2017년도
학사학위논문

국내여행 블로그의 빅데이터를 활용한 웹 서비스 시스템 구현

Development a Web Service System Utilizing Big Data on Domestic Travel Blogs

2017년 11월 27일

순천향대학교 공과대학
컴퓨터공학과

김 문 성, 김 명 수

2017년도
학사학위논문

국내여행 블로그의 빅데이터를 활용한 웹 서비스 시스템 구현

Development a Web Service System Utilizing Big Data on Domestic Travel Blogs

2017년 11월 27일

순천향대학교 공과대학
컴퓨터공학과

김 문 성, 김 명 수

국내여행 블로그의 빅데이터를 활용한 웹 서비스 시스템 구현

Development a Web Service System Utilizing Big Data on Domestic Travel Blogs

지도교수 이 상 정

이 논문을 공학사학위 논문으로 제출함

2017년 11월 27일

순천향대학교 공과대학
컴퓨터공학과

김 문 성, 김 명 수

김 문 성

김 명 수 의 공학사학위논문을 인준함

2017년 11월 27일

심 사 위 원 이 해 각 인

심 사 위 원 남 윤 영 인

순천향대학교 공과대학

컴퓨터공학과

초 록

과거의 웹 문서만 전달하는 구조에서 클라이언트와 서버를 프론트엔드와 백엔드로 분리하고 백엔드에서는 MVC 패턴을 적용해 데이터 모델과 템플릿, 비즈니스 로직을 분리한 개발 방법을 적용하기 시작하였다. 이와 함께 프론트엔드에서도 HTML, CSS, 자바스크립트를 이용해 문서의 구조와 표현, 동작을 분리하기 시작했고, 이는 단순히 정적인 화면을 제공하는 것에서 벗어나 동적인 화면을 구성하는 현재의 웹 서비스에 이르게 되었다.

본 논문에서는 파이썬을 활용하여 국내 여행과 관련된 블로그 포스트를 수집하고, Elasticsearch를 이용해 데이터의 보관과 검색, 여행지 통계를 처리한다. 백엔드는 다양한 웹 애플리케이션 프레임워크 중 MVC 패턴을 강력하게 제공하는 Ruby on Rails 프레임워크를 사용한다. 프론트엔드는 이벤트 처리와 지도를 기반으로 한 표현을 위해 jQuery와 구글 지도 API를 사용하여 클라이언트 측 기술을 구현한다. 이를 통해 블로그 포스트의 여행 지역을 언급한 횟수를 제공하는 웹 서비스 시스템을 구현한다. 해당 웹 서비스를 통해 사용자에게 지도 기반의 표현으로 직관적인 통계 정보를 제공하고, 다양한 클라이언트 이벤트 처리와 검색 기능의 제공으로 편리한 서비스 이용을 제공한다. 또한, 기간별 통계를 통해 특정 기간에 많이 언급된 여행 지역 정보로 향후 여행 계획의 수립에 도움이 될 정보를 제공한다.

키워드 : MVC 패턴, 웹 애플리케이션 프레임워크, 데이터 파이프라인

ABSTRACT

The past structure of simply delivering web documents to user has changed, dividing client and server into frontend and backend, setting MVC pattern which separates data model and templets and business logic in the backend side. At the same time, with utilizing HTML, CSS, and JavaScript to separate the structure, expression, and behavior of document, the frontend side started to undergo a change that leaves providing a static view behind and constructs dynamic view just as the modern web service.

In this paper, blog posts about domestic traveling were collected through Python, and storing, querying, statistical processing of the blog post data including traveling sites information was carried out by Elasticsearch. The backend adopted Ruby on Rails framework to utilize robust MVC pattern among a variety of web application frameworks. The frontend developed client side by utilizing jQuery and Google Map API for the expression based on event processing and map. By all these efforts, this paper develops a web service system which provides the number of times blog posts refer to traveling sites. This web service provides its users with intuitive statistical information from a map-based expression, and ensures convenient service that are enabled with multiple client event processing and searching capability. Furthermore, this web service delivers information that helps users plan a future trip with traveling sites information, referred in specific period, through period by period statistical information.

Keywords : MVC pattern, Web application framework, Data pipeline

차 례

표 차 례

그 림 차 례

제 1 장 서 론

과거의 웹 문서만 전달하는 구조에서 클라이언트와 서버를 프론트엔드와 백엔드로 분리하고 백엔드에서는 MVC 패턴을 적용해 데이터 모델과 템플릿, 비즈니스 로직을 분리한 개발 방법을 적용하기 시작하였다. 이와 함께 프론트엔드에서도 HTML, CSS, 자바스크립트를 이용해 문서의 구조와 표현, 동작을 분리하기 시작했고, 이는 단순히 정적인 화면을 제공하는 것에서 벗어나 동적인 화면을 구성하는 현재의 웹 서비스에 이르게 되었다.

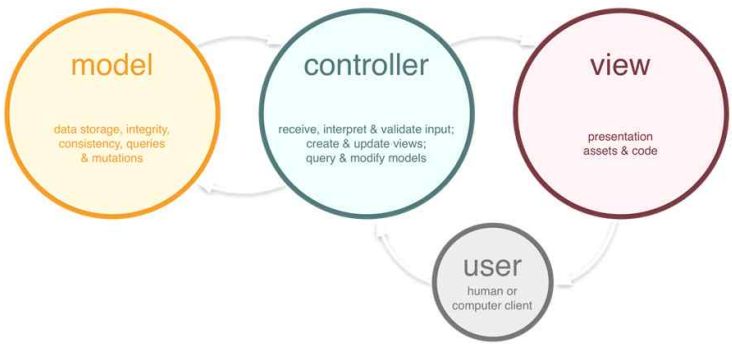
본 논문에서는 파이썬을 활용하여 국내 여행과 관련된 블로그 포스트를 수집하고, Elasticsearch를 이용해 데이터의 보관과 검색, 여행지 통계를 처리한다. 백엔드는 다양한 웹 애플리케이션 프레임워크 중 MVC 패턴을 강력하게 제공하는 Ruby on Rails 프레임워크를 사용한다. 프론트엔드는 이벤트 처리와 지도를 기반으로 한 표현을 위해 jQuery와 구글 지도 API를 사용하여 클라이언트 측 기술을 구현한다. 이를 통해 블로그 포스트의 여행 지역을 언급한 횟수를 제공하는 웹 서비스를 설계하고 구현 및 테스트를 진행한다. 본 논문의 구성은 다음과 같다. 2장에서는 이론적 배경에 대해 관련 기술과 구현 라이브러리에 관해 설명한다. 3장에서는 서비스 구조, 소프트웨어 스택, 워크플로우와 데이터 구조에 대해 설계한다. 4장에서는 3장의 설계를 바탕으로 파이썬을 기반으로 한 크롤러를 이용해 데이터를 수집하고 Elasticsearch를 이용해 처리를 구현한다. 또한, Ruby on Rails 프레임워크를 기반으로 웹 애플리케이션을 제작하며 jQuery와 구글 지도를 이용한 웹 페이지를 구현한다. 이후 구현한 크롤러와 웹 서비스에 대해 테스트를 진행한다. 마지막으로 5장에서는 결론 및 향후 연구방향을 제시한다.

제 2 장 이론적 배경

2.1 관련 기술

2.1.1 MVC(Model-View-Controller) 디자인패턴

MVC(Model-View-Controller) 디자인패턴(이하 'MVC 패턴')은 소프트웨어 공학에서 사용되는 소프트웨어 디자인 패턴이다. 애플리케이션을 세 가지 컴포넌트의 역할로 구분한 개발 방법론이며 다양한 응용프로그램에 적용되어 다양한 방법과 발전이 있었고 객체지향 분야에서 다양한 연구와 결과를 많이 응용한다. 이 패턴을 성공적으로 사용하면, 사용자 인터페이스로부터 비즈니스 로직을 분리하여 서로 영향 없이 쉽게 수정 가능한 애플리케이션의 제작이 가능하다. 입력, 출력, 처리를 간소화하여 객체 간의 연결보다 구성 요소 간의 결합력을 낮추어 개발 이후의 유지보수와 확장성에 용이하기 때문에 웹 서비스에서 대표적으로 사용되는 패턴이다.[1],[2]



[그림 1] MVC 디자인패턴

2.1.2 웹 애플리케이션 프레임워크

웹 애플리케이션 프레임워크는 다양한 프레임워크 중 동적 웹사이트, 웹 애플리케이션, 웹 서비스 개발을 지원하기 위한 목적으로 패키지들 혹은 모듈들을 모아 설계된 소프트웨어 프레임워크이다. 웹 페이지를 개발하는 과정에서 겪는 어려움을 줄이는 것을 목적으로 데이터베이스 연동, 템플릿 형태의 표준, 세션 관리, 코드 재사용 등의 기능을 포함한다.[3] 대표적인 프레임워크로 자바를 기반으로 하는 Spring과 파이썬을 기반으로 하는 Django, 자바스크립트 기반의 React.js 등이 있다. 웹 애플리케이션 프레임워크의 특성을 다음과 같이 정리 할 수 있다.

첫째, 개념을 추상화하여 여러 클래스나 컴포넌트들로 구성되어 객체를 관리하여 제공한다.

둘째, 추상적인 개념을 이용하여 개발과정에서의 문제점들을 해결 할 수 있는 방법을 정의한다.

셋째, 객체들을 관리 해주기 때문에 코드의 재사용이 높다.

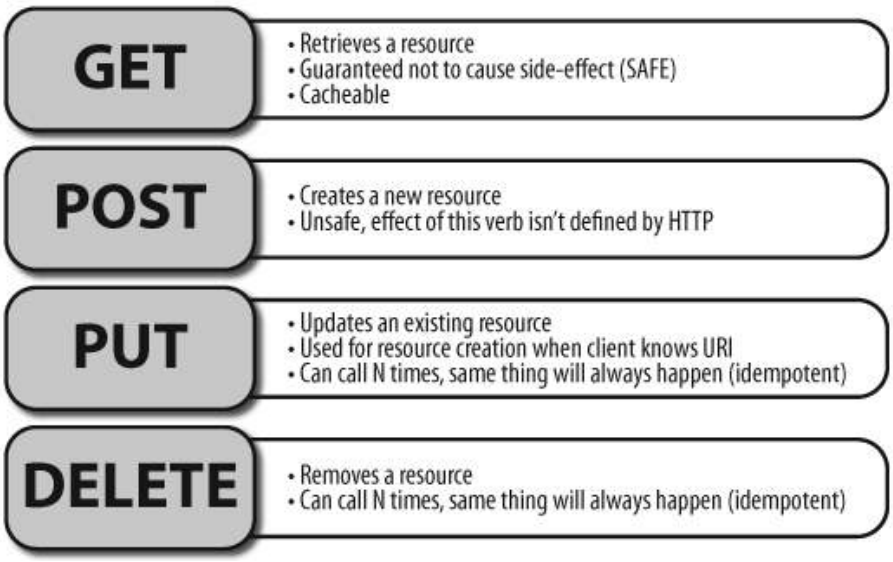
넷째, 검증된 개발 기반 환경을 제공하므로 논리적인 패턴을 조직화 한다.

2.1.3 웹 크롤링

웹 크롤링은 월드 와이드 웹을 탐색하는 컴퓨터 프로그램을 사용하여 웹 페이지를 수집하는 작업 또는 웹 사이트들에서 원하는 정보를 추출하는 것을 의미한다.[4] 그리고 해당 프로그램을 크롤러라고 한다. 크롤러는 대체로 방문한 사이트의 모든 페이지의 복사본을 생성하는 데 사용되며, 링크 검사, HTML 코드 검증과 같은 웹 사이트의 자동 유지 관리 작업을 위해 사용되기도 한다.

2.1.4 REST 아키텍처

REST 아키텍처는 REST(Representational State Transfer) 형식을 따르는 시스템을 지칭한다. REST는 월드 와이드 웹(WWW)과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 웹 아키텍처의 한 형식이다[5]. 이 용어는 Roy Thomas Fielding의 2000년 박사학위 논문에서 소개되었다. 주요한 특징으로 모든 자원에 대하여 통합 자원 식별자(URI, Uniform Resource Identifiers)를 제공하며, 웹 프로토콜인 HTTP(Hypertext Transfer Protocol)를 이용하여, 다양한 파일형식으로 데이터를 제공한다. 특히 웹에 존재하는 자원의 생성(Create), 읽기(Read), 갱신(Update) 삭제(Delete)를 HTTP 프로토콜의 기본 메서드인 GET, POST, PUT, DELETE와 대응한 URL 표현 방식으로 요구 자원에 따른 모든 서비스가 가능하다.



[그림 2] RESTful Uniform Interface

2.1.5 JSON(JavaScript Object Notation)

JSON(JavaScript Object Notation)은 자바스크립트 객체 표현식을 이용해 속성-값 쌍으로 이루어진 데이터 객체를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 경량의 데이터 교환 형식이며, 개방형 표준 포맷이다. JSON 형식의 데이터는 자료의 제한이 크지 않으며 컴퓨터 프로그램의 변수 값을 표현하는 데 적합하다. 내용이 함축적이고 최소한의 정보만을 표현하기 때문에 네트워크의 트래픽을 적게 사용할 수 있고 자료가 객체로써 표현되기 때문에 효율적으로 데이터의 구성이 가능하다. 이를 통해 XML을 대체하는 주요 데이터 포맷으로 자리를 잡았다. 특히 인터넷을 통한 자료를 요청하고 응답받는 과정에 기준이 맞춰져 있다.[6]

2.1.6 AJAX(Asynchronous JavaScript and XML)

AJAX(Asynchronous JavaScript and XML)는 기존의 웹 브라우저에서 폼을 채우고 이를 웹 서버로 제출을 하고 응답에 따라 새로운 웹 페이지를 돌려받는 방식과 달리 비동기적으로 자바스크립트에서 서버에게 요청을 보내고 웹 페이지가 아닌 데이터를 돌려받아 동적으로 웹 페이지를 갱신하는 방법이다. AJAX는 비동기적으로 동작하므로 서버의 응답을 기다리는 동안 사용자의 다양한 이벤트를 처리할 수 있고, 페이지 이동 없이 고속으로 화면을 갱신할 수 있다. 또한, 수신하는 데이터의 양을 줄이고 웹 페이지의 정보를 갱신하는 작업을 클라이언트에게 위임할 수 있는 장점이 있다.

2.2 구현 라이브러리

2.2.1 Ruby on Rails

Ruby on Rails는 스크립트형 객체지향 언어인 Ruby로 작성된 MVC(Model-View-Controller) 구조 기반의 오픈소스 풀-스택 웹 애플리케이션 프레임워크이다. Rails는 웹 애플리케이션 구현이 쉽고 코드는 적게 필요한 객체 지향의 특징을 가지고 있다. 풀-스택(Full-Stack) 프레임워크란 Rails의 모든 컴포넌트가 함께 작동하도록 하는 구조로 구현이 되어있어 처음부터 끝까지 한 언어를 사용할 수 있다는 것을 말한다. Ruby on Rails는 MVC 디자인패턴을 지원하고 템플릿을 자동으로 생성해주는 기능이 내장되어 있다. 데이터를 작성(Create), 읽기(Read), 갱신(Update), 삭제(DELETE)의 기능만을 수행하는 웹 애플리케이션은 모델에서 데이터베이스의 테이블 작성과 템플릿 생성의 단계만 끝나면 바로 개발환경 설정이 마무리되어 개발 가능하며 대부분의 웹 애플리케이션을 신속하게 작성하여 테스트할 수 있다. 이로 인해 Agile 프로세스에 초점을 맞추는 많은 스타트업 기업들이 채택하여 사용하고 있다.

[그림 3] Ruby on Rails를 이용해 개발된 App

2.2.2 Selenium

Selenium은 파이썬의 모듈 중 하나로 웹 애플리케이션의 테스트를 위한 테스트 자동화 도구이다. Selenium은 크게 세 가지의 용도로 사용할 수 있다. 첫 번째는 UI 테스트 자동화를 위해 Testcase 및 TestSuite를 관리하는 기능을 사용하여 테스트 케이스를 관리 및 실행한다. 두 번째는 실행 결과로 나온 값이 예상했던 값과 같은지 검사하는 검증 용도로 사용할 수 있다. 세 번째는 Web Driver를 이용해 웹 브라우저에 상관없이 같은 코드를 적용하며 테스트하는 용도이다. 세 가지 용도 중 Selenium은 단독으로 사용하기보다 Web Driver를 이용하여 웹 페이지를 호출하고 해당 페이지를 테스트하거나 검증하는 용도로 주로 사용한다.

2.2.3 PhantomJS

Headless 브라우저는 GUI가 없이 프로그램에서 웹 브라우저와 유사한 환경을 가졌지만 CUI(Command Line Interface)를 통해 실행하고 제어 가능한 브라우저이다. PhantomJS는 구글 크롬의 렌더링 엔진인 WebKit을 기반으로 개발된 Headless 브라우저이다. 애플리케이션 상에서 웹 페이지의 테스트, 화면 캡처, 네트워크 모니터링, 페이지 자동화 등 다양한 기능을 제공한다.

2.2.4 BeautifulSoup

BeautifulSoup는 HTML 및 XML 데이터의 구문 분석을 위한 파이썬 라이브러리이다. HTML 등의 데이터를 구분 트리로 만든 후 해당 트리에 대한 탐색,

검색 및 수정하는 방법을 제공한다. HTML 문서의 경우 이 라이브러리를 통해 HTML 태그를 탐색하고 값을 추출하는 등의 작업 수행을 가능하게 한다.

