



국민대학교
소프트웨어융합대학
소프트웨어학부

캡스톤 디자인 I

종합설계 프로젝트

프로젝트 명	옷때? (OTTE?)
팀 명	옷마이갓
문서 제목	2차 중간보고서

Version	1.3
Date	2020-05-20

팀원	황 효빈 (조장)
	송 현화
	정 예빈
	권 민수
	이 재호
	주 가구
지도교수	이 경용 교수님




CONFIDENTIALITY/SECURITY WARNING

이 문서에 포함되어 있는 정보는 국민대학교 소프트웨어융합대학 소프트웨어학부 및 소프트웨어학부 개설 교과목 캡스톤 디자인I 수강 학생 중 프로젝트 "옷때? (OTTE?)"를 수행하는 팀 "옷마이갓"의 팀원들의 자산입니다. 국민대학교 소프트웨어학부 및 팀 "옷마이갓"의 팀원들의 서면 허락없이 사용되거나, 재가공 될 수 없습니다.

문서 정보 / 수정 내역

Filename	21조2차중간보고서-옷때(OTTE).doc
원안작성자	황효빈, 송현화, 정예빈, 권민수, 이재호, 주가구
수정작업자	황효빈, 송현화, 정예빈, 권민수, 이재호, 주가구

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2020-05-18	정예빈	1.0	최초 작성	프로젝트 목표 작성
2020-05-18	권민수	1.1	최초 작성	수행 내용 및 중간 결과, 향후 추진계획 작성
2020-05-19	황효빈	1.2	최초 작성	수정된 연구내용 및 추진 방향, 고충 및 건의사항 작성
2020-05-20	황효빈	1.3	최종 수정	문서 전체 수정

	국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서	
		프로젝트 명	옷때? (OTTE?)
		팀 명	옷마이갓
		Confidential Restricted	Version 1.3 2020-MAY-20

목 차

1	프로젝트 목표.....	4
2	수행 내용 및 중간결과	5
2.1	계획서 상의 연구내용	5
2.1.1	클라이언트 – API 서버와 통신	5
2.1.2	클라이언트 – 반응형 웹페이지	5
2.1.3	서버 – 오늘의 날씨 엔드포인트 구현	5
2.1.4	서버 – 날씨정보 저장 스크립트	6
2.1.5	해외 날씨를 이용한 서비스	6
2.2	수행내용	7
2.2.1	클라이언트 – API 서버와 통신	7
2.2.2	클라이언트 – 반응형 웹페이지	10
2.2.3	서버 – 오늘의 날씨 엔드포인트 구현	12
2.2.4	서버 – 날씨정보 저장 스크립트	15
2.2.5	해외 날씨를 이용한 서비스	18
3	수정된 연구내용 및 추진 방향	21
3.1	수정사항	21
4	향후 추진계획	22
4.1	향후 계획의 세부 내용	22
4.1.1	예외처리	22
4.1.2	샘플 데이터 넣기	23
4.1.3	AWS 배포	23
5	고충 및 건의사항	24




1 프로젝트 목표

사람마다 더위, 추위를 타는 정도와 각자 소유하고 있는 옷 등이 모두 다르다. 그러므로 본 프로젝트에서는 사용자 개인 맞춤에 중점을 두어, "오늘 뭐 입지?"라는 고민을 덜어주는 웹 서비스를 만드는 것을 목표로 한다. 대략적인 기능의 흐름은 다음과 같다.

1. 사용자가 소유한 옷들은 미리 사진을 찍어 이미지 형태로 저장해 둔다. 이 때 옷의 대분류(아우터, 상의, 바지, 치마, 원피스)와 30여가지의 소분류(롱패딩, 니트 등) 카테고리가 자동으로 분석되며 옷의 별칭을 같이 등록할 수 있다.
2. 사용자는 실제로 자신이 입었던 코드를 기록할 수 있다. 심플, 스트리트 등 5가지의 스타일을 선택하여 저장할 수 있으며 코드의 별칭도 같이 등록할 수 있다.
3. 이후 사용자는 자신이 입었던 코드에 대한 리뷰를 남길 수 있다. 리뷰에는 해당 코드를 입었던 날의 체감온도, 기온, 풍속 등의 날씨 정보와 함께 코멘트가 저장된다. 또한 해당 날씨에 대한 코드의 적절성을 5단계로 선택하여 기록할 수 있다.
4. 어떤 옷을 입을지 고민이 될 때 현재 체감온도와 유사한 날에 입었던 자신의 코드와 리뷰를 통해 스스로 날씨에 적절한 옷을 매치하여 입을 수 있다.
5. 여러 사용자들이 현재 체감온도와 유사한 날에 입었던 코드의 리뷰 중 날씨에 대한 코드 적절성이 높은 리뷰를 수집한다. 이후 대분류 기준 가장 높은 비율을 차지하는 소분류 카테고리를 텍스트 형태로 추천하며 일치하는 사용자의 옷을 이미지 형태로 보여준다.
6. 해외 날씨 서비스를 통해 해외 여행 및 출장 시에도 이용할 수 있다.

