**인터넷 댓글 검열 확장프로그램 개발**

**- API와 CORS를 중심으로 -**

20928 정준영

목차

[I. 서론 3](#_Toc196252571)

[II. 확장 프로그램 4](#_Toc196252572)

[1. 정의와 기능 4](#_Toc196252573)

[2. 작동 원리 5](#_Toc196252574)

[3. 필요 요소 6](#_Toc196252575)

[III. API (Application Programming Interface) 7](#_Toc196252576)

[1. API란? 7](#_Toc196252577)

[2. API 작동 방식 9](#_Toc196252578)

[3. REST API 10](#_Toc196252579)

[1) REST와 REST API 10](#_Toc196252580)

[2) HTTP Method 12](#_Toc196252581)

[4. API 보안 13](#_Toc196252582)

[1) 클라이언트 인증 13](#_Toc196252583)

[2) CORS (Cross-Origin Resource Sharing) 14](#_Toc196252584)

[IV. 개발 18](#_Toc196252585)

[1. 개발 계획 18](#_Toc196252586)

[2. 욕설 필터 19](#_Toc196252587)

[3. 검열 시스템 22](#_Toc196252588)

[1) 시스템 구현 22](#_Toc196252589)

[2) 정규 표현식 23](#_Toc196252590)

[3) re 라이브러리 25](#_Toc196252591)

[4. API 개발 26](#_Toc196252592)

[1) AWS (Amazon Web Serviece) 26](#_Toc196252593)

[2) API 구축 27](#_Toc196252594)

[3) 로깅 (Logging) 31](#_Toc196252595)

[5. 확장프로그램 개발 33](#_Toc196252596)

[6. 테스트 35](#_Toc196252597)

[7. 코드 37](#_Toc196252598)

[V. 결론 38](#_Toc196252599)

# 서론

***1, 2학년 후속 탐구입니다.*** ***[이전 탐구 보고서](https://github.com/jjy0809/creanpl/blob/main/%EC%9A%95%EC%84%A4%20%EA%B2%80%EC%97%B4%20%EC%9D%B8%EA%B3%B5%EC%A7%80%EB%8A%A5%20%EB%AA%A8%EB%8D%B8%20%EA%B0%9C%EB%B0%9C%20%ED%83%90%EA%B5%AC.pdf)를 먼저 읽어 주세요.***

요즘 정치적/사회적 사건 사고들이 다양해지면서 뉴스, SNS, 커뮤니티 등지에서 다양한 성향의 사람들이 각자의 의견을 가지고 댓글을 작성하고 있다. 그러나 의견 차이로 인한 갈등이 심화되면서 사람들의 작성하는 댓글 들에는 욕설과 혐오 표현이 난무한다. 이러한 인터넷 환경에서 서핑하는 것은 별로 유쾌하지 않을 것이다. 특히 어린 자녀를 둔 부모는 자신의 자녀가 입에 담기 힘든 욕설과 혐오 표현들을 접하기를 원치 않을 것이다. 이러한 문제들을 해결하고자 네이버를 비롯한 검색엔진 기업들은 ‘클린봇’등의 댓글 검열 인공지능을 개발하였지만 그 효과는 미약하다. 네이버의 클린봇의 경우 직접적으로 그 수위가 높은 욕설이 댓글에 포함된 경우 댓글을 검열한다. 다르게 말하자면 간접적으로 욕설을 작성하거나 문맥을 파악해야 이해할 수 있는 비꼬는 투의 혐오 표현, 수위가 낮은 욕설들은 허용한다는 뜻이다. 네이버 뉴스에서 아무 뉴스에 들어가서 댓글창을 보면 보기 힘들 수준의 심한 욕설은 거의 없지만 여전히 약한 욕설이나 비꼬는 댓글들은 쉽게 찾을 수 있다. 이런 클란봇 등의 댓글 검열 인공지능의 문제를 보완하기 위해 이전에 직접 개발한 ‘욕설 탐지 인공지능’을 활용하여 조금 더 다양한 욕설/혐오 표현을 인식하여 댓글을 검열할 수 있는 확장프로그램을 개발하였다.

먼저 개발 과정에서 발생한 문제들을 해결하기 위해 공부하며 학습한 핵심적인 이론들에 대해서 소개한 다음 ‘욕설 탐지 인공지능’을 활용한 욕설 검열 API 개발 및 해당 API를 사용한 자동 댓글 검열 확장 프로그램을 개발하여 실제로 사용해보는 개발 과정을 소개하겠다.

본 탐구 보고서에서 사용한 코드에 대한 직접적인 설명은 생략하겠다.

# 확장 프로그램

## 정의와 기능

확장 프로그램이란 웹 브라우저[[1]](#footnote-1)에서 기존의 동작을 설정/코드 변경을 통해 변화시키거나 새로운 기능을 추가함으로써 웹 브라우저의 사용성과 편의성을 향상시키는 방법 또는 프로그램이다.[[2]](#footnote-2)

확장 프로그램의 역할 종류에는 크게 기능 변경, 기능 추가, 웹 페이지 변경, 스타일 변경이 있다. 기능 변경의 경우 기존에 있던 브라우저 기능을 변경하여 다르게 작동하도록 만드는 확장 프로그램이다. 예로 기존에 걸려있던 제한(메모리 제한, 쿠키 제한, cors 제한 등)을 변경할 수 있다. 기능 추가는 기존에 없던 새로운 기능을 추가하는 것이다. 툴바 추가, 페이지 자동 번역, 유용한 정보 제공, 프록시, vpn 등등 대부분의 우리가 사용하는 확장프로그램이 여기에 해당한다. 웹 페이지 변경은 브라우저를 변경하는게 아닌 페이지를 변경하는 프로그램이다. 특정 웹 페이지의 ui를 수정하거나 광고를 제거하는 등의 기능을 한다. 본 탐구에서 개발하고자 하는 확장프로그램의 경우 여기에 해당한다. 마지막으로 스타일 변경의 경우 브라우저의 스타일(디자인)을 변경시켜주는 확장프로그램으로 크롬 등의 브라우저는 미지원하며 파이어폭스만 지원한다. 브라우저 모양이나 레이아웃, 색깔 등을 변경할 수 있다.

## 작동 원리

확장 프로그램은 웹 사이트를 개발하는 것과 거의 동일하다. 확장 프로그램의 팝업 UI의 경우 웹 개발에 사용되는 HTML과 CSS를 사용하며, 기능 동작의 경우 JS를 사용한다. 다만 확장 프로그램의 경우 웹 사이트와는 다르게 ‘브라우저’에 맞춰서 개발되어야 하므로 일부 요소가 웹 개발과는 다르다.

우선 브라우저가 확장프로그램을 인식하기 위해 매니페스트(manifest) 파일이 필요하다. 파일은 json[[3]](#footnote-3)으로 작성된다. 매니페스트 파일이란 컴퓨터의 메타데이터(metadata)[[4]](#footnote-4)를 포함하는 파일으로 패키지가 어떻게 어떠한 구성으로 각각의 파일 이름과 확장자는 무엇이고 어떤 기능을 하는지 등등 다양한 메타 데이터와 자신(패키지)에 대한 정보들을 담아놓는다.[[5]](#footnote-5) 브라우저는 이를 인식하여 확장프로그램이 어떤 구조로 이루어져 있는지 이해하고 그 구조에 맞게 코드를 로드하여 작동한다.

브라우저가 정상적으로 확장프로그램을 인식하였으면 확장프로그램의 요청[[6]](#footnote-6)에 맞게 필요할 때 실행되어 작동하게 된다. 확장프로그램은 현재 보고있는 웹 페이지에서 백그라운드로 확장 프로그램의 js 코드를 추가하듯 동작한다. 예를 들어 광고 차단 확장 프로그램의 경우 웹 페이지를 로드 할 때 미리 지정된 광고 필터에 따라 광고 요소가 로드 될 때 해당 요소를 제거하여 광고가 페이지에 뜨지 못하도록 막는다. 이처럼 로드하는 웹 페이지의 소스보다 확장 프로그램의 소스가 우선이며, 확장 프로그램은 기존 웹사이트의 소스에서 특정 부분을 제거하거나 추가하는 기능을 수행한다.

## 필요 요소

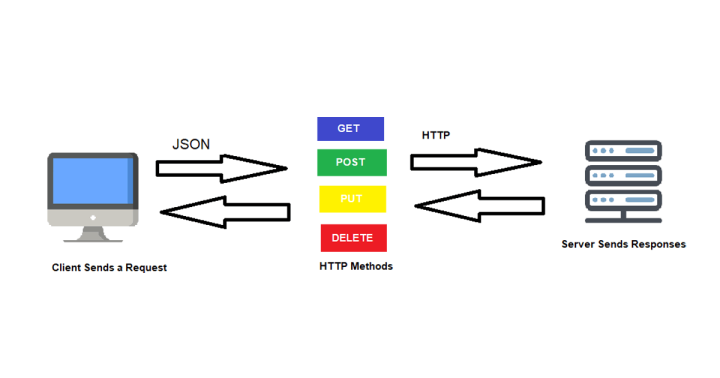
확장 프로그램을 개발하려면 확장 프로그램에 들어가야 할 요소(소스 코드)가 무엇인지 알아야 한다. 확장 프로그램에는 앞서 언급한 요소들을 포함하여 다음과 같은 요소들이 존재하며, 다음과 같은 기능들을 유기, 복합적으로 수행하여 하나의 확장 프로그램을 이룬다. 파일 이름은 본 탐구에서 개발한 확장프로그램의 각 파일 이름이며, 파일 이름이 아래와 다르더라도 이를 매니페스트 파일에 이를 명시해두면 문제가 없다. 다만 확장자의 경우 아래와 동일해야 한다. **볼드체**한 파일의 경우 필수 요소이다.

* **manifest.json**: 생략
* **background.js**: 확장 프로그램의 백엔드(백그라운드) 코드이다. 확장프로그램의 실제 작동 여부와 관계 없이 확장 프로그램을 종료하지 않는 이상 계속하여 돌아간다
* **content.js**: 실제 확장프로그램의 코드. 확장프로그램이 작동할 때 실행되는 코드이다
* popup.html: 확장 프로그램을 클릭하면 나오는 팝업 창이다. 웹 개발과 동일한 형식이다
* style.css: 팝업창의 디자인에 관련된 코드이다
* popup.js: 팝업창의 프론트엔드 js 코드이다

팝업창의 경우 웹 페이지 하나 새로 만드는 것과 동일하다. 즉 확장 프로그램에 대한 로컬 웹페이지이다. 해당 페이지에서 확장 프로그램의 옵션을 설정하거나 사용법을 익히는 등 정보 제공 및 실질적인 interaction을 수행한다.

