웹 드라이브(e 드라이브)를 통한 문서 통합 본문 검색 어플리케이션 제작

2015104187 안형욱
2016104146 이승윤

요 약

조직의 보안 측면에서 문서중앙화 관련 관심이 높아지고 있다. 그런데 대량의 파일들을 중앙화하면 파일명, 폴더명과 같은 단순 메타정보 기반의 검색은 사용자가 원하는 정보를 빠르게 검색하는 것이 불가능하다. 또한, 파일 본문 내용의 검색이 불가하므로 문서 내 본문 내용을 추출할 수 있는 기술을 기반으로 추출 완료된 문서 내용을 효율적으로 검색할 수 있는 검색 엔진 인터페이스를 개발한다.

1. 서론

1.1. 연구배경

조직의 보안 측면에서 문서중앙화에 대한 관련 관심이 높아지고 있다. 회사측면에서 주요 문서를 중앙화하여 통합하면 자산화를 할수 있음과 동시에 분실 및 휴대성의 문제에서도 손쉽게 해결할 수 있는 방안이 된다. 네트워크 드라이브처럼 사용하기 때문에 사용의 편의성도 일부 보장될 수 있는 것이다. 하지만 여러 문서들을 중앙화함으로써 생기는 다양한 문제점도 존재한다. 그 중에 하나로 대량의 파일들을 중앙화하게 되면 파일명, 폴더명과 같은 단순 메타정보 기반의 검색은 사용자가 원하는 정보를 빠르게 검색하는 것이 불가능하다. 또한 주변에서 쉽게 볼 수 있는 검색엔진 시스템에서도 파일 본문 내용의 검색기능은 찾아볼 수 없다.

따라서 대량의 파일들을 중앙화하였을때 생기는 문제점을 해결함과 동시에, 사용자 편의성 개선능력도 향상시킬 수 있도록 하기 위해 본 프로젝트는 문서 내 본문 내용을 추출할 수 있는 기술을 기반으로 추출 완료된 문서 내용을 효율적으로 검색할 수 있는 검색 엔진을 개발하는 것을 목표로 한다.

기존의 파일시스템 검색의 경우를 비교 및 대조하였을때, 윈도우 운영체제의 경우 파일명에 대한 검색은 가능하지만 문서 내부의 문자를 검색해야할 때 문서 파일을 직접 열어 검색해야 하는 불편함을 찾아 볼 수 있었다. 본 프로젝트는 이러한 파일 시스템을 사용 하였을 때와 유사한 성격을 지니고 있기 때문에, 이를 참고하여 문서중앙화를 이용하는 사용자 측면에서의 불편함을 줄이기 위하여 스토리지 내에 공용으로 사용하는 사용자들에게 문서의 제목뿐만 아니라 본문 내용까지

검색하고, 그 내용을 미리볼 수 있도록 하여 문서 내용을 효율적으로 검색할 수 있는 사용자 인터페이스, 즉 UI 를 웹을 통해 제공하도록 할 것이다.

1.2. 연구목표

산업체에서 개발한 문서 본문 내용을 추출할 수 있는 기술을 활용하여 추출된 내용을 기준으로 저희는 추출된 내용을 DB 에 담아 원하는 기준에 따라 검색 결과를 보여줄 수 있는 ElasticSearch 오픈소스를 활용하여 검색 시 파일명이 아닌 본문 내용이 포함되도록 하는 인터페이스 환경을 구현할 것이다. 또한 이를 제작함으로써 대량의 파일들을 중앙화하였을 때 생기는 파일명, 폴더명과 같은 단순 메타정보 기반의 검색만의 한계점을 넘어 사용자가 원하는 정보를 빠르게 검색할 수 있도록 하는 것을 목표로 한다.

2. 관련연구

2.1. 오픈 소스 및 개발 프레임워크

2.1.1. ELK 스택 및 Nori 형태소 분석기

ELK stack 이란 Elasticsearch, Logstash, Kibana 의 3 가지 오픈 소스 프로젝트의 앞 글자를 따서 부르는 말이다. Elasticsearch 는 JSON 기반의 분산형 RESTful 검색 엔진으로 사용하기 쉽고, 확장 가능하며,유연한 것이 특징이다. Logstash 는 여러 소스에서 동시에 데이터를 수집하여 변환한 후 Elasticsearch 로 전송하는 서버 사이드 데이터 처리 파이프라인이다. 마지막으로 Kibana 는 사용자가 Elasticsearch 에서 차트와 그래프를 이용해 데이터를 시각화할 수 있도록 해준다. 이번 프로젝트에서 ELK stack 을 사용해 문서 검색 서비스를 사용하는 사용자가 검색한 검색 데이터들을 빠르게 제공하는 것과 로그 수집 및 분석이 가능하다.

이번 프로젝트의 문서의 내용에 한글이 포함될 수 있다. 한글은 형태의 변형이 매우 복잡한 언어이다. 특히 복합어, 합성어 등이 많아 하나의 단어도 여러 어간으로 분리해야 하는 경우가 많아 한글을 형태소 분석을 하려면 반드시 한글 형태소 사전이 필요하다. Elasticsearch 6.6 버전 부터 공식적으로 Nori(노리) 라고 하는 한글 형태소 분석기를 Elastic 사에서 공식적으로 개발해서 지원을 하기 시작했다. 아래는 일반 standard tokenizer 를 사용한 예와 nori_tokenizer 를 사용한 한글 형태소 분석의 예이다.