옷때?(OTTE?)의 기대효과는 다음과 같다. 사용자는 날씨에 알맞은 코드를 할 수 있고, 어떤 옷을 입어야 할지 고민하는 시간을 단축시킬 수 있다. 또한 사용자가 소유한 옷들을 쉽게 파악할 수 있어, 자신의 옷을 효율적으로 관리할 수 있다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2 수행 내용 및 중간결과

2.1 계획서 상의 연구내용

2.1.1 클라이언트 – API 서버와 통신

클라이언트에서는 사용자 관련 데이터를 보여주기 위해 API 서버와 통신할 수 있어야 한다. 따라서 Vue.js에서 공식적으로 추천하는 axios 라이브러리를 사용한다. axios 라이브러리를 통해 필요한 요청을 보내며, 받은 응답을 이용하여 콘텐츠를 보여준다.


2.1.2 클라이언트 – 반응형 웹페이지

사용자가 데스크탑, 태블릿, 스마트폰 등 어떠한 장치로 페이지에 접근해도 불편함을 느끼지 않도록 웹 페이지를 반응형으로 만든다. 손쉬운 작업을 위해 Vue-Bootstrap 라이브러리를 이용한다.

2.1.3 서버 – 오늘의 날씨 엔드포인트 구현

메인 페이지에서 사용자의 위치에 맞는 오늘의 날씨를 보여줘야 한다. 이를 위해 서버에서는 해당 지역에 따른 오늘의 날씨 정보를 반환하는 엔드포인트를 구현해야 한다.

1. 클라이언트로부터 지역코드 정보를 받는다.
2. 해당 지역코드에 맞는 x좌표와 y좌표를 도출한다.
3. 기상청 API를 이용해 현재 ~ 6시간 후의 날씨 정보를 받아온다.
4. 받아온 데이터를 이용하여 최저/최고 온도, 최저/최고 체감온도, 평균 풍속, 평균 강수량 등을 도출한다.
5. 응답을 반환한다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2.1.4 서버 – 날씨정보 저장 스크립트


기상청 API가 제공하는 날씨 정보는 24시간 이내의 데이터만 조회 가능하기 때문에 사용자가 코디 리뷰를 등록할 때, 24시간 이내의 날짜로만 등록할 수 있도록 제한해야 한다. 따라서 일주일 이내 날짜의 코디 리뷰를 등록할 수 있도록 하기 위해 3시간마다 날씨 정보를 기상청 API를 통해 받아와 DB의 테이블에 저장하도록 한다. 그리고 일주일 이 지난 날씨 데이터는 삭제한다.

1. 지역코드, 날짜가 주어졌을 때 날씨 정보를 기상청 API로부터 받아와 DB에 저장하는 스크립트를 작성한다.
2. 작성한 스크립트를 매일 정해진 시간마다 하루에 총 8번 실행하도록 한다.
3. 스크립트가 실행될 때, 일주일 이 지난 날씨 데이터는 DB에서 삭제한다.

2.1.5 해외 날씨를 이용한 서비스

1차 중간평가의 피드백을 반영하여, 사용자가 해외 날씨를 기준으로 코디를 짜고자 할 때 도움이 될 수 있는 기능을 개발한다. 해외 도시들의 날씨 데이터를 받아올 수 있는 API를 이용하여 데이터를 받아오고, 이를 이용하여 사용자에게 서비스를 제공한다.

1. 클라이언트에서 도시 이름을 찾고자 할 때, 검색 요청에 대한 응답을 보낼 수 있는 엔드포인트를 개발한다.
2. 클라이언트에서 요청한 도시 이름과 날짜에 맞게 응답을 반환한다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2.2 수행내용

2.2.1 클라이언트 – API 서버와 통신

클라이언트 측 코드에서 API 서버의 엔드포인트를 이용하기 위해 axios 라이브러리를 이용한다. axios는 자바스크립트 Promise 객체를 이용한, HTTP 클라이언트 역할을 하는 라이브러리이다.

[참고자료] axios github 페이지 (<https://github.com/axios/axios>)

'옷 등록하기' 페이지를 예로 들면, [그림 2-1]에서 이미지를 등록하였을 때 서버 측의 옷 카테고리 추론 엔드포인트(/clothes/inference)로 요청을 보내야한다.



