**Documentation for Project toDoctoDoc**

* 모든 영문자는 대소문자에 유의한다.
* 리턴값으로 던져지는 객체에 대해서 잘 모르겠으면 콘솔에 로그 찍어보고 JSON 트리 구조를 파악하길 바람.
* <중요!!!> 복잡한 객체, 즉 스키마가 정의된 객체는 스키마 선언 보고 그에 맞는 객체화를 시킬 것!!!!!! 그냥 트리 구조 그대로 활용하지 말고 객체화를 꼭 꼭 시키자.
* 보는 법
  + 요청 인자: 이러이러한 객체를 백엔드로 “넘겨 줘야 함”
    - 참고로 로그인 되어야 작동되는 기능은 당연히 토큰을 헤더에 담아서 줘야 함!
  + 반환 인자: 이러이러한 객체가 백엔드에서 “반환 됨” (정상적인 경우라면)
* 에러시 응답 코드는 401로 통일 해놨으니 아래 표준 응답 객체의 result 칸을 보고 어떤 오류인지 판단하여 분기 처리 및 디스플레이 하자. 각종 오류에 따라 상세한 result를 보여주니까 꼭 꼭 참고!
* 모든 RESTful API는 작성만 하고 치운게 아니라 thunder client로 작동 여부랑 반환되는 객체 유효성 다 확인했으니 문제가 있다면 프론트 문제가 아닌지 한번 더 생각 해 보길…
* 기본적으로 Access Token으로 접근하고 1회 실패하면 Refresh Token으로 접근하면 되고, Refresh Token으로 접근하면 헤더에 Access\_Token에 Access Token을 담고 Refresh\_token에 Refresh Token을 담아서 줌. 그런데 마음처럼 될지는 모르겠다 여튼 백엔드 지금은 이렇게 구현 해 놨다는 정도로 알고 있으셈
* 토큰이 만료되면 응답코드 419를 반환함, 이걸로 리프레시 토큰으로 리트라이 할 수 있음
* ‘S’번 API 이후로는 오류 코드도 다양하게 리턴하고 조건 명시할게…미안하다 고생시켜서
* 600에러: 의사가 아닌데 의사 메뉴에 접근
* 601에러: 아직 인증되지 않은 의사임
* 700에러: 프리미엄 계정이 아닌데 프리미엄 계정 기능에 접근
* 모든 데이터베이스 스키마는 기본적으로 \_id라는 고유값을 가지고, 이 코드가 각종 회원 정보, 게시물 조회 등등 모든 기능의 매개체이므로 별도 명시가 없더라고 \_id라는 속성이 있음을 명심하고, 그를 활용하자.
* 800에러: 게시물 소유자와 응답 요청자가 맞지 않을 경우
* 450에러: (프리미엄 계정)일일 큐레이팅 횟수(1회) 초과
* DM 아키텍쳐: 5장을 참고

1. 표준 JSON 응답 객체
   1. 텍스트, 스크린샷, 폰트이(가) 표시된 사진

      자동 생성된 설명의 규칙을 따른다.  
      즉 {  
       error: Boolean,  
       result: String,  
       content: Object,  
       }  
      의 형태를 따른다. 그런데 Content에 바로 String이 들어갈 수도 있고 Object로 감쌀 수도 있고
2. REST API :: 환자용 앱에서 사용하는 API 리스트
   1. /mapp/login
      1. POST
      2. 요청 인자: userid,password를 body의 인자로 받음.
      3. 반환 인자: {Access Token, Refresh Token}을 content에 담아 전송
      4. 로그인 하는 기능을 제공함.
   2. /mapp/logout
      1. GET
      2. 로그아웃 하는 기능을 제공함.
   3. /mapp/register v2
      1. POST
      2. 요청 인자: id,password,password2,nickname,postcode,address,detailAddress,extraAddress,email을 body의 인자로 받음.
      3. 회원가입 하는 기능을 제공함.
      4. 반환 인자: {Access Token, Refresh Token}을 content에 담아 전송
      5. 오류 코드
         1. 402: 패스워드 미일치
         2. 403: 이미 존재하는 ID
         3. 401: 서버 내부 오류
   4. /mapp/query
      1. POST
      2. 로그인이 되어 있어야 함.
      3. 요청 인자: input을 body의 인자로 받음.
      4. 반환 인자: content에 pageContent 객체를 담아서 전송, pageContent 객체의 형태:  
         텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