- request

standard 토크나이저로 "동해물과 백두산이" 문장 분석

```
GET _analyze
{
  "tokenizer": "standard",
  "text": [
    "동해물과 백두산이"
  ]
}
```

nori_tokenizer 토크나이저로 "동해물과 백두산이" 문장 분석

```
GET _analyze
{
  "tokenizer": "nori_tokenizer",
  "text": [
    "동해물과 백두산이"
  ]
}
```

- response

standard 토크나이저로 "동해물과 백두산이" 문장 분석 결과

```
{
  "tokens" : [
    {
      "token" : "동해물과",
      "start_offset" : 0,
      "end_offset" : 4,
      "type" : "<HANGUL>",
      "position" : 0
    },
    {
      "token" : "백두산이",
      "start_offset" : 5,
      "end_offset" : 9,
      "type" : "<HANGUL>",
      "position" : 1
    }
  ]
}
```

nori_tokenizer 토크나이저로 "동해물과 백두산이" 문장 분석 결과


```
{
  "tokens" : [
    {
      "token" : "동해",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "word",
      "position" : 0
    },
    {
      "token" : "물",
      "start_offset" : 2,
      "end_offset" : 3,
      "type" : "word",
      "position" : 1
    },
    {
      "token" : "과",
      "start_offset" : 3,
      "end_offset" : 4,
      "type" : "word",
      "position" : 2
    },
    {
      "token" : "백두",
      "start_offset" : 5,
      "end_offset" : 7,
      "type" : "word",
      "position" : 3
    },
    {
      "token" : "산",
      "start_offset" : 7,
      "end_offset" : 8,
      "type" : "word",
      "position" : 4
    },
    {
      "token" : "이",
      "start_offset" : 8,
      "end_offset" : 9,
      "type" : "word",
      "position" : 5
    }
  ]
}
```


Standard 토크나이저는 공백 외에 아무런 분리를 하지 못했지만 nori_tokenizer 는 한국어 사전 정보를 이용해 "token" : "동해", "token" : "산" 같은 단어을 분리 한 것을 확인할 수 있다. 이를 통해 검색시 효과적인 자동완성 추천 단어를 추가할 수 있고, 한글에서 '을/를/이/가'와 같은 조사를 검색어에서 검색할 때 무의미한 단어로 인지하여 해당 검색어를 필터링하는 효과를 기대할 수 있다.

2.1.2. React.js

React.js 는 페이스북이 공개하고 유지보수하는 사용자 인터페이스를 만들기 위한 JavaScript 라이브러리다. React.js 의 특징으로는 컴포넌트 단위로 UI 를 구성해 효율적인 코드 관리와 Virtual DOM 을 사용하여 VDOM 으로 먼저 페이지를 그린 후, 실제 DOM 과 비교하여 다른 부분이 있으면 그부분만 변경이 되어 DOM tree 를 reload 하기 위해 계속해서 DOM 을 추적하는 것을 자동으로 해준다는 특징이 있다. 이번 프로젝트에서는 React.js 와 함께 Redux, Mantine, Styled-Components, Axios 등의 라이브러리를 함께 사용하여 문서 검색을 위한 웹사이트 프론트엔드를 구현했다.

