

Rookie Hero Project

미니 요기요 - 2차 스프린트

태형 - 2차 스프린트 구현 내용

- 주문표 관련 API 작성 마무리 (1차 스프린트 때 완성하지 못한 것들)
- 주문표 수량 수정 부분 Ajax로 동작하는 부분 자바스크립트 상에서 동작하도록 수정
- **주문 페이지** 보기 (주문자 정보 + 주문자 추가정보 + 결제정보 +주문함정보)
- **주문 페이지 정보 입력 후, 주문하기** ('결제 중' 단계 까지)
- 랜덤 메뉴 Pick 페이지 생성, 메뉴 추천 방법 결정

태형 - 랜덤 메뉴 Pick 구현 계획

- 소개

1. 사용자의 취향과 각 메뉴들의 점수를 기준으로, 랜덤으로 메뉴를 골라준다.

- 사용 방법

1. 랜덤 메뉴 Pick 하기 메뉴에 들어간다.
2. 나의 취향에 해당하고, 점수가 높은 메뉴 중 5개가 랜덤으로 선택되어 화면에 보여진다.
3. 해당 메뉴 디테일 페이지로 이동 혹은 해당 메뉴를 주문함에 담을 수 있다.

태형 - 랜덤 메뉴 Pick 구현 계획

taehyoung.kwon 님의 랜덤 메뉴 Pick



태형 - 랜덤 메뉴 Pick 구현 계획



태형 - 랜덤 메뉴 Pick 구현 계획

- 메뉴 점수 구하는 방법

1. 메뉴 점수를 구하기 위해서 '조회 수(0.5)', '좋아요(2.5)', '주문 수(1.5)'에 각각 임의의 가중치를 매긴다.
2. 사장님 추천 메뉴(2.5), 랜덤(3)
3. 메뉴당 만점을 받을 시, 10점이며, 나의 취향 중 메뉴 점수가 높은 메뉴 N개 중 5개를 랜덤으로 골라서, 추천해 준

#매운맛

#국물

#10,000원 이하

태형 - 스프린트 간 막혔던 부분

- **CSRF Token**

- API 뷰에 기존에 @csrf_exempt를 적용했던 것을, 코드리뷰 이후, **매 요청마다, csrf token을 헤더에 실어서 요청하는 방식으로 변경**
- Django의 HTTP 테스트 클라이언트는 csrf token 없이도, 요청을 거부하지 않도록, 기본 설정 (**enforce_csrf_checks=False**)

```
>>> from django.test import Client
>>> csrf_client = Client(enforce_csrf_checks=True)
```

태형 - 스프린트 간 막혔던 부분

```
if request_csrf_token == "":  
    # Fall back to X-CSRFToken, to make things easier for AJAX,  
    # and possible for PUT/DELETE.  
    request_csrf_token = request.META.get(settings.CSRF_HEADER_NAME, '')
```

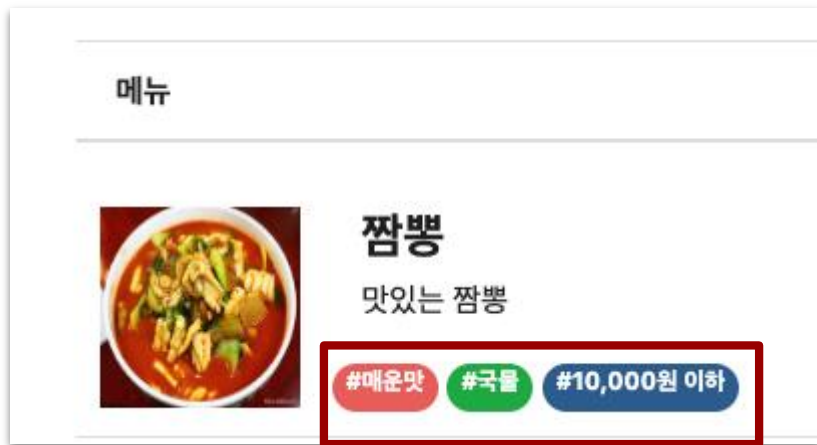
CSRF_HEADER_NAME

Default: 'HTTP_X_CSRFTOKEN'

As with other HTTP headers in **request.META**, the header name received from the server is normalized by converting all characters to uppercase, replacing any hyphens with underscores, and adding an **'HTTP_'** prefix to the name. For example, if your client sends a **'X-XSRF-TOKEN'** header, the setting should be **'HTTP_X_XSRF_TOKEN'**.