         자동 생성된 설명  
         title이 query한 내용의 제목,  
         context가 query한 내용에 대한 답변 내용임.
   5. /mapp/upload
      1. POST
      2. 로그인이 되어 있어야 함
      3. 요청 인자: title, content, content\_additional, tag를 body의 인자로 받음.
      4. 반환 인자: content에 Post 객체를 담아서 전송, Post 객체의 형태:  
         텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

         자동 생성된 설명Post 객체 스키마  
         여기에 고유한 \_id 속성까지 존재한다.
   6. /mapp/view/:id
      1. GET
      2. 로그인이 되어 있어야 함
      3. 요청 인자: id 매개변수로 Post의 \_id 속성을 받음. (예시: /mapp/view/12345)
      4. 반환 인자: content에 pageContent 객체를 담아 반환함:  
         텍스트, 스크린샷, 폰트이(가) 표시된 사진

         자동 생성된 설명  
         isOwner로 편집/삭제 기능을 활성화 시킬 수 있음.
   7. /mapp/edit/:id
      1. PATCH
      2. 로그인이 되어 있어야 함
      3. 로그인한 유저와 게시물의 작성자 id가 맞지 않으면 오류를 반환함
      4. 요청 인자: id 매개변수로 Post의 \_id 속성을 받음
      5. 게시물을 수정하는 역할을 함(따라서 PATCH후 view/로 다시 리다이렉션 해 줘야 함
   8. /mapp/delete/:id
      1. DELETE
      2. 로그인이 되어 있어야 함
      3. 로그인한 유저와 게시물의 작성자 id가 맞지 않으면 오류를 반환함
      4. 요청 인자: id 매개변수로 Post의 \_id 속성을 받음
      5. 게시물을 삭제하는 역할을 함(따라서 DELETE후 myPosts/로 다시 리다이렉션 해 줘야 함)
   9. /mapp/myPosts
      1. GET
      2. 로그인이 되어 있어야 함
      3. 반환 인자: 로그인한 유저가 작성한 전체 게시물을 Posts[ ]의 형태로 리턴함! 배열임에 주의해야 함 또 하나 중요한 점: title tag createdAt \_id만을 projection으로 리턴함.
   10. /mapp/tagSearch/:tag v2
       1. GET
       2. 로그인이 되어 있어야 함
       3. 요청 인자: tag 매개변수로 게시물의 태그를 검색하는 역할을 수행함.  
          이때, 가나다,마바사 <- 이렇게 (1가지 형태만 지원함에 주의, (가나다, 마바사) 또는 (가나다 마바사)형태는 지원하지 않음. 프론트엔드에서 스페이스나 콤마스페이스를 하나의 콤마로 통일해서 주는 파싱이 필요함) 주면 OR 조건으로 태그 검색을 실시함.
       4. 반환 인자: 해당 tag를 가진 게시물을 Posts[ ]의 형태로 리턴함! 배열임에 주의해야 함
   11. /mapp/graphBoard v2
       1. GET
       2. 로그인이 되어 있어야 함
       3. 그래프보드를 렌더링 하기 위한 객체를 반환하는데, 아래와 같은 형태임  
            
          \_tagList는 키:값 쌍을 가진 맵 형 자료구조임.  
          값은 그 태그를 가진 게시물의 수를 의미함.  
          즉 값이 클수록 큰 원을 가지게 됨.  
          \_tagGraph는 키:키 쌍을 가진 자료구조임.  
          중복된 키:키 쌍은(즉, A:B와 B:A)존재하지 않음.  
          서로 연결된 태그를 표시해 주기 위한 자료구조임.  
          이해 안되면 디스코드 남기면 바로 답장 해 줌
       4. v1과 달리 언더바가 앞에 생김에 주의!
   12. /mapp/userInfo
       1. GET
       2. 로그인이 되어 있어야 함
       3. 요청 인자: 없음
       4. 반환 인자: userinfo 객체를 반환하는데, MongoDB Projection이 되어 있어서 일부 필드만 전송됨. 전송되는 필드는 id usernick email address limits isPremium임.  
          userinfo 객체의 스키마는 아래의 스키마를 따르지만, 전송되는 필드가 한정되어 있음을 유념하길 바람. 또 address는 그 자체로 주소가 아니라 AddressSchema로 구성되어 있기에 이 역시 같이 볼 수 있도록 첨부함. 예를 들어 주소에 접근하려면 userinfo.address.address를 해야 한다는 뜻임. limits도 객체를 전송함  
          텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

