Week1_#3

배포된 API들의 Response 중에서 부족한 데이터

▼ (기능 1) [POST] 회원가입 : `/join`

```
- 기능 1 [POST] 회원가입 : `/join`

Request Header : 아직 로그인해서 JWT를 발급받기 전

Request Body :
  {
    "username":"{입력받은 유저명}",
    "email":"{입력받은 이메일}",
    "password":"{입력받은 패스워드}"
  }

Response Body :

  (회원가입 성공) {
    "success": true,
    "response": null,
    "error": null
  }

  (회원가입 실패) {
    "success" : false,
    "response" : null,
    "error" : {
      (이메일 형식 오류) {
        "message" : "이메일 형식으로 작성해주세요:email"
      }
      (비밀번호 형식 오류 - 표현식) {
        "message": "영문, 숫자, 특수문자가 포함되어야하고 공백이 포함될 수 없습니다.:password"
      }
      (이메일 중복 오류) {
        "message": "동일한 이메일이 존재합니다 : {입력받은 이메일}"
      }
      (비밀번호 형식 오류 - 길이) {
        "message": "8에서 20자 이내여야 합니다.:password"
      },
        "status" : 400
    }
  }
```

▼ (기능 2) [POST] 로그인 : `/login`

```
- 기능 2 [POST] 로그인 : `/login`

Request Header : 아직 로그인해서 JWT를 발급받기 전

Request Body :
  {
    "email":"{입력받은 이메일}",
    "password":"{입력받은 패스워드}"
  }

Response Body :

  (로그인 성공) {
    "success": true,
    "response": null,
    "error": null
  }

  (회원가입 실패) {
    "success" : false,
    "response" : null,
    "error" : {
      (이메일 형식 오류) {
        "message" : "이메일 형식으로 작성해주세요:email",
        "status" : 400
      }
      (비밀번호 형식 오류 - 표현식) {
        "message": "영문, 숫자, 특수문자가 포함되어야하고 공백이 포함될 수 없습니다.:password",
        "status" : 400
      }
      (비밀번호 형식 오류 - 길이) {
        "message": "8에서 20자 이내여야 합니다.:password",
        "status" : 400
      }
      (가입되지 않은 사용자) {
        "message": "인증되지 않았습니다",
        "status" : 401
      }
    }
  }
```

▼ (기능 3) [POST] 로그아웃 : `/logout`

```
- 기능 3 [POST] 로그아웃 : `/logout`

Request Header : 쿠키의 Authorization Key에 JWT 값을 담아보냄

Request Body :
  {
    뭘 담아보내건 같은 500 Error가 발생
  }

Response Body :

  (로그아웃 실패) {
    "success" : false,
    "response" : null,
    "error" : {
        "message": "Request method 'GET' not supported",
        "status": 500
      }
  }

=> GET으로 보내나 POST로 보내나 500 Error를 발생시킴
=> 보다 정확하고 명확한 에러 처리에 대한 구현이 필요해보임.
```

▼ (기능 4) [GET] 전체 상품 목록 조회 : `/products`

```
- 기능 4 [GET] 전체 상품 목록 조회 : `/products`

Request Header : 필요없음
Request Parameter :
  `?page={num} (default: 0)`

Response Body :
  (조회 성공) {
    "success": true,
    "response": [
      (id 1~9까지 반복) [+num*9] {
        "id": 1,
        "productName": "{해당 product_id의 제품명}",
        "description": "",
        "image": "{해당 product_id의 이미지 URL}",
        "price": {해당 product_id의 가격}
      }, x 9
    ],
    "error": null
  }

  (렌더링 가능한 페이지를 넘은 경우) {
```

```
    "success": true,
    "response": [],
    "error": null
  }
```