2.2.5 Elasticsearch

Elasticsearch는 오픈소스 검색 라이브러리인 아파치 루씬(Apache Lucene)을 기반으로 개발된 오픈소스 분산시스템으로써 Elasticsearch의 공식 홈페이지에서는 점점 많은 문제를 해결하는 분산형 RESTful 검색 및 분석 엔진이라고 소개한다.[1] Elastic Stack으로 제공되어 데이터의 저장과 검색을 제공하는 Elasticsearch를 중심으로 데이터 시각화와 전체 Elastic Stack을 탐색, 관리하는 Kibana, 다양한 소스로부터 데이터 수집을 제공하는 Logstash로 구성되어 있다. 다양한 검색 질의의 수행, 데이터의 집계, 분석, 빠른 속도의 데이터 접근을 제공하며, 클러스터의 쉬운 확장을 제공하고 높은 안전성과 고가용성을 보장한다.[7]

[그림 4] Elastic Stack의 구성

데이터의 접근은 표준 RESTful API와 JSON을 사용한다. 검색 질의, 데이터 인덱싱 등의 모든 작업이 RESTful API를 이용해 가능하며, 요청 질의와 응답 데이터는 HTTP의 몸체에 JSON 데이터를 포함하여 전달한다. 다양한 클라이언트 라이브러리를 제공하여 다른 응용 프로그램과의 쉬운 연결을 제공한다.

2.2.6 jQuery

jQuery는 HTML의 클라이언트 측 조작을 단순화하도록 설계된 크로스 플랫폼의 자바스크립트 라이브러리로 오늘날 가장 인기 있는 자바스크립트 라이브러리 중 하나이다.[8] jQuery 재단의 웹 사이트에서는 가벼운 용량과 CSS3의 선택자 표기법을 사용한 DOM 객체 탐색과 수정 그리고 크로스 브라우저를 지원하며 여러 브라우저에서 동작하는 사용하기 쉬운 API를 통해 HTML 문서 탐색과 조작, 이벤트 처리, 애니메이션, Ajax 등을 훨씬 더 간단하게 만들어 준다고 설명하고 있다.[9] jQuery의 적용 방법은 jQuery 라이브러리 파일을 서버에 저장하고 웹 페이지에 참조하는 것과 CDN을 통해 외부의 라이브러리 제공 서버에서 클라이언트가 참조하도록 하는 방법이 있다. jQuery의 가장 큰 특징은 CSS3 선택자 표기법을 기반으로 한 DOM 객체 탐색이며, 사용이 쉽고 기존 자바스크립트의 DOM 객체 탐색 방법보다 편리하게 DOM 객체 탐색을 가능하게 한다. 또한, jQuery는 기존 자바스크립트에서 제공하는 HTTPXmlRequest 객체를 이용한 AJAX의 구현보다 쉽고 간단한 코드로 AJAX의 구현을 할 수 있도록 제공한다.

```
$.ajax({
  type: "POST",
  url: "some.php",
  data: "name=John&location=Boston",
  success: function(msg){
    alert( "Data Saved: " + msg );
  }
});
```

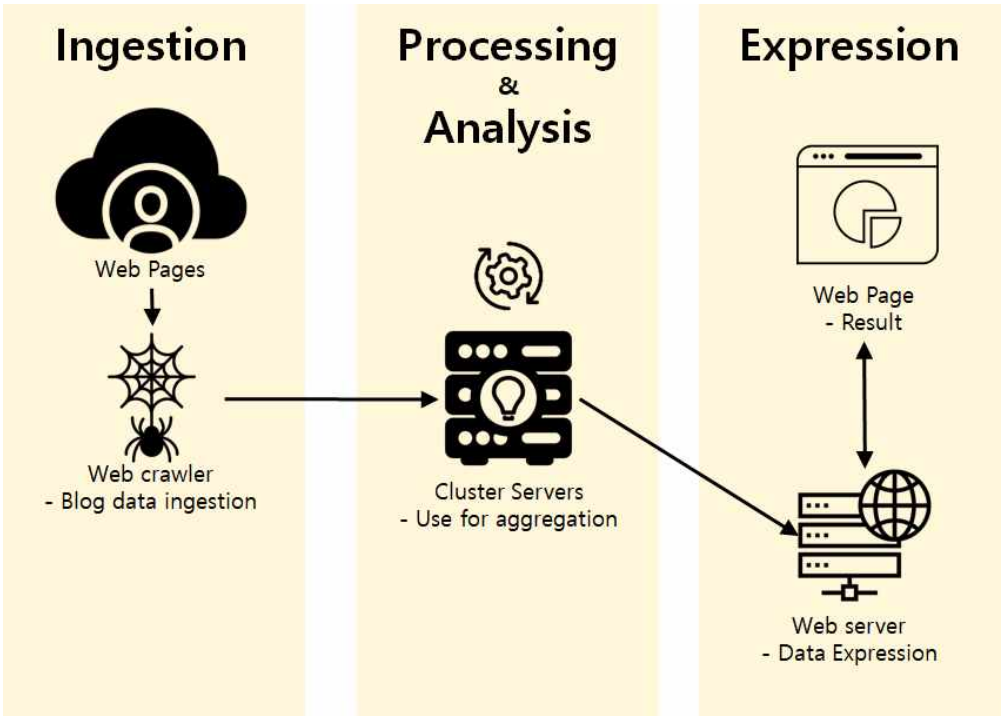
[그림 5] jQuery의 AJAX 구현 예

제 3 장 서비스 설계

3.1 서비스 구조 설계

3.1.1 시스템 구성

본 논문에서 구현한 웹 서비스는 데이터 수집, 처리, 표현 과정을 통해 사용자에게 서비스된다. 서비스를 제공하기 위한 시스템 구성 개요는 [그림 6]과 같다.

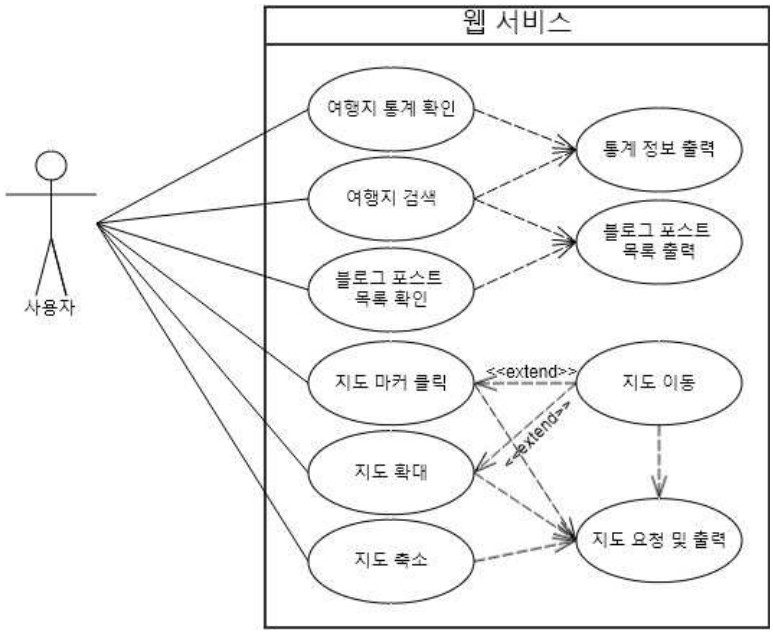


[그림 6] 시스템 구성 개요

Web Crawler는 웹으로부터 데이터를 블로그 데이터 수집하고 클러스터 서버에 저장한다. 클러스터 서버는 데이터의 저장과 색인, 분류를 통해 사용자에게 데이터가 제공될 수 있도록 처리한다. 웹 서버는 사용자에게 웹 페이지를 제공하고 사용자의 요청에 따라 클러스터 서버에 저장된 데이터를 제공한다.

3.1.2 서비스 기능 범위

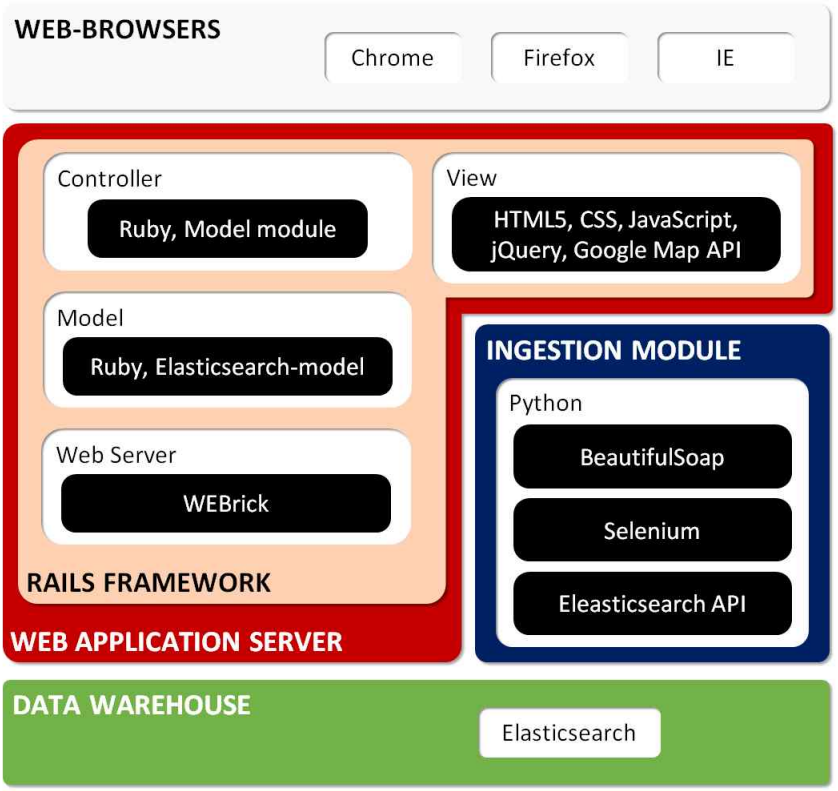
본 논문에서 구현하는 웹 서비스가 제공하는 기능은 여행지에 대한 검색, 여행지 통계 정보 제공, 지도 확대/축소, 블로그 포스트 목록 제공, 지도의 마커 클릭 기능으로 구성한다. 다음 [그림 7]은 이를 사용사례 다이어그램으로 도식화한 것이다.



[그림 7] 웹 서비스의 사용사례 다이어그램

3.1.3 서비스 소프트웨어 스택

웹 서비스를 구성하는 소프트웨어 스택은 [그림 8]과 같다.



[그림 8] 웹 서비스의 소프트웨어 스택

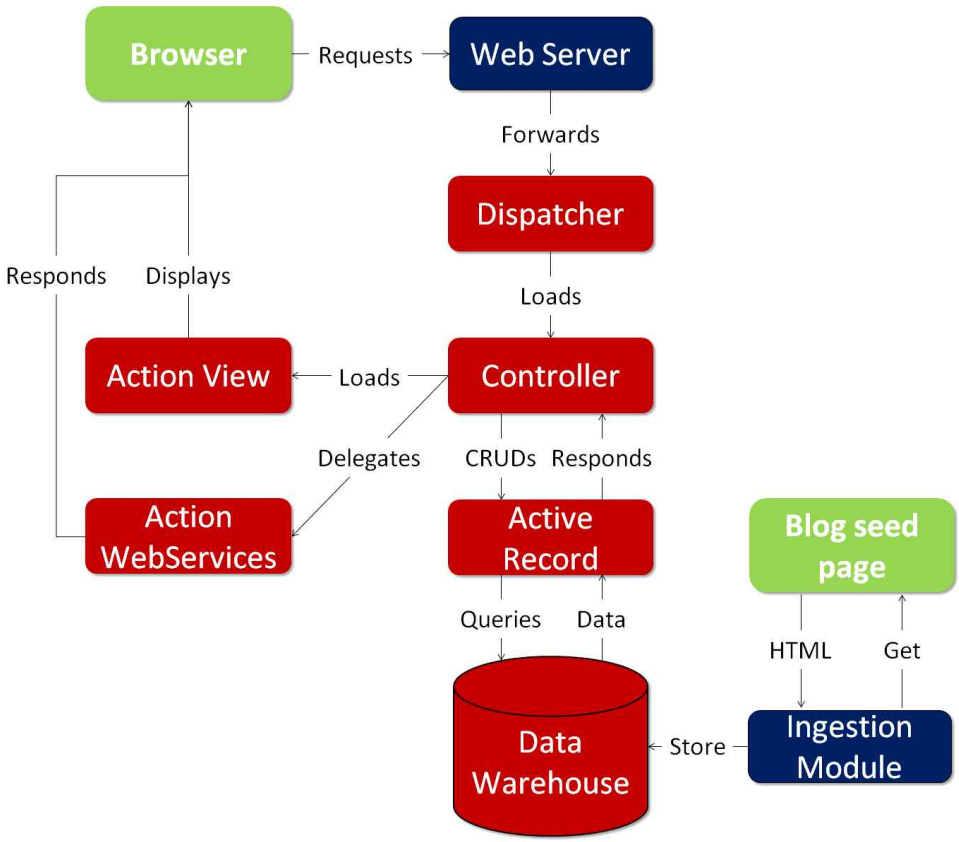
사용자는 웹 브라우저들을 통해 본 논문의 웹 서비스를 사용하게 된다. 웹 브라우저들은 현재 서비스 중인 구글의 크롬, 마이크로소프트의 인터넷 익스플로러 등의 모든 브라우저를 사용할 수 있다. 사용자에게 웹 페이지의 제공과 요청, 응답을 처리하기 위한 웹 애플리케이션 서버는 Ruby on Rails 프레임워크로 구

성된다. Ruby on Rails 프레임워크에 기본으로 제공되고 있는 WEBrick 서버를 이용해 웹 서버가 동작하며, 모델 컴포넌트는 루비를 기반으로 구현되고 Elasticsearch-model 라이브러리를 통해 Elasticsearch 클러스터 서버와 연결된다. 컨트롤러 컴포넌트는 사용자의 요청을 처리하기 위해 루비를 통해 구현하고, 모델 컴포넌트를 사용해 데이터에 접근한다. View 컴포넌트는 사용자에게 동적 웹페이지를 제공하기 위해 HTML5, CSS, JavaScript를 이용해 구현하며 jQuery를 통해 사용자의 이벤트 처리와 구글 지도 API를 통해 지도를 이용한 서비스를 제공한다. 수집 모듈은 파이썬을 이용해 구현한다. 동적 웹 페이지의 데이터 수집을 위하여 Selenium와 PhantomJS를 이용해 수집할 웹 페이지를 로드하고 BeautifulSoup를 통한 구문 분석으로 데이터를 추출한다. 이후 Elasticsearch-py 라이브러리를 이용해 Elasticsearch 클러스터 서버와 연결하고 데이터를 저장한다. 본 논문의 웹 서비스는 Elasticsearch를 사용한 Data Warehouse를 구현한다. 또한, Elasticsearch의 검색 엔진 기능과 집계 기능을 이용하여 사용자에게 검색 기능 및 집계 데이터를 제공한다.

3.1.4 서비스 워크플로우

웹 서비스는 [그림 8]과 같은 워크플로우를 가진다. 수집 모듈은 블로그의 시드 페이지를 요청하여 HTML 문서를 구문 분석하고 데이터를 수집한다. 수집된 데이터는 데이터 웨어하우스에 저장한다. 브라우저를 통해 사용자의 요청이 전달되면 Ruby on Rails 프레임워크의 웹 서버가 이를 받아 Dispatcher에게 전달한다. Dispatcher는 전달받은 요청에 따라 적합한 컨트롤러를 로드한다. 로드된 컨트롤러는 사용자 요청을 처리하는데 요청에 따라 모델 컴포넌트의 Active Record에게 CRUD(Create, Read, Update, Delete) 메서드를 전달하거나 Action

View를 로드하거나 Action WebServices에게 처리한 결과를 위임한다. View 컴포넌트는 사용자에게 Action View와 Action WebServices를 통해 처리된 웹 페이지 또는 결과를 응답한다. Active Record는 전달받은 메서드를 통해 Data Warehouse에 질의를 요청하고 응답을 처리하여 컨트롤러에게 제공한다. 다음 [그림 9]는 이를 도식화한 그림이다.