          자동 생성된 설명limit 객체 스키마  
          텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

          자동 생성된 설명User 객체 스키마  
          텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

          자동 생성된 설명Address 객체 스키마
   13. /mapp/editUserInfo
       1. PATCH
       2. 로그인이 되어 있어야 함
       3. 요청 인자: {usernick, email, postcode, address, detailAddress, extraAddress, password, password2}를 body에 담아서 전송한다.
       4. 반환 인자: 성공적으로 작동(200)했을 경우 새 newUserInfo 객체를 반환한다. 이 객체의 특성은 /mapp/info에서 봤던 것과 동일하다.
       5. 여기서 중요한 점 !!!! 요청 인자에서 password에 내용이 있으면(8글자 이상)패스워드 변경을 시도하는 것으로 인식한다. 무슨 말인지 알죠? 그래서 프론트에서 볼 때는 패스워드 입력 칸을 초기에 웹버전처럼 비워 놓는 것이 중요하다.
   14. /mapp/dupidcheck
       1. POST
       2. 요청 인자: userid를 body에 담아서 전송한다.
       3. 반환 인자: error가 false면 사용 가능 id, 402 코드를 리턴함과 동시에 JSON 응답 객체텍스트, 스크린샷, 폰트이(가) 표시된 사진

          자동 생성된 설명이거  
          의 result가 “id\_already\_exists”면 사용 불가, 그 외에는 서버 에러 처리 하면 됨.
       4. 일반 유저의 id에 대해서만 중복 검사를 함. 즉 의사용 API가 아님.
   15. /mapp/dupemailcheck
       1. POST
       2. 요청 인자: email을 body에 담아서 전송한다.
       3. 반환 인자: error가 false면 사용 가능, 402 코드를 리턴함과 동시에 JSON 응답 객체의 result가 “email\_already\_exists”면 사용 불가, 그 외에는 서버 에러 처리 하면 됨.
   16. /mapp/aichat/new v2
       1. GET
       2. 로그인 되어 있어야 함
       3. 요청 인자: 없음
       4. 반환 인자: {chatid: 채팅방 고유한 id, startingMessage: 채팅방을 시작하는 메시지}  
          
       5. 에러 종류: 401(errorAtNewAiChat)
       6. AI 기반 새 채팅방을 개설함
   17. /mapp/aichat/get/:chatid
       1. GET
       2. 로그인 되어 있어야 함
       3. 당연히 자기 게시물에 대한 접근이어야 함(후술)
       4. 요청 인자: chatid 자리에 채팅방 id가 와야 함
       5. 반환 인자: AIChat 데이터베이스 스키마 기반 객체를 던져서 준다.  
          텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

          자동 생성된 설명구조는 이러함.
       6. 저 response 객체가 특이한데, 시간 순으로 모든 채팅이 동기식으로 나열되어 있음. 정확히 말하자면 Object의 Array인데, 각 Object는 ‘role’과 ‘content’ 속성을 가짐. ‘role’이 ‘user’이면 유저, ‘assistant’ 이면 AI가 쓴 것임. content 속성은 대화 내용임. 이 데이터를 기반으로 과거에 있었던 채팅을 띄워주는 것을 구현하는게 좋을 듯 함.(물론 로컬에 저장하는 방법도 있지만…)
       7. 만약 본인 채팅이 아닌데 비정상적인 방법으로 접근하면 401에러와 함께 notYourChat을 반환함. 근데 사실 401이면 웬만하면 서버 에러로 퉁치…면 안되고 위에서 봤듯이 ID 중복 검사 이런건 메시지 내용 검토까지 해야지. 그런데 웬만하면 서버 에러로 퉁치면 백엔드만 문제 있는 것 같잖아. 그건 또 아니지.
   18. [매우 중요] Websocket 기반 AI 채팅 :: Socket.io
       1. 사용 방법
          1. 네임스페이스 /aichat 을 사용
          2. path: /msg 를 사용
          3. Query에 다음 키로 해당되는 값을 넣어줘야 함:
             1. token: 토큰
             2. chatid: 채팅방 id
          4. 그러면 연결 할 수 있음.
          5. 그다음 emit 되어지는 대화를 받으면 됨.
       2. 채팅방을 퇴장 할 때는
          1. /mapp/aichat/save
             1. POST
             2. 로그인이 필요함
             3. 요청 인자: POST body에 채팅방 ID를 chatid로 넣어줘야 함
             4. 반환 인자: 지금까지 나눈 대화 내용을 기반으로 한 채팅방의 제목을 {title: }의 객체로 반환함.
       3. 채팅방을 삭제 할 때는
          1. /mapp/aichat/delete/:id v2
             1. DELETE
             2. 로그인이 필요함
             3. 당연히 자기 게시물을 삭제하는 것이어야 함
             4. 요청 인자: id 자리에 채팅방 ID
             5. 반환 인자: 없음
             6. 채팅방을 삭제함.
             7. 오류 코드:

401: 채팅방 id와 일치하는 채팅방이 없음

402: 본인의 채팅이 아님

403: 서버 내부 알고리즘 오류

* 1. /mapp/aichat/list v2
     1. GET
     2. 로그인이 되어 있어야 함
     3. 요청 인자: 없음
     4. 반환 인자: AIChat 데이터베이스 스키마 객체 배열 [] 을 제공한다.  
        텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

        자동 생성된 설명  
        ~~저 response 객체가 특이한데, 시간 순으로 모든 채팅이 동기식으로 나열되어 있음. 정확히 말하자면 Object의 Array인데, 각 Object는 ‘role’과 ‘content’ 속성을 가짐. ‘role’이 ‘user’이면 유저, ‘assistant’ 이면 AI가 쓴 것임. content 속성은 대화 내용임. 이 데이터를 기반으로 과거에 있었던 채팅을 띄워주는 것을 구현하는게 좋을 듯 함.(물론 로컬에 저장하는 방법도 있지만…)~~
     5. v2에서는 바뀜. title, recentMessage, charCreatedAt, chatEditedAt, \_id 속성만을 projection을 통해 반환하도록 데이터 경량화. recentMessage는 가장 최근의 AI의 대답을 저장하는 데이터베이스임.
     6. 오류 종류: 401(단순 오류)  
        201: 리스트가 비어있음
  2. /mapp/curate/around?radius=”someValue”&page=”someValue” v3
     1. GET
     2. 로그인이 되어 있어야 함
     3. 요청 인자: query의 radius 속성에 km 단위로 (someValue) 검색을 희망하는 거리를 제공해야 함.  
        query의 page 속성에 페이지 번호 (최대 45)를 적어야 함. 아무래도 radius 커지다 보면 밑으로 내릴 때 마다 새로고침? 뭔말인지 알지? 해서 새 페이지 반환하도록 하는거
     4. 반환 인자: {list: list}의 객체 안에 list 속성이 list를 가지도록 반환됨.  
        텍스트, 스크린샷, 폰트이(가) 표시된 사진

        자동 생성된 설명  
        list이기에 iterable하며, 각 element는 다음과 같은 속성을 지님:  
        텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

        자동 생성된 설명  
        그리고 여기에 덧붙여서 Boolean형의 isPremiumPsychiatry 속성을 가짐. 백엔드 단에서 프리미엄 의원은 정렬하여 상위 노출시켜서 리스트를 제공하나, 프리미엄 의원이라고 강조 표시를 하거나 하는 등의 작업은 프론트에서 해야 하므로 해당 속성을 제공하는 것임. 또한 isPremiumPsychiatry가 true면 추가적으로 stars라는 property도 지님. 5점 만점의 별점을 제공함.
     5. 오류 종류
        1. 404: noSuchUser
        2. 405: errorAtAroundAlgorithm
  3. /mapp/careplus/list
     1. GET
     2. 로그인 되어 있어야 함 / 프리미엄 계정이어야 함
     3. 요청 인자: 없음
     4. 반환 인자: 데이터베이스 스키마 Curate의 [](배열) 객체를 바로 반환함.  
        각 원소 내에 있는 comments[]의 길이로 코멘트 몇 개가 달렸는지 알 수 있음.  
        텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

        자동 생성된 설명  
        이 중에서 \_id(탐색에 사용되는 고유한 속성값)과 date 속성만 projection(데이터 경량화를 위해 일부 칼럼만 반환하는 메서드 기능)해서 줌.  
        따라서 프론트엔드는 000에 신청한 큐레이팅 <- 요정도로 표시해서 주고 \_id 속성은 내부적으로 사용하면 될 듯 함.
     5. 오류 종류
        1. 500, list GET 메서드 에러
  4. /mapp/careplus/post/:id
     1. GET
     2. 로그인 되어 있어야 함 / 프리미엄 계정이어야 함
     3. 요청 인자: URI 파라미터로 Curate 스키마의 \_id값을 넣어야 함.
     4. 반환 인자: 데이터베이스 스키마 Curate에 posts, ai\_chats, comments 스키마가 JOIN된 객체를 반환함. Post랑 AIChat 스키마는 앞에서 본 적이 있지만, comments 스키마는 처음 봤을 거임. 따라서 Comment 스키마 알려줌:  
        텍스트, 스크린샷이(가) 표시된 사진

