


isCrazyConsole() 부분의 의도: vscode에서 디버그시 커서를 가지고 노는 데 있어 어려움이 있어 사용할 수 없는 환경일 때 프로그램을 새로 실행하도록 하여 해결하였습니다.

서버단과 웹소켓 커넥션을 맺고 OAuth 로그인을 위한 절차가 진행됩니다. 이때, 서버단에서는 최대 5분간 유효한 토큰을 발급하고 로그인을 위한 링크를 전달해줍니다. 클라이언트에서 해당 링크를 보여주고, 이를 통해 로그인하면 서버단에서는 해당 토큰의 상태와 아이디를 업데이트합니다. 클라이언트에서 주기적으로(1000ms) 서버단에 해당 토큰이 유효한지 확인합니다. 개발 초기에 마치 폴링처럼 구현했는데, 사실 인터럽트 방식으로 당장 개선이 가능합니다.

```
hello@DESKTOP-0BV2C6P MINGW64 /c/dev/CppLangAI_test/compfund_client (master)
$ ./client
아래 링크로 접속해 로그인하세요!
"https://comp-fund.jae.one/api/oauth/login?ticket=7kjg9iebp1"
```

로그인 링크

	
최초 로그인시	재로그인시

로그인 후에는 while loop로 돌면서 여러 메뉴들에 접근할 수 있게 했습니다.

/exit	프로그램을 종료합니다.
/my	내 정보를 출력합니다.
/rooms	내가 참여한 방의 목록을 정보와 함께 출력합니다.
/create	새로운 채팅방을 개설합니다.
/room-list	참여 가능한 방 목록을 정보와 함께 출력합니다.
/status	모든 참여자의 상태를 확인합니다.
/status <상태 메시지>	내 상태 메시지를 수정합니다.
/nickname <닉네임>	내 닉네임을 수정합니다.
/leaderboard	전체 회원의 리더보드를 확인합니다.

/my: 내 정보를 출력합니다.

[illegible]

서버단에서 매 로그인마다 프로필사진을 다운로드해 픽셀 정보를 base64인코딩해 데이터베이스에 저장하고, 클라이언트에게 전달해줍니다. URL이 변경될 때만 다운로드하도록 코드를 작성하지 않은 점이 아쉽지만 간단히 수정할 수 있는 부분입니다.

/rooms: 내가 참여한 방의 정보를 목록을 정보와 함께 표시합니다.

2995명을 선택하세요 (↑/↓, Enter | ESC):

방 제목	방 설명	최근 채팅 시간	접역자
room1	testroom	2025-06-05T13:14:05.689Z	*진재민[채팅 / Language & AI로만전공]* *Pleizz* *신해진[채팅 / Language & AI로만전공부]* shs

제목, 방 설명, 최근 채팅 시간, 참여자(닉네임)을 예쁘게 포매팅하여 보여줍니다.

가장 긴 글자에 맞추어 길이가 늘어나는데, 이를 함수(prettier)로 만들어 /room-list, /leaderboard 등에도 함께 적용했습니다.

포매팅한 글자를 전달하면 방향으로 선택할 수 있는 함수(selector)도 만들었습니다.

해당 화면에서 선택한 방으로 접근하게 됩니다.

```
MINGW64/c/dev/CppLangAI x + v
채팅방 이름: room1
shs | 5
shs | <band>
shs | 6
*Pleizz* | 안녕하세요
shs | 안녕
=== /back '뒤로 가기' /send <메시지> '메시지 보내기' ctrl '정보 보기' ===
> |
```

위는 채팅방 화면으로, 실시간으로 채팅이 추가되며, 파란색으로 선택된 채팅에 대해 ctrl 키를 누르면 상세 정보가 출력됩니다.

```
MINGW64/c/dev/CppLangAI x + v
채팅방 이름: room1
shs | 5
shs | <band>
shs | 6
*Pleizz* | 안녕하세요
shs | 안녕
=== /back '뒤로 가기' /send <메시지> '메시지 보내기' ctrl '정보 보기' ===
[채팅 디테일] 닉네임: shs | 최근 활동: 3일 전 | 전송: 4일 전 | 내용: 5 | 읽음 (3명): *Pleizz*, shs, *최재원 [재학 / Language & AI융합전공]*,
> |
```