[그림 2-1] 옷 등록하기 페이지

따라서 [그림 2-2]와 같은 코드를 작성하였다.

```

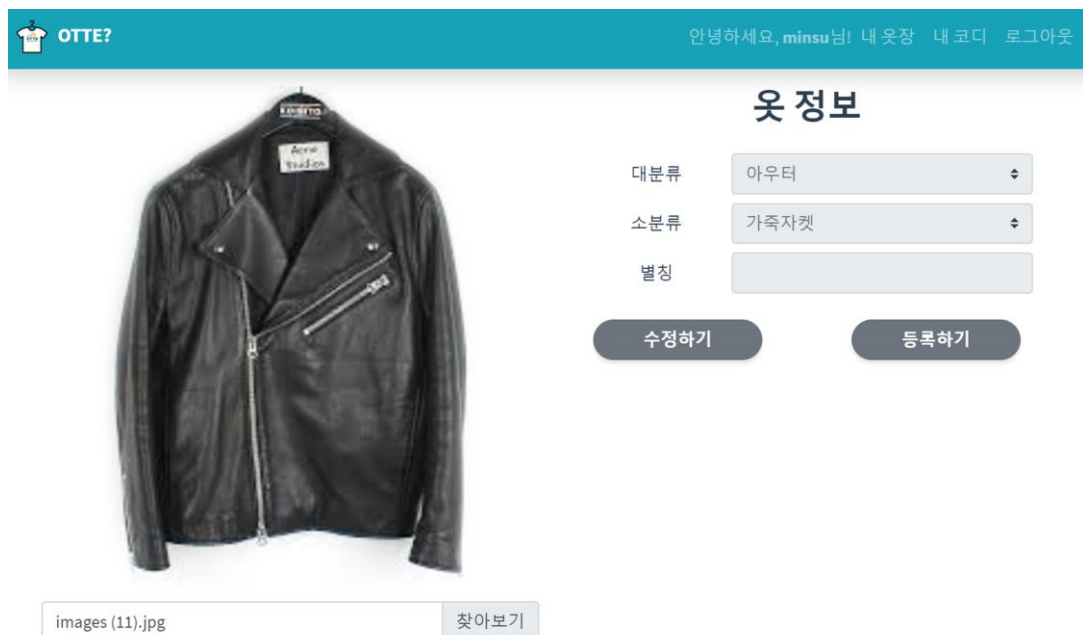
handleImageUpdate: function (event) {
  this.isLoading = true
  var imageStr = event.split(',')[1]
  var token = window.localStorage.getItem('token')
  var config = {
    headers: { Authorization: `Bearer ${token}` }
  }
  axios.post(`${const.SERVER_BASE_URL}/clothes/inference/`, { image: imageStr }, config)
    .then(response => {
      this.image = response.data.image_url
      this.analysis_props.upper = response.data.upper_category
      this.analysis_props.lower = response.data.lower_category
      this.isLoading = false
    }).catch((ex) => {
      this.alertMessage = '옷 분석에 실패했습니다. 오류가 계속 될 경우, 관리자에게 연락해주세요.'
      this.showAlert = true
      this.isLoading = false
    })
}

```


[그림 2-2] 옷 분석 요청 코드

axios의 post 메소드(axios.post(url, data, config))를 이용해서 지정한 url으로 요청을 보낸다. 이 메소드의 실행 결과는 Promise 객체를 반환하며, 이를 이용해 then으로 요청이 성공했을 때의 행동을, catch로 요청이 실패했을 때의 행동을 지정할 수 있다.

요청이 성공한 경우, 페이지가 [그림 2-3]과 같이 보이게 된다. 배경이 제거된 사진은 왼쪽에, 받아온 옷 카테고리 분석 결과는 오른쪽의 옷 정보 폼에 보여진다.



[그림 2-3] 옷 분석 성공 화면


	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

요청이 실패한 경우에는 페이지가 [그림 2-4]와 같이 보이게 된다. 원래의 페이지 모습은 그대로 유지되고, 사용자에게 요청이 실패했다는 메시지를 보여준다..



[그림 2-4] 옷 분석 실패 화면

'옷 등록하기' 페이지와 마찬가지로 다른 페이지들도 axios 라이브러리를 이용하여 성공적으로 API 서버와의 통신 기능을 구현하였다.