[그림 9] 웹 서비스 워크플로우

3.1.5 데이터 구조 설계

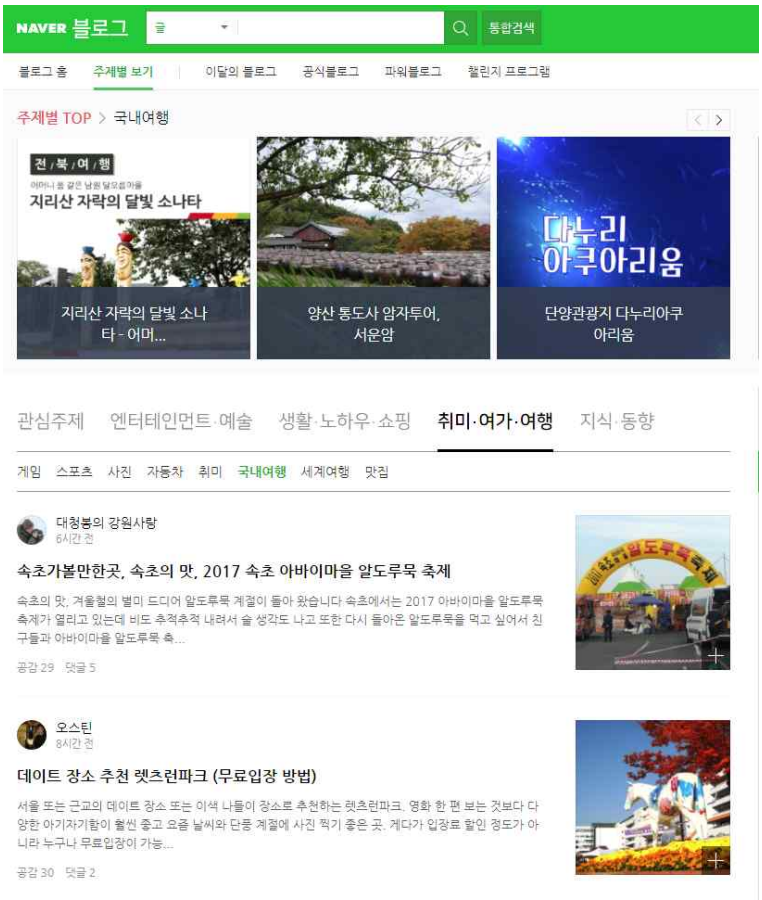
본 논문에서는 국내여행을 주제로 한 네이버 블로그 포스트를 대상으로 포스트의 작성자, 작성자의 이미지 URL, 포스트 URL, 포스트 요약, 포스트 제목, 본문을 수집한다. 수집한 데이터의 저장을 위해 설계한 데이터 구조는 다음 [그림 10]과 같다.

[그림 10] 데이터 구조

Elasticsearch는 index, type, document로 데이터를 문서 객체로 표현한다. index는 관계형 데이터베이스(이하 'RDB')의 데이터베이스, type은 RDB의 테이블, document는 RDB의 레코드와 같다. document안의 각 field들은 RDB의 column 역할을 한다. document에 설정한 값은 문서를 구분하는 기준이 되며, RDB의 기본키와 같은 역할을 한다. [그림 10]에서는 userID_postID가 blogs index, cityName이 areas index에서 문서를 구분하는 역할이다.

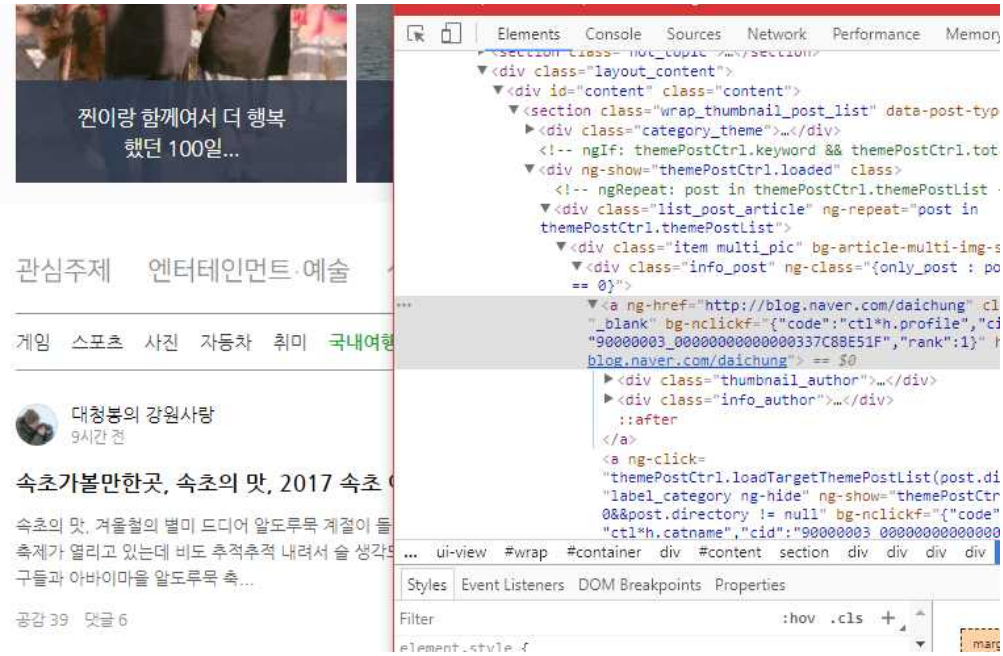
3.2 데이터 수집 설계

수집 모듈을 구현하기 위해 데이터 수집을 위한 수집 대상 선정, 웹 페이지 동작 분석 그리고 HTML 구조 분석을 선행하고 수집을 위한 알고리즘을 설계한다. 데이터 수집 대상은 네이버 블로그의 국내여행을 주제로 하는 포스트들로 설정한다. 수집 대상을 웹 크롤링에서 시드 페이지라고 한다.



[그림 11] 수집 대상의 웹 페이지의 화면

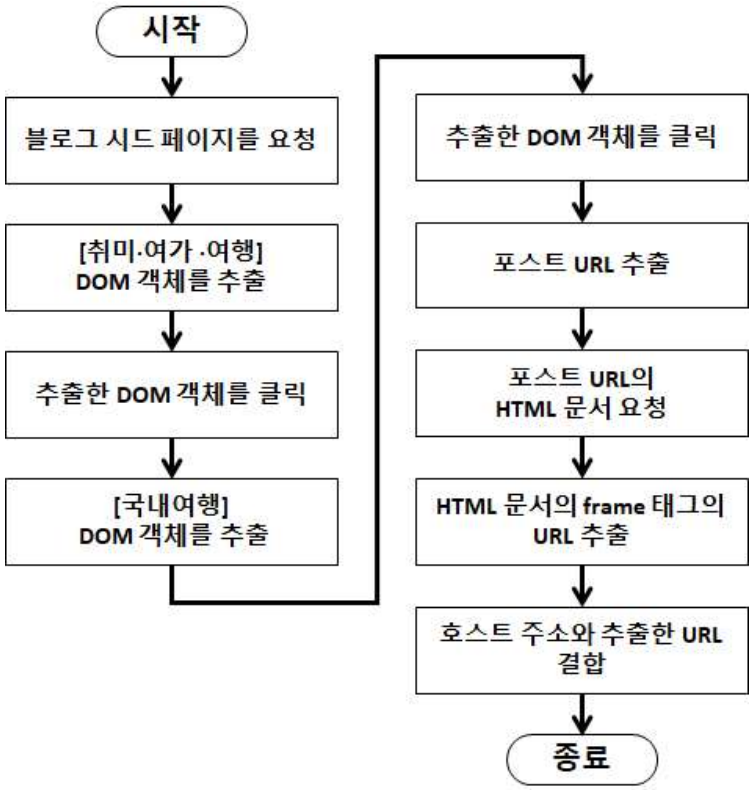
본 논문에서는 블로그 포스트들의 URL을 수집하는 1차 수집과 블로그 포스트에서 실제 필요 데이터를 수집하는 2차 수집 단계로 구현한다. 1차 수집을 위해 시드 페이지의 동작을 분석하면 링크를 통한 페이지 전환 방식에서 자바스크립트를 이용한 동적 렌더링 방식을 사용하는 것을 확인할 수 있었다. 국내여행 주제에 맞는 블로그 포스트의 목록에 접근하기 위하여 각각의 DOM 객체에 접근할 필요가 있다. 이를 위해 HTML 분석을 실행한 결과는 다음 [그림 12]와 같다.



[그림 12] 시드 페이지의 HTML 문서내용과 포스트의 주소

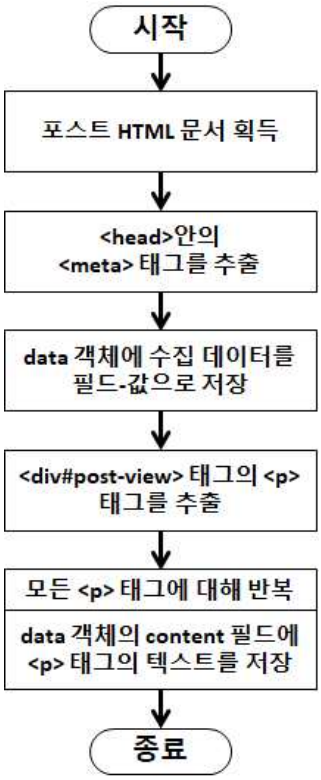
시드 페이지는 한 페이지마다 10개의 포스트로 구성되어 있고, 구글의 크롬 브라우저에 있는 개발자 도구를 이용해 HTML 문서를 분석한 결과 각 포스트의

주소는 list_post_article 클래스를 가진 태그 안에 첫 번째 <a> 태그에 있는 것을 확인하였다. 분석을 통해 얻은 포스트의 주소를 통해 블로그 포스트에 접근하면 실제 블로그 포스트를 <frameset> 태그를 통해 웹 페이지를 렌더링한 것을 확인할 수 있었다. <frameset> 태그의 주소는 호스트 주소를 제외한 나머지 부분의 URL만 기술되어 있었고, 호스트 주소와 <frameset> 태그의 주소를 결합하여 접근한 결과 같은 내용을 출력하는 실제 블로그 포스트의 HTML 문서에 접근할 수 있었다. 이를 기반으로 설계한 1차 수집 알고리즘은 다음 [그림 13]과 같다.



[그림 13] 포스트 URL을 획득하기 위한 1차 수집 알고리즘

2차 수집 단계에서는 각각의 블로그 포스트에서 본 논문에서 구현하는 웹 서비스에 필요한 실제 데이터를 수집한다. 1차 수집에서 얻은 블로그 포스트에 접근하여 동작 방식을 확인한 결과 자바스크립트를 이용한 동적 렌더링 요소는 없었고, 1차 수집과 같은 방법으로 HTML 문서를 분석한 결과 서비스에 필요한 실제 데이터의 대부분을 <head> 태그 안에 <meta> 태그에서 확인할 수 있었다. 또한, content 필드에 저장할 본문 내용은 post-view ID를 가진 태그 안에 <p> 태그로 문단 또는 문장으로 기술되어 있는 것을 확인했다. 따라서 다음 [그림 14]와 같은 데이터 수집 알고리즘을 설계하였다.



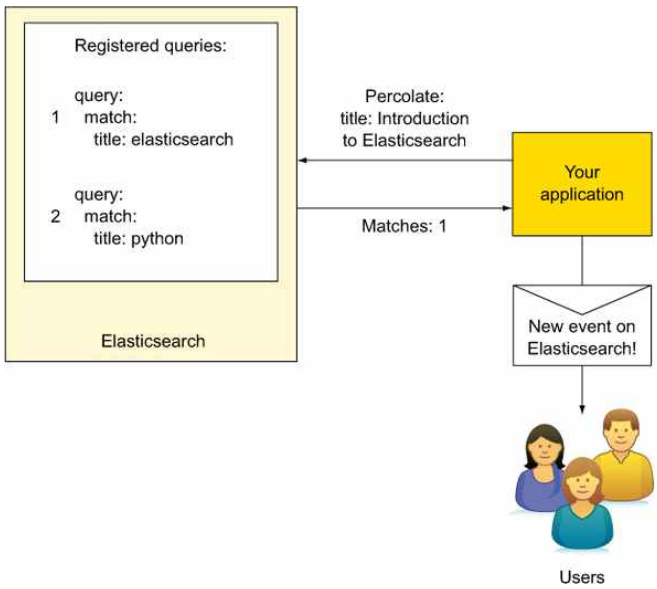
[그림 14] 웹 서비스를 위한 데이터를 획득하기 위한 알고리즘

3.3 데이터 처리 설계

본 논문에서 제안하는 여행지 통계를 제공하기 위한 데이터 처리를 위해 Elasticsearch의 percolator 기능을 사용한다. percolator는 일반적인 질의 방식의 반대로 다음과 같은 동작을 수행한다.

- 문서 대신 질의의 색인을 만든다.
- 질의 대신 문서를 Elasticsearch에 보낸다. 이 과정을 문서를 percolate 한다고 한다.
- 보통의 질의 같은 방식 대신, 문서와 일치하는 질의 목록을 받는다.

이런 이유로 percolator는 '뒤집어 검색하기'로도 정의된다.[10] 다음 [그림 15]는 percolator를 사용한 전형적인 사례를 통해 percolator의 동작을 도식화한 모습이다.

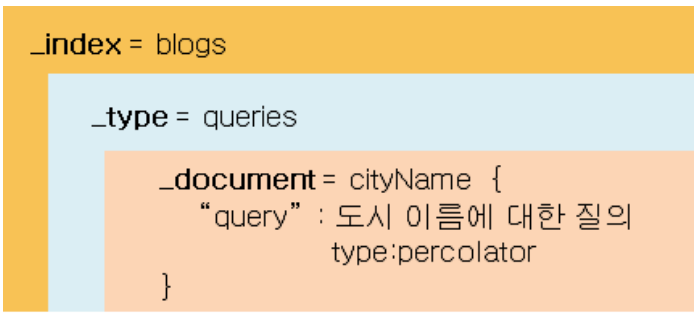


[그림 15] Percolator를 사용한 사례인 알림 기능의 동작

percolate의 구현은 다음과 같은 네 단계로 구성된다.

1. 등록한 질의가 참조하는 모든 필드에 대해 매핑이 있다는 것을 확인한다.
2. 질의의 등록을 위한 새로운 필드를 인덱스에 추가한다.
3. 질의 자체를 등록한다.
4. 문서에 percolator를 실행한다.

등록한 질의가 필드를 참조하기 위해 질의의 대상이 될 필드를 설정한다. 여기에서 질의의 대상은 데이터를 분석하기 위한 처리를 위한 필드를 뜻한다. 또한, 3단계의 질의를 등록할 때 사용되어 질의가 수행될 필드이다. 본 논문에서는 블로그 포스트의 본문 내용을 가지고 있는 blogs 인덱스의 content 필드를 질의 대상으로 설정한다. percolator의 구현을 위하여 blogs 인덱스에 다음 [그림 16]과 같은 데이터 구조를 추가한다.



[그림 16] percolator를 위한 추가 데이터 구조

percolator를 사용하기 위해 필드의 타입을 percolator로 정의한다. 그러면 이후 질의 자체를 해당 필드에 등록할 수 있게 되고, Elasticsearch는 해당 필드에 등록된 질의를 통해 문서를 percolate 하여 문서와 일치하는 질의 목록을 제공하게 된다.

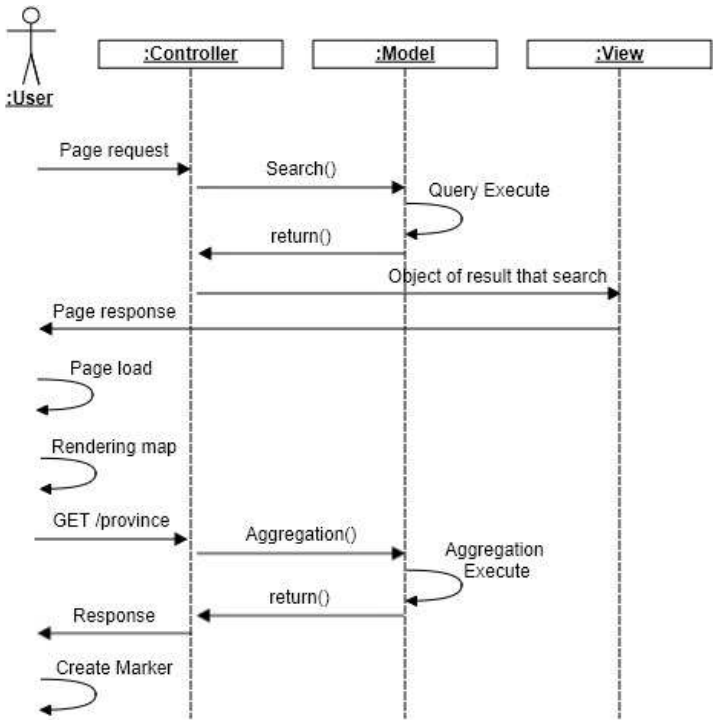
3.4 데이터 표현 설계

웹 서비스의 표현을 위해 먼저 웹 페이지의 디자인과 기능을 설계한다. 3.1.2의 서비스 기능 범위에서 언급한 내용을 구현하기 위하여 다음 [그림 17]과 같은 기능명세서를 설계하였다.