읽은 사람을 확인하는 기능을 구현하기 위해 서버단에서 사용하는 데이터베이스의 participation에 lastReadAt 필드를 추가하여 각 방에서 해당 유저의 마지막 확인 시간과 채팅 생성 시간을 비교하여 읽은 사람을 확인하도록 했습니다.

(1) 6846b38cfc731fe881642f8c	{ lastReadAt : 1749464420759 } (5 fields)	struct RoomState
🔑 _id	6846b38cfc731fe881642f8c	{
🔑 userId	6846b0e398534e988258e777	std::vector<Chat> chats;
🔑 roomId	68405d87fc731fe881641afe	std::unordered_map<std::string, time_t> participants;
📖 lastReadAt	1,749,464,420,759.0 (1.7T)	std::string roomName;
📅 joinedAt	2025. 6. 9. 오후 7:12:28 - 8 minutes ago	std::string roomId;
		std::string description;
		std::string latestChat;
		};

채팅 데이터에서 userId를 가지고 오고, RoomState.participants[userId]로 해당 유저의 접속 시간을 불러옵니다. 회원 정보는 아래와 같은 구조로 저장되어 있습니다.

```
struct Participant
{
    std::string id;
    std::string name;
    std::string nickname;
    std::string email;
    std::string picture;
    std::string status;
    time_t latest_access;
};

struct ParticipantState
{
    std::unordered_map<std::string, Participant> participants;
    std::mutex mutex;
};
```

여러 방에 참여할 수 있으므로 회원들의 정보는 한 곳에 두고, 각 방 별로 필요한 정보 (마지막 확인 시간 등)을 따로 두는 식으로 설계했습니다.

멀티 스레드 프로그램에서 하나의 콘솔을 다루는 것에 대한 고민으로, 웹소켓 수신부에서 채팅 이벤트를 받았을 때 다음과 같은 경우를 생각했습니다.

내가 보고 있는 채팅방이면 메시지를 출력해주기(정확히는 채팅방에서 채팅 개수가 변경되면 새로 렌더링하는 방식으로)

그런데 내가 그 채팅방을 안 보고 있으면 어떡하지? 이는 메시지박스를 통해 사용자에게 알려주는 방식으로 실시간성을 해치지 않으며 해결했습니다.



/create: 채팅방 이름과 설명을 입력받고 이를 생성합니다.

```
> /create
성명> 방 이름 > 방 설명
성명> 방 설명
/exit 종료 /my 내 정보 /rooms 참여할 방 목록
/create 방 만들기 /room-list 참여 가능한 방 목록
/status 모든 참여자의 상태 확인 /status <상태> 상태 메시지 설정
/nickname <닉네임> 닉네임 설정 /leaderboard 리더보드 확인

> 방이 성공적으로 생성되었습니다!
방 이름: 방성성
```

예쁘지 않게 출력되는데, 이는 웹소켓 수신 스레드에서 콘솔을 건드리기 때문입니다.

/create는 서버단에 방 생성 요청만 한 채로 끝이 나고, 수신 스레드에서 나머지 모든 절차(방 데이터 및 참여자 추가)가 진행되는데, 방을 생성/참여하는 과정이 유사해 함께 처리합니다.

