

A screenshot of a web browser window titled "VidE". The address bar shows the URL "127.0.0.1:5000". The top navigation bar includes links for "HOME" and "ARCHIVE", and a button for "Upload New Video". On the right side of the header, there are several icons and a "업데이트" (Update) button. The main content area features a large blue "VidE" logo. Above the logo, the text "ICT융합캡스톤디자인1 프로젝트 발표" (ICT integrated capstone design 1 project presentation) is displayed. Below the logo, the text ": 집중도 높은 영상 제작을 위한 편집 애플리케이션" (A video editing application for high concentration video production) is shown. In the bottom right corner, there is a list of names: 김진아, 이윤지, 고지수, and 임주연.

ICT융합캡스톤디자인1 프로젝트 발표

VidE

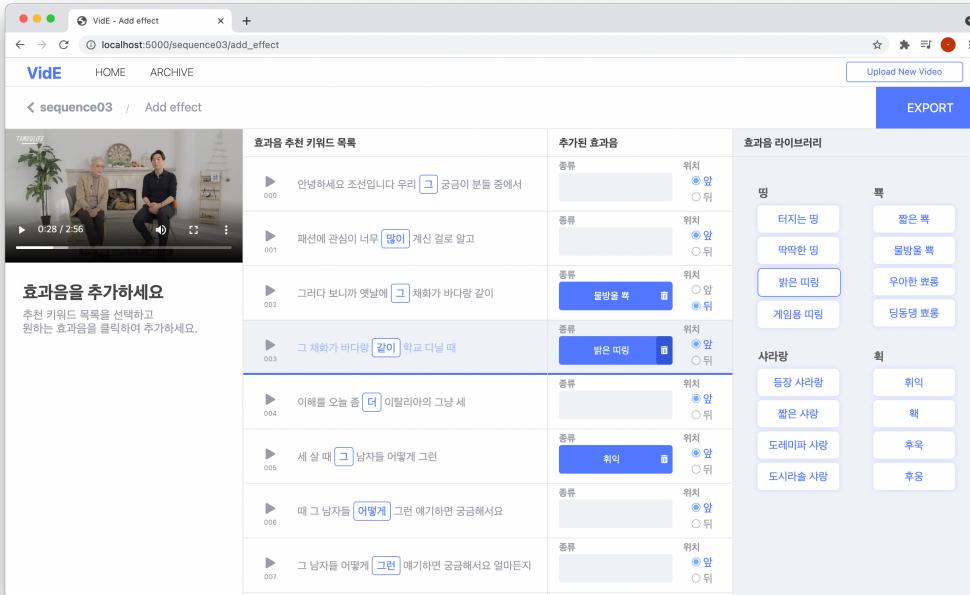
: 집중도 높은 영상 제작을 위한 편집 애플리케이션

디자인테크놀로지전공 김진아
디자인테크놀로지전공 이윤지
미디어테크놀로지전공 고지수
미디어테크놀로지전공 임주연

01 What is our Project?

비디(VidE)

집중도 높은 영상 제작을 위한 편집 애플리케이션



▲ 비디(VidE) 애플리케이션 사용 화면
(<https://youtu.be/0-L0hnjrPf4>)

현재 코로나19로 인해 비대면 온라인 강의 시스템이 활성화됨에 따라
영상으로 콘텐츠를 제작하고 주고 받는 일이 증가

BUT 영상의 집중도 떨어짐

AS-IS

TO-BE

수용자 - 집중력 있는 영상

제작자 - 영상편집 시간의 단축 및 효율화

누구나 쉽고 빠르게 집중력 있는 영상을 만들 수 있는
실용적인 툴을 만들고자 함

02 Painpoint & Solution

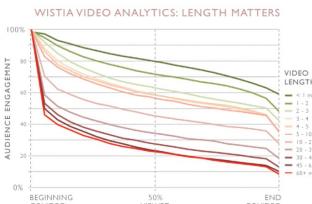
Painpoint 01

영상 집중도가 떨어진다

무음구간으로 인해 영상이
불필요하게 길어지고 속도감이 떨어짐



영상의 길이가 길수록 이탈률이 높고
집중도가 현저히 떨어지는 모습을 확인



영상의 길이를 줄이고, 속도감을 높여야함

무음구간을 삭제하는 방식으로 해결

Why not 배속기능?