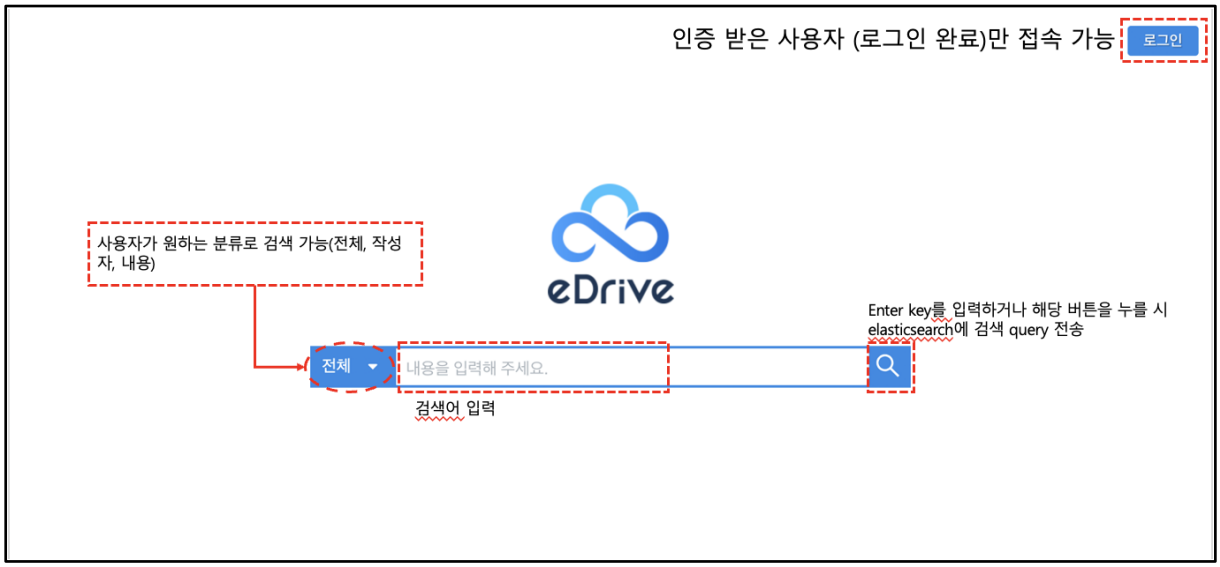
2.1.3. S3

S3 는 Amazon Simple Storage Service(Amazon S3)로 AWS 에서 제공하는 인터넷 스토리지 서비스다. 이 서비스는 개발자가 더 쉽게 웹 규모 컴퓨팅 작업을 수행할 수 있도록 설계되어 있다. 쉽게 버킷을 만들고 원하는 데이터(image, file, text 등)을 클라우드에 저장할 수 있다. 이번 프로젝트에서는 클라우드 환경에서 회사의 문서 데이터가 저장되어 있다고 가정한다. 그래서 S3 를 활용해 웹서비스에서 검색하고자 하는 파싱되기 전 hwp, docx, pdf 등의 확장자 파일을 스토리지에 저장하여 사용한다. 사용자는 웹사이트에서 원하는 키워드를 검색하여 찾고자 하는 문서를 찾을 수 있고, 웹사이트에서 제공된 결과 문서들 중에서 원하는 문서를 바로 다운로드할 수 있다. 이는 S3 에서 제공하는 public url 을 사용했다.

3. 프로젝트 내용

3.1. 시나리오

3.1.1. 홈페이지



[그림 1] 메인 화면

맨 처음 사이트에 접속하면 **[그림 1]**과 같은 모습을 확인할 수 있다. 메뉴 바를 통해 사용자가 원하는 분류에 따라 검색 할 수 있도록 지정할 수 있으며, 바로 우측에 있는 검색 바를 통해 검색어를 입력하고 검색 결과를 요청할 수 있다. 해당 검색 결과를 조회하기 위해선 우측 상단에 있는 로그인 버튼을 통해 로그인을 해야만 검색 결과 창을 확인할 수 있다.

3.1.2. 로그인 페이지

[그림 2] 로그인 화면

홈페이지의 로그인 버튼을 누르면 나오는 로그인 페이지이다. 해당 양식에 맞추어 작성 후, 로그인버튼을 누르면 홈페이지로 리다이렉트(redirect)되어 이동한다.

해당 로그인은 산업협력체의 회원 관리 폼 및 기능을 건네받아 구현할 예정이었으나, 회사 내부 문제로 인해 자료를 건네받지 못하여 완성하지 못한 상태이다.

따라서 현재 해당 폼에 대한 회원관리 기능은 각 기업의 회원관리에 적용시킬 계획이기 때문에 페이지의 인터페이스만 구현해놓은 상황이다.

3.1.3. 검색결과 페이지

[그림 3] 검색결과 페이지 1

[그림 4] 검색결과 페이지 2

홈페이지에 입력한 검색 양식에 따른 검색결과를 보여준다. 검색 결과창에 따른 각 기능은 다음과 같다.

- 로고 이미지를 클릭 시 홈페이지로 이동하기 기능
- 홈페이지에 입력한 검색어 유지 기능

- 카테고리별 검색 기준에 따른 검색 결과 출력 기능
- 정렬 기준에 따른 검색 결과 출력 기능
- 중복 키워드를 통한 검색 결과 출력 기능
- 파일 미리보기 기능
- 문서 내 이미지, 파일 경로, 파일 다운로드, 파일 본문 내용 미리보기 기능

3.2. 요구사항

3.2.1. 홈페이지

- 사용자로 하여금 직관적으로 기능을 이해할 수 있어야 한다.
- Elastic Search 를 이용하여 파싱된 데이터를 담은 결과를 검색어에 따라 분류하여 출력할 수 있도록 한다.
- 기존에 사용하였던 검색엔진 사이트와 비슷한 형태를 띄어 사용자로 하여금 친근하게 느낄 수 있도록 한다.
- 검색어 입력에 엔터키와 검색 버튼 모두를 사용할 수 있도록 한다.
- 검색어값이 없는(null) 키워드를 입력할 시, 안내문을 띄우고 이전 상태로 돌린다.

3.2.2. 로그인 페이지

- 로그인 시, 미리 ID 와 Password 양식을 placeholder 를 통해 안내하도록 한다.
- Password 를 입력시, 보안에 따라 화면에 드러내지 않고, *처리를 하여 시각적 암호화처리를 한다.
- 주어진 양식과 다른 형태의 양식을 작성할 경우, 경고표시를 띄워 안내한다.

3.2.3. 검색 결과 페이지

- 사용자로 하여금 별도의 가이드라인 없이 직관적으로 기능을 이해할 수 있어야 한다.
- Elastic Search 를 이용하여 파싱된 데이터를 담은 결과를 검색어에 따라 분류하여 출력할 수 있도록 한다.
- 파일 미리보기는 마우스를 hover 할 경우에만 출력할 수 있도록 한다.
- 파일 다운로드 기능은 검색 결과에나온 파일의 제목을 클릭할 경우 다운로드할 수 있도록 한다.
- 드롭다운 메뉴에 따라 작성자, 내용, 전체 기준에 따른 검색 결과를 출력 할 수 있도록 한다.
- 검색 결과 전체에 사용자가 원하고 편하게 볼수 있는 정보를 중심으로 출력 할 수 있도록 한다.