No.	Event	Action	Description
1	Page load	구글 지도 렌더링	
2	Page load	도/시 별 통계 정보 마커 렌더링	구글 지도 API의 마커를 활용
	Click	해당 도 위치로 지도 중심 이동, 시 단위로 지도 확대	확대 후 시 단위로 지도 마커 갱신
3	Page request	가장 최근의 10개의 블로그 포스트 목록 출력	
4	Input text	입력 힌트 삭제	
	Keypress 'enter'	검색 질의 요청	응답 후 검색 결과를 기반으로 지도 위치와 마커, 블로그 포스트 목록 갱신

[그림 17] 웹 페이지 기능 명세서

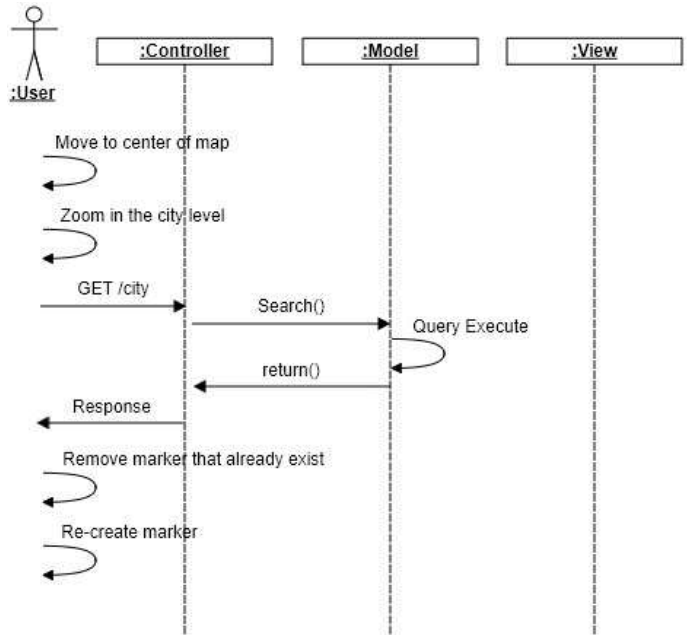
처음 웹 페이지에 접근하여 웹 페이지의 렌더링이 끝나면 구글 지도를 출력하고, 도별의 통계 정보를 지도 마커를 활용하여 출력한다. 왼쪽에는 가장 최근에 수집한 10개의 블로그 포스트 목록을 출력하고, 왼쪽 위에 검색 기능 제공을 위한 폼을 구성한다. 다음으로 사용자 요청에 따른 웹 애플리케이션의 동작을 설계한다. 다음 [그림 18]은 처음 페이지 접속 시의 웹 애플리케이션 측 동작을 시퀀스 다이어그램으로 표현한 것이다.



[그림 18] 페이지 접속에 대한 시퀀스 다이어그램

사용자의 페이지 요청이 전달되면 컨트롤러는 모델에게 질의를 전달하고 질의 결과를 기반으로 뷰가 페이지를 구성하여 응답한다. 페이지 로드가 완료되면

지도 마커 표현을 위해 REST 요청을 보내고 집계 결과를 응답하여 지도 마커를 만든다. 지도 마커 클릭에 대한 동작은 다음 [그림 19]의 시퀀스 다이어그램과 같다.



[그림 19] 지도 마커 클릭, 검색 기능에 대한 시퀀스 다이어그램

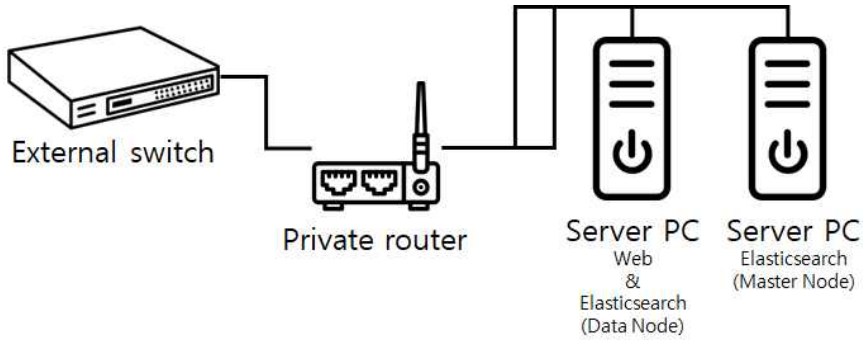
지도 마커를 클릭하면 해당 마커의 도시를 검색하게 된다. 이는 검색 창을 통한 검색과 동일하기 때문에 위 [그림 19]의 시퀀스 다이어그램은 지도 마커 클릭과 검색 기능의 동작을 같이 표현할 수 있다. 검색에 대한 동작 또한 REST 요청을 통해 실행된다. 도시 이름을 기준으로 검색에 대한 요청을 보내면 컨트롤러는 해당 도시 이름에 대한 질의를 모델에게 전달하고 모델은 이를 실행한 결과를 반환한다. 컨트롤러는 반환된 결과를 직접 JSON 형식으로 사용자에게 응답한다. 사용자는 응답된 데이터를 기반으로 지도 마커를 재구성한다.

제 4 장 서비스 구현 및 테스트

4.1 서비스 구현

4.1.1 개발 환경 구축

서비스의 구현을 위해 외부 스위치에 공유기를 연결하여 내부 라우터로 사용하였다. 서버 구축을 위하여 두 대의 데스크탑을 공유기에 연결하였다. 서버 구축에 사용한 데스크탑의 사양은 다음 [표 1]과 같다.



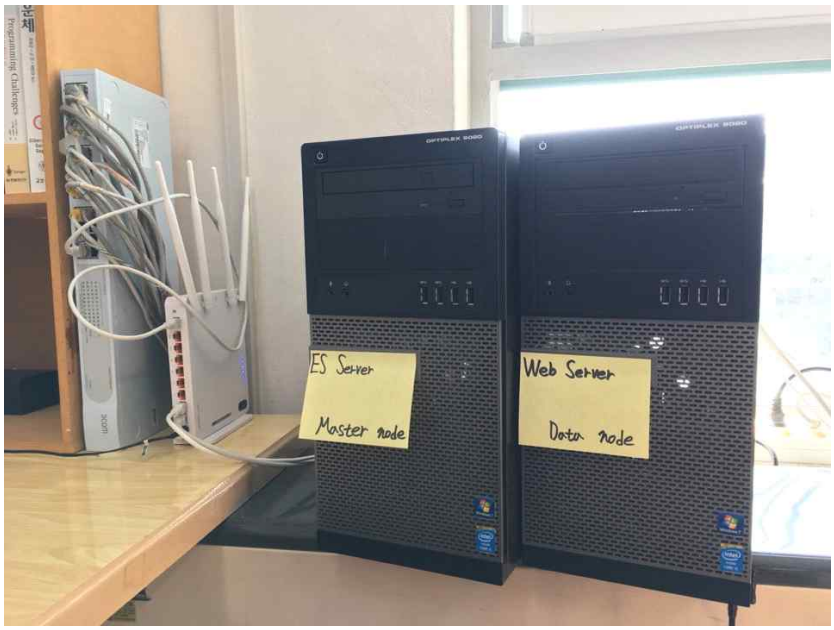
[그림 20] 웹 서비스를 위한 하드웨어 구성도

Server	CPU	RAM	HDD
Elasticsearch	Intel Core i5-4670 3.40 GHz	4 GB	1 TB
Web	Intel Core i5-4670 3.40 GHz	4 GB	1 TB

[표 1] 서버 데스크탑의 사양정보

웹 서버와 Elasticsearch 서버의 IP는 각각 192.168.0.6, 192.168.0.7로 할당하였고, 웹 서비스 접근과 Elasticsearch의 모니터링을 위해 공유기에 각각 8080과 9200 번 포트로 포트포워딩 설정을 하였다. 구성한 서버 환경은 다음과 같다.

- 운영체제: Ubuntu 14.04.5 LTS
- 웹 서버: Ruby 2.3.3, Ruby on Rails 5.1.4, WEBrick 3.10.0
- 클러스터 서버: Elasticsearch 5.1.1
  - Cluster name: es
  - Nodes name: masterNode, dataNode
  - Master node: masterNode, Data-node: Both



[그림 21] 실제 설치된 공유기와 서버 모습

웹 서비스에서 사용하는 Elasticsearch는 Ubuntu Elasticsearch패키지를 이용해 설치하였다. 설치된 패키지는 다음과 같은 경로가 설정된다.

- 환경설정 파일: /etc/elasticsearch
- 실행 및 관련 파일: /usr/share/elasticsearch

Elasticsearch를 패키지로 설치하면 자동 실행이 되지 않으므로 /etc/rc.d를 수정하여 부팅 시 자동 실행하도록 설정하였다. 클러스터를 구성하기 위해 각각의 노드의 Elasticsearch 환경설정 파일인 elasticsearch.yml 파일을 수정한다. 다음 [그림 22]는 마스터 노드의 설정한 내용이다.

```
# 클러스터의 이름 설정, 같은 이름을 갖는 노드는 클러스터에 속하게 됨
cluster.name: es
# 노드의 이름 설정, 클러스터 내에서 노드를 구별하는 이름
node.name: masterNode
# 해당 노드를 마스터 노드 여부 설정
node.master: true
# 외부에서 노드에 접속할 수 있는 네트워크 주소
network.host: 192.168.0.7, localhost
# 외부에서 노드에 접속할 수 있는 포트
http.port: 9200
# 노드 간 바인딩을 위한 포트
transport.tcp.port: 9300
# 다른 노드와 연결을 위해 ping을 보낼 노드 주소 목록
discovery.zen.ping.unicast.hosts: ["192.168.0.6:9301"]
# 최소 마스터 개수 설정
discovery.zen.minimum_master_nodes: 2
```

[그림 22] Elasticsearch 마스터 노드의 환경설정

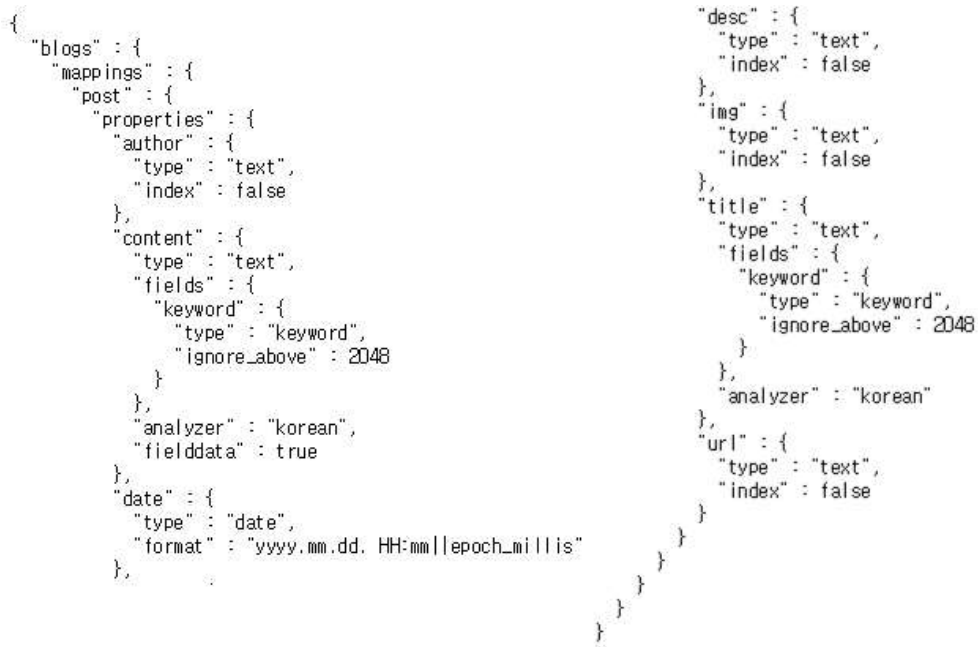
Elasticsearch는 한글을 위한 분석기를 제공하지 않기 때문에 한글 분석을 위해 오픈소스 한국어 형태소 분석기인 은전한닢 플러그인을 설치 및 적용하였다. 은전한닢 플러그인을 설치할 때 중요한 점은 Elasticsearch 버전과 플러그인의 버전이 같아야 한다. 따라서 설치한 Elasticsearch의 버전에 따라 은전한닢 5.1.1.1 버전의 플러그인을 설치하였다. 이후 다른 모든 노드에 플러그인을 추가

로 설치하였다. 설치한 한글 분석기의 적용을 위해서는 분석기를 사용할 인덱스에 custom 타입의 토크나이저를 설정하여야 한다. 다음 [그림 23]은 blogs 인덱스에 설치한 한글 분석기를 설정한 결과이다.


```
"blogs" : {
  "settings" : {
    "index" : {
      "number_of_shards" : "3",
      "provided_name" : "blogs",
      "creation_date" : "1497021048075",
      "analysis" : {
        "analyzer" : {
          "korean" : {
            "type" : "custom",
            "tokenizer" : "mecab_ko_standard_tokenizer"
          }
        }
      },
      "number_of_replicas" : "2",
      "uuid" : "mKqC9R2lQ3uy1t3SN3F1xA",
      "version" : {
        "created" : "5010199"
      }
    }
  }
}
```

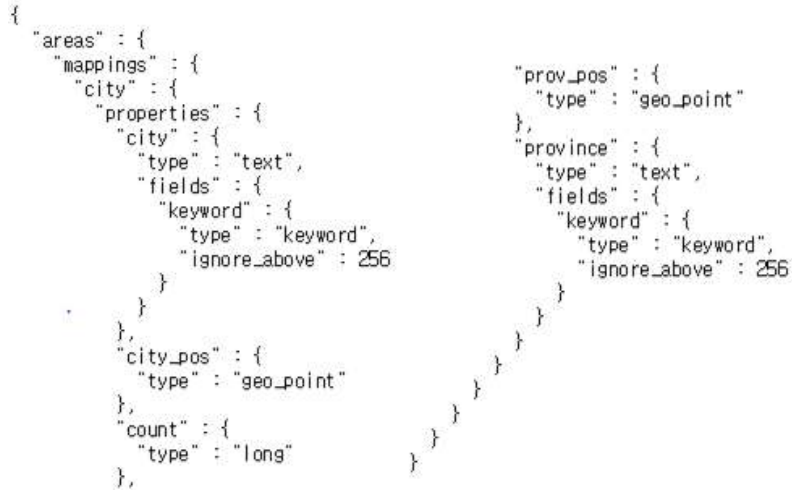

[그림 23] 한글 분석기를 설정한 blogs 인덱스의 정보

수집 데이터의 저장을 위해 Elasticsearch에 blogs 인덱스와 post 타입을 생성하였다. 타입은 문서의 데이터 구조를 명시하며, 필드의 속성, 길이 등을 정의한다. 타입의 생성을 매핑한다고 하며, Elasticsearch는 document가 색인 될 때 인덱스가 없으면 동적으로 매핑하여 데이터 구조를 생성하지만, 한글 분석기 설정, Epoch 시간 설정 등의 세부사항은 정적 매핑을 통해 구성해야 한다. 다음 [그림 24]는 blogs 인덱스의 타입을 생성하고 결과를 확인한 모습이다.



[그림 24] 매핑 적용 후 blogs 인덱스의 데이터 구조

최상단에서 blogs 필드로 인덱스 이름을 확인할 수 있고 mappings 필드 안의 post 필드로 blogs 인덱스의 타입 이름이 제대로 설정되었는지 확인할 수 있다. properties 필드에 지정된 다양한 필드들이 실제 데이터가 매핑 될 스키마이다. title과 content 필드에 한글 분석기를 설정한 내용을 확인할 수 있고, 날짜순으로 정렬하기 위해 date 필드에 epoch_millis를 설정하였다. 분석한 데이터를 기반으로 여행 지역의 언급된 횟수를 저장하기 위해 areas 인덱스를 생성하였다. areas 인덱스의 매핑 구조는 [그림 25]와 같다. areas 인덱스의 필드 중 prov_pos와 city_pos 필드는 지역의 좌표 데이터를 저장하기 위해 geo_point 타입을 적용하였다.



[그림 25] areas 인덱스의 매핑 구조

여행 지역은 동적으로 변경되지 않기 때문에 전국의 시/군 단위의 여행지역 161개를 조사하여 문서 목록을 작성하였다. Elasticsearch의 bulk API를 사용하여 문서 목록을 일괄적으로 Elasticsearch에 색인하였다.


```
{ "index": {"_index": "areas", "_type": "city", "_id": "seoul"} }
{ "prov_pos" : {"lat": 37.566535, "lon": 126.977969}, "city_pos" : {"lat": 37.566535, "lon": 126.977969},
"province": "서울특별시", "city": "서울특별시", "count": 0 }
{ "index": {"_index": "areas", "_type": "city", "_id": "busan"} }
{ "prov_pos" : {"lat": 35.179554, "lon": 129.075641}, "city_pos" : {"lat": 35.179554, "lon": 129.075641},
"province": "부산광역시", "city": "부산광역시", "count": 0 }
{ "index": {"_index": "areas", "_type": "city", "_id": "deagu"} }
{ "prov_pos" : {"lat": 35.871435, "lon": 128.601445}, "city_pos" : {"lat": 35.871435, "lon": 128.601445},
"province": "대구광역시", "city": "대구광역시", "count": 0 }
{ "index": {"_index": "areas", "_type": "city", "_id": "incheon"} }
{ "prov_pos" : {"lat": 37.456255, "lon": 126.705206}, "city_pos" : {"lat": 37.456255, "lon": 126.705206},
"province": "인천광역시", "city": "인천광역시", "count": 0 }
{ "index": {"_index": "areas", "_type": "city", "_id": "gwangju"} }
{ "prov_pos" : {"lat": 35.159545, "lon": 126.852601}, "city_pos" : {"lat": 35.159545, "lon": 126.852601},
"province": "광주광역시", "city": "광주광역시", "count": 0 }
{ "index": {"_index": "areas", "_type": "city", "_id": "deajeon"} }
{ "prov_pos" : {"lat": 36.350411, "lon": 127.384547}, "city_pos" : {"lat": 36.350411, "lon": 127.384547},
"province": "대전광역시", "city": "대전광역시", "count": 0 }
{ "index": {"_index": "areas", "_type": "city", "_id": "ulsan"} }
{ "prov_pos" : {"lat": 35.538377, "lon": 129.313596}, "city_pos" : {"lat": 35.538377, "lon": 129.313596},
"province": "울산광역시", "city": "울산광역시", "count": 0 }
{ "index": {"_index": "areas", "_type": "city", "_id": "sejong"} }
{ "prov_pos" : {"lat": 36.592878, "lon": 127.292331}, "city_pos" : {"lat": 36.592878, "lon": 127.292331},
"province": "세종특별시", "city": "세종특별시", "count": 0 }
```


[그림 26] 여행 지역을 저장하기 위한 데이터 목록

파이썬으로 구현한 웹 크롤러가 수집한 데이터를 Elasticsearch에 저장하기 위해 Elasticsearch의 파이썬 클라이언트 라이브러리인 Elasticsearch-py를 설치 후 테스트를 진행하였다.

```
es@esServer:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from elasticsearch import Elasticsearch
>>> es = Elasticsearch([{'host': 'localhost', 'port': 9200}])
>>> print(es.cluster.health())
{u'status': u'green', u'number_of_nodes': 2, u'unassigned_shards': 0, u'number_of_pendin
g_tasks': 0, u'number_of_in_flight_fetch': 0, u'timed_out': False, u'active_primary_shar
ds': 4, u'task_max_waiting_in_queue_millis': 0, u'cluster_name': u'es', u'relocating_sha
rds': 0, u'active_shards_percent_as_number': 100.0, u'active_shards': 8, u'initializing_
shards': 0, u'number_of_data_nodes': 2, u'delayed_unassigned_shards': 0}
```

[그림 27] 파이썬 IDLE을 이용한 Elasticsearch-py 테스트

[그림 27]은 파이썬 IDLE(Integrated DeveLopment Environment, 통합 개발 환경)을 이용해 Elasticsearch-py를 import하고 테스트를 진행한 모습이다. 해당 테스트에서는 Elastcisearch에 연결하여 클러스터의 상태를 요청하는 cluster.health() 메서드를 호출하여 요청을 전달한 후 응답 내용을 출력하는 것으로 설치가 성공적으로 이루어진 것을 확인하였다.