/room-list: 참여 가능한 방 목록을 정보와 함께 가져옵니다. 당장은 내가 참여하지 않은 방의 목록을 가져오는데, 방을 선택하면 앞서 언급했던 /create와 같이 서버단에 방 참여 요청만 한 채로 끝이 나고, 수신 스레드에서 /create와 함께 처리됩니다.

```
else if (type == "create-room-res" || type == "join-room-res")
```

/status: 현재 참여중인 방의 가입자들에 한정하여 상태 메시지를 보여줍니다.

```
C:\WINDOWS\system32\cmd.exe
현재 참여중인 방의 가입자만 보여줍니다.

(나) *Pleizz* | <설정되지 않음>
*신지원[재학 / Language & AI융합학부]* | <설정되지 않음>
*최재원[재학 / Language & AI융합전공]* | <설정되지 않음>
*신혜성[재학 / Language & AI융합학부]* | <설정되지 않음>
shs | 자는 중~
=====

/exit 종료 /my 내 정보 /rooms 참여한 방 목록
/create 방 만들기 /room-list 참여 가능한 방 목록
/status 모든 참여자의 상태 확인 /status <상태> 상태 메시지 설정
/nickname <닉네임> 닉네임 설정 /leaderboard 리더보드 확인

> |
```

/status <상태 메시지>: 상태메시지를 설정하는 요청을 서버단에 전송하고 종료됩니다.

```
> /status 상태설정해제
/exit 종료 /my 내 정보 /rooms 참여한 방 목록
/create 방 만들기 /room-list 참여 가능한 방 목록
/status 모든 참여자의 상태 확인 /status <상태> 상태 메시지 설정
/nickname <닉네임> 닉네임 설정 /leaderboard 리더보드 확인

> 성공적으로 상태를 업데이트했습니다.
```

서버단의 응답을 받고 업데이트가 완료되면 "성공적으로 상태를 업데이트했습니다."를 출력합니다. (정확히는 상태 업데이트가 서버단에서 여러 클라이언트로 전파될 때, 업데이트 아이디가 내 아이디와 같을 때만 출력합니다.) /create, /room-list의 방 참여 등과 같이 서버단에 요청만 한 채로 끝이 나므로 출력이 예쁘지 않습니다.

/nickname <닉네임>: 닉네임을 수정합니다. OAuth 최초 로그인시 주어진 이름 양옆에 *를 붙인 것이 닉네임의 기본 값으로, 프로그램 내에서는 채팅창에서 ctrl을 누른 경우를 제외하면 상대방의 이름이 노출되지 않도록 했습니다. 마찬가지로 서버단에 요청만 한 채로 끝이 나 출력이 예쁘지 않습니다.

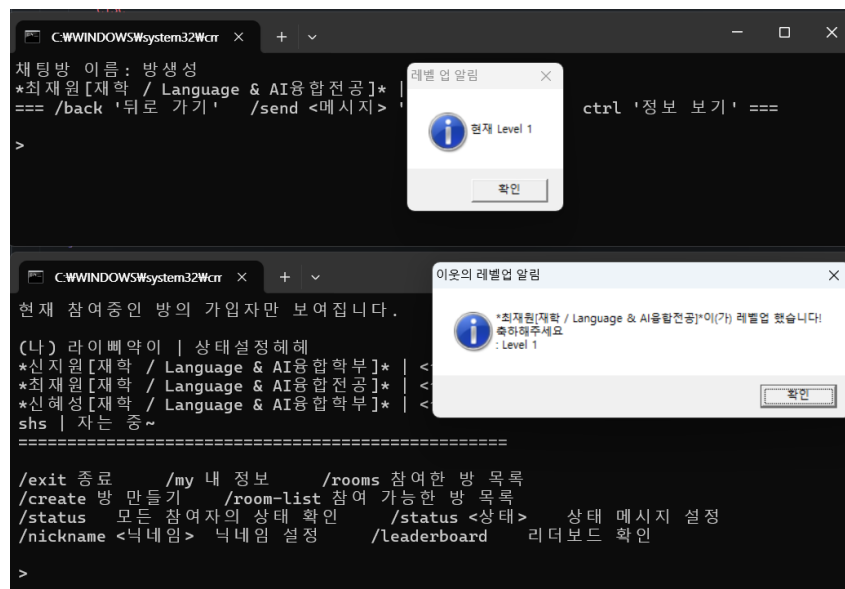
```
> /nickname 라이베약이
/exit 종료 /my 내 정보 /rooms 참여한 방 목록
/create 방 만들기 /room-list 참여 가능한 방 목록
/status 모든 참여자의 상태 확인 /status <상태> 상태 메시지 설정
/nickname <닉네임> 닉네임 설정 /leaderboard 리더보드 확인

> 성공적으로 닉네임을 업데이트했습니다.
```