4. 프로젝트 구조

4.1. 시스템 구성도

[그림 5] 시스템 구성도

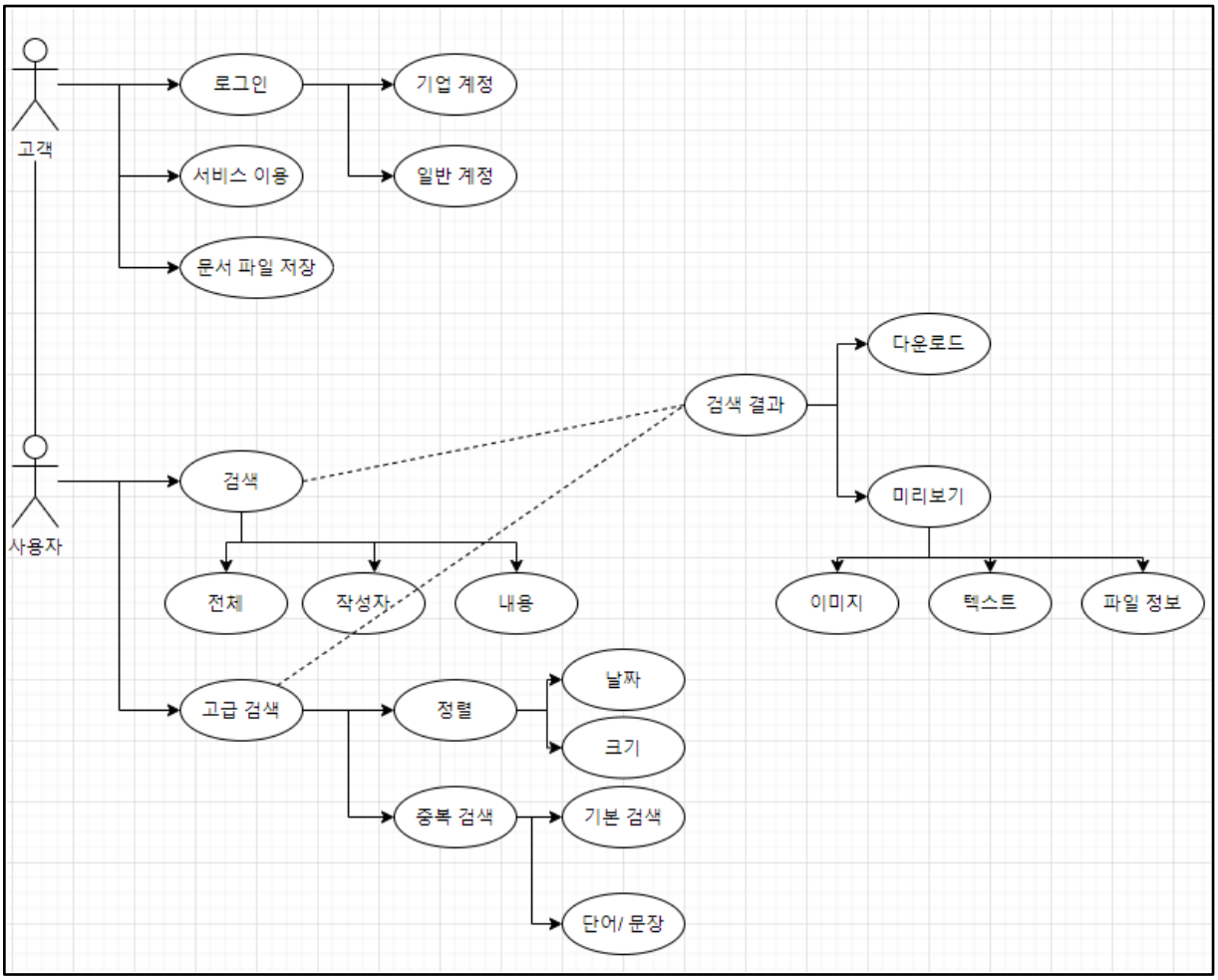
시스템 구성도는 [그림 5]과 같다. 사용자는 문서 저장소(S3)에 문서를 저장하는 것과 EC2 로 배포된 웹사이트에서 문서를 검색하는 기능을 이용할 수 있다.

S3 에 저장된 문서를 문서 파서로 파싱되어 나온 결과(현재 mock data 사용)를 elastic cloud 에 스키마 구조에 맞게 저장한다.

저장되어 나온 결과를 바탕으로 react 와 redux, mantine 을 통해 제작된 검색 인터페이스를 통해 사용자는 EC2 에서 배포된 웹사이트에서 검색을 할 경우 앞서 말한 작성된 프론트에서 request 를 처리하여 elastic cloud 에 원하는 결과값에 대한 request 를 요청하고 검색 결과를 받아 데이터를 가공하여 사용자에게 제공한다.