4.1.2 데이터 수집 구현

데이터 수집을 위해 웹 크롤러(이하 “크롤러”)를 구현하였다. 3.2절에서 분석한 결과 수집 대상 페이지의 동작이 자바스크립트를 이용한 동적 렌더링 방식을 사용하기 때문에 파이썬을 기반으로 Selenium 라이브러리와 PhantomJS를 이용해 크롤러를 구현하였다. 다음 [그림 28]은 1차 수집 알고리즘을 바탕으로 구현한 크롤러 소스 코드 일부이다.

```
#Make Browser object for request to page
browser = webdriver.PhantomJS(executable_path='/home/es/phantomjs/bin/phantomjs')
browser.implicitly_wait(2) # Wait for page load
#Request to seed page
browser.get("https://section.blog.naver.com/ThemePost.nhn")
#Click the travel subject
menus = browser.find_elements_by_css_selector('div.navigator_category a')
menus[3].click()
time.sleep(0.5) # Wait for page load
#Click the category that is domestic travel
subMenus = browser.find_elements_by_css_selector('div.navigator_category_sub[data
-set="hobby"] a')
subMenus[5].click()
time.sleep(0.5)
```

[그림 28] 블로그 시드 페이지를 렌더링 하는 소스 코드 일부

Selenium의 webdriver를 이용해 phantomJS 브라우저를 실행시킨 후 블로그 시드 페이지를 get() 메서드를 이용해 요청한다. webdriver를 사용할 때, 모든 페이지의 요청 또는 DOM 객체를 클릭한 후에는 페이지의 렌더링 시간을 기다리는 시간이 필요하고 time 라이브러리의 sleep() 메서드를 사용하여 구현하였다. 페이지가 렌더링 되고 나면 webdriver의 find_elements_by_css_seletor() 메서드를 사용해 <div.navigotor_category> 태그의 <a> 태그 객체를 찾고 [취미·여가·여행] 카테고리를 표현하는 DOM 객체를 click() 메서드로 클릭한다. 렌더링 된 세부 카테고리의 DOM 객체를 얻기 위해 다시 find_elements_by_css_selector() 메서드로 data-set 속성에 “hobby”라는 값을 갖는 <div.navigator_category_sub>의 <a> 태그를 찾아 [국내여행] 카테고리를 표현하는 DOM 객체를 획득하고 클릭한다. 이 과정을 통해 국내여행에 관련된 블로그 포스트들을 렌더링 한 시드 페이지를 획득할 수 있었다. 렌더링 된 페이지에서 포스트 URL을 추출하고 호스트 주소와 재결합하여 포스트 주소를 정규화를 실행하는 부분은 다음과 같이 구현하였다. 렌더링 된 시드 페이지는 webdriver의 page_source 필드에 저장되어 해당 필드의 값을 BeautifulSoup 객체에 전달하여 구문 분석을 실행하고 select()

메서드를 이용하여 div.list_post_article div.desc a.desc_inner에 일치하는 태그들의 값을 획득한다. 획득한 태그의 값은 포스트 주소이고 이를 urllib2 라이브러리를 이용해 포스트들의 HTML 문서를 획득한 후 BeautifulSoup에 전달한다. 다시 select()를 이용해 frame#mainframe 태그를 추출하고 해당 태그의 src 속성 값을 블로그 호스트 주소인 “http://blog.naver.com”과 결합한다. 작업 결과 포스트의 실제 주소를 얻을 수 있었다. 다음 [그림 29]는 크롤러에 구현한 소스 코드 일부이다.

```
#Save rendered html source
html = browser.page_source
#Make the object that is for parsing html source
blogBS = BeautifulSoup(html, 'html.parser')
#Scrap each post url
postLink = blogBS.select('div.list_post_article div.desc a.desc_inner')
for url in postLink:
        #Transforming to url of real post
        postBS = BeautifulSoup(urllib2.urlopen(str(url['href'])).read(), 'html.parser')
        postUri = "http://blog.naver.com" + postBS.select('frame#mainFrame')[0]['src']
```

[그림 29] 실제 포스트 URL을 추출하고 정규화 하는 소스 코드 일부

이후 포스트의 실제 주소를 획득하는 방식과 같이 포스트의 실제 주소와 urllib2를 이용해 포스트의 HTML 문서를 획득하고 BeautifulSoup에 전달한다. 수집 데이터는 select()에 head meta[property]를 인수로 하여 <meta> 태그를 획득한 후 title, desc, author, img, url 필드에 저장할 데이터를 각 태그로부터 추출하여 이를 data 객체에 필드-값으로 저장한다. date 필드에 저장할 데이터는 데이터 추출 당시의 시간을 저장한다. content 필드에 저장할 데이터를 추출하기 위해 select()에 div#post-view{postID} p를 인수로 하여 해당 포스트의 본문 내용을 추출하고 data 객체에 저장한다. 다음 [그림 30]은 상기 내용을 구현한 크롤러 소스 코드 일부이다.

```
#Transforming to url of real post
postBS = BeautifulSoup(urllib2.urlopen(str(url['href'])).read(), 'html.parser')
postUri = "http://blog.naver.com" + postBS.select('frame#mainFrame')[0]['src']

#Make the object that is for parsing blog post
contentBS = BeautifulSoup(urllib2.urlopen(str(postUri)).read().decode('ms949', 'utf8'),
'html.parser')

#Parse to content
meta = contentBS.select('head meta[property]')
tmpID = meta[3]['content'].replace("http://blog.naver.com/", "").split("/")
#Make data object for save
data = {
        "date"   : datetime.now().strftime("%Y.%m.%d. %H:%M"),
        "author" : meta[8]['content'],
        "title"  : meta[0]['content'],
        "img"    : meta[9]['content'],
        "desc"   : meta[2]['content'],
        "url"    : meta[3]['content'],
        "content": ""
}

selector = "div#post-view" + tmpID[1] + " p"
content = contentBS.select(selector)
for c in content:
        data['content'] += c.getText().strip()+" "
```

[그림 30] 실제 포스트에서 데이터를 획득하는 소스 코드 일부

수집 데이터를 추출한 다음 Elasticsearch에 저장하기 위해 Elasticsearch-py 라이브러리를 사용하여 수집한 데이터를 저장한다. 제작한 크롤러에 Elasticsearch-py를 적용한 내용은 [그림 31]과 같다.

```
...
import elasticsearch

#Connect part
es = elasticsearch.Elasticsearch([{'host': '192.168.0.7', 'port': 9200}])

        ...

try:
        #Indexing blog post data into elasticsearch
        es.create(index='blogs', doc_type='post', id=str(tmpID[0]+"_"+tmpID[1]), body=da
ta)
except elasticsearch.TransportError:
        continue
```

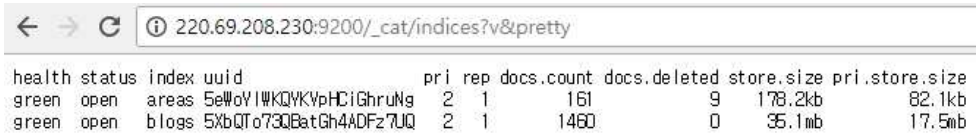
[그림 31] 크롤러에 Elasticsearch-py를 적용한 소스 코드 일부

[그림 31]에서 일부 소스 코드는 생략하였다. Elasticsearch-py 라이브러리를 import하고 Elasticsearch의 IP와 port를 설정하여 연결한다. 연결이 성공하면 Elasticsearch는 elasticsearch 객체를 반환한다. 이 객체를 통해 create(), update(), delete() 등의 다양한 메서드를 사용할 수 있다. 본 논문에서 구현한 크롤러는 create() 메서드를 사용해 수집한 데이터를 Elasticsearch에 저장하도록 구현하였다. create()는 문서를 색인할 때 같은 document ID를 가진 문서가 이미 존재하면 409 응답 코드와 문서가 이미 존재한다는 내용의 오류를 발생한다.

```
TransportError(409, u'version_conflict_engine_exception',
 u'[test1][1]: version conflict, document already exists
(current version [1])')
```

[그림 32] create()를 이용한 문서 색인 시 중복 오류 메시지

따라서 오류가 발생하였을 경우 색인을 하지 않고 다음 포스트의 데이터를 색인할 수 있도록 예외처리를 설정하였다. Elasticsearch-py에서 발생하는 오류는 elasticsearch 클래스 안에 정의되어 있고, 문서 중복 시 발생하는 오류인 TransportError를 검출하도록 하였다. 수집된 데이터가 Elasticsearch에 저장되었는지 확인하는 방법으로 Elasticsearch의 cat API를 사용하여 blogs 인덱스의 개수를 확인하였다. [그림 33]은 모든 인덱스의 정보와 문서 개수를 확인한 모습이다.



[그림 33] 모든 인덱스의 정보와 색인된 문서의 개수 확인

4.1.3 데이터 처리

Elasticsearch의 percolator 기능을 사용하기 위해 3.3절에서 추가한 데이터 구조를 바탕으로 blogs 인덱스에 queries 타입을 추가한다.

```
{
  "blogs" : {
    "mappings" : {
      "queries" : {
        "properties" : {
          "query" : {
            "type" : "percolator"
          }
        }
      }
    }
  }
}
```

[그림 34] percolator를 위해 blogs 인덱스에 추가한 queries 타입

다음으로 추가한 queries 타입에 따라 질의를 등록한다. 본 논문에서는 블로그 포스트의 내용에서 지역 이름을 검출하기 위해 행정구역 분류를 기준으로 전국의 시/군을 조사하였고, 모든 지역을 질의로 등록하기 위해 질의 목록을 작성하였다. 질의 목록의 query 필드에 term 질의를 등록한다. 이는 content 필드에 해당 지역 이름이 있으면 질의의 ID를 응답한다. 질의 목록을 이용해 질의를 등록하기 위해 Elasticsearch의 bulk API를 사용하였다. [그림 35]은 bulk API를 사용해 질의를 등록하기 위한 모습이다. 또한, 질의의 등록을 확인하기 위해 임의의 데이터를 구성하여 등록한 질의에 percolate를 실행한 결과는 [그림 36]와 같다.

```
{ "index": {"_index": "blogs", "_type": "queries", "_id": "seoul"} }
{ "query": {"term": {"content": "서울"}} }
{ "index": {"_index": "blogs", "_type": "queries", "_id": "busan"} }
{ "query": {"term": {"content": "부산"}} }
{ "index": {"_index": "blogs", "_type": "queries", "_id": "deagu"} }
{ "query": {"term": {"content": "대구"}} }
{ "index": {"_index": "blogs", "_type": "queries", "_id": "incheon"} }
{ "query": {"term": {"content": "인천"}} }
{ "index": {"_index": "blogs", "_type": "queries", "_id": "gwangju"} }
{ "query": {"term": {"content": "광주"}} }
{ "index": {"_index": "blogs", "_type": "queries", "_id": "deajeon"} }
{ "query": {"term": {"content": "대전"}} }
{ "index": {"_index": "blogs", "_type": "queries", "_id": "ulsan"} }
{ "query": {"term": {"content": "울산"}} }
{ "index": {"_index": "blogs", "_type": "queries", "_id": "sejong"} }
{ "query": {"term": {"content": "세종"}} }
...
```

[그림 35] bulk API를 사용해 질의를 등록하기 위한 질의 목록

```
es@esServer:~$ curl -XGET 'http://localhost:9200/blogs/queries/_percolate?pre
tty' -d '{
"doc": {
  "content": "서울 대공원 나들이를 가고싶다."
}
}'
{
  "took" : 3,
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "total" : 1,
  "matches" : [
    {
      "_index" : "blogs",
      "_id" : "seoul"
    }
  ]
}
```

[그림 36] 등록한 질의를 확인하기 위한 percolate 실행과 응답

percolator 기능을 크롤러에 적용하여 데이터의 수집과 처리를 함께 구현하였다. Elasticsearch-py에서 percolator 기능을 사용하기 위해서 perform_request() 메서드를 사용하였다. 이 메서드는 Elasticsearch-py 라이브러리가 지원하지 않는 API에 대해서 구현할 수 있도록 elasticsearch에게 REST API를 이용해 요청을 전달한다. [그림 37]은 perform_request()를 사용하여 percolator를 구현한 크롤러의 소스 코드 일부이다.

```
response = es.transport.perform_request(
                                method='GET',
                                url='/blogs/queries/_percolate',
                                body={ "doc": { "content" : data['content'] } })
```

[그림 37] perform_request()를 사용해 percolator를 구현한 소스 코드 일부

크롤러에 구현한 percolator는 포스트 내용에 포함된 여행 지역들의 질의 목록을 응답한다. 질의 목록에서 각 질의의 ID는 areas 인덱스에 저장된 문서의 ID와 같게 설정하였고, 따라서 count 필드의 값을 증가시킬 areas 인덱스에 색인된 문서의 ID는 질의의 ID를 사용하였고 update() 메서드를 이용하여 구현하였다. [그림 38]은 해당 내용의 구현한 소스 코드 일부이다.

```
    for ids in response['matches']:
            es.update(index='areas', doc_type='city', id=ids['_id'],
                    body={ "script": "ctx._source.count += 1" })
```

[그림 38] 여행 지역 언급횟수를 증가시키는 update() 구현 소스 코드 일부

데이터의 수집과 처리를 구현한 크롤러를 반복으로 예약 실행하기 위하여 crontab 기능을 이용하였다. crontab은 리눅스의 프로세스 예약 데몬으로 등록해 놓은 작업 명령을 설정한 시간에 반복 실행한다. 예약 작업을 작성한 내용은 다음 [그림 39]와 같다.

```
SHELL=/bin/bash
PATH=/usr/lib/python2.7:/usr/lib/python2.7/plat-x86_64-linux-gnu:/usr/lib/
python2.7/lib-tk:/usr/lib/python2.7/lib-old:/usr/lib/python2.7/dist-packag
es
HOME=/home/es/crawl

0 * * * * export DISPLAY=:0 && /usr/bin/python /home/es/crawl/scrap.py >
```

[그림 39] 예약 작업을 작성한 crontab의 내용

SHELL에 사용할 쉘을 지정하고 PATH에 sys.path로 확인한 파이썬과 관련된 모든 환경변수를 등록하였다. HOME에는 crontab에서 명령어를 실행할 디렉터리를 지정하였다. 지정하지 않으면 기본값으로 /etc/passwd에 설정된 사용자 홈 디렉터리로 설정된다. 마지막 줄에 crontab의 등록형식을 따르는 예약 작업 명령을 설정한다. crontab에 기술한 작업 명령은 1시간마다 구현한 크롤러를 실행하겠다는 의미이다. export DISPLAY=:0는 Selenium의 webdriver를 사용하기 위해 설정하였다.

4.1.4 데이터 표현

웹 서비스를 제공하기 위해 Ruby on Rails(이하 “레일즈”)를 기반으로 웹 애플리케이션을 제작하였다. 레일즈 애플리케이션을 생성하면 다음 [그림 40]과 같은 구조를 확인할 수 있다.

```
Gemfile      README.md  app/  config/    db/   log/          public/  tmp/
Gemfile.lock Rakefile   bin/  config.ru  lib/  package.json  test/    vendor/
```

[그림 40] 레일즈 애플리케이션의 구조

레일즈 애플리케이션에서 app 디렉터리를 확인하면 models, controllers, views 디렉터리를 확인할 수 있다. 레일즈는 해당 디렉토리들을 사용해 MVC 패

턴의 구현을 제공한다. 각각의 컴포넌트는 각각의 디렉토리에 생성되고 레일즈는 애플리케이션의 동작에서 자동으로 컴포넌트간의 메시지를 처리한다. 또한, 레일즈는 generator 명령어를 통해 각각의 컴포넌트의 기본 기능을 적용한 클래스 파일을 쉽게 생성할 수 있도록 제공한다. 이를 통해 레일즈는 MVC 패턴을 강력하게 제공하는 프레임워크임을 확인할 수 있다.