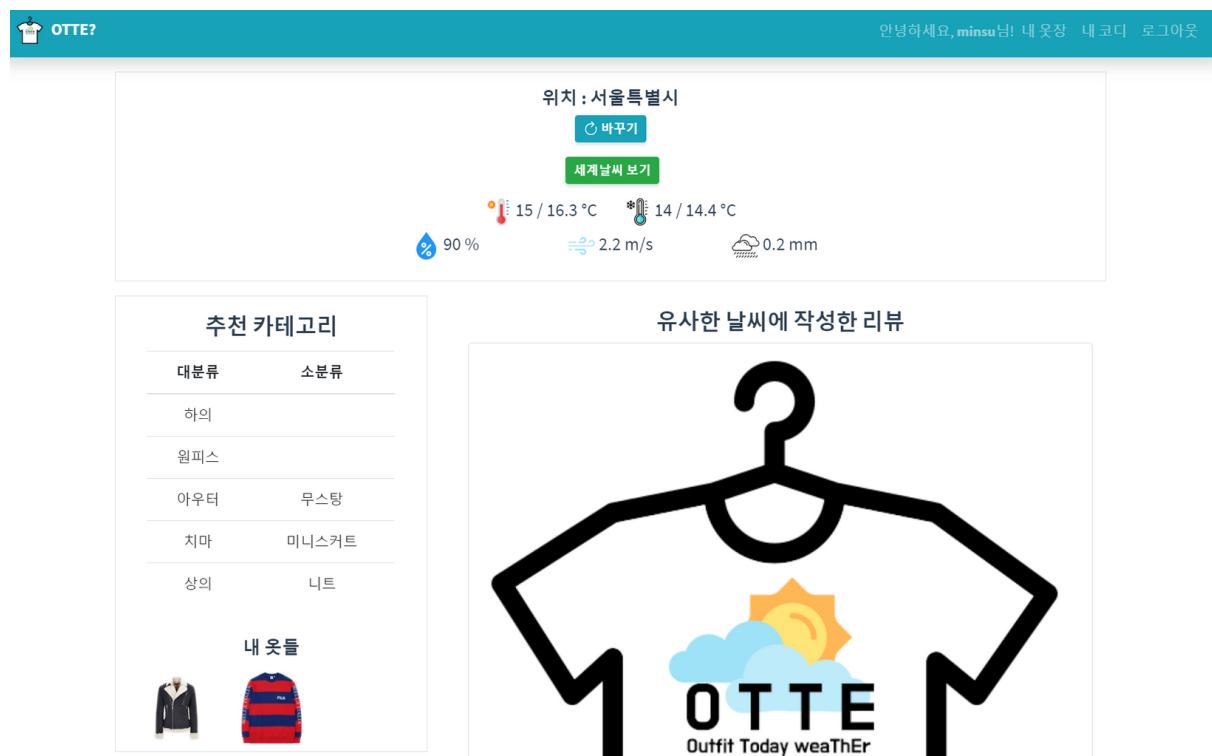
 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2.2.2 클라이언트 – 반응형 웹페이지

반응형 웹 페이지는 Bootstrap 라이브러리를 이용하여 구현하였다. Bootstrap 라이브러리에서 제공하는 Vue 컴포넌트를 사용함으로써, 손쉽게 브라우저의 크기 변화 또는 기기 변화에 대응할 수 있었다. Bootstrap은 화면 크기를 다섯가지 브레이크포인트(xs, sm, md, lg, xl)로 나누어 개발자가 각각의 크기에 맞게 원하는 페이지 구성을 할 수 있게 해주며, 이를 활용하여 페이지를 구성하였다.

[참고자료] Vue-Bootstrap 홈페이지 (<https://bootstrap-vue.org/>)

'메인' 페이지로 예를 들면, 일반적인 데스크탑의 브라우저에서는 [그림 2-5]와 같은 페이지 구성을 보인다.



[그림 2-5] 메인 페이지

해당 페이지는 [그림 2-6]과 같은 코드로 구성된다.

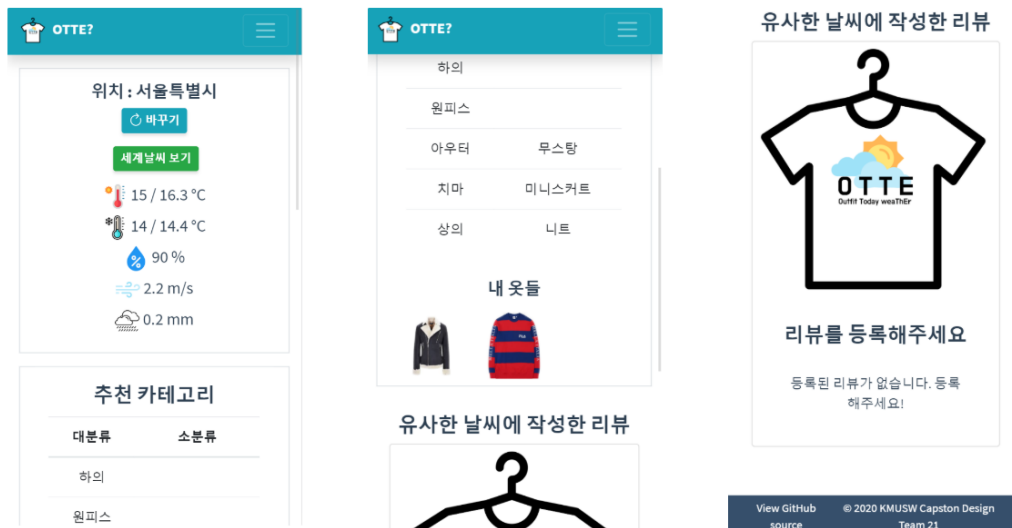
```

<b-container>
  <b-row cols=1>
    <b-col cols=12>
      <!-- 날씨 컴포넌트 -->
      <WeatherComponent class="border mb-3 p-3" :weatherData.sync="weatherProps" />
    </b-col>
  </b-row>
  <b-row cols=1 cols-md=2>
    <b-col cols=12 lg=4>
      <!-- 추천 카테고리 컴포넌트 -->
      <RecommendedCategoriesComponent class="border" :categories="recommendedCategories" />
    </b-col>
    <b-col class="mt-3 mt-lg-0" cols=12 lg=8>
      <!-- 리뷰 컴포넌트 -->
      <h4 class="mt-3 pb-0">유사한 날씨에 작성한 리뷰</h4>
      <ReviewListComponent :reviews="userReviews" />
    </b-col>
  </b-row>
</b-container>