# API (Application Programming Interface)

## API란?

 [[7]](#footnote-7)

API(Application Programming Interface)란 두 소프트웨어의 구성 요소끼리 서로 통신할 수 있게 하는 메커니즘이다. 예를 들어 ‘날씨 앱’을 개발한다고 해보자. 날씨 앱 개발자가 전국 지역 곳곳의 날씨를 모두 **직접** 측정할 수는 없을 것이다.이때 사용 할 수 있는 것이 바로 API이다. 직접 데이터를 수집할 수 없는 상황에서 미리 수집된 데이터를 가져올 수 있다. 날씨의 경우 기상청 데이터베이스에 각 지역에 대한 날씨 정보가 세세히 실시간으로 기록되고 있다. 이 데이터를 가져올 수 있다면 날씨 앱에서 user에게 날씨 정보를 제공할 수 있을것이다. 개발자는 기상청 API에 request(요청)한다. 그러면 기상청 API가 돌아가고 있는 서버에서는 이 요청을 받아서 필요한 경우 인증 과정을 거친 다음 요청 받은 정보를 response(응답)한다. 개발자는 이 응답 받은 데이터를 사용자에게 보여줌으로써 날씨 앱을 개발할 수 있게 되었다.[[8]](#footnote-8) 이때 응답 데이터를 리소스라고 부른다.

이처럼 API는 데이터를 요청하는 **Client**와 이를 받아서 데이터를 제공해주는 **Application** (Server) 양측간의 상호 통신이다. 이때 클라이언트는 요청할 때 보내는 packet(패킷)에 다양한 데이터를 담을 수 있다. 예를 들어 기상청 API 서버에 ‘대구’의 지난 3개월간의 미세먼지 수치만을 받아오고 싶다면 request packet에 원하는 데이터에 대한 정보를 담아서 보내면 API 서버에서는 이를 분석하고 이에 따라 데이터를 모아서 다시 클라이언트로 보내준다. 이때 만약 API 서버에서 요구하는 필수 데이터가 패킷에 포함되어 있지 않거나 데이터 구조가 API 서버의 요구 사항과 다를 경우 제대로 응답하지 못하거나 reject될 수 있다.

최근에는 대부분의 Application이나 Web 등 다양한 서비스에서 개발할 때 각자의 역할을 수행하는 독립된 백엔드 API 서버를 구축하고 해당 서버에 API요청을 보내 작동하도록 구축한다. 이 분야의 대표적인 예로 아마존[[9]](#footnote-9)(Amazon)이 있다.

## API 작동 방식

API의 작동 방식에는 크게 4가지로 분류할 수 있다.[[10]](#footnote-10)

* **SOAP API**: 단순 객체 접근 프로토콜[[11]](#footnote-11)을 사용하여 클라이언트와 서버 양측이 XML[[12]](#footnote-12)을 사용하여 데이터를 주고 받는다. 보안이나 데이터 전송 방식에 있어서 여러 제약과 표준들이 정해져 있어서 보안에는 좋지만 웹에서는 사용하기 어려워 최근에는 거의 사용하지 않는다[[13]](#footnote-13)
* **RPC API**: 원격 프로시저 호출이라 함. 클라이언트가 마치 클라이언트의 로컬 소스인 것처럼 직접 서버의 함수를 호출하여 사용함[[14]](#footnote-14). 프로그래밍 언어에서 이부 라이브러리를 import하여 사용하는 방법과 비슷함
* **Websocket API**: 실시간 통신이 가능한 protocol을 사용하여 클라이언트와 서버가 한 번 연결 된 이후 계속하여 통신이 가능하다. 채팅이나 온라인 게임 등 실시간으로 데이터를 주고받아야 하는 상황에서 사용할 수 있다[[15]](#footnote-15)
* **REST API**: **3.REST API**에서 자세히 설명하겠다

## REST API

### REST와 REST API

REST(**Re**presentational **S**tate **T**ransfer)란 API에 제약 조건을 걸어주는 소프트웨어 architecture이다. 인터넷과 같은 복잡한 네트워크에서 지정된 형식만을 사용하여 고성능으로 통신하기 위해 만들어졌다. 이때 사용하는 통신 프로토콜은 http 또는 https이다. REST에서 통신을 할 때 먼저 요청 packet에 리소스를 명시해야한다. 그리고 요청을 보낼 때 특정한 HTTP Method를 이용하여 보내야한다. 이러한 정보들을 담은 메타데이터는 header에 담겨져 있으며 주로 사용하는 확장자는 json이다.[[16]](#footnote-16) 다음은 REST API가 작동하는 방식이다.

1. 클라이언트가 서버에 요청을 전송한다. 클라이언트는 서버측에서 미리 명시한 형식에 따라 요청 형식(프로토콜과 Method 등)을 지정한다
2. 서버가 클라이언트를 인증하고 해당 요청을 수행할 수 있는 권한이 클라이언트에 있는지 확인한다(API크래딧 등을 사용하는 유료 API이거나 특정 사용자만 사용할 수 있는 Private API인 경우 필요한 과정으로Public인 경우 생략 가능하다)
3. 서버가 요청을 수신하고 내부적으로 처리한다
4. 서버가 클라이언트에 응답을 반환한다. 응답에는 응답 코드가 포함되어 클라이언트측에 요청 성공 여부도 알려주며, 클라이언트가 요청한 요청 패킷도 함께 포함되어 있다. [[17]](#footnote-17)

이때 2와 3번에서 볼 수 있듯 서버가 요청을 수신하기 전 클라이언트 인증 과정을 먼저 거치게된다. 해당 내용에 대해서는 **3) 클라이언트 인증** 에서 다루겠다.

REST API의 경우 Stateless(무상태) 아키텍쳐이다. 무상태란 서버와 클라이언트간의 통신은 모두 독립적이며 서로 다른 요청끼리 분리되어 실행된다. 즉 클라이언트 A가 보낸 요청과 클라이언트 B가 보낸 요청은 서로 독립적으로 분리되어 서버가 응답하며 두 응답에는 서로의 요청 내용에 영향받지 않는다.

또한 REST API의 경우 Layer화(계층화) 할 수 있다. 클라이언트와 서버 사이에 다른 중개 서버 또는 아키텍쳐를 삽입하여도 정상적으로 작동할 수 있다. [[18]](#footnote-18)

이러한 특징들로 인해 REST API의 경우 다양한 장점이 있다. 먼저 기존 웹에서 사용하는 HTTP Protocol을 사용하여 별도의 인프라를 구축할 필요도 없을 뿐더러 웹에서 API를 사용하거나 구축하기도 쉽다. 또한 형식이 지정되어 있으므로 서비스간의 충돌이나 문제가 적고 안정적이다.

또한 REST에는 다음과 같은 구성요소가 존재한다.

* **EndPoint**: 통신이 끝나는 지점, 즉 최종 목적지(서버)이다. 클라이언트는 서버의 EndPoint로 요청을 보내게 되며 이때 EndPoint의 형식은 웹 페이지의 도메인 주소와 동일하다(URL 형식이다). API 서버의 URL이 각각 고유하게 존재하며, 클라이언트는 이 URL에 request하여 API를 호출한다. URL에 요청을 보낼 때 header에는 메타데이터가, body에는 요청 내용이 담겨져 있어야 한다.
* **HTTP Method**: REST는 앞서 말했듯 HTT/HTTPS 프로토콜을 사용한다. 이때 데이터 전송 방식에 여러 Method(방법)이 존재하는데 이 방법에 따라 데이터를 처리하는 방법과 과정이 약간씩 다르다
* **리소스 응답**: 클라이언트의 요청을 받아 서버는 해당 요청 사항에 맞게 작업을 한 다음 결과물을 다시 클라이언트에게 response한다. 이때 resource는 json, xml 등 다양한 데이터 확장자를 통해 전달 될 수 있다. [[19]](#footnote-19)

### HTTP Method

HTTP 요청을 보낼 때 사용할 수 있는 Method에는 대표적으로 다음과 같은 방법들이 존재한다. 용도에 맞게 Method를 택하여 사용해야 한다.

* **GET**: 서버 측 리소스를 요청. 서버측에서 데이터만 받아오는 경우 사용한다. 예를 들어 기상청 서버에서 날씨 정보를 가져오는 경우 사용된다
* **POST**: 서버에 데이터를 전송. 주로 서버측에서 데이터를 저장해야 하는 경우 사용한다. 예를 들어 회원가입 서버에 신규 회원에 대한 정보를 등록할 때 사용한다
* **PUT**: 서버측의 데이터를 수정(갱신)할 때 사용한다. 예를 들어 회원 정보를 수정할 때 사용한다
* **Patch**: PUT과 비슷하지만 리소스의 특정 부분만 수정한다
* **DELETE**: 데이터를 삭제할 때 사용한다. 예를 들어 회원 탈퇴를 하여 회원 정보를 삭제해야 할 때 사용한다
* **Options**: 클라이언트와 서버간 통신 옵션을 확인할 때 사용된다. 추후 설명할 CORS에서 주로 사용된다 [[20]](#footnote-20) [[21]](#footnote-21) [[22]](#footnote-22)

## API 보안

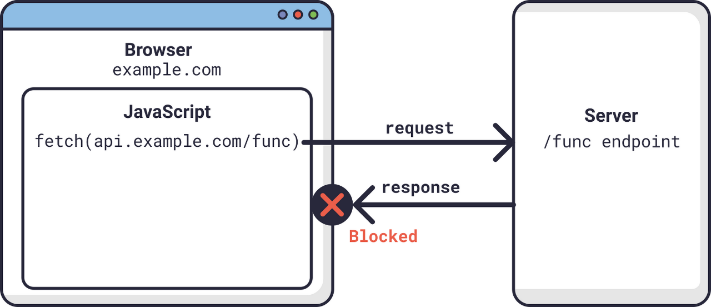
### 클라이언트 인증

앞서 말했듯이 API를 호출할 때 서버에서는 클라이언트가 API를 호출할 권한이 되는지를 검증하기 위해 여러 인증 과정을 거친다. 이러한 인증 과정은 Private API 뿐만 아니라 Public API 또한 필요하다. 클라이언트의 요청 권한(OAuth)을 포함하여 클라이언트에서 보낸 요청이 올바른 형식인지(API 스키마), 기준치를 넘겨서 과도하게 호출하지는 않았는지(API Key) 등등 API마다 각자의 다양한 인증 과정을 거친다.

* **API Key**: 클라이언트와 서버만이 알고있는 고유 문자열. API요청을 보낼 때 API Key도 함께 첨부하여 본인이 누구인지 밝히는 방법으로 일종의 보안번호 같은 개념이다. API Key를 발급 받기 위해서는 우선 서버측에 사용자를 등록하여 API 사용 권한을 얻어야 한다. API Key는 Key만 있으면 API를 쉽게 호출 가능하여 사용이 편리하고 쉽다는 장점이 있지만 API Key가 유출된다면 타인이 API Key 주인의 신원으로 API를 무단 이용 할 수 있다는 보안상 단점도 존재한다
* **ID/PW**: 대부분의 사람들이 일상 중에 사용하는 방법으로 사용자 아이디(ID)와 비밀번호(PW)를 통해 인증한다. API Key와의 차이점은 ID와 PW는 사용자가 직접 설정할 수 있다. API Key의 경우 문자열이 규칙이 없고 길어서 외우고 다니는건 불가했지만, ID/PW의 경우 자신이 직접 설정하므로 외우기가 쉽다는 장점이 있다. 다만 이는 오히려 보안상에서는 단점으로 작용하여 API Key에 비해 비밀번호가 유출되기 쉽고, 주로 API Key보다 짧은 경우가 대부분이라 Brute Force(무차별 대입 공격)에도 취약하다
* **OAuth 토큰**: OAuth란 ID나 비밀번호 등의 자격 증명 데이터를 공유하지 않고도 권한을 부여하는 표준 프로토콜이다[[23]](#footnote-23). 클라이언트가 API 서버에 요청을 보낼 때 신뢰 가능한 인증 서버에 OAuth 인증을 요청하고 해당 응답을 토큰을 이용하여 API 서버에 전달한다. 데이터 유출 위험이 없어 보안에 강력하지만 사용하기 까다로워 Public API에서는 사용하지 않는다 [[24]](#footnote-24)

### CORS (Cross-Origin Resource Sharing)

교차 출처 리소스 공유(**C**ross-**O**rigin **R**esource **S**haring, CORS)란 브라우저에서 서로 다른 출처로부터 데이터를 공유하는 것을 허용하도록 브라우저와 서버가 접근 허가를 해주는 보안 메커니즘을 말한다.