배속기능 사용 시, 발음이 뭉개져 전달력 저하

영상에서 말로만 정보를 전달하는 경우,
내용의 강조 정도와 집중도가 떨어짐



유튜브 정보전달형 영상 콘텐츠 분석 결과,
많은 영상에서 주의 집중의 수단으로
'효과음'을 사용하는 것을 확인

32개 영상분석결과 총 1498회 사용



영상 속 키워드에 효과음 추가

Painpoint 02

영상 편집 툴 사용이 어렵다

영상 편집 툴에 대한 사전 지식 없이
처음 접하는 경우,
기능 사용에 어려움을 겪음



초보자가 속도감 있고, 효과음이 추가된
영상을 제작하는 일은 쉽지 않음

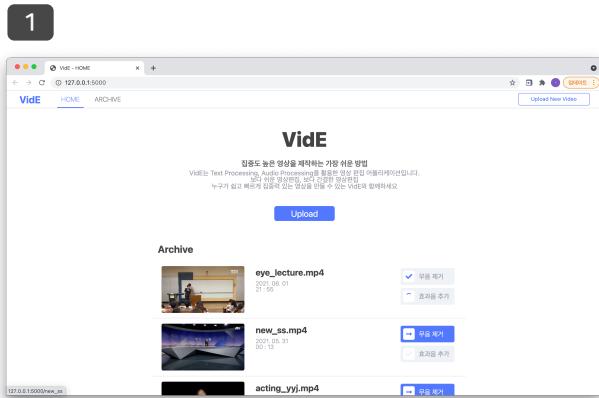


직관적인 UI 설계

Solution

'[오디오 데이터 분석](#)'과 '[텍스트 데이터 처리](#)'를 활용하여 [무음 구간 삭제](#)와 [효과음 추가](#)를 용이하게 하며,
[직관적인 UI](#)를 통해 처음 영상 편집을 접하는 사람도 쉽게 사용할 수 있는 애플리케이션 제작

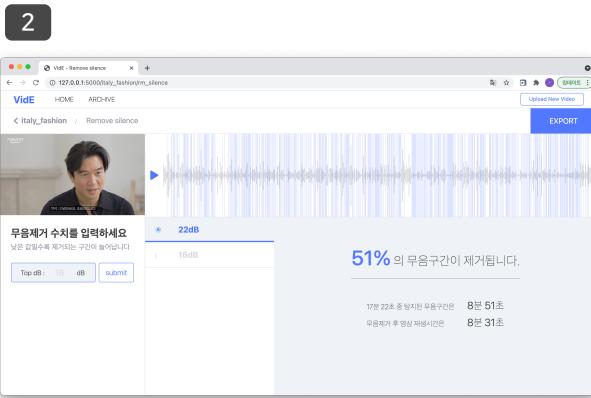
03 User Interaction Flow



VidE 홈화면

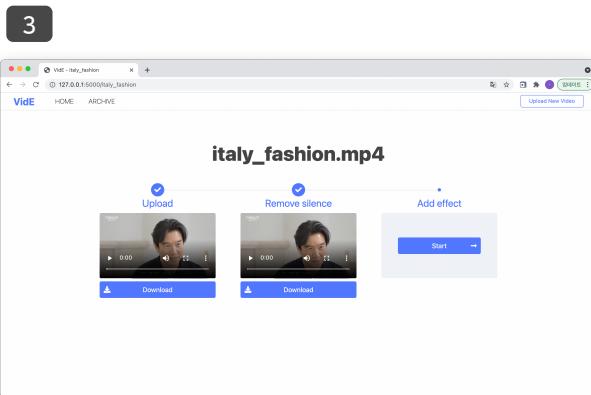
영상 업로드

지난 작업물 확인



[italy_fashion.mp4](#)

A screenshot of the WeVideo mobile application interface. At the top, there are two buttons: 'Upload' on the left and 'Remove silence' on the right. Below these are two video preview windows. Each window shows a man's face from the chest up, with a play button, a timestamp of '0:00', and a volume slider. At the bottom of each window is a blue bar with a download icon and the word 'Download'. The background of the app is white.