```

[그림 2-6] 메인 페이지 코드

크게 3파트로 나누어진 부분들에 각각의 브레이크 포인트에서 차지하기를 원하는 열의 수를 지정해주었다. 따라서 모바일 환경에서(즉, 가장 작은 브레이크포인트에서) 메인 페이지는 [그림 2-7]과 같은 구성을 갖게 된다.



[그림 2-7] 메인 페이지(모바일)

같이 행에 배치되었던 부분들이 화면에 맞게 각각 다른 행에 배치된 모습을 볼 수 있다. '메인' 페이지와 마찬가지로, 다른 페이지들도 Bootstrap 라이브러리의 Vue 컴포넌트들을 이용하여 성공적으로 반응형으로 구현하였다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2.2.3 서버 – 오늘의 날씨 엔드포인트 구현

1. 클라이언트로부터 지역코드 정보를 받는다.

```
@action(detail=False, methods=['get'])
def current_weather(self, request, *args, **kwargs):
    """
    An endpoint that returns weather data for
    location based on query parameter and current time
    """
    # Get Location.
    location = request.query_params.get('location')
```

[그림 2-8] 지역코드 받아오기

GET 메소드를 사용하며, 따라서 지역 코드는 쿼리스트링 형태로 전달받는다.

2. 해당 지역코드에 맞는 x좌표와 y좌표를 도출한다.

```
with open('apps/api/locations/data.json') as json_file:
    json_data = json.load(json_file)

x = (json_data[str(location)]['x'])
y = (json_data[str(location)]['y'])
```

[그림 2-9] x, y 좌표 도출


지역 코드별로 x좌표와 y좌표가 등록되어 있는 JSON파일을 이용해 도출한다.

3. 기상청 API를 이용해 현재 ~ 6시간 후의 날씨를 받아온다.

```
api_url = url + key + numOfRows + typeOfData + date + time + nx + ny
data = urllib.request.urlopen(api_url).read().decode('utf8')
data_json = json.loads(data)
parsed_json = data_json['response']['body']['items']['item']
```

[그림 2-10] 기상청 API에서 응답 받아오는 코드

환경 변수로 등록되어 있는 API키와 함께 요청 정보를 지정한 기상청 API의 URL로 보내고, 받아온 정보를 Python 딕셔너리 형태로 저장한다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

4. 받아온 데이터를 이용하여 최저/최고 온도, 최저/최고 체감온도, 평균 풍속, 강수량 등을 도출한다.

```
# 체감 온도 계산
wci = getWCI(t, v)
wci = round(wci, 2)

# api 초단기예보 중 최저 최고온도 구하기
wci_max = getWCI(float(max), float(max_WSD))
wci_max = round(wci_max, 2)
wci_min = getWCI(float(min), float(min_WSD))
wci_min = round(wci_min, 2)

passing_data['MAX'] = max
passing_data['MIN'] = min
passing_data['WCI'] = wci # current wind chill temp
passing_data['WCIMIN'] = wci_min
passing_data['WCIMAX'] = wci_max
```

[그림 2-11] 체감온도 계산

받아온 풍속, 강수량의 평균값을 구하고, 체감온도를 계산한다.

5. 응답을 반환한다.

```
# Return response
return Response({
    'temperature': temperature,
    'min_temperature': min_temp,
    'max_temperature': max_temp,
    'chill_temp': sense,
    'min_chill_temp': min_sense,
    'max_chill_temp': max_sense,
    'humidity': humidity,
    'wind_speed': wind_speed,
    'precipitation': precipitation,
}, status=status.HTTP_200_OK)
```

[그림 2-12] 응답 반환

클라이언트에게 JSON 형태로 응답을 반환한다.

 <div> <p>국민대학교</p> <p>소프트웨어학부</p> <p>캡스톤 디자인 I</p> </div>	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

이렇게 서버 측에 구현된 오늘의 날씨 엔드포인트는 메인페이지에서 사용된다.