4.2. UML Diagram 을 통한 시스템 모델링

4.2.1. Use Case Diagram



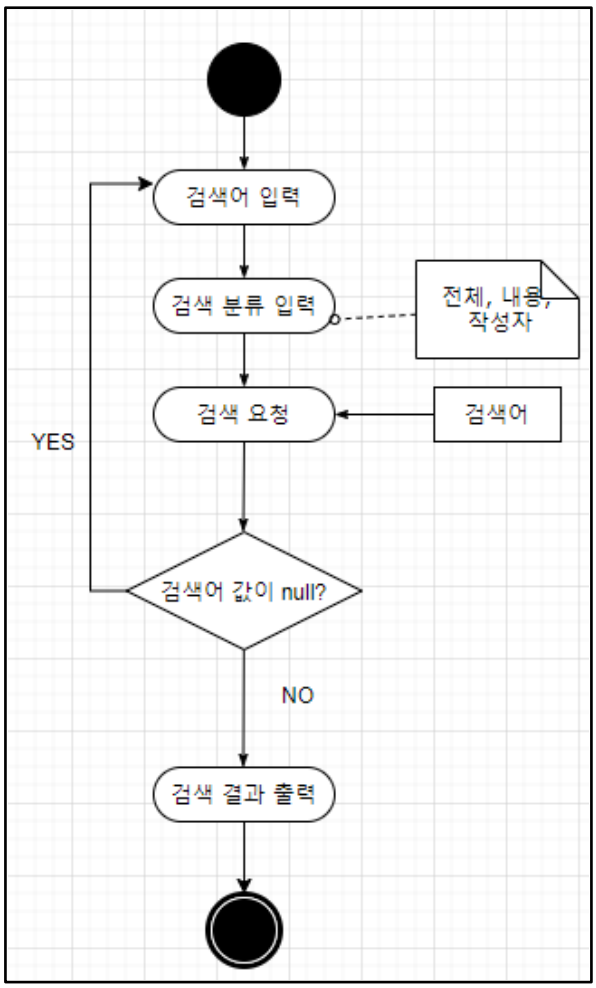
[그림 6] Use Case Diagram

Usecase 는 간단하다. 고객은 기존에 존재하는 기업 혹은 일반 계정을 통해 해당 문서 파일들이 저장되어있는 곳에 접속할 수 있으며, 계정에 연결되어있는 저장소를 기준으로 검색엔진 서비스를 이용할 수 있다. 그 서비스를 이용하는 사용자는 검색 기능을 통해 검색 결과를 확인 할 수 있다. 검색 기능은 크게 기본 검색 기능과 고급 검색 기능으로 나뉜다. 기본 검색은 전체, 작성자, 내용에 따라 필터링하여 검색을 할 수 있으며, 고급 검색의 경우 기본검색 기능을 통해 나온 검색결과에서 다시한번 필터링을 해주는 역할을 한다. 그 역할로 정렬, 중복 검색 등의 기능이 포함된다.

검색을 통해 나온 결과물들은 이미지, 텍스트, 파일정보를 열람할 수 있으며, 해당 파일을 다운로드 할 수 있다.