 [[25]](#footnote-25)

여기서 출처란 도메인 뿐만 아니라 포트, 프로토콜도 모두 포함한다. 예를들어 <https://daegun.hs.kr/api> (기본 포트는 5000 이라 가정)라는 API 서버가 존재한다 해보자. 이때 어떤 출처가 https 프로토콜, daegun.hs.kr 도메인, 포트 5000이라면, 위의 API 서버와 동일한 출처라고 할 수 있다. 그러나 만약 프로토콜이 http거나, 도메인이 daegun.ms.kr 또는 yeungnam.hs.kr이거나, 포트가 1234라면 동일한 출처라고 할 수 없다. 프로토콜, 도메인, 포트 세가지 모두 동일해야 동일 출처라고 할 수 있는 것이다. 여기서 달라도 허용되는것은 Path([https://daegun.hs.kr/**study**](https://daegun.hs.kr/study) 등) 뿐이다.[[26]](#footnote-26)

이러한 메커니즘은 과거 XSS(**c**ross-**s**ite **s**cripting, 사이트 간 스크립팅) 공격(외에도 다양한 이유가 있지만 언급 안하겠다)을 막기위해 개발되었다. XSS란 웹 취약점 중 하나로 웹사이트 개발자(관리자)가 아닌 제3자(공격자)가 웹 페이지에 악성 코드를 삽입 할 수 있는 취약점이다. 예를 들어 어떤 커뮤니티 사이트가 존재한다고 해보자. 이 사이트에 *“안녕하세요”* 라는 게시글을 작성한다면 실제 HTML상 구조로는 *<span>안녕하세요</span>* 라고 작성되어 타인에게 표시될 것이다.

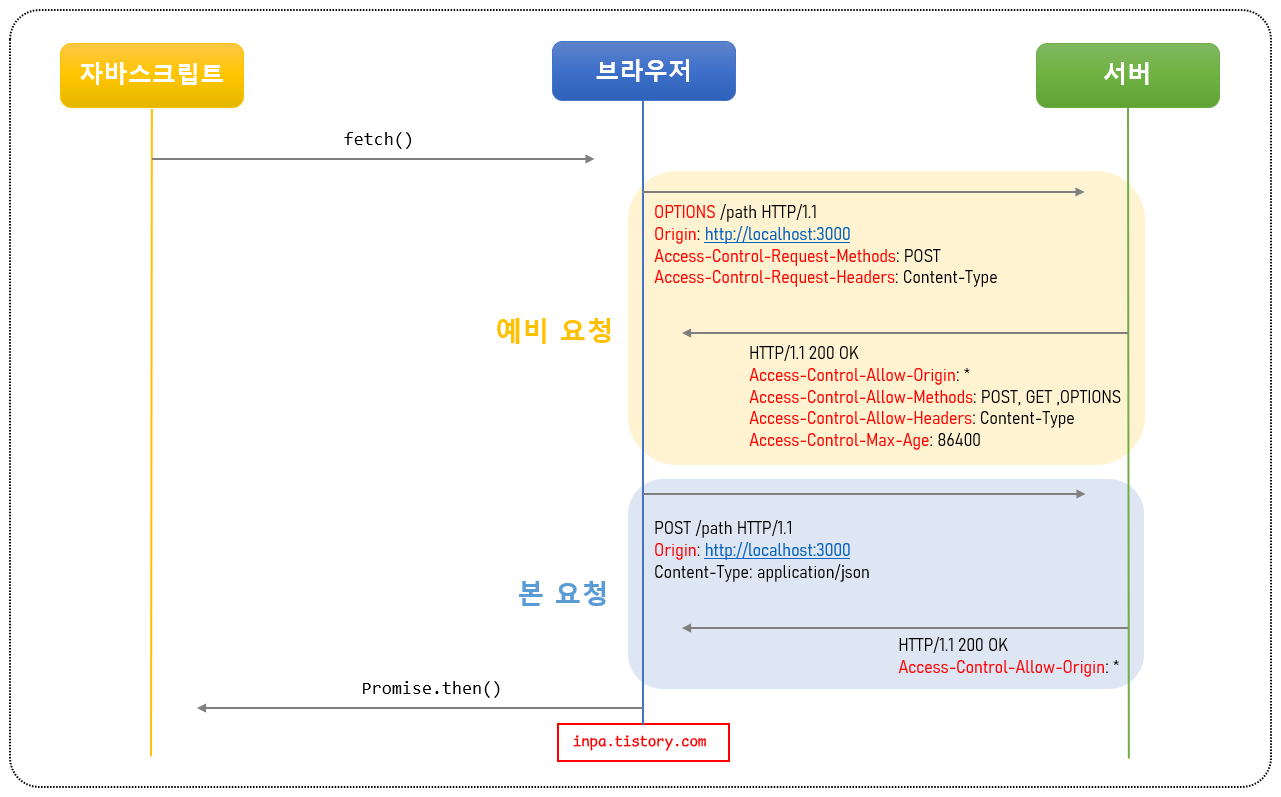
그러나 만약 공격자가

*“안녕하세요.* *<img src=’abc’ onerror=’this.src=”http://hacking.site.com/”'></img>[[27]](#footnote-27)”*

라고 작성했다면, HTML 구조상에서는

*<span>안녕하세요.* ***<img -생략-></img>****</span>* 이라고 표시될 것이다.

HTML 특성상 span 태그 안에있는 img 태그 또한 새로운 태그라고 인식 인식하여 해당 img 태그가 **코드**로써 실행될 것이다[[28]](#footnote-28). [[29]](#footnote-29)그러나 이러한 공격을 위해서는 위 코드에서 보이듯 해킹 사이트를 입력하여 해당 사이트에 데이터를 전송하거나 가져와야만 한다. 만약 CORS가 없었다면 커뮤니티 사이트와 해킹 사이트는 **다른 출처**지만 이에 대한 보안 프로토콜은 존재하지 않으므로 그대로 해킹 사이트와 데이터를 통신하게 된다. 그러나 CORS가 적용된다면 커뮤니티 사이트와 해킹 사이트는 동일 출처가 아니므로 데이터 통신 자체가 거부된다. 이러한 원리를 통해 XSS를 CORS로 방어할 수 있다[[30]](#footnote-30).

 [[31]](#footnote-31)

CORS의 작동 과정은 다음과 같다.

1. 클라이언트에서 외부 서버에 요청을 보내기 전, 브라우저가 자신의 출처를 담은 OPTIONS 요청을 먼저 서버에 보낸다. 이를 예비 요청(Preflight Request)라고 한다
2. 서버는 예비 요청에 대한 응답으로 서버가 허용하는 출처 등의 정보를 브라우저에 보낸다
3. 브라우저는 자신의 정보와 서버가 허용하는 정보를 비교하여 클라이언트가 보내려고 하는 요청이 CORS 정책에 위배되지 않는지 확인한다
4. 만약 CORS가 허용되었다면 브라우저는 처음에 클라이언트가 보내려고 했던 본 요청을 서버에 보낸다
5. 만약 정보가 달라 CORS 요청이 거부되었다면, 브라우저는 애러를 일으키고 본 요청을 하지 않는다

이때 1, 2번에서 주고받는 예비 요청과 응답에는 다음과 같은 요소들이 포함되어 있다.

* Origin(브라우저) / Access-Control-Allow-Origin(서버)

: 요청을 보낸 브라우저의 출처 / 서버가 허용하는 출처들의 목록

* Access-Control-Request-Method

: 실제 본 요청의 HTTP Method / 서버가 허용하는 HTTP Methods 목록

* Access-Control-Request-Headers

: 실제 본 요청에 사용할 Headers / 서버가 허용하는 Headers 목록

여기서 중요한 사실은 CORS는 브라우저에서**만** 판단한다는 점이다. 서버는 브라우저에게 자신이 허용하는 출처만 알려주는 것이지 서버에서 CORS에 위배되는 요청인지 판단하지 않는다. 브라우저는 결국 사용자의 컴퓨터에서 작동되므로 사용자가 강제로 브라우저의 보안 기능을 제거하거나 공격자가 CORS 헤더를 조작하는 등 브라우저의 CORS를 무력화 할 수 있다면 해당 보안 프로토콜은 작동하지 않게 된다. 또한 프록시 서버를 이용하게 된다면 출처를 위장할 수 있어 CORS가 제대로 작동하지 않게된다.[[32]](#footnote-32) [[33]](#footnote-33) [[34]](#footnote-34)

# 개발

이제 실제 확장프로그램의 개발 과정을 시간 순서대로 소개하겠다. 코드 자체에 대한 설명/해석은 생략하고 어떠한 과정과 사고 방식을 통해 어떻게 문제들을 해결하여 최종 결과물을 완성해냈는지를 서술하겠다.

## 개발 계획

기존에 개발한 ‘욕설 탐지 인공지능 모델(이하 인공지능)’을 사용하여 사용자의 웹 페이지에 뜨는 댓글들을 확장프로그램을 통해 검열해야 한다. 그러나 인공지능은 파이썬으로 작성되어 있고, 확장프로그램상에서 파이썬으로 인공지능을 구동하여 예측을 수행하는 것은 불가하다. 따라서 확장프로그램에서 파이썬 및 인공지능 모델을 사용할 수 있는 방법을 찾아보았고 API를 통해 외부와 통신하여 작업을 수행한다면 확장프로그램 내에서는 수행 불가하던 작업도 가능하단 사실을 알게되었다. 또한 인공지능의 정확도는 수집한 데이터 내에서는 우수하지만 수집이 안된 새로운 욕설과 은어는 인공지능이 학습하지 못하여 제대로 탐지를 못해낸다는 문제점과 욕설이 포함되었음에도 탐지를 못해내는 경우도 간혹 발생하여 이에 대한 해결책이 필요하다. 이 문제를 해결하기 위해 댓글 데이터를 활용한 ‘욕설 및 혐오 표현 필터’를 만들어서 함께 병행하여 사용하기로 하였다. 편의상 앞으로 인공지능을 사용한 욕설 탐지는 ‘인공지능 검열’ 으로, 필터 속 단어 리스트와 댓글속 단어 대조를 통한 욕설 탐지는 ‘필터 기반 검열’로 칭하겠다.