/leaderboard: 채팅을 입력할 때마다 서버단에서 exp를 1씩 증가시키고, 레벨이 변경된 경우 해당 유저가 참여한 모든 방의 유저에게 전파합니다.

<pre> await propagateEvent("message", { roomId, chat: { userId: user.id, text: chat.text, createdAt: chat.createdAt, roomId: chat.roomId, }, }); room.latestChat = new Date(); await room.save(); const curLevel = getLevel(user.exp); user.exp += CHAT_EXP; await user.save(); const newLevel = getLevel(user.exp); if (newLevel > curLevel) { propagateEvent("level-up", { userId, level: newLevel }); } </pre>	<pre> } else if (type === "level-up") { const uids = await getUserIdsInSameRooms(userId); for (const { ws } of sockets.filter(v => uids.includes(v.userId))) { ws.send(JSON.stringify({ type: "level-up-event", data: { userId, level }, })); } } </pre>
채팅 처리 후 exp 처리	propagateEvent의 type이 "level-up"인 경우

클라이언트가 이 이벤트를 받았을 때, 그 아이디에 따라 다르게 분기 처리했습니다.



서버단에서 exp만을 저장하고 레벨은 계산하도록 하여 관리자가 파라미터 COEFFICIENT, EXPONENTIAL, CHAT_EXP를 수정하더라도 문제가 없도록 설계했습니다. 현재 exp별 레벨은 다음과 같습니다.

exp until	level
2	0
8	1
15	2
24	3
33	4

이렇게 적용된 레벨과 exp를 보여주는 기능입니다.

exp에 따라 서버단에서 정렬되었으며, 관계라는 개념을 도입하였습니다.

```
C:\WINDOWS\system32\cmd.exe
<< 리더보드 >>
같은 방에 참여하고 있을 경우 이웃으로 표기되며, 그렇지 않은 경우에는 낯선 사람으로 표기됩니다.

등수 | 관계 | 닉네임 | 레벨 | 경험치
1등 | 이웃 | shs | 2 | 11
2등 | 이웃 | *최재원[재학 / Language & AI융합전공]* | 1 | 3
3등 | 이웃 | *신혜성[재학 / Language & AI융합학부]* | 0 | 1
3등 | 낯선 사람 | *은혜* | 0 | 1
3등 | 낯선 사람 | *박은혜[재학 / Language & AI융합학부]* | 0 | 1
3등 | 나 | 라이베악이 | 0 | 1
4등 | 이웃 | *신지원[재학 / Language & AI융합학부]* | 0 | 0
4등 | 낯선 사람 | *hufs* | 0 | 0

/exit 종료 /my 내 정보 /rooms 참여한 방 목록
/create 방 만들기 /room-list 참여 가능한 방 목록
/status 모든 참여자의 상태 확인 /status <상태> 상태 메시지 설정
/nickname <닉네임> 닉네임 설정 /leaderboard 리더보드 확인

> |
```

클라이언트단 소스코드: https://github.com/jaeewon/compfund_client.git

서버단 소스코드: https://github.com/jaeewon/compfund_backend.git

모든 기능의 구현을 목적으로 하여 다소 C++스럽지 않게 완성되었지만 모든 기능이 잘 작동하여 잘 마무리할 수 있었습니다.

지난 시간에 배운 내용을 적용하지 못해 아쉬움이 큼니다.

한국외국어대학교 Language & AI 융합전공 202402050 최재원

2025.06.10.

본 문서는 최종 완성된 버전이 아닙니다.