모델 컴포넌트는 Elasticsearch와 연동되어 Elasticsearch에 저장된 문서들을 조회하고 질의를 수행하는 역할을 한다. 모델 컴포넌트와 Elasticsearch와의 연동을 위해 elasticsearch-model이라는 Gemfile로 라이브러리를 애플리케이션에 포함한다. 따라서 elasticsearch-model 라이브러리를 애플리케이션에서 사용하겠다는 선언을 Gemfile에 gem 'elasticsearch-model'과 같이 명시하였다. 이후 라이브러리의 설치 및 애플리케이션에 추가하기 위하여 bundle install 명령을 실행하면 Gemfile에 명시되어 있는 라이브러리 목록을 읽어 들여 라이브러리의 설치와 애플리케이션에 추가를 진행한다.

```
Using elasticsearch-api 5.0.4
Using elasticsearch-transport 5.0.4
Using elasticsearch-model 5.0.1
```

[그림 41] bundle install 실행 중 메시지

본 논문에서 제안하는 웹 서비스는 blogs 인덱스와 연결되는 blog 모델 클래스와 areas 인덱스와 연결되는 area 모델 클래스를 구현하였다. 다음 [그림 42]는 blog 모델 클래스를 구현한 소스 코드이다.

```
require 'elasticsearch/model'

class Blog < ApplicationRecord
  include Elasticsearch::Model
    index_name 'blogs'
    document_type 'post'
    mappings dynamic: 'true' do
    end

  def as_indexed_json(options={})
    {
      "title"    => _source.title,
      "img"      => _source.img,
      "desc"     => _source.desc,
      "url"      => _source.url,
      "date"     => _source.date,
      "content"  => _source.content,
      "author"   => _source.author
    }
  end
end
Elasticsearch::Model.client = Elasticsearch::Client
.new host: '192.168.0.7:9200', log: true
```

[그림 42] blog 모델 클래스의 소스 코드

require를 이용해 elaticsearch-model 라이브러리를 모델 클래스에 추가하였다. 이후 include를 통해 라이브러리의 Elasticsearch::Model 모듈을 클래스의 인스턴스로 추가하였다. 모듈을 포함하면 해당 클래스가 모듈의 클래스, 메서드를 사용할 수 있다. Elasticsearch::Model 모듈을 추가하고 생성하면서 모델과 연결할 Elasticsearch의 인덱스와 타입을 지정하였다. 명시적으로 지정하지 않을 때는 기본값으로 모델 클래스의 이름으로 인덱스와 타입을 연결한다. mappings는 데이터 구조를 발견할 때의 처리방법을 명시하는데 dynamic 속성으로 방법을 지정한다. dynamic 속성은 세 개의 값을 가질 수 있다. 첫째, true는 새로운 필드를

발견하면 모델 클래스에 추가한다. 둘째, false는 새로운 필드를 발견하면 추가하지 않는다. 따라서 명시적으로 필드를 추가해야 하는 작업이 필요하다. 셋째, strict는 새로운 필드를 발견하면 오류를 발생한다. 여기에서는 dynamic 속성을 true로 설정하였다. as_indexed_json() 메서드는 Elasticsearch 서버의 응답 결과를 루비의 해시 데이터 형식으로 변환한다. 응답 결과의 필드 중 _source 필드의 값을 사용하여 해시를 구성하도록 구현하였고 이를 컨트롤러와 뷰 컴포넌트가 사용할 때 기존 루비의 해시 형식의 데이터를 사용하는 것과 같이 Elasticsearch의 응답 결과를 편리하게 사용할 수 있다.

```
require 'elasticsearch/model'

class Area < ApplicationRecord
  include Elasticsearch::Model
  index_name 'areas'
  document_type 'city'
  mappings dynamic: 'true' do
  end

  def as_indexed_json(option={})
    {
        "province" => _source.province,
        "city"     => _source.city,
        "prov_pos" => _source.prov_pos,
        "city_pos" => _source.city_pos,
        "count"    => _source.count.to_s
    }
  end
end
```

[그림 43] area 모델 클래스의 소스 코드

마지막에 있는 Elasticsearch::Client 모듈을 통해 Elasticsearch 서버로 접속하여 애플리케이션이 실행되면 Elasticsearch와 연결을 갖도록 구성하였다. 기본 연결 설정은 localhost:9200으로 연결을 시도한다. 일반적으로 웹 서버와 데이터 서버는 분리되어 구축되기 때문에 localhost 주소를 사용할 수 없다.

Elasticsearch 서버와 연결을 위해 사용되는 클래스 모듈은 Elasticsearch::Client 이다. 이 객체는 새로 생성될 때 지정한 host 필드의 정보를 가지고 Elasticsearch 서버와 연결을 구성한다. elasticsearch-model 라이브러리는 Elasticsearch 서버와 연결하는 다양한 방법을 제공한다. 하나는 모델 클래스에 직접 Elasticsearch::Client 객체를 생성하여 서버와 연결하는 것이다. 다른 하나는 Elasticsearch::Model 모듈의 client 필드에 Elasticsearch::Client 객체를 저장하여 서버와 연결을 구성하는 것이다. 이 중 Elasticsearch::Model 모듈을 사용해 서버와 연결을 구성한 이유는 Elasticsearch::Model 모듈에 Elasticsearch::Client 객체를 생성하여 서버와 연결을 구성한 객체를 저장하면 Elasticsearch::Model을 사용하는 모든 모듈에 같은 설정을 적용할 수 있기 때문이다. areas 인덱스와 연결되는 area 모델 클래스도 blog 모델 클래스와 같은 설정을 하였다. 이렇게 설정한 모델 컴포넌트들을 rake db:migrate 명령어를 통해 애플리케이션에 실제 적용하였다.

다음으로 컨트롤러 컴포넌트를 구현하였다. 본 논문에서 구현한 웹 서비스는 1개의 컨트롤러 클래스를 가진다. 컨트롤러 클래스는 사용자 인터페이스로부터 전달된 요청을 처리하고 응답을 제공하기 위하여 사용자 요청에 따른 메서드 동작 방식을 가진다. 따라서 요청 URL과 컨트롤러 클래스의 메서드를 매핑하는 라우팅을 설정해야 한다. 라우팅의 설정은 config 디렉터리의 routes.rb 파일에 기술하며, 레일즈 애플리케이션은 routes.rb 파일의 내용을 기반으로 요청을 실행할 컨트롤러 클래스와 메서드를 찾아 전달한다. 다시 말하면 3.3절의 서비스 워크플로우에서 언급한 디스패처의 기능을 수행하기 위한 참고자료가 라우팅 정보이다. 웹 서비스를 위해 적용한 라우팅 정보는 다음 [그림 44]와 같다.

```
Rails.application.routes.draw do

  root 'test#index'
  get '/province' => 'test#province'
  get '/city' => 'test#city'
  get '/search' => 'test#search'
  get '/search/:key(.:format)' => 'test#search'
end
```

[그림 44] 레일즈 애플리케이션의 routes.rb 파일의 내용

routes.rb 파일은 특별한 설정이 없이 [METHOD] [CONTROLLER/URL]을 기술하면 해당 컨트롤러 클래스의 URL명과 같은 이름을 가진 컨트롤러 메서드로 지정된다. [그림 46]과 같이 [URL] => [CONTROLLER]#[ACTION] 형식을 사용하여 라우팅을 설정하면 URL과 컨트롤러 메서드의 이름을 다르게 사용할 수 있다. 마지막에 있는 URL의 형식을 보면 '/search/:key(.:format)'으로 되어 있다. 이는 RESTful 인터페이스를 구현하는 라우팅 형식이다. 본 논문에서 구현한 웹 서비스는 search URI에 대해 REST API를 구현하였다. /search URI를 기술하고 :key로 요청 데이터를 받아들인다. (.:format)은 요청 데이터의 형식을 자유롭게 받는다는 의미이다. 다음 [그림 45]는 rake routes로 설정한 라우팅 정보를 확인한 모습이다.

```
web@webServer:~/webProjects/origin$ rake routes
  Prefix Verb URI Pattern             Controller#Action
    root GET  /                       test#index
province GET  /province(.:format)     test#province
    city GET  /city(.:format)         test#city
  search GET  /search(.:format)       test#search
         GET  /search/:key(.:format)_test#search
```

[그림 45] 명령 프롬프트 환경에서 확인한 라우팅 정보

컨트롤러 클래스는 설정한 라우팅 정보에 따라 요청을 처리하기 위해 4개의 메서드를 가진다. 각 메서드에 대한 간단한 설명은 다음과 같다.

- index: 사용자가 '/'(Root URL)을 요청했을 때, 즉 처음 웹 서비스에 접근하였을 때의 요청을 처리한다.
- province: 웹 페이지가 렌더링 될 때 Elasticsearch의 aggregation 기능을 이용해 시/군의 언급횟수를 도 단위로 집계하여 제공한다.
- city: 1번 이상 언급된 시/군 단위의 언급횟수를 제공한다.
- search: 사용자로부터 검색하고자 하는 지역 이름을 전달받아 Elasticsearch에 검색 질의를 전달하고 그 응답을 반환한다.

index 메서드는 위에서 구현한 blog 모델 클래스를 사용해 데이터에 접근하였다. elasticsearch-model의 API 문서를 참조하면 Elasticsearch로 전달하는 모든 요청은 질의 DSL 방식의 사용을 권고한다. 따라서 질의 DSL 방식을 사용하여 Elasticsearch에 검색 질의를 전달하였다. 다음 [그림 46]은 구현한 index 메서드 소스 코드이다.

```
def index
  res = Blog.search(
              query: { match_all: {} },
              sort: { "date": "desc" },
              size: 10
      )
  @result = res.results
end
```

[그림 46] index 메서드의 소스 코드

Elasticsearch::Model 모듈을 구현한 blog 모델 클래스를 사용하여 블로그 인덱스에 접근하였다. 모듈을 구현하였기 때문에 Elasticsearch::Model 모듈에서 제

공하는 모든 메서드를 사용할 수 있다. 여기서는 search() 메서드를 사용하여 Elasticsearch에게 검색 질의를 요청하였다. 요청한 검색 질의는 전체 문서를 date 필드의 내림차순 정렬한 후 상위 10개의 문서를 검색하는 내용이다. 요청에 대한 응답은 모델 클래스에서 정의한 as_indexed_json() 메서드로 인해 실제 문서의 데이터가 저장된 _source 필드의 내용이 해시 형식으로 재구성되어 res에 해시 배열로 저장된다. 이 res변수의 내용을 레퍼런스 변수인 @result에 저장하도록 하였다. 레퍼런스 변수는 뷰 컴포넌트에게 응답 데이터를 갖는 객체를 제공하기 위한 변수이다.

```
def province
  res = Area.search(
            size: 0,
            aggs: {
              "provinces": {
                terms: {
                  field: "province.keyword",
                  order: { "sum_cnt": "desc" },
                  size: 20
                },
                aggs: {
                  "sum_cnt": {
                    sum: { "field": "count" }
                  },
                  "prov_pos": {
                    top_hits: {
                      size: 1,
                      _source: { includes: ["prov_pos"] }
                    }
                  }
                }
              }
            })
  render json: res.aggregations['provinces']['buckets']
end
```

[그림 47] province 메서드의 소스 코드

다음으로 province 메서드는 위의 [그림 47]과 같이 구현하였다. index와 같

이 검색 질의를 이용해 aggregation 기능을 구현하였다. province는 area 모델 클래스를 사용해 areas 인덱스에 접근하였다. index 메서드와 다르게 size를 0으로 함으로써 실제 문서의 내용은 응답받지 않도록 설정하였다. 질의 DSL은 areas 인덱스의 문서들을 province 필드로 그룹을 만들고, 임시로 생성한 sum_cnt 필드를 기준으로 내림차순 정렬을 실행하여 20개의 결과를 요청하였다. sum_cnt 필드를 위해 내부 집계를 사용하였다. sum_cnt 필드는 count 필드의 합을 저장하는데 이 때, 위에서 만든 그룹별로 합을 계산한다. 그리고 도 단위의 좌표를 결과에 포함하기 위해 top_hits 필드를 사용하여 _source 필드에 있는 prov_pos 필드를 결과에 추가하도록 하였다. 응답 데이터에서 메타 데이터를 제외한 실제 필요한 집계 결과 데이터는 res 객체의 aggregations 필드 안에 provinces 필드의 buckets 필드로 저장되어 있었다. 이를 컨트롤러 메서드에서 처리 결과를 출력하기 위한 다양한 응답 메서드 중 하나인 render 메서드를 사용하여 JSON 형식의 데이터로 사용자에게 응답하도록 하였다.

city 메서드는 province 메서드와 같이 area 모델 클래스를 사용해 areas 인덱스에 접근하였다. 해당 메서드는 집계를 사용하지 않고 필터 기능을 사용하여 검색 질의를 요청하였다. 필터 기능은 bool 필드를 사용하여 구현하였는데 bool 필드는 검색된 문서의 내용에 filter 필드를 이용해 특정 조건을 적용해 검색된 문서를 걸러내는 역할을 한다. 이 메서드에서 적용한 질의 DSL은 모든 문서에서 count 필드의 값이 0보다 큰 문서를 검색하도록 하였다. 이때, count 필드를 기준으로 내림차순 정렬하도록 하였고, areas 인덱스의 모든 문서의 개수가 161개이므로 필터링 된 모든 문서를 요청하기 위해 size는 200으로 설정하였다. 사용자 요청에 대한 응답은 province 메서드와 같이 JSON 형식의 데이터로 응답하였다. 다음 [그림 48]은 city 메서드를 구현한 소스 코드이다.

```
def city
  res = Area.search query: {
                        bool: {
                          must: {
                            match_all: {}
                          },
                          filter: {
                            range: {
                              "count": {
                                "gt": 0
                              }
                            }
                          }
                        }
                      },
                      sort: { "count": "desc" }, size: 200
  render json: res.results
end
```

[그림 48] city 메소드의 소스 코드

사용자의 검색 요청을 처리하기 위한 search 메서드는 blog 모델 클래스와 area 모델 클래스를 모두 사용한다. 요청 파라미터는 key라는 이름으로 전달되 params 객체로 읽어온 후 이를 이용하여 Elasticsearch에 검색 질의를 요청한다. blogs 인덱스에는 content 필드에 key가 포함되어 있으면 _score 필드를 기준으로 내림차순 정렬하여 응답하도록 질의 DSL을 구성한다. _score 필드는 질의의 내용과 문서가 일치한 연관 점수를 가지고 있다. 연관 점수는 0에서 1사이의 실수 값으로 표현되며 높을수록 연관이 깊다고 판단한다. 검색 결과는 상위 10개로 제한하여 결과적으로 요청 검색어와 가장 유사한 문서 10개를 요청한다. 여행 지역은 검색어가 포함된 시/군 이름에 대해 일치하는 문서를 요청하였다. 응답할 데이터는 blogs 인덱스와 areas 인덱스의 응답 데이터를 res 해시 객체에 저장하여 JSON 형식의 데이터로 응답하도록 하였다.

```
def search
  key = params[:key]

  resPost = Blog.search(
                    query: { match_phrase: { content: key } },
                    sort: { "_score": "desc" },
                    size: 10
            )
  resCity = Area.search query: { match: { city: { type: "phrase_prefix", query: key } } }

  res = {
    "posts" => resPost,
    "city" => resCity
  }
  render json: res
end
```

[그림 49] search 메서드의 소스 코드

뷰 컴포넌트는 내장 루비를 구현하는 HTML과 CSS, 자바스크립트를 이용해 하나의 웹 페이지를 구현한다. 본 논문에서 구현하는 웹 페이지의 리소스는 app/assets/ 디렉터리에 stylesheets/, javascripts/ 디렉터리에 위치하였다. 레일즈는 리소스를 웹 페이지에 쉽게 참조하기 위해 assets 파이프라인을 제공한다. 레일즈 4버전 이상부터는 Sprockets 라이브러리를 사용해서 구현 가능하다. 해당 라이브러리는 application.js, application.css를 각각 javascripts/와 stylesheets/ 디렉터리에 생성해 놓는다. 두 파일은 자바스크립트와 CSS를 애플리케이션에 포함하기 위한 지시자를 명시하는 매니페스트 파일이다. 매니페스트 파일은 주석으로 구성된다. application.js는 '//', application.css는 '/* ~ */로 구성되고, 설정 명령은 주석 안에 =<지시자> <매개변수>로 기술한다. 지시자 중 require_tree는 상대 경로 표현으로 리소스의 경로를 지정해야 한다. 상대 경로 표현을 사용할 때 시작이 되는 현재 디렉터리 경로는 각각 stylesheets/와 javascripts/ 디렉터리이다. CSS에서 폰트와 이미지 참조를 못 하는 오류가 발생 시 Sprocket의 헬퍼를 사용한다. 이 헬퍼는 CSS 파일에서만 사용할 수 있다. 폰트의 경우 font-path(), 이미지의 경우 image-path()와 같은 헬퍼로 참조 오류를 해결할 수 있다.

