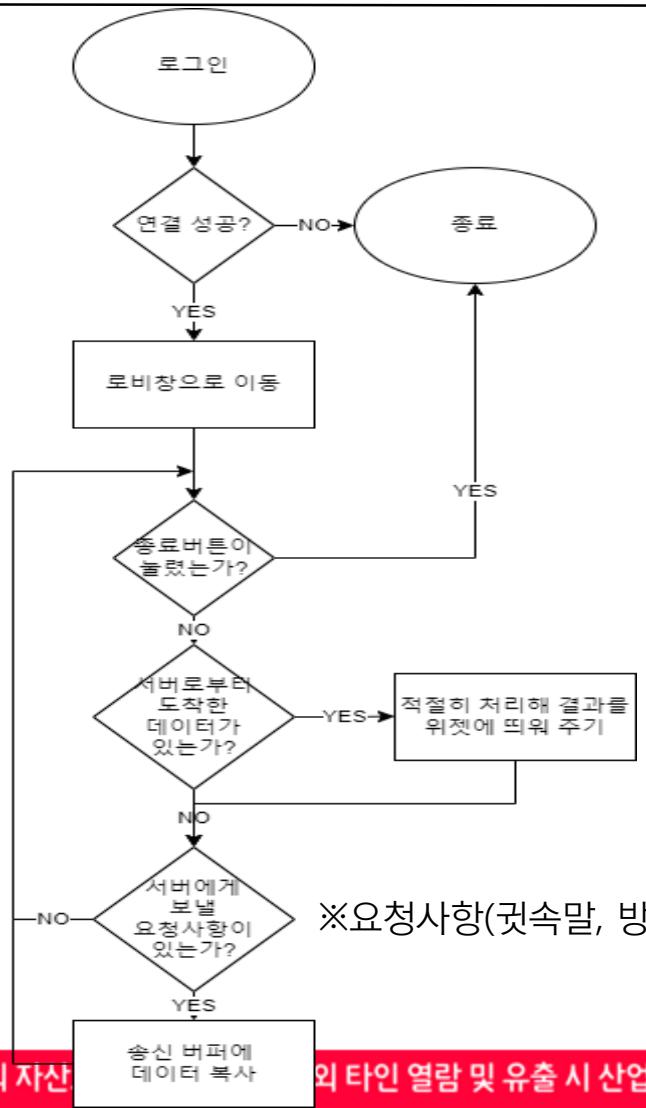
메시지 수신

- SocketComp에서 서버로 부터 온 메시지를 분석하고, User에 있는 적절한 함수를 호출
- User에선 소유한 위젯에게 메시지를 전달
- 각 위젯에서 메시지를 적절히 처리

메시지 송신

- 각 위젯의 버튼이 눌리는 순간
e.g) 유저리스트 요청)
User의 SocketComp를 통해 서버에 메시지를 전달

실행 흐름



※요청사항(귓속말, 방 개설 등등)은 로비에서 버튼이 눌릴 때 마다 발생

통신 처리

USocketComponent

역할

서버와 통신하기 위해 사용하는 액터 컴포넌트

주요 함수

- SendMsg
- ProcessingSend
- ProcessingRecv

SendMsg

역할

송신 큐에 데이터를 삽입 – 삽입된 데이터는 추후 Processing Send에서 처리

```cpp
void USocketComponent::SendMsg(const FString& Msg)
{
    UE_LOG(LogTemp, Log, TEXT("Input Send Msg To Queue: %s"), *Msg);

    FSendBuffer newMsg;
    newMsg.LeftPos = 0;
    newMsg.RightPos = Msg.Len() + ADDITIONAL_PACKET_SIZE;
    for (int i = 0; i < Msg.Len(); ++i)
    {
        newMsg.Buffer[i] = Msg[i];
    }
    newMsg.Buffer[Msg.Len()] = 'Wr';
    newMsg.Buffer[Msg.Len() + 1] = 'Wn';
    newMsg.Buffer[Msg.Len() + 2] = 'WO';
    //텔넷 클라이언트와 동일한 메시지 형식을 지켜주기 위해서 추가적인 문자를 붙입니다.
    SendQ.Enqueue(newMsg);
}
```

```cpp
void ULobbyWidget::UserListButtonClickedCallback()
{
    if (User)
    {
        LobbyWidgetSwitcher->SetActiveWidgetIndex(USER_LIST_INDEX);

        User->SendMsg(USER_LIST_REQ_COMMAND);
    }
}
```