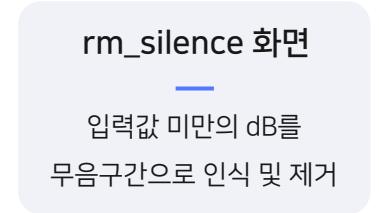


Process 화면

작업진행상황

원본 영상 및 무음 제거 영상 다운로드

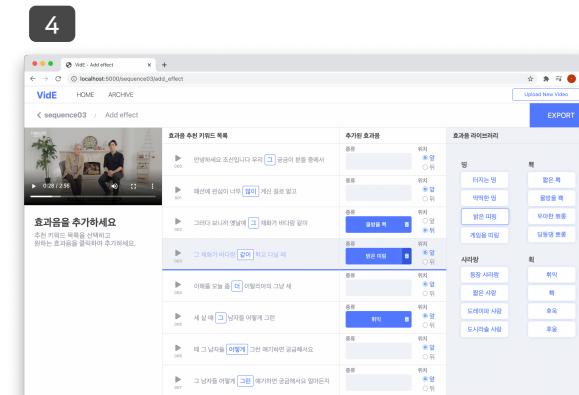
다음단계(효과음 추가)로 이동



rm_silence 화면

입력값 미만의 dB를

무음구간으로 인식 및 제거



add_effect화면

STT를 활용하여 구간 및 키워드 추출

효과음 라이브러리 제공

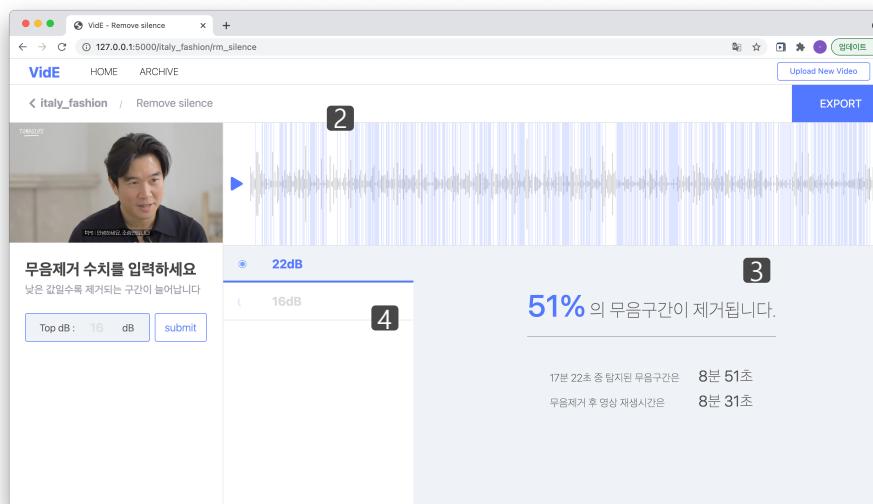
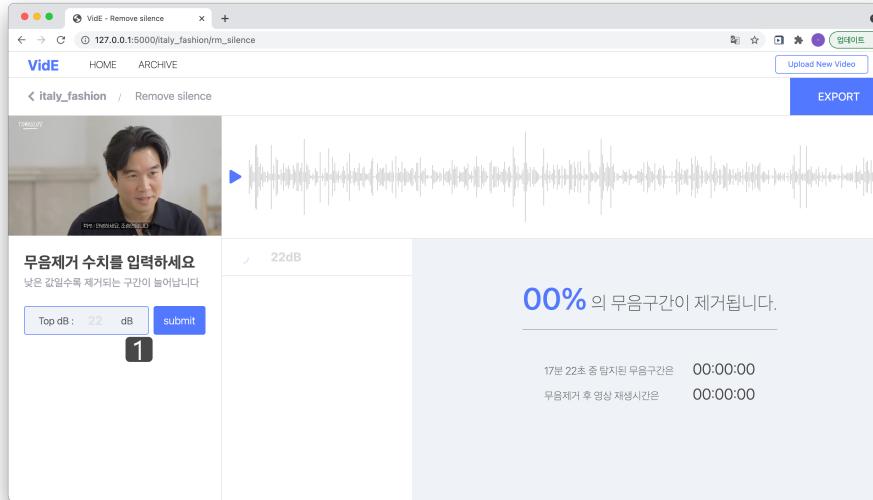
효과음 추가

▶ 편집된 영상 OUTPUT

04 무음 제거 UI 설명

#1 과정 자동화

#2 현 상태 이해를 위한 적절한 피드백



Considerations

#1

1

사용자가 값을 조절할 수 있게 하자

#2

2

작업결과를 쉽게 알 수 있게 하자

3

얼마나 줄어들었는지 수치로 알려주자

4

입력값에 따라 다양한 결과를
쉽게 볼 수 있게 하자

결과를 이해하고 수정할 수 있도록 하는 것

Solutions

무음제거 수치를 입력하세요

낮은 값일수록 제거되는 구간이 늘어납니다

Top dB : 16 dB submit

원하는 값을 입력한다

오디오의 전반적인 파형



차지하는 비율

51% 의 무음구간이 제거됩니다.