        자동 생성된 설명  
        근데 여기서 doctor이 레퍼런스로 들어가 있음.   
        정리하자면, Curate는 Post[], AIChats[], Comment[] 스키마를 레퍼런스로 갖고 있고 Comment는 Doctor 스키마를 레퍼런스로 갖고 있음.  
        따라서 2중첩 JOIN(물론 SQL에서의 JOIN과 MongoDB에서의 populate는 약간 다르지만)된 것을 반환함.  
        즉 의사의 이름에 접근하려면 curate.comment[].doctor.name으로 접근해야 함.  
        Post, AIChat, 스키마 까먹었으면 위에 있으니까 보면 됨.  
        그리고 doctor의 property들 중 개인정보도 있으니까 당연히 projection(일부 필드만 걸러내는 것)을 함. name, address, phone, email, \_id만 패스함. \_id는 나중에 의사 프로필 기능 구현할 때 의사 프로필 사진 눌러서 의사 프로필 페이지로 들어갈 때 쓸 수 있을 것 같음.
     5. 에러 코드
        1. 800: 자신의 post가 아님
        2. 500: 서버 내부 알고리즘 에러
  5. /mapp/careplus/curate
     1. POST
     2. 로그인이 되어 있어야 함 / 프리미엄 계정이어야 함
     3. 요청 인자: 없음
     4. 반환 인자: Curate 스키마의 \_id값 (\_id 속성으로 반환함)  
        따라서 /post로 리다이렉션 해 주는건 프론트엔드의 몫임.
     5. 하는 역할: 큐레이팅 게시물을 등록함. 자동으로 최근 5개의 포스트와 ai 채팅을 큐레이팅 게시물에 등록함.
     6. 오류 코드
        1. 400: 큐레이팅 알고리즘 에러
  6. /mapp/careplus/post/:id
     1. DELETE
     2. 로그인이 되어 있어야 함 / 프리미엄 계정이어야 함
     3. 요청 인자: URI 파라미터 값으로 Curate 스키마의 \_id 속성을 받음
     4. 반환인자: 없음
     5. 오류 코드 종류
        1. 401: 등록된 유저 없음
        2. 402: 그런 큐레이팅 없음
        3. 403: 본인 큐레이팅이 아님
        4. 400: 큐레이팅 삭제 알고리즘 오류

1. REST API :: 의사용 앱에서 사용하는 API 리스트
   1. /mapp/doctor/dupemailcheck
      1. POST
      2. 로그인이 되어 있지 않아야 작동 가능
      3. 요청 인자: body에 email이라는 이름의 객체가 존재하고, 그 객체가 email을 담고 있어야 함.
      4. 반환 인자: 응답 코드 200이면 중복되는 이메일이 없다는 뜻임.
      5. email은 후술할 ID와 다르게 의사, 환자 전체의 데이터베이스를 대상으로 검색함.
      6. 오류 코드 종류
         1. 402: 이메일 중복
         2. 401: 서버 오류
   2. /mapp/doctor/dupidcheck
      1. POST
      2. 로그인이 되어 있지 않아야 작동 가능
      3. 요청 인자: body에 id라는 이름의 객체가 존재하고, 그 객체가 id를담고 있어야 함.
      4. 반환 인자: 응답 코드 200이면 중복되는 id가 아니라는 뜻임.
      5. 의사의 데이터베이스를 기준으로만 검색함.
      6. 오류 코드 종류
         1. 402: ID 중복
         2. 401: 서버 오류
   3. /mapp/doctor/login
      1. POST
      2. 로그인이 되어 있지 않아야 작동 가능
      3. 요청 인자: body에 {userid, password}라는 이름의 객체가 존재하고, 그 객체가 각각 상응하는 정보를 담고 있어야 함
      4. 반환 인자: 정상 응답(200)이라면 {token, refreshToken} 을 content에 담고 있음
      5. 오류 코드 종류
         1. 601: 현재 인증절차 진행 중인 계정
         2. 403: 등록된 유저 없음
         3. 401: 서버 오류
   4. /mapp/doctor/register
      1. POST
      2. 로그인이 되어 있지 않아야 작동 가능
      3. 요청 인자: body에 {id, password, password2, name, phone, personalID, doctorID, postcode, address, detailAddress, extraAddress, email}이라는 이름의 객체가 존재하고(대소문자 필히 유의!!) 그 객체가 각각 상응하는 정보를 담고 있어야 함. 여기서 personalID는 주민번호를 뜻하고, doctorID는 의사 면허 번호를 뜻함.
      4. 반환 인자: 정상 응답(200)이면 Register pending 상태.
      5. 오류 코드 종류
         1. 400: pw =/= pw2이거나, 중복되는 의사 정보가 존재하는 경우
         2. 401: 서버 오류
   5. /mapp/doctor/myComments
      1. GET
      2. 로그인이 되어 있고 의사여야 작동됨.
      3. 요청 인자: 없음
      4. 반환 인자: 정상 응답 (200)이면 Comment 스키마의 배열 [ ]을 반환함.  
         comment 스키마는 다음과 같음:  
         텍스트, 스크린샷이(가) 표시된 사진

