쪽지 기능 Api list

message

POST	**/message/send**	쪽지 보내기
POST	**/message/get/partnerUserIdGroup**	가장 최근에 보내거나 받은 메시지가 있는 상대방 10명의 대화 가져오기
POST	**/message/get/partnerUserId**	상대방과의 대화 가져오기(읽음 상태로 변경 처리 포함)
POST	**/message/delete**	메시지 삭제

message-block-user

POST	**/message/block-user/create**	차단 유저 등록
POST	**/message/block-user/get**	차단 유저 가져오기
POST	**/message/block-user/delete**	차단 유저 삭제

Message Table 구조

type : 0(보냄) 1(받음)

	Field	Type	Not Null	Default Value
	created_at	datetime(6)	Not Null	CURRENT_TIMESTAMP
	updated_at	datetime(6) ON UPDATE CURRENT_TIMESTAMP(6)	Not Null	CURRENT_TIMESTAMP
	deleted_at	datetime(6)		
P	id	int	Not Null	
	user_id	int	Not Null	
	partner_user_id	int	Not Null	
	message	varchar(255)	Not Null	
	type	int	Not Null	
	is_read	int	Not Null	

가장 최근에 보내거나 받은 쪽지가 있는 10명의 쪽지 가져오기 - Service 부분

```
async getMessageByPartnerUserIdGroup(
  getMessageInDto: GetMessageInDto,
): Promise<GetMessageOutDto[]> {
  const session = this.requestContextProvider.get('session');
  const userMessageRepository = getCustomRepository(
    MessageRepository,
    gameTypeOrmModuleOptions[session.gameDbId].database,
  );

  const messageBlockUserRepository = getCustomRepository(
    MessageBlockUserRepository,
    gameTypeOrmModuleOptions[session.gameDbId].database,
  );

  const blockUsers = await messageBlockUserRepository.findByUserId(
    session.userId,
  );
  let blockUserIds: number[];
```

메시지 차단 유저를 제외하기 위해 차단 유저의 user Id를 가져 옴

```
const blockUsers = await messageBlockUserRepository.findByUserId(
  session.userId,
);
let blockUserIds: number[];

//id in or not in 쿼리 시에 array 가 비어 있으면 에러남.
if (!blockUsers.length) {
  blockUserIds = null;
} else {
  blockUserIds = blockUsers.map((row) => row.blockUserId);
}
const result = await userMessageRepository.findByPartnerUserIdGroup(
  session.userId,
  blockUserIds,
  getMessageInDto.offset,
);

const messageDto = result.map((it) => {
  const messageDto = MessageDto.fromEntity(it);
  messageDto.createdAt = it.createdAt;
  return messageDto;
});
```

Sub Query를 사용하여 가장 최근에 보내거나 받은 쪽지가 있는 대상을 찾음.

```
const queryBuilder = this.createQueryBuilder( alias: 'message');
// getRaw or getRawMany 를 사용하여 얻은 데이터를 dto나 entity화 시키기 위해
// selectionAliasName 을 프로퍼티 이름과 동일하게 맞춰줘야 한다.
const result = await queryBuilder
  .select( selection: 'message.created_at', selectionAliasName: 'createdAt') SelectQueryBuilder<Message>
  .addSelect( selection: 'message.id', selectionAliasName: 'id') SelectQueryBuilder<Message>
  .addSelect( selection: 'message.userId', selectionAliasName: 'userId') SelectQueryBuilder<Message>
  .addSelect( selection: 'message.partnerUserId', selectionAliasName: 'partnerUserId') SelectQueryBuilder<Message>
  .addSelect( selection: 'message.message', selectionAliasName: 'message') SelectQueryBuilder<Message>
  .addSelect( selection: 'message.type', selectionAliasName: 'type') SelectQueryBuilder<Message>
  .addSelect( selection: 'message.isRead', selectionAliasName: 'isRead') SelectQueryBuilder<Message>
  .from( entityTarget: (subQuery : SelectQueryBuilder<any> ) => {
    return subQuery
      .select( selection: 'm1.partnerUserId', selectionAliasName: 'partnerUserId') SelectQueryBuilder<any>
      .addSelect( selection: ' MAX(m1.created_at) ', selectionAliasName: 'created_at') SelectQueryBuilder<any>
      .from(Message, aliasName: 'm1') SelectQueryBuilder<Message>
      .where( where: 'm1.userId=:userId', parameters: { userId: userId }) SelectQueryBuilder<Message>
      .andWhere( where: 'm1.partner_user_id!=:userId', parameters: { userId: userId }) SelectQueryBuilder<Message>
      .andWhere( where: 'm1.partner_user_id NOT IN (:blockUserIds)', parameters: {
        blockUserIds: blockUserIds ? blockUserIds : false,
      }) SelectQueryBuilder<Message>
      .groupBy( groupBy: 'm1.partnerUserId') SelectQueryBuilder<Message>
      .orderBy( sort: 'created_at', order: 'DESC') SelectQueryBuilder<Message>
      .limit( limit: 10) SelectQueryBuilder<Message>
      .offset(offset);
  }, aliasName: 'm2') SelectQueryBuilder<unknown>
  .where( where: 'message.partnerUserId=m2.partnerUserId') SelectQueryBuilder<unknown>
  .orderBy( sort: 'createdAt', order: 'DESC') SelectQueryBuilder<unknown>
  .getRawMany();
```