17분 22초 중 탐지된 무음구간은 8분 51초
무음제거 후 영상 재생시간은 8분 31초

탐지된 무음구간의 시간

총 변환될 비디오시간



입력했던 값을 클릭

- (1)무음제거 오디오
(2)무음구간표시
(3)결과수치값

변경

비교해서 들어보기

05 무음 제거 구현방법

#1 과정 자동화

: 무음구간 제거 구현 및 코드

오디오 구간 Split

오디오 (Max dB - top_db) 미만의 dB level 구간을 무음구간으로 처리

무음구간 시작점 & 끝점 리스트
비무음구간 시작점 & 끝점 리스트

```
def split(tdb, id):
    """ 오디오 무음 제거 후 해당 구간 타임스탬프 반환 """
    # get non mute intervals
    tdb = int(tdb)
    y, sr = librosa.load(get_file_path(f'{id}.wav'))
    non_mute_intervals = librosa.effects.split(y, top_db=tdb)
    # get mute intervals
    temp = [t/sr for t in non_mute_intervals[0].flatten()]
    temp = np.insert(temp, 0, 0) # temp, 0번째 위치, 0 값
    mute_intervals = np.append(temp, len(y)/sr) # temp, 종료 시간값
    split_data = [
        {
            "id": id,
            "sr": sr,
            "intervals": [
                {
                    "mute": mute_intervals.tolist(),
                    "non_mute": non_mute_intervals.tolist()
                }
            ]
        }
    ]
    return split_data
```

비디오 Concat

비무음구간 영상 frame을 받아옴

비무음구간 비디오를 합침

```
def silence_export(id, video_name, data):
    """ 영상 무음 제거 """
    tdb, sr, intervals = data
    clip = mp.VideoFileClip(get_file_path(video_name))
    clip_list = [clip.subclip(i*sr/int(sr), i+1*sr) for i in intervals['non_mute']]
    final_clip = mp.concatenate_videoclips(clip_list)
    try:
        path = get_path(id, f'rm{tdb}dB_{id}.mp4')
        print(path)
        final_clip.write_videofile(path,
                                   temp_audiofile='temp-audio.m4a',
                                   remove_temp=True,
                                   codec="libx264", audio_codec="aac")
    except:
        pass
    return {
```