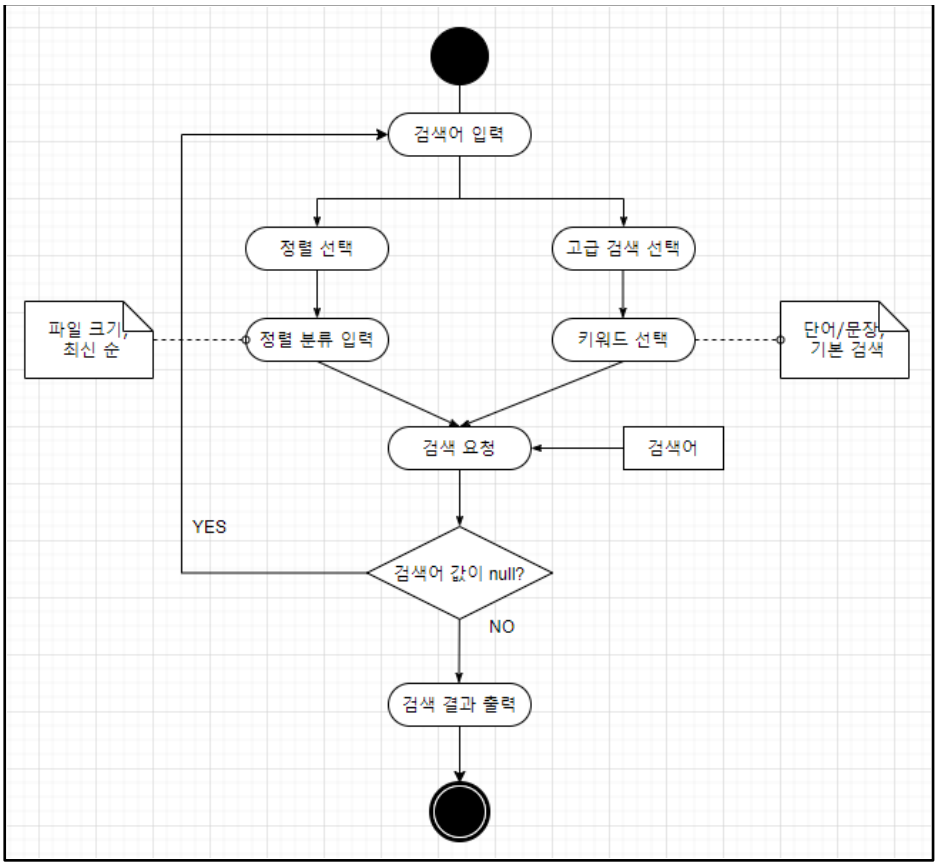
4.2.2. Activity Diagram

4.2.2.1 기본 검색



[그림 9] Activity Diagram - 기본 검색

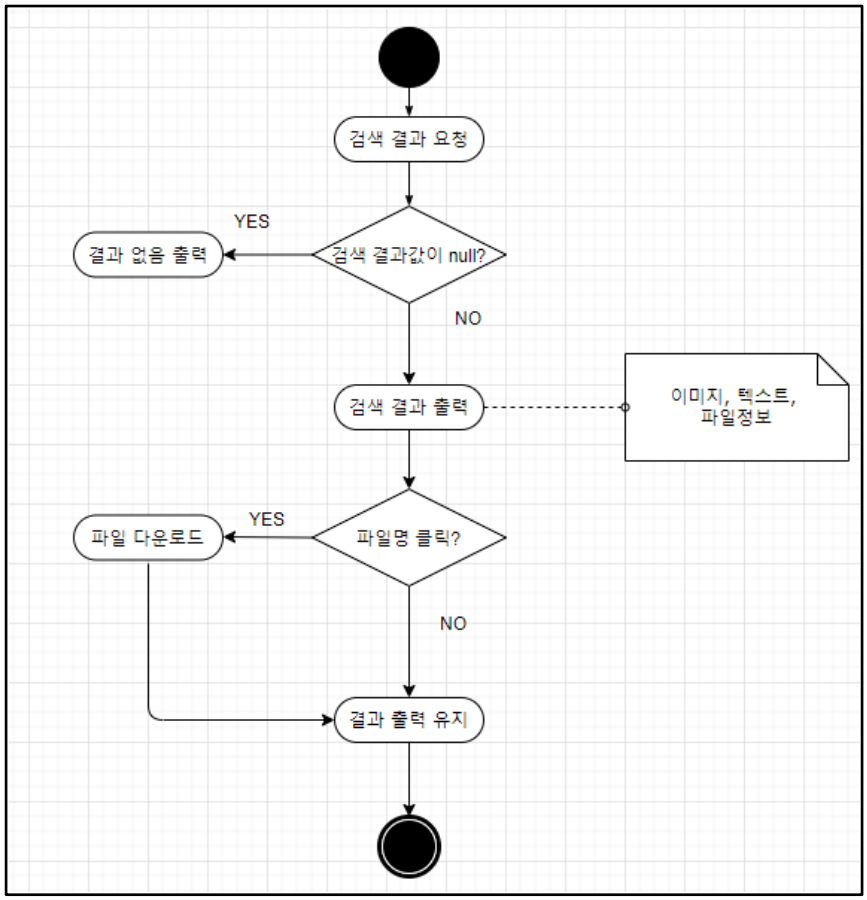
4.2.2.2 고급 검색



[그림 10] Activity Diagram - 고급 검색

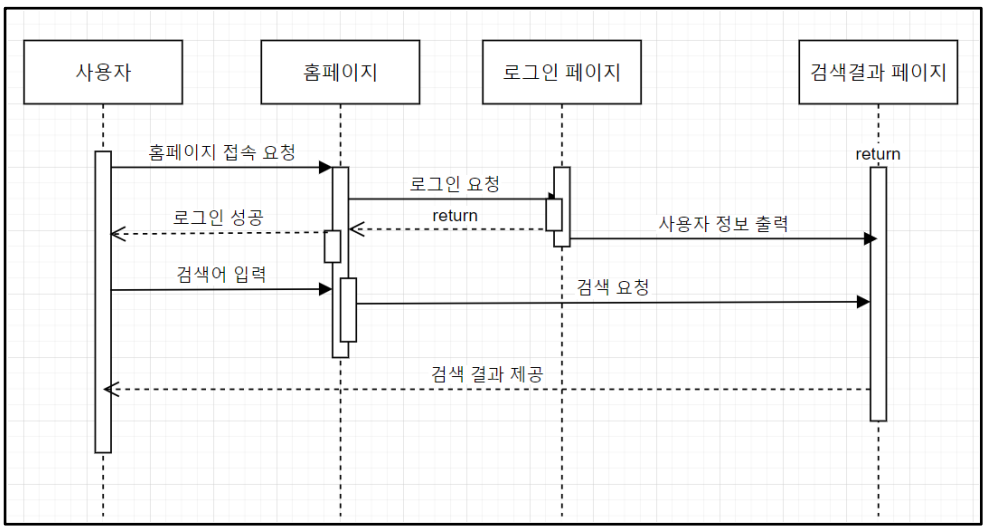
기본 검색 Activity Diagram 을 토대로, 원하는 검색 기준을 선택하여 검색 결과를 출력시킬 수 있도록 한다.