[그림 2-13] 메인 페이지 날씨 컴포넌트

사용자는 [그림 2-14]와 같이 검색을 통해 원하는 지역을 지정할 수 있다.




[그림 2-14] 지역 검색 화면

지역을 바꾸게 되면, '오늘의 날씨' 엔드포인트로 새로운 요청을 보내고 응답을 받아와, [그림 2-15]와 같이 페이지가 변하게 된다.



[그림 2-15] 지역이 바뀐 날씨 컴포넌트

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2.2.4 서버 – 날씨정보 저장 스크립트

1. 지역 코드, 날씨가 주어졌을 때 날씨 정보를 기상청 API로부터 받아와 DB에 저장하는 스크립트를 작성한다.

Django Python 스크립트 형식으로 스크립트를 작성하였으며, 국내에 존재하는 모든 지역 코드(총 3780개)에 대하여 x좌표와 y좌표가 겹치지 않는 지역들만 기상청 API로 데이터 요청을 보냈다.

[그림 2-16]과 같이 작성된 코드로 받아온 데이터는 Django Model API를 통해서 DB에 들어가게 된다.

```
for location in range(3780):
    new_x = int((json_data[str(location)][ 'x' ]))
    new_y = int((json_data[str(location)][ 'y' ]))

    weather_filtering = Weather.objects.filter(date=now[0:10], time=conv_time[0:2], x=new_x, y=new_y)
    if weather_filtering.exists():
        # TODO(hyobin) : print문 지우기
        print("old x: ", new_x, " y : ", new_y)
        Weather.objects.create(location_code=location, date=now[0:10], time=conv_time[0:2], x=new_x, y=new_y,
                                temp=weather_filtering[0].temp, sensible_temp=weather_filtering[0].sensible_temp,
                                humidity=weather_filtering[0].humidity, wind_speed=weather_filtering[0].wind_speed,
                                precipitation=weather_filtering[0].precipitation)

        continue

    # TODO(hyobin) : print문 지우기
    print("new x: ", new_x, " y : ", new_y)

    try:
        response = get_weather_date(now, str(location))

        # WCI 체감온도 T3H 기온 WSD 풍속 REH 습도 R06 강수량
        Weather.objects.create(location_code=location, date=now[0:10], time=conv_time[0:2], x=new_x, y=new_y,
                                temp=response['T3H'], sensible_temp=response['WCI'], humidity=response['REH'],
                                wind_speed=response['WSD'], precipitation=response['R06'])

    except KeyError:
        err_location_code.append(location)
```

[그림 2-16] 날씨 저장 스크립트

2. 작성한 스크립트를 매일 정해진 시간마다 총 8번씩 실행하도록 설정한다.

Python의 schedule 패키지를 이용하여, 지정한 시간마다 날씨 정보를 받아와 DB에 저장하는 스크립트를 실행하도록 하였다. 여기서 실행시간은 기상청의 API 문서에 근거하여, 확실하게 데이터가 들어오는 시간(2, 5, 8, 11, 14, 17, 20, 23시)을 기준으로 각각 30분 뒤에 해당 시간의 날씨 데이터를 요청하도록 하였다.


```
# basetime 30분 뒤 마다 basetime에 대한 날씨정보를 저장
schedule.every().day.at("02:30").do(run)
schedule.every().day.at("05:30").do(run)
schedule.every().day.at("08:30").do(run)
schedule.every().day.at("11:30").do(run)
schedule.every().day.at("14:30").do(run)
schedule.every().day.at("17:30").do(run)
schedule.every().day.at("20:30").do(run)
schedule.every().day.at("23:30").do(run)
```

[그림 2-17] schedule 패키지를 이용한 작업 예약

[그림 2-17]의 스크립트 실행으로 시간대별로 3780개씩의 데이터가 잘 들어갔음을 [그림 2-18]에서 확인할 수 있다.

	date	time	count(id)
▶	2020-05-09	2	3780
	2020-05-09	5	3780
	2020-05-09	8	3780
	2020-05-09	11	3780
	2020-05-09	14	3780
	2020-05-09	17	3780
	2020-05-09	20	3780
	2020-05-09	23	3780

[그림 2-18] MySQL Workbench를 통한 데이터 확인

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

3. 스크립트가 실행될 때, 일주일이 지난 날씨 데이터는 DB에서 삭제한다.