=> 요청 가능한 페이지를 넘겨도 오류가 발생하지 않는다

=> 페이지네이션을 구현하려면 Front-end와 요청한 페이지 넘버를 서로 확인할 필요성도 있어보인다

▼ (기능 5) [GET] 개별 상품 상세 조회 : `/products/{product_id}`

```
- 기능 5 [GET] 개별 상품 상세 조회 : `/products/{product_id}`

Request Header : 필요없음
Request Parameter : 없음

Response Body :
  (조회 성공){
    "success": true,
    "response": {
            "id": {해당 product_id},
            "productName": "{해당 product의 제품명}",
            "description": "",
            "image": "{해당 product_id의 이미지 URL}",
            "price": {해당 product_id의 가격},
            "starCount": {해당 product_id의 rate},
            "options": [
        (해당 product의 option 개수만큼 반복) {
          "id": {해당 option_id},
          "optionName": "{해당 option_id의 옵션명}",
          "price": {해당 product_id의 가격}
        }, [x N]
            ]
    },
    "error": null
    }

  (보유한 product_id 데이터 범위를 넘어선 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "해당 상품을 찾을 수 없습니다 : {요청한 product_id}",
      "status": 404
    }
  }
```

▼ (기능 8) [POST] 장바구니 담기 : `/carts/add`

```
- 기능 8 [POST] 장바구니 담기 : `/carts/add`
Request Header : 쿠키의 Authorization Key에 JWT 값을 담아보냄
Request Body :
  [
    (client가 선택한 option의 개수만큼 반복) {
      "optionId": {선택한 option의 id},
      "quantity": {선택한 option의 개수}
    }, [x N]
  ]

Response Body :
  (장바구니 담기 성공) {
    "success": true,
    "response": null,
    "error": null
  }


  (JWT가 유효하지 않은 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "인증되지 않았습니다",
      "status": 401
    }
  }

  (동일한 옵션을 장바구니에 다시 추가할 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "장바구니 담기 중에 오류가 발생했습니다 : could not execute statement; SQL
 [n/a]; constraint [\\"PUBLIC.UK_CART_OPTION_USER_INDEX_4 ON PUBLIC.CART_TB(USER_ID N
ULLS FIRST, OPTION_ID NULLS FIRST) VALUES ( /* key:1 */ 1, 1)\\"; SQL statement:\\nin
sert into cart_tb (id, option_id, price, quantity, user_id) values (default, ?, ?, ?,
?) [23505-214]]; nested exception is org.hibernate.exception.ConstraintViolationExcep
tion: could not execute statement",
      "status": 500
    }
  }

  (보유한 범위를 넘어선 option_id를 요청할 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "해당 옵션을 찾을 수 없습니다 : {요청한 option_id}",
```

```
      "status": 404
    }
  }
```

▼ (기능 9) [POST] 장바구니 보기 : `/carts`

```
- 기능 9 [POST] 장바구니 보기 : `/carts`

Request Header : 쿠키의 Authorization Key에 JWT 값을 담아보냄
Request Body : 없음

Response Body :
  (장바구니 조회 성공) {
    "success": true,
    "response": {
      "products": [
        (장바구니에 담긴 product_id의 개수만큼 반복) {
          "id": {해당 옵션을 포함하는 product_id},
          "productName": "{해당 product의 제품명}",
          "carts": [
            (해당 product에서 장바구니에 추가된 옵션 수만큼 반복) {
              "id": {cart_id = 장바구니에 추가된 순서대로 Auto-Increment},
              "option": {
                "id": {장바구니에 추가된 option_id},
                "optionName": "{해당 option_id의 옵션명}",
                "price": {해당 option_id의 가격}
              },
              "quantity": {장바구니에 추가한 수량},
              "price": {옵션 가격 x 추가한 수량}
            }, [x N1]
          ]
        }, [x N2]
      ],
      "totalPrice": {장바구니에 추가된 모든 price의 합}
    },
    "error": null
  }


  (JWT가 유효하지 않은 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "인증되지 않았습니다",
      "status": 401
    }
  }
```