태형 - 개선해야할 부분

- 이미 작성한 API의 **ORM 최적화**
- 주문표 내, **메뉴별 취향 태그 출력**(현재는 하드코딩)



태형 - 3차 스프린트 계획

- 미니 요기요
 - 주문 내역 페이지
 - 주문 내역 디테일 페이지
 - 주문 내역 디테일 페이지 내 재주문 기능
- 랜덤 메뉴 Pick
 - 메뉴 모델 수정(조회수, 좋아요, 주문수, 점수)
 - 랜덤 메뉴 Pick 화면 구현
 - 점수에 임의의 값 저장 후, 랜덤 메뉴 추천 기능 구현

소라



-요기요 Gift Coupon(가칭: 쿠폰죤yo)

- 소개

- 요기요 사이트에서 기프티콘 및 상품권처럼(ex.스타벅스 기프티카드) 구매하고 **선물**할 수 있는 쿠폰
- 사용기간이 남아 있는 경우 미니요기요 회원끼리 **양도**가능.

- 사용방법

- 10000원단위로 금액을 지정하여 '요기요 gift coupon' 선물
- 입력한 받는이와 받는이의 email로 쿠폰 발송.
- 선물 받은 사람은 회원가입 또는 로그인 후에 쿠폰번호를 등록하면 해당 쿠폰등록.
- 선물받은 쿠폰은 다른 회원에게 양도 가능.
- 주문시 쿠폰 적용 가능 *해당 금액 이상 구매시 이용 가능

소라

- 2차 스프린트 구현 내용 - 요기요 gift 쿠폰기능 구현 시작
 - 요기요 gift 쿠폰
 - 쿠폰 관련 모델 생성
 - 요기요 gift 쿠폰 선물하기(이메일 전송)
 - 해당 쿠폰의 **쿠폰 코드** 발송
 - 요기요 gift 쿠폰 등록
 - 받은 요기요 gift 쿠폰 보기(받은 쿠폰)
 - 사용 가능한 쿠폰의 경우 양도 가능
 - 사용 완료시 및 양도시 표기
- 개선할 내용
 - mocking을 통한 좀 더 튼튼한 testcase 작성 ex) gift coupon 이메일 전송

소라

- 3차 스프린트 계획
 - 요기요 gift 쿠폰
 - 보낸 요기요 gift 쿠폰 보기(보낸 쿠폰)
 - 선물한 쿠폰(구매한 쿠폰)& 양도받은 쿠폰 보기
 - 요기요 gift coupon 양도하기
 - 받은 gift coupon이 유효기간이 남아 있는 경우 양도 가능
 - 양도할 사람의 **아이디**를 입력하여 양도 가능
 - 양도시 양도 받은 사람에게 표시

명훈

- 2차 스프린트 구현 내용
 - 기상청 API 요청 후 응답 값 가공해서 날씨 데이터 받아오기
 - Order 테이블로부터 사용자의 주소, 주소의 날씨와 일치하는 주문(메뉴명, 가격 등) 받아 오는 ORM 작성

명함

- Grid 테이블에 동네 대한 x, y 좌표 데이터를 넣음
 - 사용자 주소(동)의 x, y 좌표를 기상청 **API**에 요청 후 날씨 데이터를 받아와야 하기 때문

행정동 이름 ex 서초2동:

서초2동

X 좌표:

61

Y 좌표:

125

명훈

- 기상청 API 사용

API 요청 시 나오는 데이터 중에 category 값이 SKY인 객체의 fcstValue(맑음, 구름 많음 등)를 가져옴

```
{
  "response": {
    "header": {
      "resultCode": "0000",
      "resultMsg": "OK"
    },
    "body": {
      "items": {
        "item": [
          {
            "baseDate": "20190516",
            "baseTime": "1700",
            "category": "POP",
            "fcstDate": "20190516",
            "fcstTime": "2100",
            "fcstValue": 0,
            "nx": 97,
            "ny": 76
          },
          {
            "baseDate": "20190516",
            "baseTime": "1700",
            "category": "PTY",
            "fcstDate": "20190516",
            "fcstTime": "2100",
            "fcstValue": 0,
            "nx": 97,
            "ny": 76
          },
          {
            "baseDate": "20190516",
            "baseTime": "1700",
            "category": "REH",
            "fcstDate": "20190516",
            "fcstTime": "2100",
            "fcstValue": 70,
            "nx": 97,
            "ny": 76
          },
          {
            "baseDate": "20190516",
            "baseTime": "1700",
            "category": "SKY",
            "fcstDate": "20190516",
            "fcstTime": "2100",
            "fcstValue": 1,
            "nx": 97,
            "ny": 76
          },
          {
            "baseDate": "20190516",
            "baseTime": "1700",
            "category": "T3H",
            "fcstDate": "20190516",
            "fcstTime": "2100",
            "fcstValue": 21,
            "nx": 97,
            "ny": 76
          },
          {
            "baseDate": "20190516",
            "baseTime": "1700",
            "category": "UUU",
            "fcstDate": "20190516",
            "fcstTime": "2100",
            "fcstValue": -0.6,
            "nx": 97,
            "ny": 76
          }
        ]
      },
      "numOfRows": 6,
      "pageNo": 1,
      "totalCount": 184
    }
  }
}
```