사용 예시

- 위젯에서 AUser의 SendMsg 함수를 통해 메시지 송신이 이루어짐
- AUser는 SocketComponent의 SendMsg를 호출해줌

ProcessingSend

역할

송신 큐에 있는 데이터들을 송신 버퍼에 복사

```
if (!Socket)
    {
        return;
    }
    while (!SendQ.IsEmpty())
    {
        int32 sendingSize = 0;
        FSendBuffer buffer;
        SendQ.Peek(buffer);

        bool isSuccess = Socket->Send((uint8*)buffer.Buffer.GetData() + buffer.LeftPos,
            buffer.RightPos - buffer.LeftPos, sendingSize);
        if (!isSuccess)
        {
            EndGame();
        }
        FString str(buffer.Buffer.GetData());
        UE_LOG(LogTemp, Log, TEXT("Send: %s"), *str);
        buffer.LeftPos += sendingSize;
        if (buffer.LeftPos == buffer.RightPos) //만약 데이터를 송신 버퍼에 복사하는데 성공했다면
        {
            SendQ.Pop(); //큐에서 빼내고 다음 데이터를 송신합니다.
            continue;
        }
        break;
    }
```

송신 버퍼가 가득 차서 데이터 복사에 실패하면 다음에 다시 복사를 시도

ProcessingRecv

역할

수신 버퍼에 있는 데이터들을 읽고, 패킷 처리 함수를 호출

```
int32 recvSize = 0;
    bool isSuccess = Socket->Recv((uint8*)RecvBuffer.GetData() + RecvRightPos, MAX_BUFFER_SIZE - RecvRightPos, recvSize);

    if (!isSuccess)
    {
        EndGame();
    }

    RecvRightPos += recvSize;

    int leftPos = 0;
    for (int i = 0; i < RecvRightPos; ++i)
    {
        if (RecvBuffer[i] == END_SIGN) //엔드 사인 발견!(패킷 완성)
        {
            std::string packet(RecvBuffer.GetData() + leftPos, RecvBuffer.GetData() + (i - 1)); //패킷엔 엔드사인과 캐리지리턴을 띠어냅니다.
            leftPos = i + 1; //엔드 사인 다음을 left로 삼습니다.
            ProcessingPacket(packet);
        }
    }
    //완성된 패킷이 뭉쳐있는 경우를 처리해줍니다.

    if (leftPos != RecvRightPos) //완성되지 않은 패킷이 존재하는 경우
    {
        std::transform(RecvBuffer.GetData() + leftPos, RecvBuffer.GetData() + RecvRightPos, RecvBuffer.GetData(), [](const char Ch) {return Ch; });
        //완성되지 않은 패킷을 버퍼의 맨 앞으로 옮겨 다음 수신 때 완성된 패킷을 만들 수 있게 합니다.
    }
    else
    {
        UE_LOG(LogTemp, Log, TEXT("Packet Complete"));
        memset(RecvBuffer.GetData(), '\0', MAX_BUFFER_SIZE);
        RecvRightPos = 0;
    }
    //모든 패킷을 정상처리했을 경우 버퍼와 RecvRightPos를 초기화시킵니다.
```