레일즈 애플리케이션을 생성하면 app/views/layouts/ 디렉터리를 제공하며 모든 뷰에 공통으로 적용 가능한 레이아웃을 제공하는 application.html.erb 템플릿 파일을 생성한다. 레일즈의 뷰를 구성하는 동작은 application.html.erb 템플릿을 기반으로 HTML 문서의 필요 리소스를 참조하고 <body> 태그에 있는 내장 루비 코드를 통해 개발자가 작성한 뷰를 가져와 렌더링한다.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Trip</title>
    <%= csrf_meta_tags %>

    <%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track': 'reload' %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

[그림 50] 기본으로 생성된 application.html.erb 템플릿 파일의 내용

Assets 파이프라인의 매니페스트 파일을 application.html.erb 파일에서 참조한다. 이를 통해 웹 페이지는 관련 리소스를 적재하고 렌더링할 수 있게 된다. [그림 50]에서 stylesheet_link_tag가 application.css 파일을 참조하고, javascript_include_tag가 application.js 파일을 참조한다. 두 매니페스트 파일에 명시된 리소스 경로를 기반으로 레일즈 애플리케이션은 리소스를 복제한 임시 파일을 public/ 디렉터리에 위치한다. 브라우저는 이 복제된 리소스를 요청하여 페이지 렌더링에 사용한다.

컨트롤러 컴포넌트를 생성하면 레일즈는 app/views/ 디렉터리에 컨트롤러 컴포넌트의 이름과 같은 디렉터리를 생성한다. 뷰를 위한 HTML 문서는 이 디렉터리 안에 위치하면 된다. 레일즈가 application.html.erb 템플릿을 사용해 HTML

의 기본 구조를 만들기 때문에 개발자는 <body> 태그에 표현할 내용만을 구현하면 된다. 뷰를 구현할 때 사용하는 확장자 .erb는 Embedded Ruby를 뜻하며 루비 컴파일러는 erb 파일의 루비 코드를 해석하여 HTML에서 사용할 수 있게 변환해준다. [그림 51]은 본 논문에서 구현한 application.html.erb 파일의 내용이다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Origin</title>
    <%= csrf_meta_tags %>

    <%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track': 'reload' %>
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700,800" rel="stylesheet">
  </head>
  <body>
    <%= yield %>
  </body>
  <!-- Google Map API -->
  <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDPICZjQ8FRUkDxEmlgLwITXnjiTjieDUw" async defer ></script>
  <%= javascript_include_tag 'application', 'data-turbolinks-track' => 'reload' %>
</html>
```

[그림 51] 구현한 웹 서비스의 application.html.erb 파일의 내용

본 논문의 웹 서비스는 구글 지도를 기반으로 여행 지역을 표현하기 위해 구글 지도 API를 사용한다. 하단의 주석으로 표현한 부분은 구글 지도 API를 신청하여 받은 API 키를 적용한 모습이다. 뷰를 개발할 때 HTML 문서 파일의 이름은 해당 디렉터리의 이름을 가진 컨트롤러의 메서드와 같아야 한다. 컨트롤러가 사용자의 요청을 처리하고 나면 render 메서드를 이용해 응답을 명시적으로 지정하지 않으면 기본으로 컨트롤러 클래스와 같은 이름을 가진 디렉터리에 요청을 처리한 메서드와 같은 이름의 HTML 문서를 응답한다. 상기 구현한 컨트롤러 클래스에서는 index 메서드만 render 메서드를 사용하지 않았기 때문에 뷰 페이지의 이름은 index.html.erb로 설정하였다. 컨트롤러 클래스의 다른 메서드는

뷰 페이지가 렌더링 되고 난 후 자바스크립트 코드를 통해 Ajax로 요청을 전달하기 때문에 응답을 JSON 형식의 데이터를 반환하는 render 메서드를 사용하였다.

```
<% @result.each do |row| %>
<div class="list-group-item">
  <a href="<%= row.url %>" target="_blank">
    <div class="list-header">
      <img class="img-reponsive img-circle" src="<%= row.img %>" alt="..." width=
"32px" height="32px">
      <span><%= row.author %></span>
    </div>
    <div class="list-body">
        <h4 class="heading"><strong><%= row.title %></strong></h4>
        <p class="desc"><%= row.desc[0,.80] %></p>
    </div>
  </a>
</div>
<hr style="color:lightgray"/>
<% end %>
```

[그림 52] 구현한 뷰 페이지 index.html.erb 파일의 소스 코드 중 일부

뷰 페이지 소스 코드에서 중요한 것은 다른 서버 페이지 언어들과 같이 HTML 코드에 네이티브 언어를 사용할 수 있다는 것이다. 루비 언어의 문법을 사용해 반복 작업을 수행하여 코드의 재사용을 통해 빠른 개발을 가능하게 한다. 또한 [그림 52]에서 컨트롤러 클래스의 index 메서드에서 정의한 레퍼런스 변수인 @result 변수를 직접 HTML 코드에서 사용하는 모습을 확인할 수 있다. 해당 코드에서는 @result에 저장된 해시 배열 데이터를 반복하여 img, author, title, desc 필드의 값을 가지고 HTML 문서를 구성하도록 하였다. [그림 52]의 코드는 최근 10개의 블로그 목록을 구성한다.

본 논문에서 구현한 웹 서비스는 상기 언급한 바와 같이 구글 지도를 기반으로 표현된다. 구글 지도를 구현한 자바스크립트가 참조되면 google.map 클래스를 사용할 수 있다. 다음 [그림 53]은 google.map 클래스를 이용해 구글 지도를 초기화하고 웹 페이지에 렌더링하는 소스 코드 일부이다.

```
function initMap() {
  map = new google.maps.Map(document.getElementById('map'), {
    center: {
      lat: 36.381404,//36.769876,
      lng: 128.004450//126.931744
    }, //페이지 로딩시의 위치, Geolocation 값
    zoom: 7, //지도 확대/축소 수준, 한스텝에 1씩, 1:세계, 5:대륙, 10:시/군/구, 15:도로,
 20:건물
    mapTypeControl: false,
    fullscreenControl: false,
    maxZoom: 10,
    minZoom: 7
  });

  //marker, marker 1로 차트 처럼 표시!
  google.maps.event.addListener(map, 'zoom_changed', function() {
    zoomLevel = map.getZoom();
    console.log(zoomLevel);
    switch( zoomLevel ){
      case 10:
        cityMarker(12, 12);
      case 9:
        cityMarker(9.5, 8.5);
        break;
      case 8:
        map.setCenter({lat:36.381404, lng:128.004450});
        provinceMarker(16, 4);
        break;
      case 7:
        map.setCenter({lat:36.381404, lng:128.004450});
        provinceMarker(12, 3);
        break;
    }
  });
}
```

[그림 53] 구글 지도를 생성하고 초기화 하는 자바스크립트 소스 코드 일부

google.maps.Map() 메서드를 사용해 구글 지도 객체를 생성한다. 첫 번째 인수는 구글 지도를 웹 페이지에 렌더링하기 위한 DOM 객체로 map ID를 가진 객체를 지정하였고, 두 번째 인수는 구글 지도의 초기화에 사용되는 다양한 설정을 가진 객체이다. 본 논문에서 구현한 구글 지도를 렌더링하는 자바스크립트는 <div#map> 태그에 구글 지도를 렌더링하고 처음 렌더링 시 구글 지도에 중심이 될 좌표를 전국의 중심 좌표로 지정하였다. 그리고 전국의 모습을 보여주기 위하여 7 수준의 확대/축소 값을 설정하였다. 최대, 최소 확대/축소는 최대 10, 최소

7 수준으로 지정하였고, 확대/축소 이벤트가 발생하면 지도 위에 마커를 새로 생성하는 이벤트를 등록하였다.

```
function provinceMarker(markSize, fontSize){
  jQuery.ajax({
    type: "GET",
    url: "/province",
    success: function(data){
      setMapOnAll(null);
      markers = [];
      for(var i = 0; i < data.length; i++){
        var pos = data[i].prov_pos.hits.hits[0]._source.prov_pos;
        areaLat = pos.lat;
        areaLng = pos.lon;
        var tmpMark = new google.maps.Marker({
          position: {lat: areaLat, lng: areaLng},
          icon: {
            path: google.maps.SymbolPath.CIRCLE,
            scale: Math.log(data[i].sum_cnt.value) * markSize,
            fillColor: getRandomColor(),
            fillOpacity: .75,
            strokeColor: 'white',
            strokeWeight: .5
          },
          label: {
            color: 'white',
            fontSize: (Math.log(data[i].sum_cnt.value)*fontSize) + 'pt',
            fontWeight: 'bold',
            text: data[i].key+"("+data[i].sum_cnt.value+")"
          },
          map:map
        });
        tmpMark.addListener('click', function() {
          pos = this.getPosition();
          map.setCenter(pos);
          map.setZoom(9);
        });
        markers.push(tmpMark);
      }
    }
  });
}
```

[그림 54] 지도에 마커를 표시하는 소스코드 중 일부

여행 지역 언급 횟수는 지도에 표현하는 마커를 응용하여 구현하였다. Ajax(Asynchronous JavaScript and XML)을 이용해 웹 애플리케이션에게 요청을 보내도록 구현하였다. [그림 54]는 도 단위의 정보를 요청해 응답받은 데이터

를 기반으로 지도에 마커를 구성하는 소스 코드이다. 컨트롤러 클래스에서 정의한 도 단위의 정보 요청을 처리하는 province 메서드를 호출하기 위해 /province URL에 요청을 전달한다. 성공적인 응답이 전달되면 지도에 구성했던 모든 마커를 지우고 다시 마커를 구현하였다. 마커를 클릭하였을 때 해당 위치를 지도의 중심으로 이동하고 지도를 확대하여 시/군 단위의 정보를 표현할 수 있도록 click 이벤트를 등록하였다. 지도 마커 클릭, 검색 기능과 지도 확대/축소에 대한 기능도 해당 구현 기능과 유사하게 구현하였다.

4.2 서비스 테스트

4.2.1 데이터 수집 로그 확인

크롤러가 데이터의 수집을 성공적으로 실행하고 있는지 확인하였다. 먼저, 크롤러가 기록하는 데이터 수집 로그를 확인한 모습이 [그림 55]이다.

```
scrap.log                  scrap_20171105050001.log  scrap_20171106092547.log
scrap_20171104020001.log   scrap_20171105060001.log  scrap_20171106092602.log
scrap_20171104030001.log   scrap_20171105070002.log  scrap_20171106092703.log
scrap_20171104040001.log   scrap_20171105080003.log  scrap_20171106092714.log
scrap_20171104050001.log   scrap_20171105090002.log  scrap_20171106093241.log
scrap_20171104060001.log   scrap_20171105100002.log  scrap_20171106100001.log
scrap_20171104070001.log   scrap_20171105110004.log  scrap_20171106110002.log
scrap_20171104080001.log   scrap_20171105120003.log  scrap_20171106120001.log
scrap_20171104090001.log   scrap_20171105130002.log  scrap_20171106130001.log
scrap_20171104100001.log   scrap_20171105140001.log  scrap_20171106140002.log
scrap_20171104110001.log   scrap_20171105150002.log  scrap_20171106150001.log
```

[그림 55] 크롤러의 수집 로그 목록

크롤러는 1시간마다 반복 실행되면서 [그림 55]와 같은 로그를 남긴다. 로그의 내용은 수집한 블로그 포스트의 ID와 중복 여부, 색인 성공 여부, 색인한 총

포스트의 개수 등을 기록한다. scrap.log에는 크롤러가 예기치 못한 오류를 발생하였을 때 해당 오류를 기록한다. 오류가 발생하지 않으면 scrap.log는 데이터를 갖지 않는다.

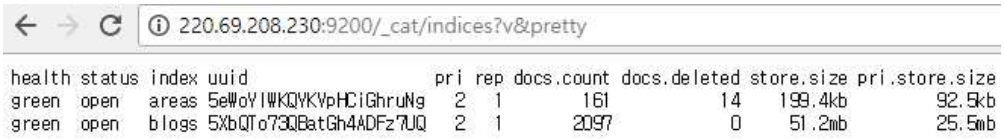
```
[2017-11-07 06:01:20] Complate to parse - sweetpurin_221133998150
[2017-11-07 06:01:20] Complate to percolate - sweetpurin_221133998150
[2017-11-07 06:01:20] Complate to parse - 9piga10_221133969803
[2017-11-07 06:01:21] Complate to percolate - 9piga10_221133969803
[2017-11-07 06:01:21] Complate to parse - choluck01_221133729056
[2017-11-07 06:01:21] Complate to percolate - choluck01_221133729056
[2017-11-07 06:01:21] Click the next
[2017-11-07 06:01:21] Complate stored which 36 posts scrapped
[2017-11-07 06:01:21] Scrapping end, Process time: 79.5105638504s
```

[그림 56] 크롤러의 수집 로그 중 일부

수집 로그 목록을 확인한 결과 구현한 예약 작업에 따라 매시간 0분에 실행되는 것을 확인할 수 있었다. 수집 로그의 내용을 확인한 결과 중복되는 포스트를 제외하고 평균 35개의 데이터를 수집하는 것을 확인하였다. 구현한 크롤러는 시간당 200개의 포스트를 수집하도록 구현하였으나, 네이버 블로그의 국내여행 관련 내용의 갱신 속도로 인해 중복된 포스트가 많이 발생하고 그로 인해 수집 성능이 떨어진 것으로 분석한다.

4.2.2 Elasticsearch 데이터 확인

Elasticsearch에 저장된 수집 데이터를 확인하기 위해 우선 블로그 인덱스에 색인 된 문서의 개수를 확인하였다. Elasticsearch의 cat API를 사용하여 블로그 인덱스에 색인 된 문서의 개수를 확인한 모습이 [그림 57]이다.



[그림 57] 블로그 인덱스의 색인 된 문서 개수

11월 07일 오전 7시를 기준으로 확인한 결과 총 2,097개의 문서가 색인되어 있는 것을 확인할 수 있었다. 색인 된 문서 중 161개는 percolator를 위해 등록한 질의 문서이므로 확인 시점에 수집한 데이터의 개수는 총 1,936개이다. 색인 된 문서를 확인하기 위해 Elasticsearch 서버에 search API를 실행하였다. 특별한 질의를 지정하지 않았기 때문에 모든 문서를 응답하는 것을 확인할 수 있다. 다음 [그림 58]은 search API를 실행한 결과이다.


```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  },
  "hits" : {
    "total" : 1936,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "blogs",
        "_type" : "post",
        "_id" : "grant2_221132061719",
        "_score" : 1.0,
        "_source" : {
          "img" :
"http://blogpfthumb.phinf.naver.net/MjAxNzEwMTdfNTcg/MDAxNTA4MjQ1
BdjONeEFnne_A4g.JPEG.grant2/profileImage.jpg",
          "title" : "남양주 가볼만한곳 수종사와 운길산, 묵언으로
          "url" : "http://blog.naver.com/grant2/221132061719",
          "author" : "그랜트",
          "content" : "남양주 가볼만한곳, 10월 22일(일)의 수종사
```


[그림 58] 블로그 인덱스에 search API를 실행한 결과

실행한 결과 내용 중 hits 필드의 total 필드에서 검색 질의를 통해 선택된 문서의 개수를 확인할 수 있다. blogs 인덱스의 색인 된 문서 개수와 같은 1,936 개를 응답한 것을 확인할 수 있었다.

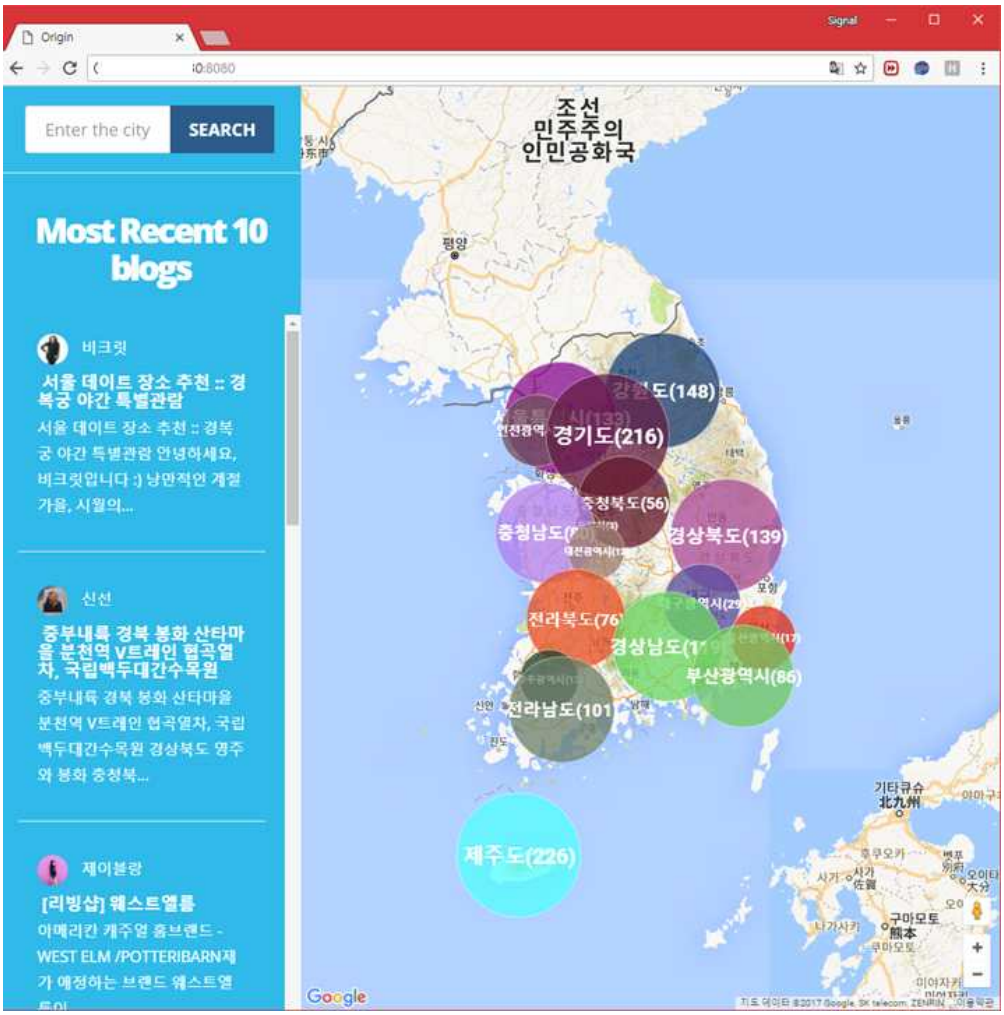
4.2.3 웹 서비스 동작 테스트

웹 서비스를 제공하기 위하여 레일즈 프레임워크에 포함된 WEBrick 서버를 이용해 웹 서버를 실행한다. 레일즈 애플리케이션의 서버를 동작시키는 명령은 rails server이고 -p 옵션을 통해 포트 번호를 지정하거나 -d 옵션을 통해 웹 서버를 데몬으로 실행시킬 수 있다. [그림 59]는 레일즈 서버를 실행한 모습이다.