[그림 2-19]의 스크립트를 통해, 일주일이 지난 데이터는 모두 삭제한다.


```
def ago():
    today = datetime.date.today()
    last_week = today - datetime.timedelta(days=7)
    data = Weather.objects.filter(date__lte=last_week)
    # TODO(hyobin) : print문 지우기
    print('delete date : ', last_week)
    data.delete()
```

[그림 2-19] 지난 정보 삭제 코드

지난 날씨 데이터의 삭제는 매일 00시에 실행된다.

```
# 현재 기준 7일 전의 날씨정보 삭제
schedule.every().day.at("00:00").do(ago)
```

[그림 2-20] schedule 패키지를 통한 삭제 작업 예약

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2.2.5 해외 날씨를 이용한 서비스

1. 클라이언트에서 해외 도시 이름을 찾고자 할 때, 검색 요청에 대한 응답을 보낼 수 있는 엔드포인트를 개발한다.

먼저 클라이언트에서 쿼리스트링 형태로 보낸 search값을 받아온다.

```
# Get query parameters.
search = request.query_params.get('search')
```

[그림 2-21] search 쿼리스트링 값

이후 해당 검색 키워드가 포함된 도시 이름들을 JSON 파일에서 추출하고, 이를 리스트 형태로 저장한다.

```
# Open JSON file for location results.
with open('apps/api/locations/cities_20000.json', 'rt', encoding='UTF-8') as json_file:
    data = json.load(json_file)

# Get results total count & initial list containing search keyword.
results = []
count = 0
for city in data:
    if search in city['city_name']:
        count += 1
        results.append({
            'id': city['city_id'],
            'location' : city['city_name']
        })
```

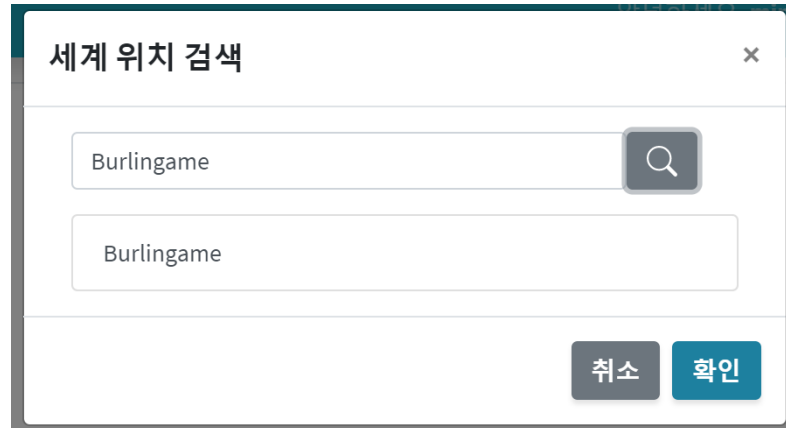
[그림 2-22] 검색 결과 도출

마지막으로 클라이언트에게 결과를 반환한다.

```
# Return response.
return Response({
    'count': count,
    'next': offset + limit_count,
    'results': final_results,
}, status=status.HTTP_200_OK)
```

[그림 2-23] 검색 결과 반환

클라이언트에서는 위의 엔드포인트를 이용하여 [그림 2-24]와 같이 검색 결과를 받아온다.



[그림 2-24] 메인페이지 세계 위치 검색

- 클라이언트에서 요청한 도시 이름과 날짜에 맞게 응답을 반환한다.

우선 클라이언트에서 쿼리스트링 형태로 전달해 준 도시 이름과, 날짜를 받아온다.


```
# Get Location.
city_name = request.query_params.get('city_name')
forecast_date = request.query_params.get('date')
```

[그림 2-25] 쿼리스트링 값 받아오기

환경 변수에 저장된 API키와 지정된 API의 URL을 통해 응답을 받아오고, 이를 Python Dictionary 형태로 가공한다. 여기서 API는 [Weatherbit.io](https://weatherbit.io)에서 제공하는 16일 예보를 사용한다.

```
api_url = url + city_id_url + key
data = urllib.request.urlopen(api_url).read().decode('utf8')
data_json = json.loads(data)
parsed_json = data_json['data']
```

[그림 2-26] Weatherbit API 이용

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

이후 필요한 데이터를 채워넣어 응답결과를 클라이언트 쪽으로 반환한다.

```
return Response({
  'temperature': temperature,
  'min_temperature': min_temp,
  'max_temperature': max_temp,
  'chill_temp': sense,
  'min_chill_temp': min_sense,
  'max_chill_temp': max_sense,
  'humidity': humidity,
  'wind_speed': wind_speed,
  'precipitation': precipitation,
  }, status=status.HTTP_200_OK)
```

[그림 2-27] 응답 반환

클라이언트는 [그림 2-28]과 같은 형태로 응답을 받아 사용자에게 보여준다.




[그림 2-28] 메인 페이지 세계날씨 컴포넌트

옷 카테고리 추천과 날씨에 적절한 코디 제안도 이 결과에 따라 변한다.