         자동 생성된 설명  
         originalID는 게시물(큐레이팅 원본 게시물)의 ID를 뜻하니까, 자기가 쓴 코멘트 누르면 바로 그 게시물로 이동할 수 있도록 처리할 것.
      5. 오류 코드 종류
         1. 401: 서버 내부 에러
   6. /mapp/doctor/doctorInfo
      1. GET
      2. 로그인이 되어 있고 의사여야 작동됨.
      3. 요청인자: 없음
      4. 반환인자: 정상 응답(200)이면 Doctor 스키마에 password, chats, refreshToken을 제외한 필드의 projection이 되어 있음(로그 찍어보면 무슨 말인지 알 것)  
         또한 주민번호(personalID)와 의사 면허 번호(doctorID)는 편집 못하게 막아놓을 것.
      5. 오류 코드 종류
         1. 401: 서버 내부 에러
   7. /mapp/doctor/editDoctorInfo
      1. PATCH
      2. 로그인이 되어 있고 의사여야 작동됨.
      3. 요청인자:  
           
         password 변경 요청이 없으면 password 객체 자체를 안 넘겨주면 됨.
      4. 반환인자: 정상 응답(200). content 객체 내에 반환되는 내용 없음. doctorInfo refresh해서 로드하길 바람.
      5. 오류 코드 종류
         1. 401: 서버 내부 에러
   8. /mapp/doctor/curate?radius=”someValue”
      1. GET
      2. 로그인이 되어 있고 의사여야 작동됨
      3. 요청 인자: query 자리에 someValue로 km 단위로 된 반경 값을 입력
      4. 반환 인자: 정상 응답(200)이면 다음과 같은 형태의 객체를 반환함:  
         텍스트, 스크린샷, 폰트이(가) 표시된 사진

         자동 생성된 설명  
         이미 읽은 큐레이팅 게시물인지 아닌지는 isRead 속성이 true면 읽은 것, false면 아직 읽지 않은 것으로 받아들이면 됨.  
         comment의 개수는 comments[] 배열의 길이 속성인 .length로 접근하면 됨.  
         흔히 알고 있는 Curate 스키마, 그러니까 아래 이거  
         텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

