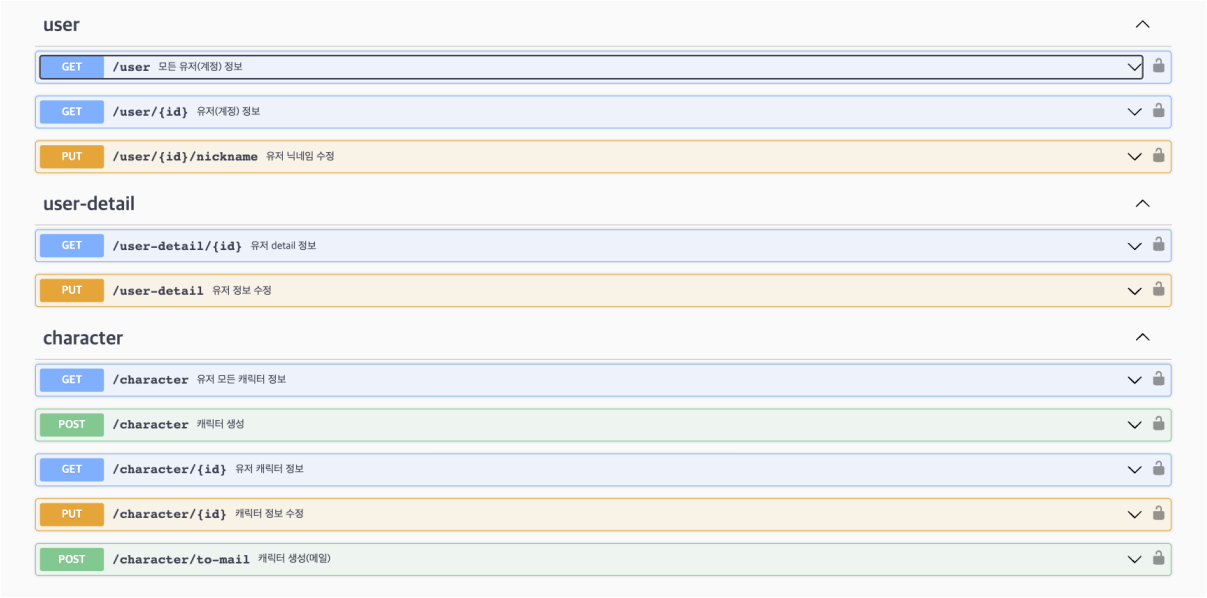
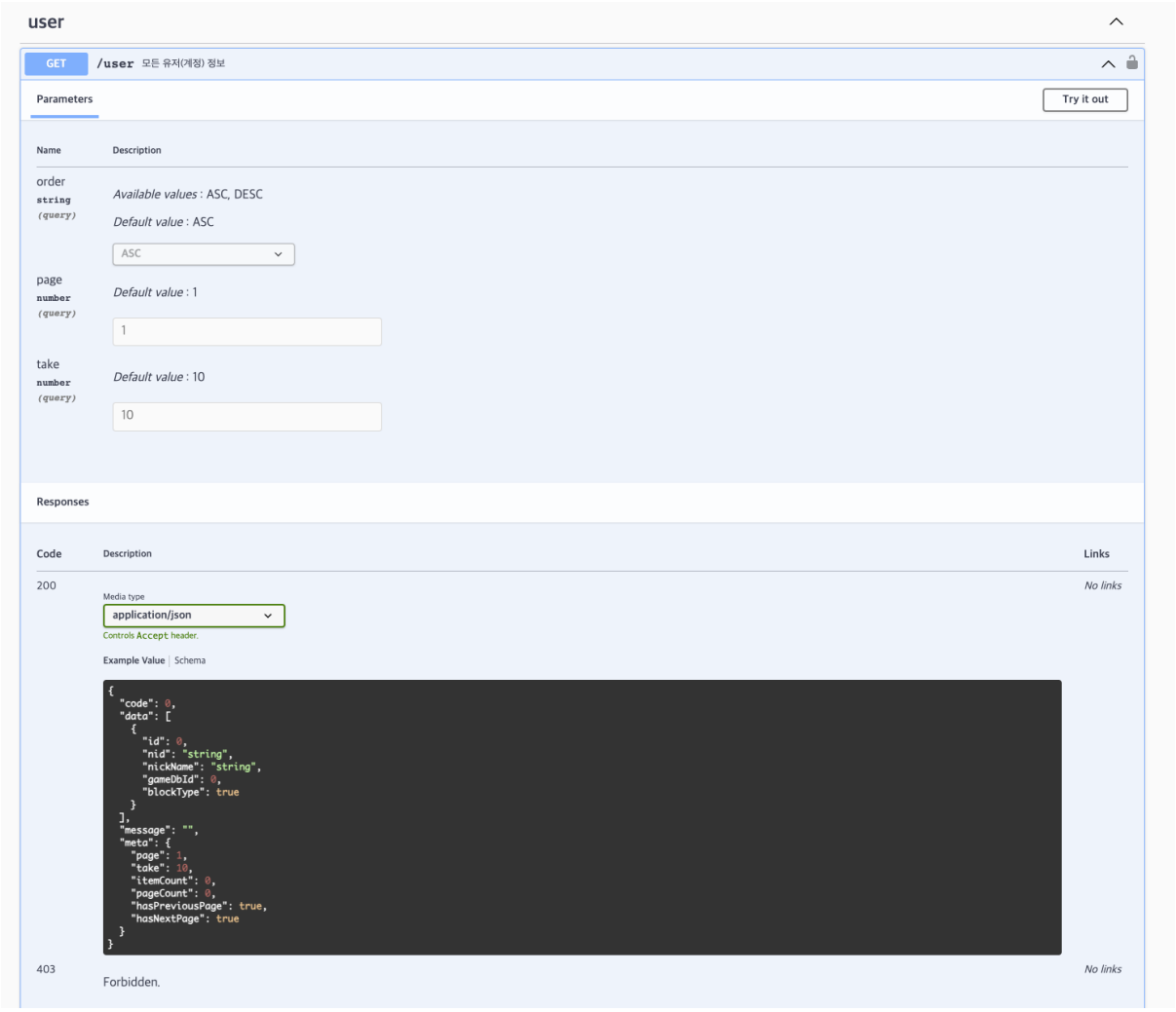
다음과 같이 Swagger 를 사용하여 Api 문서화를 진행 하였습니다.