```
web@webServer:~/webProjects/origin$ rails server
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.10.0 (ruby 2.3.3-p222), codename: Russell's Teapot
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

[그림 59] 레일즈 애플리케이션의 웹 서버 실행 모습

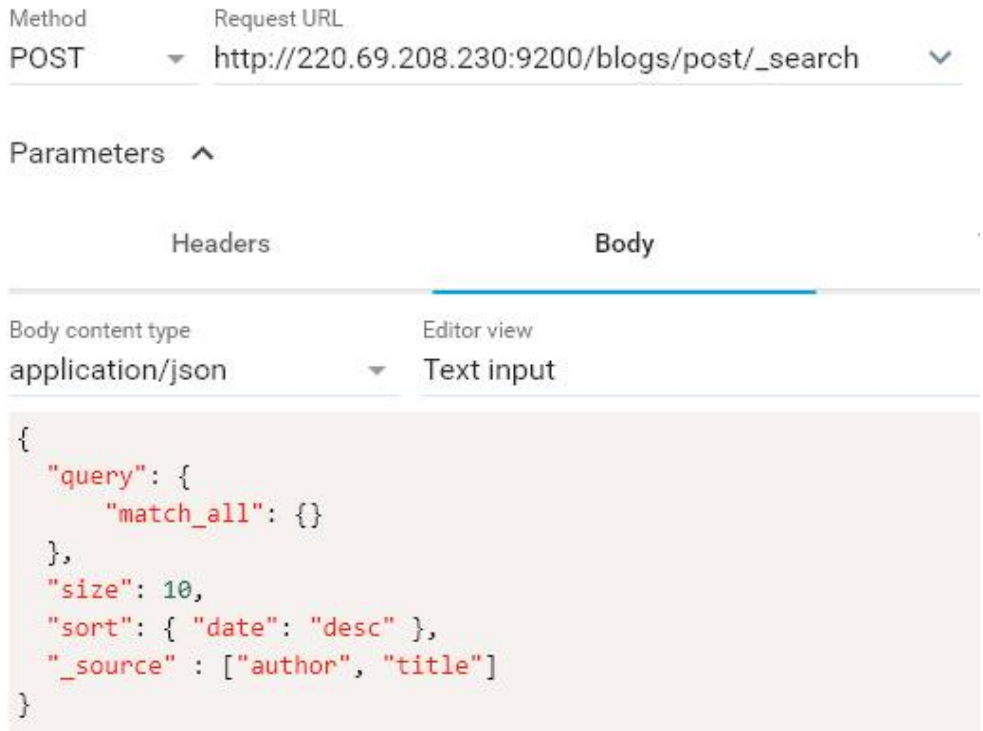
실행 로그 중 Environment 부분은 현재 웹 애플리케이션의 상태를 표현한다. 개발 환경에서 테스트를 위해 웹 서버를 동작하였으므로 Environment는 development를 나타내는 것을 확인하였다. Listening on tcp://0.0.0.0:3000은 웹 서버의 호스트 주소와 포트 번호를 나타낸다. 다음 [그림 60]은 크롬 웹 브라우저를 이용해 웹 서비스에 접근한 모습이다.



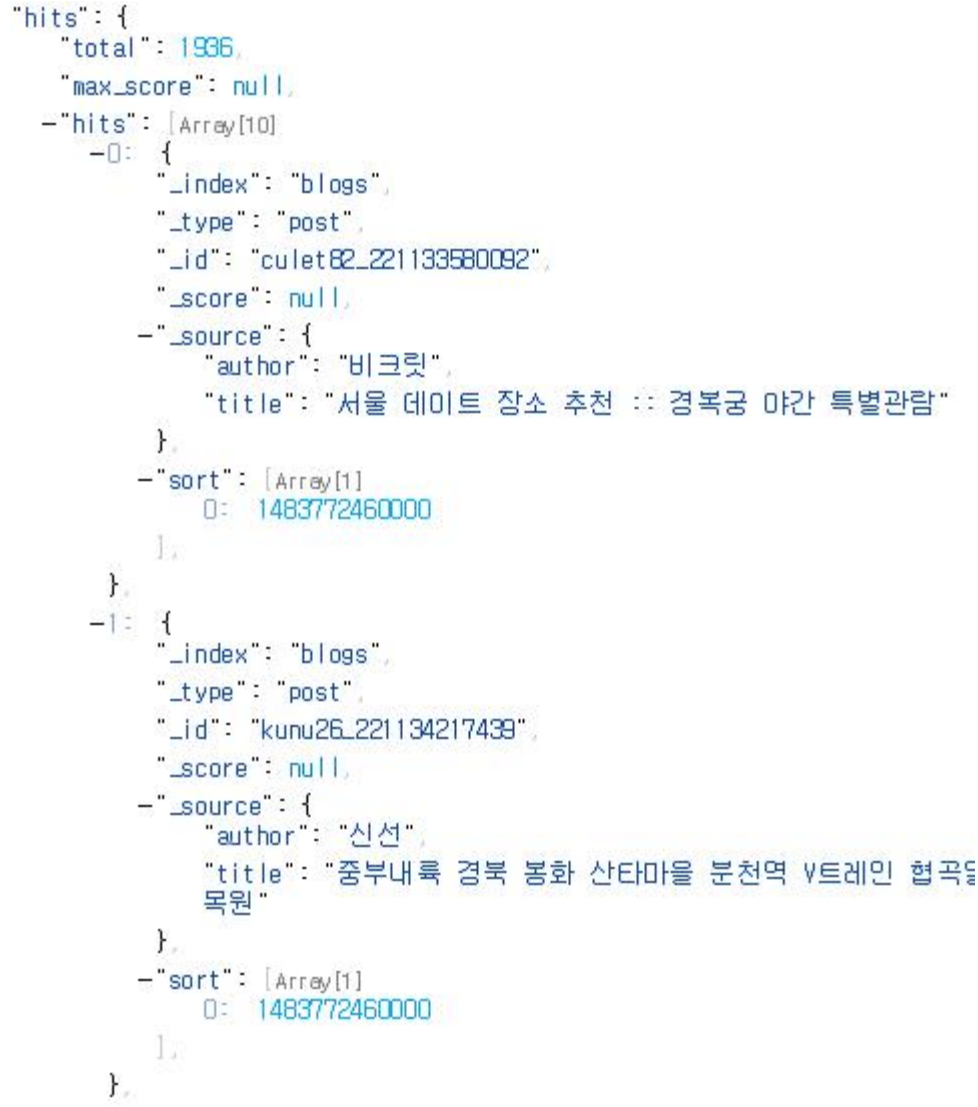
[그림 60] 크롬 웹 브라우저로 접속한 웹 서비스의 모습

접속한 index 페이지의 결과가 제대로 출력된 것인지 확인하기 위해 Elaticsearch 서버에 4.1.4절에서 구현한 컨트롤러 클래스에 적용한 질의 DSL를 search API를 사용해 그 응답을 확인하였다.

API의 실행은 크롬 웹 확장 프로그램인 Advenced REST Client를 사용하였다. 표준 REST 아키텍처는 GET 메소드에 요청 몸체를 허용하지 않기 때문에 POST 메소드를 사용하여 검색 질의를 요청하였다. [그림 61]은 Elasticsearch에 search API를 실행한 모습이고 [그림 62]는 search API를 실행 결과 모습이다.



[그림 61] 블로그 인덱스에 search API 실행



[그림 62] search API 실행 결과

[그림 60]의 웹 서비스 실행결과에서 좌측에 있는 블로그 포스트 목록과 [그림 62]의 응답 결과가 일치함을 확인하였다. 이는 사용자 요청을 전달받아 컨트롤러 컴포넌트가 모델 컴포넌트에 질의를 전달한 후 모델 컴포넌트가 Elasticsearch 서버에게 질의를 요청하고 그 응답을 반환하여 사용자에게 제대로 응답하였다는 의미이다.

제 5 장 결론 및 향후 과제

웹 서비스는 네트워크에 접속을 통해 어디서든 접속할 수 있다는 장점과 기기에 종속되지 않는 독립적인 사용자 인터페이스를 제공하기 때문에 모니터링, 분석 등 다양한 분야에 사용된다. 또한, 웹 서비스는 서비스가 제공되는 특징 상 프레젠테이션 로직과 비즈니스 로직의 구별이 확실하고 지속적인 사용자 피드백에 따라 각 로직의 수정과 유지보수가 빠르게 이루어진다는 점에서 MVC 패턴은 웹 서비스에서 필수 불가결한 요소로 자리하게 되었다.

본 논문에서는 국내여행에 관련된 블로그 포스트를 수집하고 여행 지역의 언급횟수를 분석하여 통계를 제공하는 웹 서비스를 설계 및 구현하였다. Python을 기반으로 구현한 동적 웹 페이지 크롤러는 crontab으로 작성한 예약 작업에 맞춰 정상 동작하는 것을 확인하였다. Elasticsearch의 percolator를 사용한 데이터 처리 및 가공 작업을 통해 여행 지역의 언급횟수를 확인하였다. Ruby on Rails를 이용해 구현한 웹 서비스의 구조가 MVC 패턴을 따라 만들어지는 것을 확인할 수 있었고, 웹 브라우저를 통해 접속한 웹 페이지에서 외부 API인 구글 지도 API와 jQuery가 클라이언트 측에서도 정상 동작함을 확인하였다.

본 논문에서는 단일 웹 페이지를 이용해 웹 서비스를 구현하였다. MVC 패턴을 기반으로 웹 서비스를 제작하며 각각의 로직이 분리되었을 때의 장점을 느낄 수 있었고 보다 큰 규모의 웹 서비스에서 더 강한 능력을 발휘할 것이라 생각한다. 또한, Elasticsearch의 percolator를 사용한 단순 분류 작업을 구현했으나 향후 머신 러닝을 이용한 토픽 모델링, 감성 분석, 텍스트 마이닝 기법을 적용한다면 보다 신뢰도 높은 분류 서비스를 구축할 수 있을 것이라 판단한다.

감 사 의 글

늘 익숙했던 풍경을 뒤로 하고 이제 사회를 향해 발을 내딛어야 한다는 순간이 왔다는 것이 실감이 나질 않습니다.

본 논문을 적극 지도해주시며 항상 올바른 마음가짐과 열정을 가질 수 있게 해주신 이상정 교수님께 감사를 드립니다. 그리고 처음 프로그래밍 언어를 배우며 프로그램 개발에 흥미를 갖게 해주신 천인국 교수님, 어렵고 힘들지만 프로그램의 구조와 언어의 깊이에 대해 이해하게 해주신 하상호 교수님, 하드웨어의 기초와 전자회로에 기본에 대해 깨닫게 해주신 홍인식 교수님을 비롯하여 이해각 교수님, 남윤영 교수님에게도 깊은 감사를 표합니다. 학교에 복학하고 같은 관심사를 나누며 서로를 응원하던 진우와 성아 그리고 학우들에게 즐거운 추억과 저를 더 발전할 수 있게 해준 마음에 고마움을 전합니다. 4학년이 되어 들어왔지만 반겨주고 어렵지 않게 저를 대해준 영상처리 연구실의 식구들 운기, 명수, 혜린이, 승훈이, 은혜, 에녹이와 그리고 영재에게도 즐거운 학교생활을 만들어줘서 고맙다는 말을 전합니다. 2017년 학생회 활동을 하며 새로운 인연을 만든 우리 집행부 사람들 광전이, 석희, 준호, 동민이, 진혁이, 동근이, 재현이, 경준이, 지훈이, 승찬이, 영서, 유정이, 소미, 그리고 선희에게 부족한 부학회장이지만 잘 따라줘서 고맙고 학교생활을 하며 가장 즐거웠던 순간을 만들 수 있어서 또 고맙다는 말을 전합니다. 언급했던 모든 이들의 행복을 진심으로 바라며 계속해서 인연을 이어 나갔으면 좋겠습니다.

마지막으로 본 논문의 구현과 작성을 함께해준 명수에게 감사하고 수고했다고 전하며 저를 항상 믿고 응원해주신 부모님께 사랑한다는 말을 전합니다.

감 사 의 글

2012년 3월에 입학하고 어느덧 6년이라는 세월이 흘러 졸업을 앞두고 있습니다. 학교를 졸업하면 사회로 나아가는 환경의 변화에 걱정도 많이 되고 한편으로는 설렘도 있습니다. 학교생활을 한번 돌이켜 보니 원 없이 공부도 해봤고, 놀기도 해봤다고 생각했는데 항상 마무리에 아쉬움이 남는 점은 어쩔 수 없는 것 같습니다. 대학 생활 동안 행복했고 소중하고 감사한 사람들 덕분에 많이 배우고 성장할 수 있었습니다. 많은 분의 도움을 받아 졸업을 할 수 있게 되었습니다. 항상 열정으로 수업해주시고 가장 많은 상담을 받았던 천인국 교수님과 본 논문을 지도해주신 이상정 교수님, 그리고 홍인식 교수님, 이해각 교수님, 하상호 교수님께도 깊은 감사를 표합니다. 제가 막연히 공부를 하고자 문을 두드리고 들어갔던 영상 처리 연구실, 지금은 졸업했지만 연구실에 들어가서 만났던 현호 형, 승연이 형, 은지, 소은이와 연구실에 있는 맨날 괴롭혔던 승훈이, 게임 없으면 죽는 운기형, 공부 열심히 하는 은혜랑 승훈이가 놀리는 에녹이까지 다들 정말 즐겁게 연구실 생활을 할 수 있게 해주셔서 감사하고 다들 원하고자 하는 일을 이뤘으면 좋겠습니다. 그리고 이번 작품과 논문작성에서 제일 고생한 문성이 형에게 감사를 보냅니다. 입학해서 지금까지 많은 추억과 소중한 사람들이 되어버린 12학번 동기들 석희, 준호, 동민, 진혁, 동근, 그리고 정신 못 차린 김성훈, 커피 파트너 주환이 형 그동안의 시간과 추억을 모두 좋은 기억으로 담아 모두의 앞날에 좋은 일만 가득하길 바랍니다. 그리고 힘들 때 항상 의지가 되고 없어서는 안 될 존재가 되어버린 소중한 인연이 된 주이, 그리고 못난 아들 뒷바라지하느라 고생하신 엄마 아빠에게 진심으로 사랑하고 고맙다고 전하고 싶습니다.

참 고 문 헌

[1] 위키백과, "모델-뷰-컨트롤러",
https://ko.wikipedia.org/wiki/%EB%AA%A8%EB%8D%B8-%EB%B7%B0-%EC%BB%A8%ED%8A%B8%EB%A1%A4%EB%9F%AC

[2] 이선숙,“MVC 모델을 적용한 자동문제 출제 시스템 설계 및 구현”,성신여자대학교 대학원 석사학위논문, 2006

[3] 위키백과, "웹 애플리케이션 프레임워크",
https://ko.wikipedia.org/wiki/%EC%9B%B9_%EC%95%A0%ED%94%8C%EB%A6%AC%EC%BC%80%EC%9D%B4%EC%85%98_%ED%94%84%EB%A0%88%EC%9E%84%EC%9B%8C%ED%81%AC

[4] 위키백과, "웹 크롤러",
https://ko.wikipedia.org/wiki/%EC%9B%B9_%ED%81%AC%EB%A1%A4%EB%9F%AC

[5] 이선숙,“MVC 모델을 적용한 자동문제 출제 시스템 설계 및 구현”,성신여자대학교 대학원 석사학위논문,2006

[6] majid, http://www.majidkhosravi.com/xml-vs-json-android/

[7] Elasticsearch, https://www.elastic.co/kr/products/elasticsearch

[8] 위키백과, “jQuery”, https://ko.wikipedia.org/wiki/JQuery

[9] jQuery, “https://jquery.com”

[10] 라두 게오르게, 매튜 리 한만, 로이 루소, Elasticsearch In Action, 에이콘, 2016