명훈

- x, y 좌표 가져오기

정규 표현식으로 사용자의

주소에서 동을 추출하고,

Grid 테이블에서 동을

검색해서 x, y 좌표를 추출

```
def get_dong(user_address):
    try:
        dong_pattern = re.compile(r'^[가-힣]+[1-9]*동$')
        addrs = user_address.split(' ')
        for addr in addrs:
            if re.fullmatch(dong_pattern, addr):
                return addr
    except ValueError:
        return JsonResponse(
            {
                "message": "주소에 동이 존재하지 않습니다",
            },
            status=HTTPStatus.BAD_REQUEST,
        )

def get_x_y_grid(user_dong):
    try:
        x_y_grid = get_object_or_404(Grid, name=user_dong)
        return x_y_grid
    except:
        return JsonResponse(
            {
                "message": "동에 대한 x, y 좌표 정보가 없습니다.",
            },
            status=HTTPStatus.BAD_REQUEST,
        )
```

명훈

기상청 API 요청 및 응답

```
def get_user_dong_weather(nx, ny):  
    service_key = get_env_var('WEATHER_API_SERVICE_KEY')  
    now = datetime.now()  
    now_date = now.strftime('%Y%m%d')  
    now_hour = int(now.strftime('%H'))  
  
    if 0 <= now_hour < 6:  
        base_date = str(int(now_date) - 1)  
    else:  
        base_date = now_date  
    base_hour = get_base_time(now_hour)  
  
    num_of_rows = '6'  
    base_date = base_date  
    base_time = base_hour  
    nx = str(nx)  
    ny = str(ny)  
    num_of_rows = num_of_rows  
    _type = 'json'  
    api_url = 'http://newsky2.kma.go.kr/service/S  
        &base_date={}&base_time={}&nx={}&ny={}&_type=  
        service_key, base_date, base_time, nx, ny,  
  
    data = urlopen(api_url).read().decode('utf8')  
    json_data = json.loads(data)  
    sky = get_sky_info(json_data)  
    return sky
```

```
def get_base_time(hour):  
    hour = int(hour)  
    if 6 <= hour < 9:  
        temp_hour = '02'  
    elif 9 <= hour < 12:  
        temp_hour = '05'  
    elif 12 <= hour < 15:  
        temp_hour = '08'  
    elif 15 <= hour < 18:  
        temp_hour = '11'  
    elif 18 <= hour < 21:  
        temp_hour = '14'  
    elif 21 <= hour < 24:  
        temp_hour = '17'  
    elif 0 <= hour < 3:  
        temp_hour = '20'  
    else:  
        temp_hour = '23'  
    return temp_hour + '00'
```

기상청 기준 시
간으로 변환

```
def get_sky_info(data):  
    try:  
        weather_info = data['response']['body']['items']['item']  
        if weather_info[3]['category'] == 'SKY':  
            return weather_info[3]['fcstValue']  
        elif weather_info[5]['category'] == 'SKY':  
            return weather_info[5]['fcstValue']  
    except KeyError:  
        return JsonResponse(  
            {  
                "message": "기상청 서버로부터 날씨 정보를 가져오는 중 문제가 발생하여 날씨 정보를 받아  
            },  
            status=HTTPStatus.INTERNAL_SERVER_ERROR,  
        )
```

응답 데이터에서
날씨 정보 추출

- 사용자 주소와 현재 날씨에 맞는 주문 가져오기
 - 테이블 join, 메뉴 이름으로 그룹화, 각 메뉴 총수량 구하기, 수량 내림차순 정렬, 주문수 상위 5

```
menu_list = (Cart.objects
    .prefetch_related('order')
    .filter(order__address__contains=user_dong, order__weather=user_dong_weather)
    .values('cartitem__menu__name')
    .annotate(menu=F('cartitem__menu__name'), quantity=Sum('cartitem__quantity'))
    .order_by('-quantity')
    .values('menu', 'quantity',
            price=F('cartitem__menu__price'), img=F('cartitem__menu__img'),
            id=F('cartitem__menu__id'), detail=F('cartitem__menu__detail'),
            restaurant=F(
                'cartitem__menu__restaurant__id'),
            )
   )[:5]
```

```
{'menu': '갈비천왕', 'quantity': 24, 'price': 20000, 'img': 'menu/2019/05/01/갈비천왕.jpg', 'id': 2, 'detail': '갈비양념이 들  
어간 치킨', 'restaurant': 1}
```

```
{'menu': '볼케이노', 'quantity': 7, 'price': 22000, 'img': 'menu/2019/05/05/볼케이노.png', 'id': 4, 'detail': '매콤한 치킨',  
'restaurant': 1}
```