위의 해결책을 포함하여 확장프로그램을 개발하기 위해 다음과 같은 계획을 수립하고 실행하게 되었다.

1. 7가지의 카테고리에 대한 필터 단어 리스트 개발
2. 인공지능과 필터를 결합하여 텍스트의 욕설 정도를 수치화 해주는 ‘욕설 검열 시스템’ 개발
3. 욕설 검열 시스템 API 개발
4. API를 사용하여 확장프로그램 개발

## 욕설 필터

인공지능을 학습시키기 위해 수집했던 댓글 데이터[[35]](#footnote-35)를 활용하여 사람들이 자주 사용하는 욕설 및 혐오 표현 등의 단어에 대한 필터 리스트를 만들어 보겠다. 이전 탐구와 마찬가지로 디시인사이드(이하 디시) 댓글을 욕설 댓글로, 네이버 뉴스(이하 뉴스) 댓글을 비욕설 댓글로 가정[[36]](#footnote-36)하겠다.

이전 탐구에서 인공지능을 개발할 때 학습 데이터 라벨링을 위해 욕설 단어 리스트를 만들었었다. 그러기 위해 수많은 댓글 데이터에서 단어들을 추출해내어 인공지능이 문장을 학습할 때 사용하는 메커니즘 중 하나인 단어의 빈도수로 단어에 가중치를 주는 방법에 착안하여 모든 댓글을 단어 단위로 분리한 다음 각 단어가 수집된 댓글 데이터에서 총 몇 번 등장하는지를 count하였다. 이때 은/는/이/가 등의 조사는 제거하였다. 이 단어 리스트에서 디시에만 존재하고 뉴스 댓글에는 존재하지 않는 빈도수가 일정 수치 이상의 단어를 욕설로 규정하고 욕설 리스트를 최종적으로 생성했었다. 이제 이렇게 만들어진 욕설 단어 리스트를 이제 필터로 만들어 보겠다.

만들고자 하는 확장프로그램이 어린이들이 주요 target user이므로 이들에게 맞춰서 욕설뿐만 아니라 기타 어린이들에게 해가 되는 단어를 선별해내야 한다. 따라서 필터를 다음과 같이 7+1개의 카테고리로 분리하여 욕설 리스트 속 단어를 분류해보았다. 일부 필터간 중복되는 단어가 포함될 수 있으며, 이는 의도된 바이다. 괄호안은 실제 확장프로그램에서 사용하는 필터 명칭이다.

* **욕설(swear\_words)**: 일반적인 욕설 ex) 새끼, 병신, 시발
* **비하(demean\_words)**: 남을 비하하는 표현 ex) 짱깨, 빨갱이, 조센징
* **공격적(aggres\_words)**: 어린이가 보기에 다소 부적절한 단어

ex) 부적절 표현(꼬움, 처먹, 죽어), 범죄 관련 단어(살인, 마약, 토토) 및 은어(도박(쩐주[[37]](#footnote-37), 마개[[38]](#footnote-38), 뽀찌[[39]](#footnote-39)), 마약(아이스[[40]](#footnote-40), 떨[[41]](#footnote-41), 물뽕[[42]](#footnote-42)))

* **성(sexual\_words)**: 성적인 단어 ex) 섹스, 창녀, 몰카
* **정치(politic\_words)**: 정치적인 단어

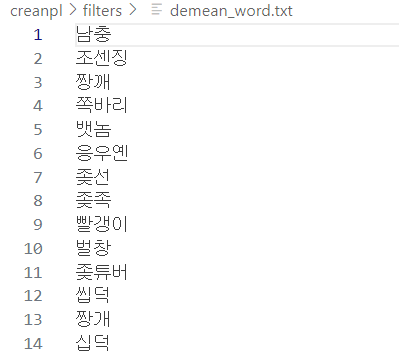
ex) 정치인(윤석열, 문재인, 박근혜), 비하(용산총독[[43]](#footnote-43), 찢[[44]](#footnote-44), 문죄인)

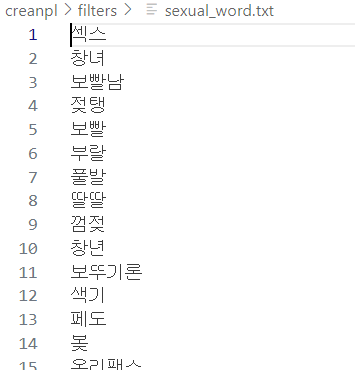
* **일베[[45]](#footnote-45)(ilbe\_words)**: 커뮤니티발 은어/표현 ex) 개추[[46]](#footnote-46), 노짱[[47]](#footnote-47), 우흥[[48]](#footnote-48)
* **젠더 갈등(gender\_words)**: 성차별/성갈등을 일으키는 단어

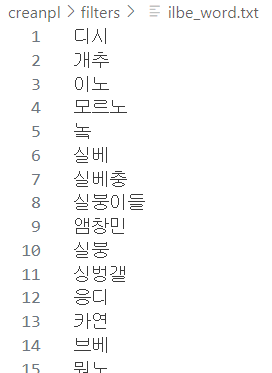
ex) 한녀, 퐁퐁남, 김치

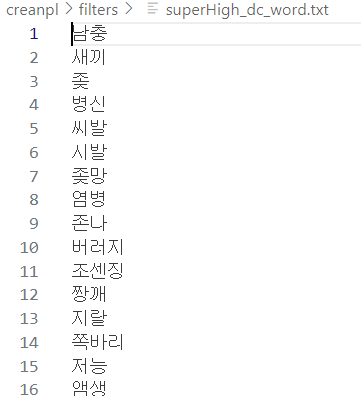
* **고빈도 욕설 필터(super\_high\_freq\_words)**: 위의 단어들 중 빈도수가 상대적으로 높은 단어 64개 선별

위의 필터에는 추출한 단어 리스트를 분류한 것 외에도 추가로 직접 부적절하다 판단되는 다양한 단어들을 인터넷에서 찾아 추가하였다. 아래는 실제 필터 리스트의 모습 일부이다. 파일 형식은 txt이며 순서는 상단 우측부터 하단 좌측까지 위의 8가지 필터 순서와 동일하다.

## 검열 시스템

### 시스템 구현

이제 개발한 인공지능과 필터들을 사용하여 실제로 어떤 텍스트(댓글)이 들어오면 해당 텍스트의 욕설 정도를 수치화하는 시스템을 개발해볼 것이다. 시스템은 다음과 같은 메커니즘으로 작동하여 단순히 댓글에 욕설이 있는지만 필터링 하는 것이 아닌 욕설을 다양한 방법으로 우회하려는 시도도 감지하여 검열할 수 있게 만들었다.

1. 텍스트를 필터로 필터링
2. 고빈도 필터로 추가 필터링
3. 숫자, 공백, 특수문자를 제거한 후 감지 ex) 욕1설, 욕 설, 욕@설 -> 욕설
4. 초성을 제거한 후 감지 ex) 욕ㅇㅇㅇ설, 욕ㄱㄴㄷ설 -> 욕설
5. 초성으로 작성된 욕설 감지 ex) ㅇㅅ -> 욕설
6. 역순으로 작성된 욕설 감지 ex) 설욕 -> 욕설
7. 풀어서 작성된 욕설 감지 ex) ㅇㅛㄱㅅㅓㄹ -> 욕설
8. 발음을 장음화한 욕설 감지 ex) 요오옥서어얼 -> 욕설
9. 일부만 초성으로 작성된 욕설 감지 ex) ㅇ설, 욕ㅅ -> 욕설
10. 인공지능으로 문장 전체의 문맥을 파악 후 부적절 여부 판단

다음과 같은 욕설을 감지해내는 메커니즘을 사람들의 욕설 우회 방식을 분석하고 직접 여러 번 테스트하여 연구를 한 내용을 토대로 완성하였다. 각 단계별로 욕설이 탐지되면 각자 지정된 점수를 합산하여 최종적으로 문장에 대한 부적절 정도를 나타내는 점수를 수치화하여 보여준다. 해당 메커니즘을 파이썬 코드로 구현하기 위해서 파이썬 정규표현식 라이브러리[[49]](#footnote-49)인 re 와 한글에 대한 다양한 기능을 제공해주는 hgtk[[50]](#footnote-50) 라이브러리을 사용했다. 여기서 정규표현식과 re 라이브러리에 대해 자세히 알아보겠다.

### 정규 표현식

정규 표현식(regular expression)이란 특정한 규칙을 지닌 문자열의 집합을 표현하는데 사용하는 형식 언어로 정규식이라고 부르기도 한다. 특정 문자열에서 어떠한 특정 **패턴**을 찾고 싶을 때 사용한다.

기본적으로 패턴 양옆에 슬래시(/)를 붙여서 하나의 패턴임을 표현한다. 또한 대괄호([])를 양옆에 붙이면 대괄호 안의 패턴을 만족하는 문자를 찾는다는 뜻으로 문자 클래스라고 부른다. 다음은 대표적으로 사용되는 메타 문자[[51]](#footnote-51)이다.

* **\*** : 와일드 카드. 모든 문자를 의미한다 ex) [sch\*l] -> school, schl, schooool, schabcdl 등
* **.** : \*과 비슷하지만 하나의 문자만 의미. 개행 문자(\n)을 제외한 모든 문자를 의미
* **^** : 대괄호 안에서 ^는 부정 연산자 not을 의미한다. 대괄호 밖에서는 찾으려는 문자열의 시작 인덱스를 의미한다

ex) [^school] -> school을 제외한 모든 문자열 / ^[school] -> I love school > 7

* **|** : 또는, OR ex) [school|class] -> I like school, I love class, classroom in school 등
* **$** : 찾으려는 문자열의 끝 인덱스
* **\s** : 공백(tab 포함)
* **\d** : 숫자
* **\D** : 숫자를 제외한 문자. 즉 [^\d]
* **\w** : 문자(대문자, 소문자, 언더스코어(\_))
* **-** : 범위 ex) [ㄱ-ㅎ] -> 모든 자음 / [1-8] -> 1부터 8까지의 숫자
* **+** : 문자 하나 이상. 와일드 카드(\*)와는 다르게 비어있는 것을 허용하지 않는다 ex) [scho+l] -> school (x) / school, schooool, schoal (o)
* \ : escape 문자. 위의 문법을 무시하고 텍스트로 인식하고 싶을 때 사용
* **{}** : n, m은 지연수일 때, {n}은 문자 n개, {n,}은 문자 n개 이상, {n, m}은 문자 n개 이상 m개 이하
* 이 외에도 매우 다양한 문법들이 존재한다. [[52]](#footnote-52) [[53]](#footnote-53) [[54]](#footnote-54) [[55]](#footnote-55) [[56]](#footnote-56) [[57]](#footnote-57)

정규표현식을 활용하면 형식이 지정된 텍스트를 쉽게 감지 가능하다. 예를들어 전화번호를 xxx-xxxx-xxxx 형식으로 입력 받는다 했을 때 정규표현식으로 *^[\d{3}-\d{4}-\d{4}]$* 으로 표현 가능하다.