4.2.2.3 검색 결과



[그림 11] Activity Diagram - 검색 결과

4.2.4. Sequence Diagram



[그림 12] 전체 Sequence Diagram

Sequence Diagram 은 다음과 같다. 먼저 사용자는 해당 웹사이트의 도메인을 입력하여 홈페이지에

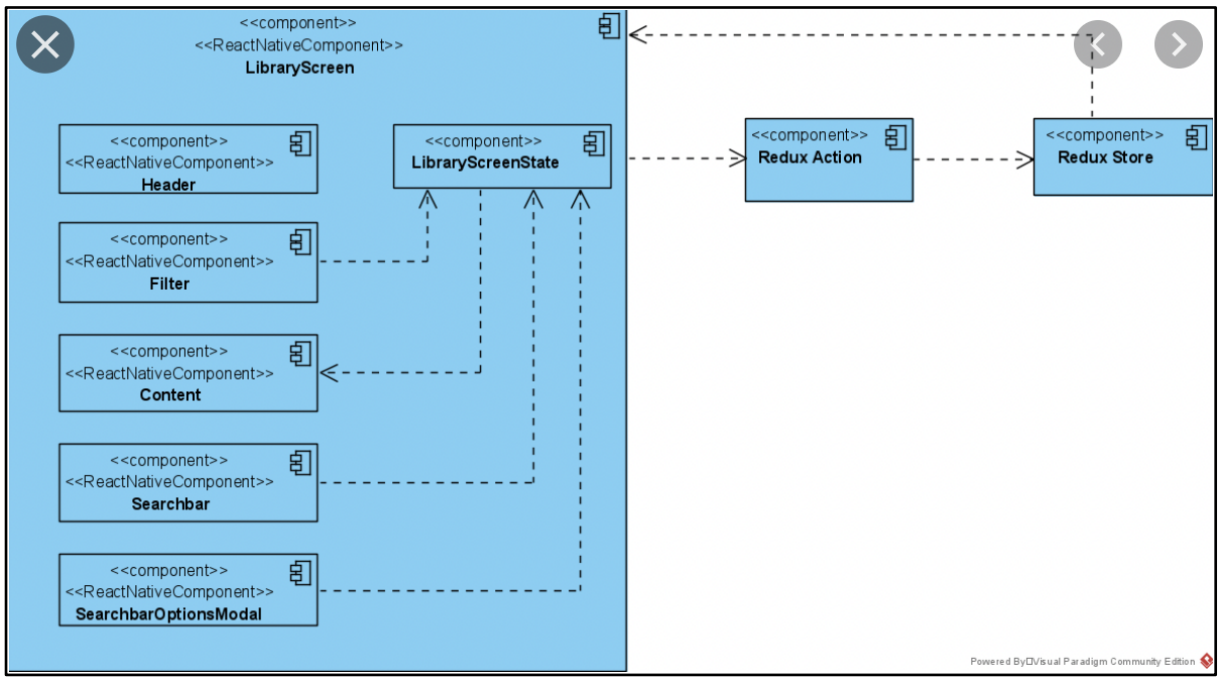
접속할 수 있다. 그 후, 검색을 하기위해 로그인 페이지로 이동하며, 로그인 페이지에서 해당 계정과 입력한 정보가 동일한 경우, 로그인 성공 여부를 알려주며, 사용자가 검색어를 입력하면 그에 따른 검색결과를 보여줄 수 있도록 검색결과 페이지를 통해 정보를 제공해준다.

4.2.5. Table 정보

- **eDrive Engine Schema**

Engine Schema	
열명	**타입**
id	number
file_path	text
images	text
filename	text
last_edited_date	date
writer	text
created_date	date
content	text

4.2.6. Component Diagram



[그림 14] 페이지 출력에 관한 Component Diagram(참고 그림)

전반적으로 다른 Diagram 들과 동일하며, 각각의 Component 에서 사용되는 대략적인 Interface 및 Artifact, 즉, 활용 정보에 관한 부분을 요약한 Diagram 이다. 이는 React,Redux 의 특성에 맞추어 제작한 Diagram 이다.

5. 프로젝트 결과

5.1. 기본 검색

사용자는 검색창에서 검색 옵션을 '전체', '작성자', '내용' 중에서 선택하여 찾고자 하는 문서를 검색할 수 있다. 아래의 예시는 문서의 내용을 기준으로 검색한 문서의 결과이며 문서의 결과는 검색창에 입력한 검색어와 매칭되는 것이 많은 순으로 나열된다.

[그림 15] 기본 검색 결과 페이지

5.2. API 호출 및 연동

기본 검색 결과를 보여주기 위해 프론트엔드에서 검색을 할 경우 elastic cloud 에 저장한 문서 결과를 보여줄 수 있도록 올바른 endpoint 와 API key 를 활용해 검색 요청을 보낸다. 아래는 검색 요청을 보내는 프론트 코드의 일부와 그에 따른 response 결과이다.

```
import axios from 'axios';

const esInstance = axios.create({
  baseURL: process.env.REACT_APP_API_ENDPOINT,
  headers: {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${process.env.REACT_APP_SEARCH_KEY}`,
  },
});

export const esApi = {
  search: async searchWord => {
    const res = await esInstance.post(
      `/api/as/v1/engines/${process.env.REACT_APP_ENGINE_NAME}/search`,
      {
        query: searchWord,
      }
    );

    return res.data;
  },
};
```