[그림 2-29] 세계날씨에 맞게 바뀐 추천 카테고리

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

3 수정된 연구내용 및 추진 방향

3.1 수정사항

1) 해외 날씨 서비스 추가 구현


기존에는 국내 날씨 API만 구현하여 날씨에 따른 코디 제안과 옷 카테고리 추천 서비스를 국내에서만 사용할 수 있다는 제한이 있었다. 하지만 1차 중간 평가의 피드백을 수용하여 해외 날씨 API를 추가적으로 구현하였다. 따라서 해외 여행이나 해외 출장을 가는 경우 해당 도시의 날씨에 적절한 옷을 입을 수 있도록 해외 날씨 서비스를 추가적으로 구현하였다. 현재 날짜 기준 16일 이후까지의 해외 날씨 예보 조회가 가능하며 해외 날씨 서비스로는 해당 도시의 날씨 정보 제공, 해당 도시의 날씨에 적절한 코디 제안 및 옷 카테고리 추천 기능이 있다.

2) 국내 날씨 API 정확성

국내 날씨 API 중 정확도가 가장 높아 채택한 기상청 API는 몇몇 시간대에 응답이 비어있는 경우가 있기 때문에 확실하게 데이터를 받아올 수 있는 시간(2시, 5시, 8시, 11시, 14시, 17시, 20시, 23시)에만 API를 호출하기로 하였다. 하지만 해당 시간에 API를 호출하여도 응답이 비어, 오류가 발생하는 경우가 있었다. 따라서 확실하게 데이터를 받아올 수 있는 시간보다 30분 뒤에 API를 호출하도록 하였다.

3) 코디 리뷰 작성 기간

국내 날씨 정보는 DB에 7일 간 저장되며 이후에는 삭제되도록 하였다. 따라서 사용자가 코디 리뷰를 남길 수 있는 날짜는 현재 기준 일주일 이내로만 가능하도록 제한하였다. 1차 중간보고서에는 코디 리뷰를 남길 때 날짜를 7일보다 더 이전으로 선택한다면 날짜를 다시 선택하라는 경고문을 띄우도록 한다고 하였다. 하지만 현재는 7일보다 더 이전의 날짜를 선택할 수 없도록 수정하여 구현하였다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

4 향후 추진계획

4.1 향후 계획의 세부 내용


4.1.1 예외처리

개발한 페이지와 엔드포인트에 대한 예외 상황을 생각해보고, 가능한 경우 예외 처리 코드를 추가한다. 예외 처리 작업은 다음 절차를 따른다.

1. Github Issue 등록

cody add page exception handling #146

 Closed mskwon1 opened this issue 4 days ago · 0 comments



 mskwon1 commented 4 days ago · edited ▾

Member 😊 ...

- ☒ change name of file
- ☐ ~~use sample image in assets~~
- ☒ add login message & loading page
- ☒ add register successful message & loading page
- ☒ error handling on clothes register
- ☒ error handling on clothes inference
- ☒ delete console.log
- ☒ limit file type to image files

[그림 4-1] Github Issue 등록

예외 처리 대상을 제목에 기재한 뒤, 내용에 어떤 부분을 구체적으로 처리할지 체크리스트 형태로 남긴다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

2. 코드 작성


새로운 브랜치를 생성하여 앞서 Issue에 작성했던 체크리스트에 맞게 작업하며 완료한 것은 체크 표시, 못하거나 하지 않을 것은 취소선 표시, 새로 추가할 것은 Issue에 업데이트 해준다.

3. Pull Request 등록 및 Merge

add cody add page exception handling #147

Merged mskwon1 merged 7 commits into `master` from `#146-cody-exec` 3 days ago

Conversation 0 Commits 7 Checks 0 Files changed 5

 mskwon1 commented 4 days ago Member

resolves #146

작업을 완료했으면 브랜치를 push 한 뒤, 작업 대상 Issue를 resolves 명령어와 함께 태그해준다. 팀원의 코드 리뷰를 받은 뒤, 브랜치를 merge하고 삭제한다.

4.1.2 샘플 데이터 넣기

옷 카테고리 추천 등의 기능은 기존 데이터가 어느정도 있어야 제대로 동작할 수 있기 때문에 가짜 데이터가 아닌 실제와 유사한 데이터를 넣어준다.

4.1.3 AWS 배포

클라이언트(SPA)는 AWS Amplify에, API 서버는 AWS EC2에 각각 배포하여 Public DNS로 접근 가능하도록 한다.

 국민대학교 소프트웨어학부 캡스톤 디자인 I	2차 중간보고서		
	프로젝트 명	옷때? (OTTE?)	
	팀 명	옷마이갓	
	Confidential Restricted	Version 1.3	2020-MAY-20

5 고충 및 건의사항

국내 날씨 API 중 가장 정확도가 높은 기상청 API를 채택하여 사용하였다. 하지만 실제로 해당 API를 사용하는 과정에서 기상청 서버 측의 오류가 간헐적으로 발생하는 고충이 있었다.