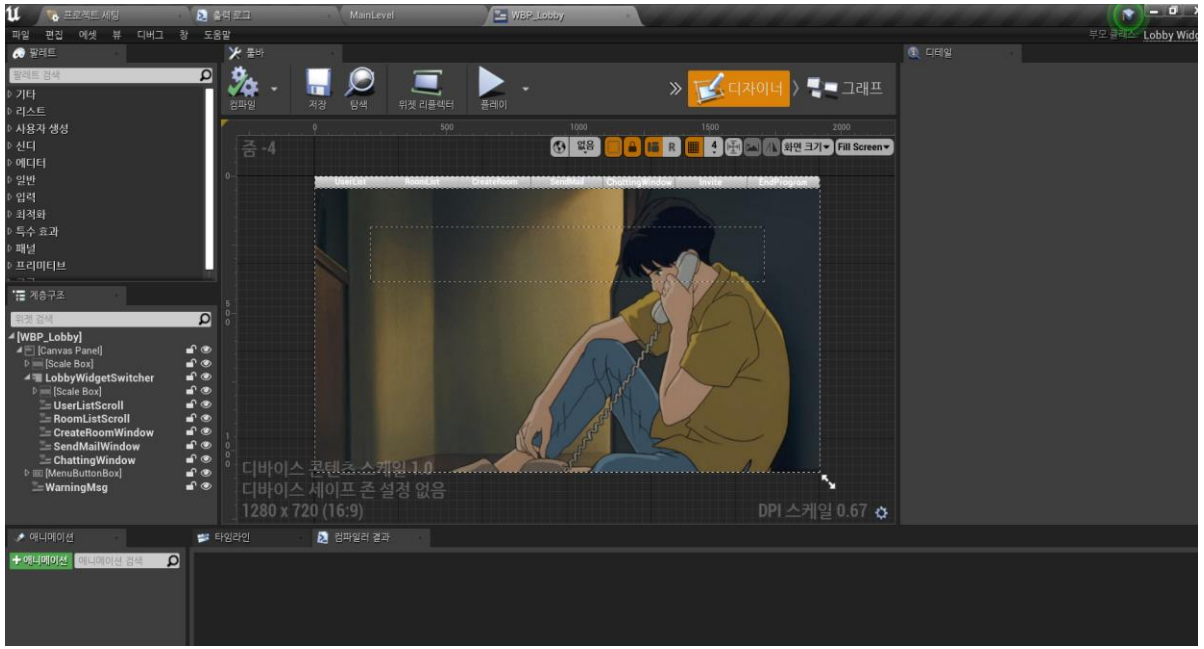
이전 수신 때 패킷이 잘린 경우 잘린 부분 부터 다시 수신

- 완성된 패킷이 뭉쳐온 경우 하나씩 처리
- ProcessingPacket 함수에서 각 패킷을 적절히 처리

완성되지 않은 패킷은 다음 수신 때 이어 받을 수 있도록 처리

위젯

구현 방식



디자인 작업은 Widget Blueprint

```cpp
public:
	UPROPERTY(meta = (BindWidget))
	UWidgetSwitcher* LobbyWidgetSwitcher = nullptr;
	UPROPERTY(meta = (BindWidget))
	UUserListWidget* UserListScroll = nullptr;
	UPROPERTY(meta = (BindWidget))
	UUserListWidget* RoomListScroll = nullptr;
	UPROPERTY(meta = (BindWidget))
	UCreateRoomWindowWidget* CreateRoomWindow = nullptr;
	UPROPERTY(meta = (BindWidget))
	USendMailWindowWidget* SendMailWindow = nullptr;
	UPROPERTY(meta = (BindWidget))
	UChattingWindowWidget* ChattingWindow = nullptr;
```

```cpp
UserListButton->OnClicked.AddUniqueDynamic(this, &ULobbyWidget::UserListButtonClickedCallback);
RoomListButton->OnClicked.AddUniqueDynamic(this, &ULobbyWidget::RoomListButtonClickedCallback);
CreateRoomButton->OnClicked.AddUniqueDynamic(this, &ULobbyWidget::CreateRoomButtonClickedCallback);
SendMailButton->OnClicked.AddUniqueDynamic(this, &ULobbyWidget::SendMailButtonClickedCallback);
```

기능 구현은 C++

LobbyWidget

주요 버튼

- 각 버튼이 눌릴 때 마다 WidgetSwitcher의 Active Widget Index가 바뀌며 적절한 위젯을 보여줌
- UserList: 서버로부터 유저 정보를 얻어 옴. 세부 정보도 확인 가능
- RoomList: 서버로부터 방 정보를 얻어 옴. 세부 정보 확인 가능
- CreateRoom: 방 생성 창으로 넘어가 유저가 채팅방을 생성할 수 있게 해줌
- SendMail: 귓속말 보내기 창으로 넘어가 유저가 귓속말을 보낼 수 있게 해줌
- ChattingWindow: 유저가 속한 채팅방의 채팅 내역 확인과 채팅을 보낼 수 있는 채팅창을 보여 줌

감사합니다