[그림 16] 검색 요청 코드

그림 16] 검색 요청에 대한 응답 결과

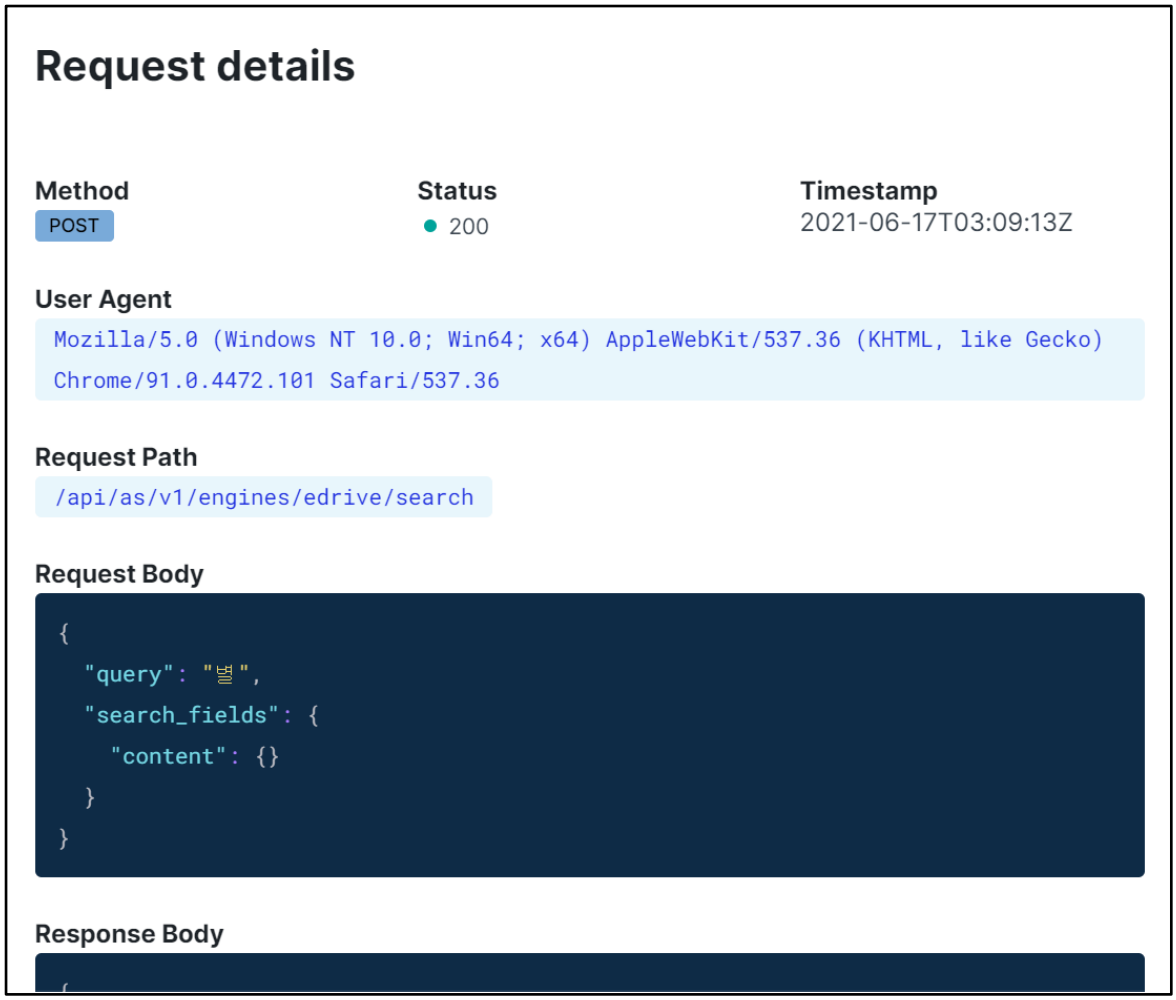
5.3. elastic search 를 활용한 검색 로깅

Elastic Cloud 를 활용해 하루동안 얼마나 많은 검색 쿼리를 요청했는지 알 수 있다.

[

그림 17] 날짜별 요청한 쿼리 수

요청한 쿼리 수를 파악하는 것뿐만아니라 세부적인 request detail 을 확인할 수 있고, 이를 통해 사용자가 어떤 문서를 많이 검색했는지와 같은 정보를 얻을 수 있고, 이를 통해 실시간 인기 검색어와 같은 추가적인 기능 구현이 가능하다.



[그림 18] 요청한 쿼리 디테일 정보

6. 결론

S3 스토리지에 저장된 문서 파일의 내용과 작성자 등의 조건으로 웹사이트에서 사용자가 검색하여 결과를 얻고 웹사이트 상에서 바로 해당 파일을 다운로드 하는 것이 가능해졌다. 또한 문서의 메타데이터 정보를 미리 확인하고, 문서 내에서 보유하고 있는 이미지들 또한 확인이 가능하다.

6.1. 기대효과

문서중앙화를 이용하는 사용자를 위해 문서의 제목뿐만 아니라 본문 내용, 작성자 등의 메타데이터를 이용해 검색이 가능하도록 했다. 그리고 문서의 내용을 미리볼 수 있도록 하여 문서 내용을 효율적으로 검색 할 수 있는 사용자 인터페이스를 웹을 통해 제공함으로써 문서 검색 및 획득의 접근성을 높여 효율적으로 문서 관리를 할 수 있도록 도움을 줄 수 있다.

6.2. 추후 연구 방향

추후에는 부족했던 기능에 대한 개선작업이 이루어질 예정이다. 현재는 mock data 를 활용해 문서의 내용을 파싱되었다고 가정하고 프로젝트를 진행했다. 이부분을 실제로 hwp, docx, pdf 등의 문서를 직접 파싱하여 데이터를 정제하고 바로 elasticsearch 에 정제된 데이터를 삽입할 수 있도록 완벽한 파이프라인을 구축하는 것을 목표로 한다. 최종적으로는 사용자가 s3 에 문서를 추가하면 백엔드에서는 해당 이벤트를 감지해 문서를 파싱한 후 정제하여 elasticsearch 에 업로드하고, 바로 사용자가 업로드한 문서를 검색할 수 있는 방향으로 구현할 예정이다.

또한 기능 구현 부분에서 완성이 미흡했던 부분을 보완하고 고급 검색 기능(조건 검색, 날짜 범위 설정 검색 등)을 추가하고, elasticsearch cloud 에서 제공하는 로깅을 활용해 실시간 인기 검색, 사용자 추천 검색어, 검색어 자동 완성 등의 사용자의 UX 를 높여주기 위한 기능 추가를 목표로 한다.

7. 참고문헌

[1] React.js 공식 사이트: https://ko.reactjs.org/

[2] ElasticSearch : https://www.elastic.co/kr/

[3] Mantine : https://mantine.dev/