Guild

guild

POST	`/guild/create`	길드 생성
POST	`/guild/get/my-guild`	유저의 길드 정보
POST	`/guild/get/ids`	여러 길드 정보(id Array)
POST	`/guild/get/name`	길드 정보(guild name)
POST	`/guild/get/recommended`	추천 길드
POST	`/guild/get/top-rank`	길드 랭킹 (누적 경험치 top 50)
POST	`/guild/update/signup-type`	길드 가입 타입 변경
POST	`/guild/update/introduction`	길드 소개글 변경
POST	`/guild/update/notice`	길드 공지사항 변경
POST	`/guild/update/emblem`	길드 엠블렘 변경
POST	`/guild/update/name`	길드명 변경
POST	`/guild/dismantling`	길드 해체

guild-signup

POST	`/guild/signup`	길드 가입 신청
POST	`/guild/signup/get/guild-list`	유저가 가입 신청한 길드 목록
POST	`/guild/signup/get/user-list`	길드에 가입 신청한 유저 목록
POST	`/guild/signup/permit`	길드 가입 신청 승인
POST	`/guild/signup/reject`	길드 가입 신청 거부
POST	`/guild/signup/cancel`	유저의 길드 가입 신청 취소

guild-invite

POST **/guild/invite** 길드 초대 생성(보내기)

POST **/guild/invite/get/user-list** 길드에서 초대한 유저

POST **/guild/invite/get/guild-list** 유저를 초대한 길드 목록

POST **/guild/invite/cancel** 길드의 길드 초대 취소

POST **/guild/invite/reject** 유저의 길드 초대 거부

POST **/guild/invite/accept** 유저의 길드 초대 승락(길드 가입)

guild-member

POST **/guild/member/get** 길드 맴버 정보

POST **/guild/member/update/type** 길드원 직위 변경

POST **/guild/member/withdrawal** 길드 탈퇴

POST **/guild/member/exile** 길드 맴버 추방

POST **/guild/member/attendance** 길드 출석체크

guild-board

POST **/guild/board/create** 길드 게시판 글 작성

POST **/guild/board/get** 길드 게시판 게시글 가져오기

POST **/guild/board/delete** 길드 게시판 글 삭제

guild-emblem

guild-log

guild-reward-box

POST **/guild/reward-box/get** 길드 보상 상자 정보 가져오기

POST **/guild/reward-box/confirm** 길드 보상 상자 수령

길드 생성 Method - Transaction 처리를 통해 길드 생성과 길드 맴버 생성 처리

```
async createGuild(createGuildInDto: CreateGuildInDto): Promise<GuildDto> {
  //동일한 길드 이름이 존재하는가?
  await this._checkGuildName(createGuildInDto.guildName);
  const queryRunner = await TransactionHelper.getQueryRunner(
    guildTypeOrmModuleOptions.name,
  );
  let result;
  await queryRunner.startTransaction();
  try {
    // 길드 생성
    const guild = new Guild();
    guild.guildName = createGuildInDto.guildName;
    guild.emblem = createGuildInDto.emblem;
    guild.introduction = createGuildInDto.introduction;
    guild.signUpType = EGuildSignUpType.Permit;
    result = await queryRunner.manager.save(guild);
    // 길드 생성 후 자신을 길드 맴버에 마스터 타입으로 추가
    const guildMember = new GuildMember();
    guildMember.guildId = result.id;
    guildMember.userId = this.requestContextProvider.get('session').userId;
    guildMember.memberType = EGuildMember.Master;
    guildMember.attendance = 1;
    guildMember.attendanceTime = TimeUtil.now();
    await queryRunner.manager.save(guildMember);
    await queryRunner.commitTransaction();
  } catch (e) {
    await queryRunner.rollbackTransaction();
    throw new InternalServerErrorException(e.message);
  } finally {
    await queryRunner.release();
  }

  return GuildDto.fromEntity(result);
}
```

길드 리워드 시스템 - 보상 지급 부분

Sharding 되어 있는 Game DB 에 각각 보상을 지급 하기 위한 로직

TypeOrm 의 Query Runner를 사용해 Transaction 처리 하였습니다.

```
const guildQueryRunner = await TransactionHelper.getQueryRunner(
  guildTypeOrmModuleOptions.name,

);
await Promise.all(
  gameDbIds.map(async (gameDbId) => {
    await gameDbQueryRunner[gameDbId].startTransaction();
  }),
);

await guildQueryRunner.startTransaction();
```

예외 처리를 사용해 Commit - RollBack 진행 하였습니다.

```
await guildQueryRunner.startTransaction();

try {
  await Promise.all(
    gameDbIds.map(async (gameDbId) => {
      await gameDbQueryRunner[gameDbId].manager.save(
        gameDbRewardMails[gameDbId],
        { reload: false },
      );
    }),
  );
  const deleteResult =
    await this.guildRewardBoxRepository.deleteByGuildId(
      confirmGuildRewardBoxInDto.guildId,
      guildQueryRunner.manager,
    );

  if (!deleteResult.affected) {
    throw new InternalServerErrorException(
      InternalErrorCode.GUILD_REWARD_BOX_CONFIRM_FAIL,
      description: 'GUILD_REWARD_BOX_CONFIRM_FAIL',
    );
  }
  await Promise.all(
    gameDbIds.map(async (gameDbId) => {
      await gameDbQueryRunner[gameDbId].commitTransaction();
    }),
  );
  await guildQueryRunner.commitTransaction();
} catch (e) {
  await Promise.all(
    gameDbIds.map(async (gameDbId) => {
      await gameDbQueryRunner[gameDbId].rollbackTransaction();
    }),
```