▼ (기능11) [POST] 주문 : `carts/update`

```
- 기능11 [POST] 주문 : `carts/update`

Request Header : 쿠키의 Authorization Key에 JWT 값을 담아보냄
Request Body :
  [
    (수정한 품목 수만큼 반복){
      "cartId": {수정한 품목의 cartId},
      "quantity": {주문하기 화면에서 수정한 quantity}
    }, [x N]
  ]


Response Body :
  (장바구니 수정 성공) {
    "success": true,
    "response": {
      "carts": [
        (장바구니에 추가된 옵션의 개수만큼 반복){
          "cartId": {cartId},
          "optionId": {해당하는 option_id},
          "optionName": "{해당 option_id의 옵션명}",
          "quantity": {장바구니에 추가한 수량},
          "price": {옵션 가격 x 추가한 수량}
        }, [x N]
      ],
      "totalPrice": {장바구니에 추가된 모든 price의 합}
    },
    "error": null
  }

  (장바구니 수정 실패) {
    "success": false,
    "response": null,
    "error": {
      "message": "장바구니에 없는 상품은 주문할 수 없습니다 : {요청한 cart_id}",
      "status": 400
    }
  }

  (JWT가 유효하지 않은 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "인증되지 않았습니다",
      "status": 401
    }
```

```
  }
```

▼ (기능12) [POST] 결제 : `orders/save`

```
- 기능12 [POST] 결제 : `orders/save`

Request Header : 쿠키의 Authorization Key에 JWT 값을 담아보냄
Request Body : 없음

Response Body :
  (결제 성공) {
    "success": true,
    "response": {
      "id": {order_id = 주문한 순서대로 Auto-Increment},
      "products": [
        (주문된 product_id의 개수만큼 반복){
          "productName": "{해당하는 product_id의 상품명}",
          "items": [
            (구매한 상품 중 해당 product_id에 포함되는 개수만큼 반복){
              "id": {cart_id},
              "optionName": "{해당하는 품목의 옵션명}",
              "quantity": {구매한 개수},
              "price": {옵션 가격 x 구매한 개수}
            }, [x N1]
          ]
        }, [x N2]
      ],
      "totalPrice": {구매한 상품의 모든 price 총합}
    },
    "error": null
  }

  (장바구니에 내역이 존재하지 않을 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "장바구니에 아무 내역도 존재하지 않습니다",
      "status": 404
    }
  }

  (JWT가 유효하지 않은 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "인증되지 않았습니다",
      "status": 401
    }
```

```
  }
```

▼ (기능13) [GET] 주문 결과 확인 : `orders/{orders_id}`

```
- 기능13 [GET] 주문 결과 확인 : `orders/{orders_id}`
Request Header : 쿠키의 Authorization Key에 JWT 값을 담아보냄
Request Body : 없음

Response Body :
  (조회 성공) {
    "success": true,
    "response": {
      "id": {order_id = 주문한 순서대로 Auto-Increment},
      "products": [
        (주문된 product_id의 개수만큼 반복){
          "productName": "{해당하는 product_id의 상품명}",
          "items": [
            (구매한 상품 중 해당 product_id에 포함되는 개수만큼 반복){
              "id": {cart_id},
              "optionName": "{해당하는 품목의 옵션명}",
              "quantity": {구매한 개수},
              "price": {옵션 가격 x 구매한 개수}
            }, [x N1]
          ]
        }, [x N2]
      ],
      "totalPrice": {구매한 상품의 모든 price 총합}
    },
    "error": null
  }

  (해당 주문 내역이 존재하지 않을 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "해당 주문을 찾을 수 없습니다 : {요청한 order_id}",
      "status": 404
    }
  }

  (JWT가 유효하지 않은 경우) {
    "success": false,
    "response": null,
    "error": {
      "message": "인증되지 않았습니다",
      "status": 401
    }
```

```
  }
```

▼ [POST] 이메일 중복 체크 : `/check`

```
- [POST] 이메일 중복 체크 : `/check`

Request Header : 아직 로그인해서 JWT를 발급받기 전

Request Body :
  {
    "email":"{입력받은 이메일}",
  }

Response Body :

  (중복 체크 통과) {
    "success": true,
    "response": null,
    "error": null
  }

  (실패) {
    "success" : false,
    "response" : null,
    "error" : {
      (이메일 형식 오류) {
        "message" : "이메일 형식으로 작성해주세요:email",
        "status" : 400
      }
      (중복된 이메일 오류) {
        "message": "동일한 이메일이 존재합니다 : ssar@nate.com",
        "status": 400
      }
    }
  }
```