개수 2

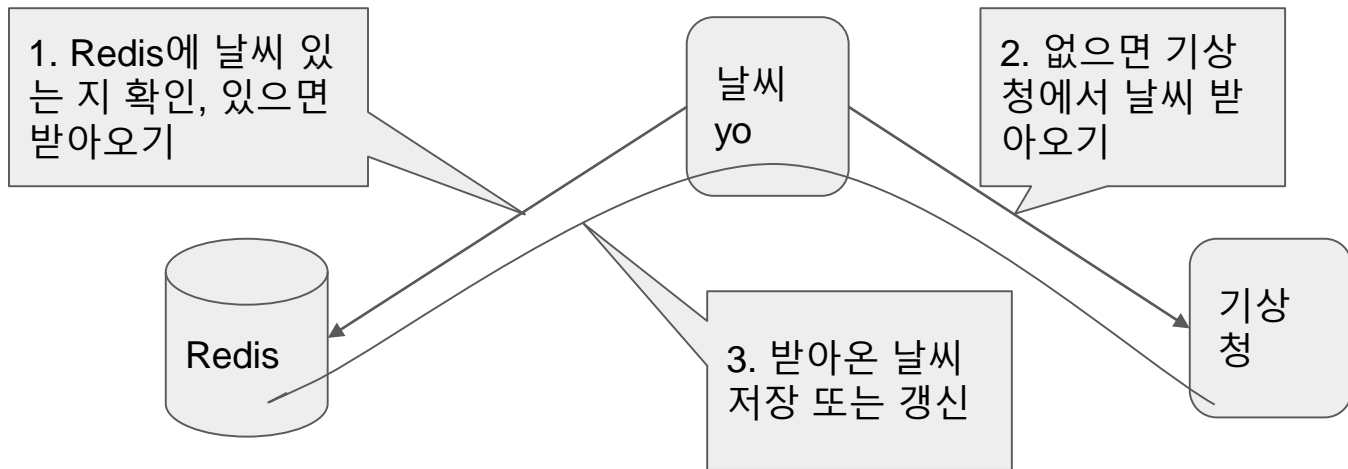
- 실행되는 Query : 테이블 join 및 menu grouping, 수량 집계

```
SELECT "menu_menu"."name" AS "menu",
       SUM("cart_cartitem"."quantity") AS "quantity",
       "menu_menu"."price" AS "price",
       "menu_menu"."img" AS "img",
       "cart_cartitem"."menu_id" AS "id",
       "menu_menu"."detail" AS "detail",
       "menu_menu"."restaurant_id" AS "restaurant"
FROM   "cart_cart"
INNER JOIN "order_order"
    ON ("cart_cart"."id" = "order_order"."cart_id")
LEFT OUTER JOIN "cart_cartitem"
    ON ("cart_cart"."id" = "cart_cartitem"."cart_id")
LEFT OUTER JOIN "menu_menu"
    ON ("cart_cartitem"."menu_id" = "menu_menu"."id")
WHERE  ("order_order"."address"::text LIKE '%서 초2동%' AND "order_order"."weather" = 3)
GROUP BY "menu_menu"."name",
         "menu_menu"."price",
         "menu_menu"."img",
         "cart_cartitem"."menu_id",
         "menu_menu"."detail",
         "menu_menu"."restaurant_id"
ORDER BY "quantity" DESC
LIMIT 5
```

명훈

- 개선할 내용

- 날씨 데이터가 필요할 때마다 기상청 API에 요청하는 문제 해결
 - 주소와 현재 시간에 맞는 날씨를 Redis에 저장
- 날씨 데이터 필요 시 Redis에서 주소, 시간을 검색하고 일치하는 게
 - 있으면 날씨 데이터 추출
 - 없으면 기상청에서 데이터 받아오기



명훈

- 다음 스프린트 계획
 - 5월 4, 5주차 :
 - Redis를 써서 효율적으로 날씨 데이터 저장 및 불러오기