### re 라이브러리

위의 정규 표현식을 파이썬에서 사용하기 위해선 re 라이브러리가 필요하다. 다음은 대표적인 Method[[58]](#footnote-58)이다.

* **match**: 문자열의 처음부터 정규식과 매치
* **fullmatch**: 문자열의 처음부터 끝까지 완벽하게 정규식과 일치
* **search**: 문자열 중 일부가 정규식과 매치. 여러 개가 있을 경우 제일 앞에 있는 부분만 반환
* **sub**: 문자열에서 정규식과 매치되는 부분을 교체
* **split**: 정규식과 매치되는 부분을 분할

**1) 시스템 구현**에서 가장 많이 사용한 메서드는 sub이다. 텍스트를 전처리할 때 즉 텍스트에서 일정 부분을 제거하고 싶을 때 사용했으므로 sub을 이용해서 정규식과 매치되는 부분을 제거하여 시스템을 구현했다. sub 메서드의 필수 파라미터는 정규식 패턴, 교체 문자열, 검사할 문자열이 있다. 예를 들어 숫자를 모두 #으로 교체하고 싶을 때 *re.sub([*\*d], ‘#’, text)* 로 표현한다. [[59]](#footnote-59) [[60]](#footnote-60) [[61]](#footnote-61) [[62]](#footnote-62)

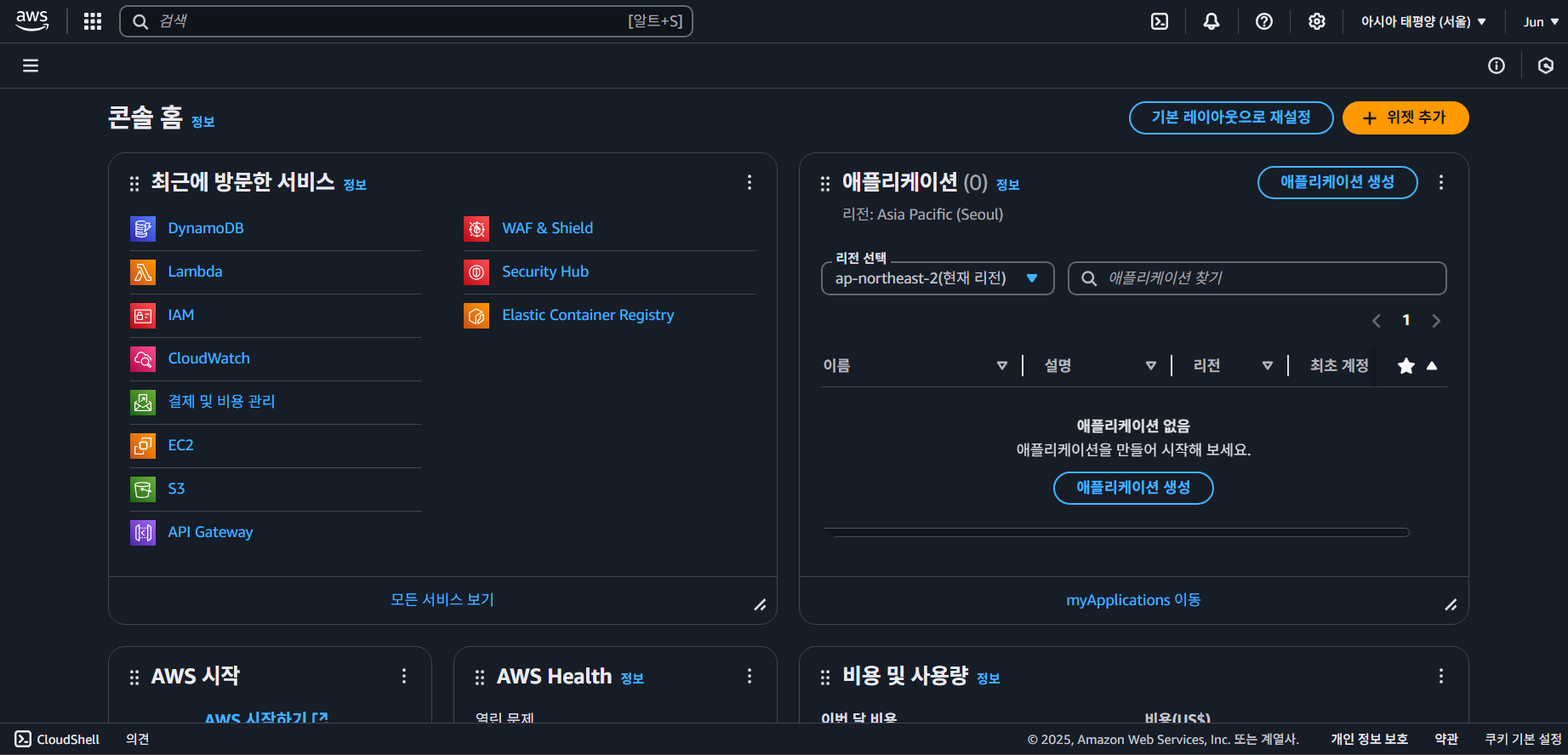
## API 개발

### AWS (Amazon Web Serviece)

우선 API를 구축하여 자유자재로 호출하기 위해선 **III. API (Application Programming Interface)**에서 언급했듯이 **서버**가 필요하다. 그러나 서버를 24시간 구동 시키려면 매월 지출액이 최소 몇십 만원이다. 고작 확장프로그램을 개발하겠다고 이러한 돈을 수익 모델도 없이 지출하는 건 말이 안된다. 이러한 문제를 해결하기 위해 다양한 방법을 찾아보다가 아마존 웹 서비스(**A**mazon **W**eb **S**erviece, 이하 AWS)에서 제공하는 서버리스 아키텍쳐인 Lamda의 존재를 알게되었다.

AWS는 인터넷 쇼필몰로 유명한 Amazon의 자회사로 세계 3대[[63]](#footnote-63) Iaas 플랫폼 중 하나이다. 서버를 제공하는 등 다양한 클라우드 컴퓨팅 서비스가 존재한다. 다음은 대표적인 서비스 목록이다.

* EC2(Elastic Compute Cloud): 컴퓨팅 서버
* **Lambda**: 이벤트가 발생하여 호출될 때만 코드를 실행하는 웹 엔진 서비스
* S3(Simple Storage Service): 웹 스토리지 서비스
* **DynamoDB**: SQL 데이터베이스
* **API Gateway**: HTTP 요청으로 람다 호출, 프록시 등 여러 작업을 수행할 수 있는 서비스



위는 AWS 콘솔 모습이다. 콘솔에서 다양한 서비스를 제어할 수 있다.

### API 구축

이제 API를 실제로 개발해보겠다. 앞서 언급했듯 API가 구동되는 서버는 AWS의 Lambda(이하 람다)를 사용할 것이다. 람다는 실제 서버를 구동하는게 아닌 호출할 때 마다 새로운 인스턴스 환경을 생성하므로 기존 ‘서버’와는 다르다. 즉 물리적인 서버가 없다[[64]](#footnote-64). 이러한 점은 API를 호출할 때 아주 약간의 딜레이가 발생한다는 단점도 있지만 매번 새로운 환경을 생성하므로 동기적으로 여러 요청들을 처리할 수 있고, 이전 요청과 독립적인 환경이 구축되므로 충동이 없다는 장점도 있다. 또한 무엇보다 가격이 24시간 하루 종일 작동되어야 하는 컴퓨팅 서버와는 다르게 호출될 때 잠깐 실행되고 닫히기 때문에 매우 저렴하다. 이러한 여러점들을 종합하여 고려해본 결과 람다를 이용하여 API를 구축하기로 결정하였다.

우선 API를 구동시킬 람다를 생성해보겠다. AWS 콘솔에서 람다 함수를 새로 생성하려면 다음과 같은 옵션들을 설정해줘야 한다.

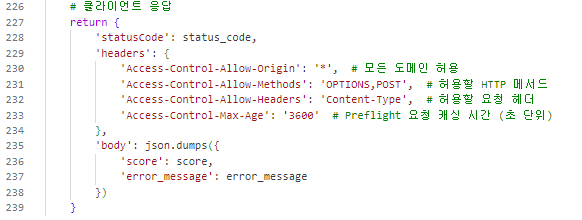


위 화면을 보면 함수 이름, 런타임, 아키텍쳐, 권한, 그 외 추가 구성 옵션들로 구성되어 있다. 개발할 API는 파이썬 기반으로 작동하므로 우선 런타임을 Python으로 설정해줬다. 아키텍쳐의 경우 보편적으로 사용하는 x86\_64[[65]](#footnote-65)를 사용하였다. 여기서 추가로 API를 호출하기 위한 엔드포인트를 구성하기 위해 ‘함수 URL 활성화’ 옵션도 켜줘야한다. 권한은 URL을 지닌 모두가 호출할 수 있도록 Public API로 생성할 것이다.

다음으로 람다 함수에 실행 코드를 작성해야 한다. 람다 함수를 올바르게 실행하기 위해서는 핸들러를 설정해줘야 한다. 여기서 람다 함수 파일명은 lambda\_function.py로 설정했고, 핸들러 함수명은 lambda\_handler()로 설정했다. 만약 누군가 함수 URL을 통해 람다를 호출한다면 이는 lambda\_finction.py 내부에 있는 lambda\_handler() 함수를 호출한것과 같다. 핸들러 함수에서 요청을 받으면 이를 파이썬으로 해석할 수 있어야 한다. 요청은 http 프로토콜로 들어오며 header, body로 구분되어 있다. 여기서 API가 실제로 필요로하는 데이터는 body에 들어있다.

body = event.get('body') 로 요청 데이터의 body를 가져온다. 이후 이를

body\_data = json.loads(body) 로 기존의 json 형식으로 작성된 body를 파이썬이 해석할 수 있도록 가져와 body\_data로 저장한다. 이후 기존에 개발한 검열 시스템 코드를 사용하여 동일한 방식으로 처리를 한 다음 결과를 응답한다. 결과의 응답은 핸들러 함수의 return[[66]](#footnote-66)으로 시행한다.

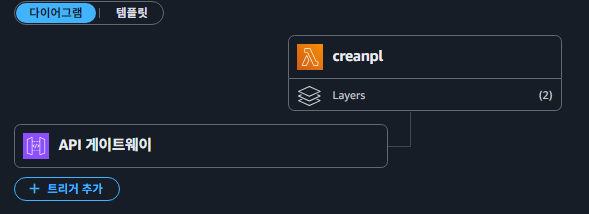


위 코드가 응답 코드이며 응답 결과를 알리는 상태 코드[[67]](#footnote-67)(status code)와 CORS 정보가 담긴 headers, 실행 결과(욕설 여부 점수 데이터)가 담긴 body로 구성되어 있다.