         자동 생성된 설명에서 user의 일부 필드만 INNER JOIN되고 그마저도 projection으로 불필요한 필드를 빼낸 객체가 리턴됨.   
         즉, JSON Content에 curation 자체의 \_id 속성과 “”업데이트””(생성 일자 아님!) 일자 속성인 date, 그리고 user.usernick 속성만 남아있음. \_id, data, user.usernick 요 3개 꼭 기억하삼.  
         그래서 000님이 신청한 큐레이팅 하고 밑에 업데이트 일자: 0000-00-00 00:00 하면 될 듯. 이건 디자인 정해지면 그때 가서 해도 좋고.
      5. 오류 코드 종류
         1. 403: 큐레이팅 알고리즘(서버 오류)오류
   9. /mapp/doctor/curate/details/:id
      1. GET
      2. 로그인이 되어 있고 의사여야 작동됨
      3. 요청 인자: id에 curate 게시물 객체의 \_id 속성을 입력
      4. 반환 인자: 정상 응답(200)이면 posts, ai\_chats, comments가 JOIN 된 Curate 객체를 반환.
      5. 사용 목적: 큐레이팅 게시물 조회
      6. 오류 코드
         1. 401: \_id 속성에 맞는 게시물이 없음
         2. 403: 서버 내부 알고리즘 오류
   10. /mapp/doctor/curate/comment/:id
       1. POST
       2. 로그인이 되어 있고 의사여야 작동됨
       3. 요청 인자: id에 curate 게시물 객체의 \_id 속성을 입력, body에 comment라는 이름의 객체로 코멘트가 존재해야 함.
       4. 반환 인자: 정상 응답(200). content 내에는 아무런 객체도 반환하지 않음.  
          따라서 댓글을 단 후에 /mapp/doctor/curate/details/:id를 새로고침 해 주는 작업이 필요.
       5. 사용 목적: 게시물에 댓글을 달 때 쓰는 함수임.
       6. 오류 코드
          1. 401: \_id 속성에 맞는 게시물이 없거나 코멘트가 1자 미만임.’
          2. 402: 1개의 게시물에 1명이 2개 이상의 코멘트를 달려고 시도하는 경우(도배 방지를 위한 것)
          3. 403: 서버 내부 알고리즘 오류
   11. /mapp/doctor/curate/commentModify/:id
       1. PATCH
       2. 로그인이 되어 있고 의사여야 작동됨
       3. 요청 인자: id에 comment 코멘트 객체의 \_id 속성을 입력, body에 comment라는 이름의 객체로 수정될 코멘트가 존재해야 함
       4. 반환 인자: 정상 응답(200). content 내에는 아무런 객체도 반환하지 않음.  
          따라서 수정한 후에 /mapp/doctor/curate/details/:id를 새로고침 해 주는 작업이 필요.
       5. 사용 목적: 게시물에 내가 단 댓글을 수정할 때 쓰는 함수임.
          1. 내가 쓴 댓글인지 어떻게 아는가?
             1. 댓글의 doctor 속성과 나의 로그인 정보에서 얻을 수 있는 \_id 속성이 같은지 확인한다.
       6. 오류 코드
          1. 401: 본인이 단 코멘트가 아님.
          2. 403: 서버 내부 알고리즘 오류
   12. /mapp/doctor/curate/commentModify/:id
       1. DELETE
       2. 로그인이 되어 있고 의사여야 작동됨
       3. 요청 인자: id에 comment 코멘트 객체의 \_id 속성을 입력
       4. 반환 인자: 정상 응답(200). content 내에는 아무런 객체도 반환하지 않음.  
          따라서 삭제한 후에 /mapp/doctor/curate/details/:id를 새로고침 해 주는 작업이 필요.
       5. 사용 목적: 게시물에 내가 단 댓글을 삭제할 때 쓰는 함수임.
          1. 내가 쓴 댓글인지 어떻게 아는가?
             1. 댓글의 doctor 속성과 나의 로그인 정보에서 얻을 수 있는 \_id 속성이 같은지 확인한다.
       6. 오류 코드
          1. 401: 본인이 단 코멘트가 아님
          2. 403: 서버 내부 알고리즘 오류
2. ~~DM 관련 API~~
   1. ~~환자용~~
      1. ~~/mapp/careplus/dm/list~~
         1. ~~GET~~
         2. ~~로그인이 되어 있고 프리미엄 계정이어야 작동됨.~~
         3. ~~요청 인자: 없음~~
         4. ~~반환 인자: 정상 응답(200)이면 Chat 스키마 중 doctor의 이름(\_id 속성도 제공)과 date가 projection된 배열을 리턴하거나, 채팅이 없다면 201 및 emptyChatList라는 메시지를 반환함. 무슨 말인지 잘 모르겠으면 JSON 로그에 찍어보길. pretty하게 출력해주는 외부 플러그인도 많은걸로 아니까 눈에 잘 안들어오면 그런걸로 다듬어서 한번 보길 바람.~~
         5. ~~Chat 스키마는 아래 API에 설명되어 있음.~~
         6. ~~오류 코드~~
            1. ~~403: 서버 내부 알고리즘 오류~~
      2. ~~/mapp/careplus/dm?uid=”someValue”&did=”someValue”~~
         1. ~~GET~~
         2. ~~로그인이 되어 있고 프리미엄 계정이어야 작동됨~~
         3. ~~요청 인자: URI의 쿼리에 uid로 유저의 \_id와 did로 작성한 코멘트에 같이 등록된 doctor(doctor의 \_id 속성임; 잘 모르겠으면 상기 Comment 스키마 참조)의 \_id 속성을 받음. 즉 유저 \_id가 1234고 의사 \_id가 5678이면  
            /mapp/careplus/dm?uid=1234&did=5678  
            인 것임.~~
         4. ~~반환 인자: 정상 응답(200)이면 {chatid: 채팅방 고유번호(Chat 스키마의 \_id 속성)을 반환함.  
            이 때, 기존에 채팅방이 존재하지 않았을 경우 새 채팅방을 만듦.  
            Chat 스키마:  
            텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