전체 조회 시 Pagination 적용하는 부분



Controller 부분

```
@Get( path: '/')
@ApiResponseEntity( options: {
  type: UserDto,
  isPagination: true,
  summary: '모든 유저(계정) 정보',
})
async getUsers(
  @Query() pageOptionsDto: PageOptionsDto,
): Promise<ResponseEntity<UserDto[]>> {
  const [findUsersOutDto, pageMetaDto] = await this.userService.getUsers(
    pageOptionsDto,
  );
  return new ResponseEntity<UserDto[]>()
    .ok()
    .body(findUsersOutDto)
    .setPageMeta(pageMetaDto);
}
```

Service 부분

```
async getUsers(
  pageOptionsDto: PageOptionsDto,
): Promise<[UserDto[], PageMetaDto]> {
  const { order, page, take } = pageOptionsDto;
  const skip = (page - 1) * take;
  const [users, itemCount] = await this.userRepository.customFindAndCount(
    take,
    skip,
    order,
  );

  const usersDto = users.map((it) => {
    if (it.gameDbId !== 0) return UserDto.fromEntity(it);
  });

  const pageMetaDto = new PageMetaDto( {pageOptionsDto, itemCount}: { pageOptionsDto, itemCount });
  return [usersDto, pageMetaDto];
}
```

Pagination을 적용한 Repository Method

```
async customFindAndCount(
  take: number,
  skip: number,
  orderBy: 'ASC' | 'DESC',
): Promise<[User[], number]> {
  return await this.createQueryBuilder( alias: 'user')
    .take(take)
    .skip(skip)
    .orderBy( sort: 'user.id', orderBy)
    .getManyAndCount();
}
```

메일을 통한 재료 생성 - Controller

```
@Post( path: '/to-mail')
@ApiResponseEntity( options: {
  type: MailDto,
  summary: '재료 생성(메일)',
})
async createToMail(
  @Body() createMaterialInDto: CreateMaterialInDto,
): Promise<ResponseEntity<MailDto>> {
  await this.materialService.checkMaterialData(
    createMaterialInDto.materialID,
  );
  const createMailInDto = new CreateMailInDto();
  createMailInDto.userId = createMaterialInDto.userId;
  createMailInDto.mailType = EMailType.None;
  createMailInDto.goodsType = EGoodsType.Material;
  createMailInDto.goodsId = createMaterialInDto.materialID;
  createMailInDto.goodsValue = createMaterialInDto.count;

  const mailDto = await this.mailService.create(createMailInDto);

  return new ResponseEntity<MailDto>().ok().body(mailDto);
}
```

Sharding 되어 있는 Game DB에 접근 하기 위해 Typeorm Module을 커스텀 해서 사용하였습니다 (getCustormRepository 부분)

```
async create(createMailInDto: CreateMailInDto): Promise<MailDto> {
  const user = await this.userRepository.findById(createMailInDto.userId);
  if (!user) {
    throw new InternalServerErrorException(
      InternalErrorCode.USER_NOT_FOUND,
      description: 'USER_NOT_FOUND',
    );
  }
  const mailRepository = getCustomRepository(
    MailRepository,
    gameTypeOrmModuleOptions[user.gameDbId].database,
  );
  const mail = await mailRepository.save(createMailInDto);

  return MailDto.fromEntity(mail);
}
```