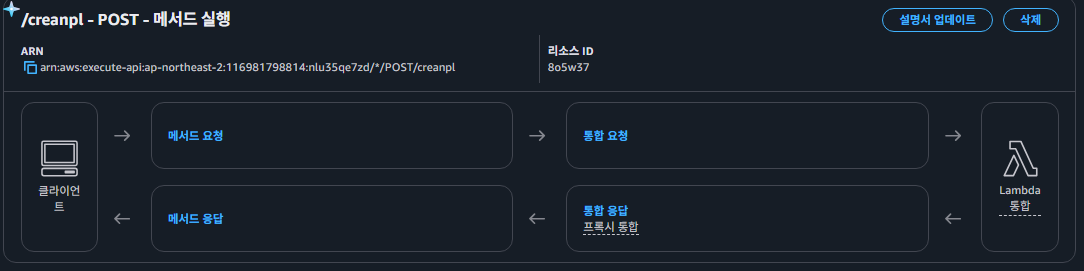
구축된 람다 API는 다음과 같이 작동한다

1. 클라이언트에서 요청을 보낸다. 이때 headers에는 CORS 정보가, body에는 검사 할 텍스트 데이터와 필터 정보(리스트 형태로 각 인덱스가 각 필터를 의미, 1이면 필터를 on / 0이면 팔터를 off)가 포함되어 있다
2. 람다는 요청을 받아 핸들러 함수를 호출한다
3. 핸들러 함수에 event 파라미터로 들어온 데이터 중 body 데이터를 사용하여 옵션에 맞게 텍스트를 검사하여 검열 여부를 결정한다
4. 해당 텍스트에 욕설 정도가 어느정도 인지를 자연수 점수로 책정하여 결과값을 반환한다
5. 클라이언트에 반환값으로 응답한다

여기다 API GateWay(이하 게이트)를 추가하여 엔드포인트를 수정해보겠다. 먼저 다음과 같이 트리거[[68]](#footnote-68)를 설정한다.



이제 게이트에서 요청을 받으면 람다를 호출하도록 만들어 보자. 우선 API의 요청은 POST 요청을 사용할 것이다. 게이트를 이용한 POST 요청은 다음과 같은 방법으로 실행된다.

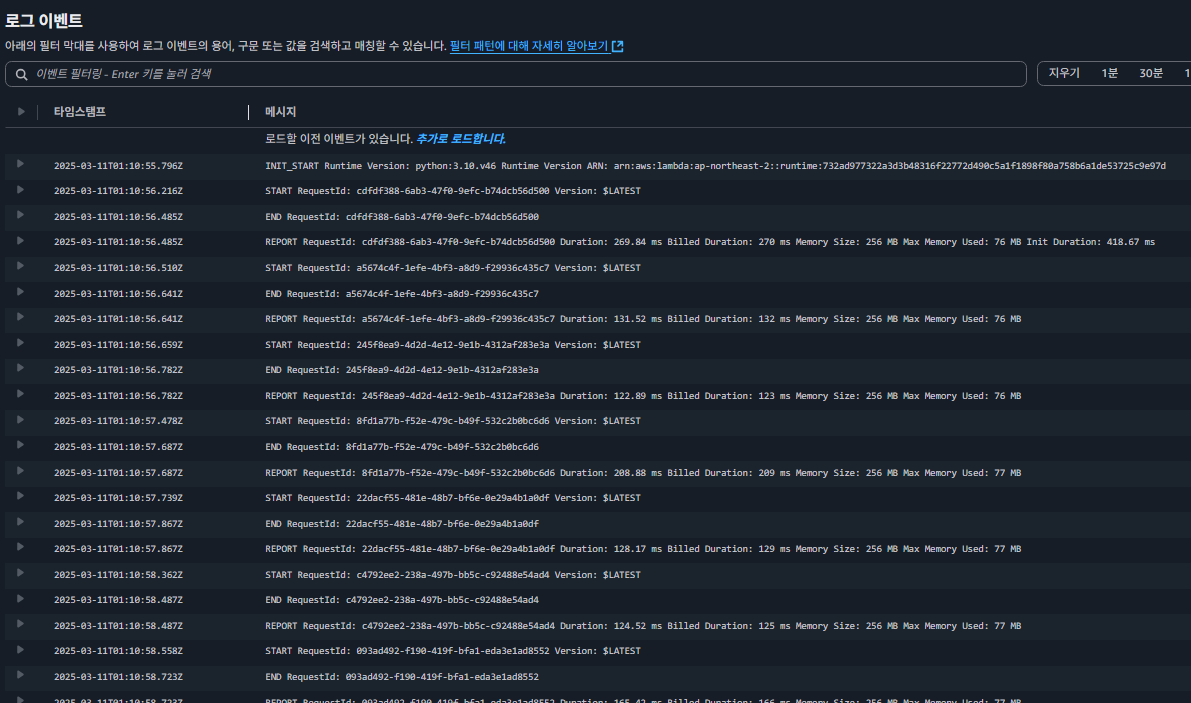


POST 요청을 받으면 게이트를 거쳐 람다를 호출하여 요청을 처리한 다음 다시 게이트를 거쳐 클라이언트에게 결과를 응답하게 된다. 현재는 설정해두지 않았으나 CORS를 비롯한 요청, 경로, 파라미터, 쿼리, 헤더 등등에 대한 정책 설정이 가능하며 API 사용량 통계나 API Key 생성 및 권한 인증 등 기존 람다로는 한계가 있던 다양한 작업들을 수행 가능하다.

### 로깅 (Logging)

이번엔 API를 호출할 때 마다 로그를 생성해보겠다. 로그(Log)란 기록이라는 의미로 전반적인 모든 활동에 대한 기록을 의미한다. 쉽게 말하자면 컴퓨터로 어떤 웹사이트에 접속하여 마우스를 어떻게 움직이고, 어떤 페이지에 몇 초간 머물고, 마우스 스크롤을 얼마나 하고, 검색은 무엇을 하고, 어떤 상품을 구매하였는지 등 다양한 정보가 모두 로그이다. 이 외에도 사용자의 활동이 아닌 기기 자체의 활동도 모두 로그로 저장하며 모든 전자기기는 무조건 로그를 기록한다. 이러한 로그 데이터는 단순 AS나 통계를 넘어 빅데이터와 마케팅 분야에서도 활용된다.

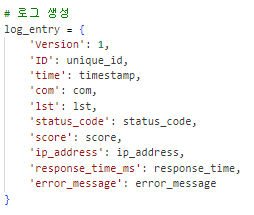
로그 기록은 람다를 생성하면 자동으로 AWS에서 CloudWatch라는 서비스를 이용하여 로깅을 해준다. 그러나 아래와 같이 로그 내용을 인간이 보기엔 매우 불편하다.



원하는 정보를 깔끔하게 제공해주지 않는다. 즉 통계 처리가 불편하다. 이러한 문제를 해결하기 위해 AWS의 데이터베이스인 DynamoDB(이하 디비)를 사용할 것이다. 디비는 SQL기반으로 작동한다.

우선 디비를 새로 생성하기 위해 파티션 키(partition key)와 정렬 키(sort key)를 지정해줘야 한다. 두 키는 기본 키(primary key)로 각각 데이터 검색과 정렬에 사용된다. 즉 DB에서 데이터를 검색하기 위해 party를 구분하는 기준을 파티션 키로, 열을 정렬하기 위한 기준으로 정렬 키를 미리 지정해두는 것이다. 해당 설정은 변경할 수 없다. 우선 파티션은 API의 버전 업데이트 가능성을 위해 각 버전별로 구분하기 위해 API 버전을 파티션[[69]](#footnote-69)으로, 시간 순으로 정렬하기 위해 생성된 ID를 정렬키로 설정하였다. 여기서 ID는 고유하게 생성되며 시간 관계가 드러나도록 1970년부터 지금까지 흘러간 경과 시간[[70]](#footnote-70)(ms)를 이용하여 생성한다. 단위가 ms이므로 요청이 0.001초 사이 동시에 들어오지 않는 이상 고유성이 깨질 일은 없다.

기본 키 이외에



와 같은 요소들이 로깅된다. 위에서부터 차례대로 버전, ID, 요청 시간, 요청받은 텍스트 데이터(댓글), 필터 옵션 리스트, 응답 코드, 결과값(점수), 클라이언트 IP 주소, 응답하는데 소요된 시간, 에러 메시지 이다.

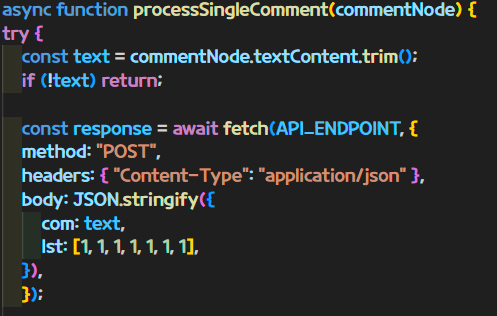
람다에서 디비에 데이터를 쓰거나 읽기 위해서는 AWS에서 스토리지와 파이썬간 연동을 위해 미리 만들어진 boto3 라이브러리를 사용해야 한다.

## 확장프로그램 개발

이제 위의 API를 사용하여 확장프로그램을 만들어 보겠다.

먼저 content.js에서 사용자가 현재 보고 있는 페이지에서 텍스트 요소들을 가져와서 이를 요소별로 API를 이용하여 욕설이 들어있는지를 판단한다. 이때 점수가 일정 수치(10점)를 넘긴다면 해당 텍스트를 ‘욕설’로 간주하고 검열한다. 검열은 해당 텍스트를 [차단된 내용입니다]로 변경시킨다. 사용자의 화면이 변할 때 마다(페이지 이동, 동일 페이지 내 요소 변동 모두 포함) 이를 감지하여 새롭게 각 요소들을 검사하여 검열한다. 만약 변동사항이 없는 요소의 경우 API를 재호출하여 리소스와 시간을 잡아먹는 일을 줄이고자 건너 뛰도록 설계하였다.

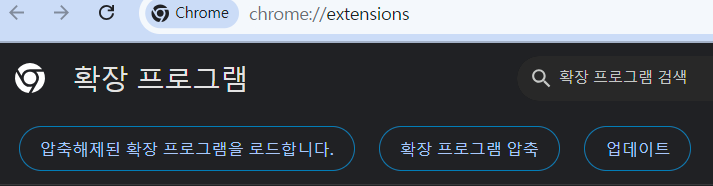
확장프로그램에서 API로 요청을 보내는 코드는 다음과 같다.



API 엔드포인트에 POST 요청을 보낸다. 이때 json 형식으로 body를 구성하여 요소 텍스트(댓글)과 필터 옵션(모든 필터 on) 리스트를 포함한다.

아직 실제로 배포하기엔 비용 문제를 비롯하여 여러 문제점들이 있어 배포는 추후에 기회가 된다면 하도록 하겠다. 테스트용으로 개발하였으므로 댓글 검열은 현재 네이버 뉴스의 댓글 더보기 페이지[[71]](#footnote-71)에서만 되도록 설정되어 있다. 팝업 UI는 간단하게 On/Off 할 수 있는 체크박스만 배치해두었다. 체크를 해제하면 확장프로그램도 작동하지 않게된다.

확장프로그램 설치법은 다음과 같다.

1. 크롬 또는 웨일 브라우저에서 chrome://extensions/에 접속한다
2. 