            자동 생성된 설명~~
         5. ~~사용 목적: 코멘트에 “이 의사와 대화할래요!”든 “이 의사와 예약 신청” 등등…아무튼 적당한 말로 의사랑 DM 할 수 있는 소통창구를 만들고 그걸 누르면 uid와 코멘트에 있는 did를 기반으로 채팅방 고유번호를 반환한다.~~
         6. ~~오류 코드~~
            1. ~~401: 본인의 uid가 아님~~
            2. ~~403: 서버 알고리즘 오류~~
      3. ~~/mapp/careplus/dm/:chatid~~
         1. ~~GET~~
         2. ~~위의 API와 매우 유사하지만 쿼리 변수 방식이 아닌 파라미터 변수 방식임에 유의!~~
         3. ~~요청 인자: 채팅방 고유번호(Chat 스키마의 \_id 속성)~~
         4. ~~반환 인자: Chat 스키마 그 자체를 반환함~~
         5. ~~사용 목적: 채팅방 내용 불러오기를 위한 API임.~~
         6. ~~오류 코드~~
            1. ~~401: 채팅방 고유번호 오류~~
            2. ~~402: 본인의 채팅이 아님~~
            3. ~~403: 서버 알고리즘 오류~~
   2. ~~의사용~~
3. Socket.io 기반 지속 가능한 스케일러블 채팅 구현
   1. Key idea: DM List에 들어왔을 때부터 Socket Connection 시작!  
      그러면 다음과 같이 event를 정의할 수 있음(메서드가 아닌 이벤트임에 주의! 예를 들어 connection::chatList는 연결 상태에서 chatList 이벤트를 발생시킨다는 것임!)
   2. NOTE: 의사 측에서 호출 할 때는 아래 이벤트 명 뒤에 \_doctor 의 접미사, 유저 측에서 호출 할 때는 아래 이벤트 명 뒤에 \_user 의 접미사를 붙인다.  
      단, chatList는 공용 API이므로 접미사를 붙이지 않고 사용한다.
   3. message의 Socket.io 기준 path는 /msg이며,  
      namespace는 /dm 이다.  
      반드시 handshake시의 query 객체에 .token으로 토큰을 넣어 줘야 한다!
      1. connection::chatList
         1. DM List에 들어와 있는 상태.
         2. 채팅방 화면 새로고침 용도로 사용.
         3. 요청 인자: 없음
         4. 반환 인자: unreadChats 데이터 전송 (구조는 recentMessage: “someValue”, unread: “someValue” 객체의 배열임. JSON 뜯어보면 바로 알 수 있음.)
      2. connection::joinChat
         1. 채팅방에 입장할 때 호출함.
         2. DB에서 채팅 내용을 불러옴.
         3. 요청 인자: data 영역에 방 번호를 plain text로 전달.
         4. 반환 인자: Chat 스키마 덩어리 하나
      3. connection::leaveChat
         1. 채팅방을 떠날 때 호출함.
         2. 요청 인자: data 영역에 방 번호를 plain text로 전달.
         3. 반환 인자: 없음
      4. connection::sendChat
         1. 메시지를 전송 할 때 호출함.
         2. 요청 인자: data 영역에 {roomNo: 방 번호, message: 메시지}로 구성된 객체를 전달해야 함.
         3. 반환 인자: 없음
         4. 액션:   
            채팅방에 두 명이 모두 있는 상태에서, 환자가 보낼 때는 recvChat\_doctor 이벤트를 발생시키고, 의사가 보낼 때는 recvChat\_user 이벤트를 발생시킨다. 이것으로 환자와 의사가 메시지를 송수신 받을 수 있도록 구성 해 놓았다.  
            메시지 객체는 다음과 같이 던져진다:  
              
            role은 보내는 사람의 역할이다.  
            또한, 채팅방에 한 명만 있는 상태에서는, 환자가 보낼때는 unread\_doctor 이벤트를 발생시키고, 의사가 보낼 때는 unread\_user 이벤트를 발생시킨다.  
            이것으로 읽지 않은 채팅을 프론트에서 구현 할 수 있다.