‘압축해제된 확장 프로그램을 로드합니다’를 클릭한다

1. [구글 드라이브](https://drive.google.com/file/d/1XB7jyKDl1jF5_ZEaAFCdyToY_p-JW0T4/view?usp=sharing)에서 다운로드한 폴더를 압축 해제하여 업로드한다

현재 인공지능 모델의 경우 람다 함수의 코드내에 삽입시 API 호출이 오래걸리는 cold start문제가 발생하여 필터 기반 검열만 사용중인 상태이다. 이 문제는 해결할 수 있지만 비용적인 문제로 하지 않겠다.

확장 프로그램의 이름은 creanpl로 정하였다. 깨끗한을 의미하는 clean과 댓글을 의미하는 reply를 합하여 만들었다. 또한 확장프로그램 아이콘 또한 직접 디자인했다.

## 테스트

확장 프로그램을 실제로 사용하여 잘 작동하는지 테스트해보도록 하겠다.

 [[72]](#footnote-72)

확장 프로그램 Off (좌) / 확장 프로그램 On (우) 이다. 어린이에게 부적절하다고 판단되는 댓글을 자동으로 검열해준다. 점수를 10점까지 허용해주므로 부적절한 댓글이 모두 완벽하게 검열되지는 않는다. 그렇다고 이 점수 기준치를 너무 낮춰버리면 댓글이 검열되는 빈도 수가 너무 많아져서 웹 서핑이 힘들어 질것이다. 이러한 점수 기준 옵션과 필터별 사용 여부 등 다양한 옵션은 추후 확장프로그램의 팝업 UI에서 On/Off 외에도 다양한 옵션을 추가한다면 위의 문제들을 해결할 수 있을것이다.

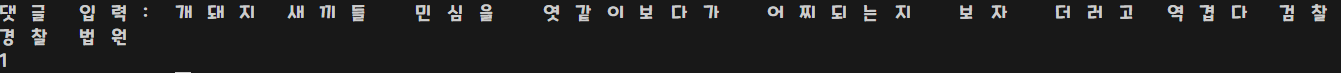
아래는 위 댓글들을 검열한 로그 기록의 일부이다



일부 요소들은 숨겨둔 상태이다. API로 검사한 댓글들이 다양한 정보들과 함께 문제없이 잘 로깅되고 있는 것을 확인할 수 있다. 문맥을 파악하여 검열하는 인공지능 모델이 현재 API상 빠져있어서 단순 필터 기반 검열이라 아직 성능은 미흡하다.

*개돼지 새끼들 민심을 엿같이보다가 어찌되는지 보자 더러고 역겹다 검찰 경찰 법원*

위 댓글의 경우 점수가 22점으로 검열되었다. 누가봐도 어린이에게 보여주고 싶지 않은 댓글로 욕설을 잘 탐지해내었다 할 수 있다.



해당 댓글을 인공지능 모델로 예측해봐도 1(욕설)이라는 올바른 결과가 출력된다.

*ㄹㅇㅋㅋ 즈그 빨개이 수장은 가만히 놔둠 ㅋㅋㅋ*

위 댓글은 점수가 0으로 측정되었다. 그러나 어린이에게 보여주기엔 부적절해 보인다.



인공지능은 올바르게 1로 예측해냈다.

*당신은 안중근 같은 애국자 입니다*

위 댓글의 점수는 0으로 측정되었다.

 인공지능 모델 또한 0으로 예측했다.

이로써 개발한 확장프로그램이 인공지능 모델이 포함된다면 처음 목적에 맞게 잘 작동한다고 할 수 있다. 직접 댓글을 작성해서 API가 잘 작동하는지 테스트하려면 [다음 사이트](https://lrc2ppt.kr/creanpl_api_test/api_test_web.html)에 접속하여 직접 체험해볼 수 있다.

## 코드

API 코드와 확장 프로그램 및 인공지능 모델 예측 코드는 [깃허브](https://github.com/jjy0809/creanpl)에 업로드 되어 있다. 그 중 본 탐구의 가장 핵심 코드인 API 코드는 [여기서](https://github.com/jjy0809/creanpl/blob/main/lambda_function.py) 볼 수 있다.

# 결론

본 탐구에서는 확장프로그램과 API에 대해 알아보고 이전 탐구 활동에서 개발한 ‘욕설 탐지 인공지능 모델’을 가지고 후속탐구로서 ‘댓글 검열 확장 프로그램’을 개발해보았다.

확장 프로그램이 무엇인지와 어떻게 개발하는지, API가 무엇이고 어떤 종류가 있는지, REST API는 무엇이며 요청을 보내는 HTTP Method에는 어떤 것들이 있는지, API 인증 방법들과 CORS 정책이 무엇인지, 기존의 인공지능을 활용하여 어떠한 매커니즘을 거쳐 욕설 검열 시스템을 개발했는지, AWS Lambda와 DynamoDB를 이용하여 어떻게 API를 구축하였는지, API를 사용하여 웹 페이지상의 댓글을 검열하는 확장 프로그램을 개발하는 과정은 어떠하였는지, 그리고 그 결과는 어떠한지를 알아보고 직접 실습해 보았다.

비용상의 문제로 실제 배포는 아직 해보진 못했지만 나중에 기회가 된다면 실제로 사람들이 이용할 수 있도록 무료로 배포할 계획이다. 많은 사람들이 이 확장 프로그램으로 조금 더 쾌적한 환경에서 서핑을 즐길 수 있었으면 좋겠다.

비록 비용 문제상 API에 인공지능 모델까지는 이식하지 못하여 확장프로그램의 성능이 실제 성능보다 떨어졌으나, 테스트 과정에서 대체로 문제가 있는 부적절한 댓글을 잘 검열하였고, 또한 확장프로그램으론 검열하지 못했어도 인공지능 모델로 외부에서 따로 예측을 하였을 땐 잘 검열하는 모습을 보여줘 API에 인공지능 모델을 이식할 수 있다면 실제 원했던 제성능이 나올것으로 기대되어 목표를 달성했다고 생각한다.

거의 10개월가량 진행된 욕설 검열 확장프로그램 ‘creanpl’ 프로젝트로 과정 중간중간에 여러 문제점들로 힘든점도 많았지만 그 결과물이 의도대로 잘 작동하는 것을 보니 뿌듯하고, 나의 결과물로 다른 사람이 도움을 받을 수 있다 생각하니 보람되는 것 같다. 몇시간 혹은 며칠동안 하나의 문제를 해결하겠다고 계속 고민하고 여러 방법을 시도하고 공부하여 이러한 결과물을 만들어 낼 수 있다는 사실을 알게 되어 앞으로도 포기하지 않고 꾸준히 노력하면 원하는 목표를 이룰 수 있음을 직접 느낄 수 있었다.

1. 인터넷망에서 정보를 검색하기 위한 프로그램으로 웹 검색, 방문 기록, 북마크 등의 다양한 기능들을 제공하고 있다. 대표적으로 크롬, 엣지, 웨일, 파이어폭스 등이 있다. 최근에는 브라우저에서 비밀번호/개인정보 저장, 광고/트레킹 차단, 웹 문서 내용 요약, 인공지능 비서 등 더욱 고차원적인 편의 기능들을 제공하고 있다 ([웹 브라우저, 네이버 지식백과](https://terms.naver.com/entry.naver?docId=3571813&cid=59088&categoryId=59096)) [↑](#footnote-ref-1)
2. [브라우저 확장, 위키피디아](https://ko.wikipedia.org/wiki/%EB%B8%8C%EB%9D%BC%EC%9A%B0%EC%A0%80_%ED%99%95%EC%9E%A5) [↑](#footnote-ref-2)
3. JavaScript Object Notation. 자바스크립트 기반 저장 형식 또는 확장자. 다양한 분야에서 데이터를 처리할 때 사용하며, 인간과 컴퓨터 모두 json 데이터를 쉽게 이해할 수 있다는 장점이 있음 ([JSON이란?, Oracle](https://www.oracle.com/kr/database/what-is-json/)) [↑](#footnote-ref-3)
4. 다른 데이터를 정의하고 기술하는 데이터 또는 다양한 형식의 다른 데이터의 내용 또는 구조를 설명하는 데이터. 즉 데이터를 설명하는 데이터 ([메타데이터, TTA정보통신용어사전](https://terms.tta.or.kr/dictionary/dictionaryView.do?subject=%EB%A9%94%ED%83%80+%EB%8D%B0%EC%9D%B4%ED%84%B0)) [↑](#footnote-ref-4)
5. [매니페스트 파일, 위키피디아](https://ko.wikipedia.org/wiki/%EB%A7%A4%EB%8B%88%ED%8E%98%EC%8A%A4%ED%8A%B8_%ED%8C%8C%EC%9D%BC) [↑](#footnote-ref-5)
6. 해당 사항은 매니페스트 파일에 기재된다. 작동 조건, 실행 가능한 사이트(도메인), 실행 방식 등이 있다. [↑](#footnote-ref-6)
7. <https://www.apisec.ai/blog/rest-api-and-its-significance-to-web-service-providers> [↑](#footnote-ref-7)
8. [API란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/api/) [↑](#footnote-ref-8)
9. 서비스 초기 단계부터 아마존의 모든 웹 사이트의 기능들을 모두 각자의 독립된 API로 구축하여 API를 상호적으로 호출하며 웹 페이지를 동작시켰다고 한다. 아를 서비스 지향 아키텓쳐(SOA)라고 한다 ([아마존 웹 서비스, 3, 배경과 탄생, 나무위키](https://namu.wiki/w/%EC%95%84%EB%A7%88%EC%A1%B4%20%EC%9B%B9%20%EC%84%9C%EB%B9%84%EC%8A%A4#s-3)) [↑](#footnote-ref-9)
10. [API란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/api/) [↑](#footnote-ref-10)
11. **S**imple **O**bject **A**ccess **P**rotocol. 확장성 마크업 언어(XML)와 하이퍼텍스트 전송 규약(HTTP) 등을 기반으로 하여 다른 컴퓨터에 있는 데이터나 서비스를 호출하기 위한 통신 규약 ([단순 객체 접근 프로토콜, 정보통신용어사전](https://terms.tta.or.kr/dictionary/dictionaryView.do?subject=%EB%8B%A8%EC%88%9C+%EA%B0%9D%EC%B2%B4+%EC%A0%91%EA%B7%BC+%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C)) [↑](#footnote-ref-11)
12. **Ex**tensible **M**arkup **L**anguage. 데이터를 정의하는 규칙을 제공하는 마크업 언어. 컴퓨터와 인간 모두 쉽게 이해할 수 있으며 형식이 동일한 마크업 언어인 HTML과 비슷하지만, 독자적으로 어떠한 기능을 수행하는 컴퓨터 언어로 사용될 수는 없는 단순 데이터 저장용 언어이다 ([XML이란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/xml/)) [↑](#footnote-ref-12)
13. <https://ideatec.co.kr/ipaas_news_/?bmode=view&idx=6617531> [↑](#footnote-ref-13)
14. [RPC와 REST의 차이점은 무엇인가요?, AWS](https://aws.amazon.com/ko/compare/the-difference-between-rpc-and-rest/) [↑](#footnote-ref-14)
15. <https://f-lab.kr/insight/websocket-vs-rest-api> [↑](#footnote-ref-15)
16. [REST API(레스트 API)란?, Red Hat](https://www.redhat.com/ko/topics/api/what-is-a-rest-api) [↑](#footnote-ref-16)
17. [RESTful API란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/restful-api/) [↑](#footnote-ref-17)
18. [API란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/api/) [↑](#footnote-ref-18)
19. [[서버] REST API란?, 하나몬](https://hanamon.kr/rest-api/) [↑](#footnote-ref-19)
20. <https://velog.io/@awesome-hong/HTTP-Options-%EC%9A%94%EC%B2%AD%EC%9D%80-%EB%AD%98%EA%B9%8C> [↑](#footnote-ref-20)
21. <https://inpa.tistory.com/entry/WEB-%F0%9F%8C%90-HTTP-%EB%A9%94%EC%84%9C%EB%93%9C-%EC%A2%85%EB%A5%98-%ED%86%B5%EC%8B%A0-%EA%B3%BC%EC%A0%95-%F0%9F%92%AF-%EC%B4%9D%EC%A0%95%EB%A6%AC> [↑](#footnote-ref-21)
22. [RESTful API란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/restful-api/) [↑](#footnote-ref-22)
23. [OAuth란?, CloudFlare](https://www.cloudflare.com/ko-kr/learning/access-management/what-is-oauth/) [↑](#footnote-ref-23)
24. [API 보안이란?, CloudFlare](https://www.cloudflare.com/ko-kr/learning/security/api/what-is-api-security/) [↑](#footnote-ref-24)
25. <https://inpa.tistory.com/entry/WEB-%F0%9F%93%9A-CORS-%F0%9F%92%AF-%EC%A0%95%EB%A6%AC-%ED%95%B4%EA%B2%B0-%EB%B0%A9%EB%B2%95-%F0%9F%91%8F> [↑](#footnote-ref-25)
26. [CORS란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/cross-origin-resource-sharing/) [↑](#footnote-ref-26)
27. Img 태그를 이용하여 abc라는 출처에서 이미지 데이터를 가져와 표시하는 HTML 코드이다. 그러나 abc라는 출처는 올바르지 않기 때문에 Error가 발생 할 것이고, 그렇게 된다면 onerror가 실행되어 해당 태그의 src를 해킹 사이트로 변경시켜 해킹 사이트로부터 사용자 데이터를 탈취하는 코드로 변하게 된다 [↑](#footnote-ref-27)
28. 해당 방식은 SQL Injection 공격과도 비슷한 방법이다. SQL Injection은 SQL DB에서 데이터를 탈취하기 위해 비정상적인 문자열을 대입하여 코드로써 동작하게 하고, XSS는 웹 페이지 상에 악성 코드를 삽입하기 위해 비정상적인 문자열을 웹 페이지 어딘가에 삽입하여 코드로써 동작하게 한다 [↑](#footnote-ref-28)
29. [사이트 간 스크립팅, 위키피디아](https://ko.wikipedia.org/wiki/%EC%82%AC%EC%9D%B4%ED%8A%B8_%EA%B0%84_%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8C%85) [↑](#footnote-ref-29)
30. 물론 CORS가 아니더라도 사용자의 입력을 HTML의 코드로써 인식하지 못하도록 조치를 취해둔다면 공격자가 비정상 문자열을 삽입하더라도 코드가 아닌 텍스트로만 인식되어 작동되지 않도록 할 수 있다 [↑](#footnote-ref-30)
31. <https://inpa.tistory.com/entry/WEB-%F0%9F%93%9A-CORS-%F0%9F%92%AF-%EC%A0%95%EB%A6%AC-%ED%95%B4%EA%B2%B0-%EB%B0%A9%EB%B2%95-%F0%9F%91%8F> [↑](#footnote-ref-31)
32. [CORS란 무엇인가요?, AWS](https://aws.amazon.com/ko/what-is/cross-origin-resource-sharing/) [↑](#footnote-ref-32)
33. [교차 출처 리소스 공유 (CORS), Mozilla](https://developer.mozilla.org/ko/docs/Web/HTTP/Guides/CORS) [↑](#footnote-ref-33)
34. <https://inpa.tistory.com/entry/WEB-%F0%9F%93%9A-CORS-%F0%9F%92%AF-%EC%A0%95%EB%A6%AC-%ED%95%B4%EA%B2%B0-%EB%B0%A9%EB%B2%95-%F0%9F%91%8F> [↑](#footnote-ref-34)
35. [이전 탐구 보고서](https://github.com/jjy0809/creanpl/blob/main/%EC%9A%95%EC%84%A4%20%EA%B2%80%EC%97%B4%20%EC%9D%B8%EA%B3%B5%EC%A7%80%EB%8A%A5%20%EB%AA%A8%EB%8D%B8%20%EA%B0%9C%EB%B0%9C%20%ED%83%90%EA%B5%AC.pdf) 참고 [↑](#footnote-ref-35)
36. 해당 가정이 문제가 없다는 근거는 이전 탐구 참고 [↑](#footnote-ref-36)
37. 고리대금업자 [↑](#footnote-ref-37)
38. 딜러 [↑](#footnote-ref-38)
39. 돈을 딴 사람이 잃은 사람에게 주는 차비 개념의 소량의 돈. 개평이라고도 부른다 [↑](#footnote-ref-39)
40. 메스암페타민. 필로폰, 히로뽕, 작대기 등으로도 불림 [↑](#footnote-ref-40)
41. 대마. 잡초, 단풍 등으로도 불림 [↑](#footnote-ref-41)
42. GHB. 흔히 데이트 폭력 약물로 알려져 있음 [↑](#footnote-ref-42)
43. 윤석열 前 대통령 취임 당시 청와대 이전을 시행한 윤석열을 이르는 멸칭 [↑](#footnote-ref-43)
44. 이재명. [해당 논란](https://namu.wiki/w/%EC%9D%B4%EC%9E%AC%EB%AA%85%20%ED%98%95%EC%88%98%20%EC%9A%95%EC%84%A4%20%EB%85%BC%EB%9E%80) 참고 [↑](#footnote-ref-44)
45. 카테고리는 일베(일간베스트 커뮤니티)이지만 타 커뮤니티 은어도 포함됨 [↑](#footnote-ref-45)
46. 개념글 추천. 디시 용어 [↑](#footnote-ref-46)
47. 故 노무현 前 대통령 생전 노무현 지지자들이 사용하던 애칭. 사후 일베에서 노무현을 지칭할 때 사용하면서 의미가 변질됨 [↑](#footnote-ref-47)
48. 노무현 대통령이 퇴임 후 자신의 고향인 봉하를 소개하는 영상에서 부엉이 바위를 소개하며 부엉이 울음소리를 흉내 낼 때 말한 표현. 이후 노무현이 부엉이 바위에서 투신한 뒤 일베에서 고인모독의 목적으로 사용 [↑](#footnote-ref-48)
49. 라이브러리, 모듈, 패키지는 서로 다른 의미이지만 본 탐구에서는 모두 라이브러리로 통일하여 사용하겠다. 실질적으로 개발자들도 구분 없이 혼용하여 사용하며 구분하는게 딱히 의미가 없다 [↑](#footnote-ref-49)
50. [hangul-toolkit, 깃허브](https://github.com/bluedisk/hangul-toolkit) [↑](#footnote-ref-50)
51. 문자를 나타내는 문자. 메타데이터와 비슷한 개념 [↑](#footnote-ref-51)
52. <https://velog.io/@ninto_2/%EC%A0%95%EA%B7%9C%ED%91%9C%ED%98%84%EC%8B%9D> [↑](#footnote-ref-52)
53. <https://wikidocs.net/4308> [↑](#footnote-ref-53)
54. <https://blog.naver.com/wjddl7899/223816157246> [↑](#footnote-ref-54)
55. [정규 표현식, Mozila](https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Regular_expressions) [↑](#footnote-ref-55)
56. [정규 표현식, 나무위키](https://namu.wiki/w/%EC%A0%95%EA%B7%9C%20%ED%91%9C%ED%98%84%EC%8B%9D) [↑](#footnote-ref-56)
57. [정규 표현식, 위키피디아](https://ko.wikipedia.org/wiki/%EC%A0%95%EA%B7%9C_%ED%91%9C%ED%98%84%EC%8B%9D) [↑](#footnote-ref-57)
58. 메서드. 특정 함수, 클래스, 모듈, 라이브러리, 패키지에서 내부에 있는 함수를 의미. foo 라이브러리 내의 bar 함수를 호출하고 싶다면 foo**.**bar() 로 표현한다. [↑](#footnote-ref-58)
59. <https://yganalyst.github.io/data_handling/memo_6/> [↑](#footnote-ref-59)
60. <https://whatisthenext.tistory.com/116> [↑](#footnote-ref-60)
61. <https://engineer-mole.tistory.com/189> [↑](#footnote-ref-61)
62. <https://docs.python.org/3.13/library/re.html#functions> [↑](#footnote-ref-62)
63. **AWS (Amazon) – 30%,** Azure (MS) – 21%, GCP (Google) – 12% / 2024 [점유율](https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/) 순 [↑](#footnote-ref-63)
64. 이를 서버리스 아키텍쳐라고 한다 [↑](#footnote-ref-64)
65. 보편적으로 사용하는 Intel, AMD CPU가 해당 아키텍쳐에 해당한다 [↑](#footnote-ref-65)
66. 파이썬에서 함수의 끝을 알리는 코드로 return 0 은 함수가 종료되고 0을 반환한다는 뜻이다. return이 나오면 해당 함수는 무조건 종료된다 [↑](#footnote-ref-66)
67. http 요청에서 요청이 어떻게 처리되었는지 여부를 3자리 숫자로 표현한 것. 표준으로 정해진 코드가 존재한다. 예를들어 200은 요청 처리 성공, 404는 찾을 수 없음, 500은 내부 서버 오류이다 (<https://hongong.hanbit.co.kr/http-%EC%83%81%ED%83%9C-%EC%BD%94%EB%93%9C-%ED%91%9C-1xx-5xx-%EC%A0%84%EC%B2%B4-%EC%9A%94%EC%95%BD-%EC%A0%95%EB%A6%AC/>) [↑](#footnote-ref-67)
68. 람다 이벤트를 생성 할 트리거로 함수 URL 호출 외에도 여러 트리거를 생성할 수 있다 [↑](#footnote-ref-68)
69. 아직까지 API 버전이 교체된 적은 없어서 의미는 없다. 본 탐구에서는 파티션을 나눌 이유가 없어 파티션 키가 무엇인지가 상관이 없다. [↑](#footnote-ref-69)
70. 파이썬 내장 라이브러리인 time 라이브러리에서 기본 매서드인 tume() 매서드가 작동하는 원리이다. 이를 그대로 사용하여 ID를 생성한다 [↑](#footnote-ref-70)
71. <https://n.news.naver.com/article/comment/> 뒤에 뉴스 ID를 첨부해야 됨 [↑](#footnote-ref-71)
72. 2025년 2월 13일에 [어느 뉴스 기사](https://n.news.naver.com/article/comment/021/0002690010)의 댓글 중 일부 [↑](#footnote-ref-72)