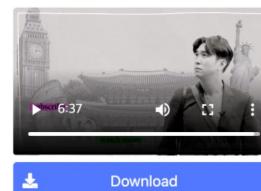
Upload



Download



Remove silence



Download

OUTPUT :
무음구간이 제거된
하나의 영상 결과물

What is Moviepy

Video Processing에 최적화된 Python 라이브러리

What is Wavesurfer

웹 오디오와 html5 위에 구축된 Javascript 라이브러리
입력된 영상의 오디오에 따른 파형 출력

#2 현 상태 이해를 위한 적절한 피드백

: 무음구간 제거 결과요약 구현 및 코드

wavesurfer 라이브러리를 활용한 시각화

무음구간 시각화

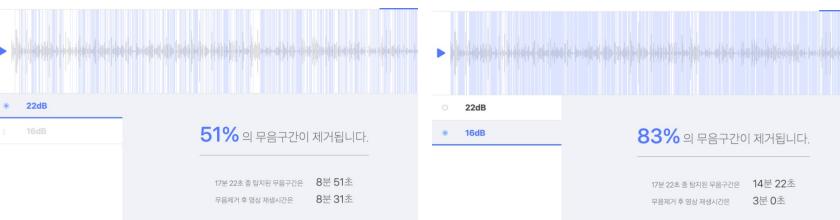
```
const updateWaveform = (intervals) => {
    // set non-mute-intervals to mute-intervals
    wavesurfer.clearRegions();
    var mute_total_time = 0;
    for (i=0; i<intervals.length-1; i+=2){
        mute_total_time += intervals[i+1]-intervals[i];
        wavesurfer.addRegion({
            start: intervals[i],
            end: intervals[i+1],
            color: 'hsla(30, 30, 30, 0.15)',
            drag: false,
            resize: false
        });
    }
}
```

```
const showDuration = (intervals) =>{
    var totalMute = 0;
    const duration = document.getElementById('video').duration;
    const percent = document.getElementById('percent');
    const silenceDuration = document.getElementById('silence-duration');
    const removedDuration = document.getElementById('output-duration');

    //무음구간 시간계산
    for (i=0; i<intervals.length-1; i+=2){
        totalMute += intervals[i+1]-intervals[i];
    }
    var minute = parseInt(totalMute / 60);
    var second = Math.round(totalMute % 60);
    var total = duration - totalMute;

    percent.innerText = Math.round((totalMute/duration)*100)+"%";
    silenceDuration.innerText = `${minute}분 ${second}초`;
    removedDuration.innerText = `${parseInt(total / 60)}분 ${parseInt(total % 60)}초`;
}
```

OUTPUT :
입력한 값에 따라
결과를 업데이트

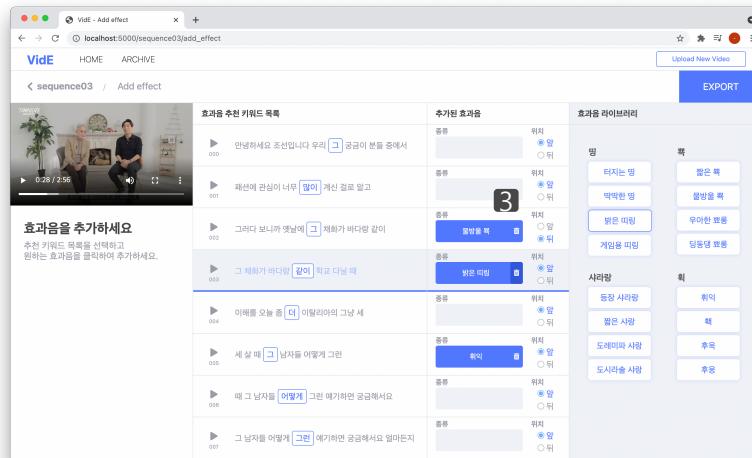
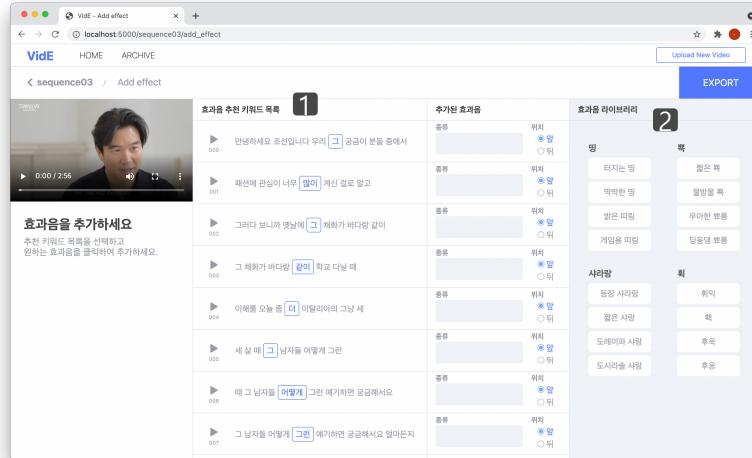


06 효과음 추가 UI 설명

#1 과정 자동화

#2 현 상태 이해를 위한 적절한 피드백

Considerations



#1,2

1

어느 내용의 구간인지 쉽게 알게 하자

키워드를 쉽게 인지할 수 있게 하자

2

다양한 효과음을 사용할 수 있게하자

오디오를 쉽게 추가할 수 있게 하자

바로바로 확인할 수 있게하자

3

어떤 효과음이 추가됐는지 알게하자

추가시점을 결정할 수 있게 하자

효과음을 삭제할 수 있도록 하자

Solutions

STT Json파일 파싱 후 키워드 필터링을 거쳐 앞뒤 3단어씩 함께 재구조화함

키워드와 주변 맥락을 함께 보여줌

효과음 추천 키워드 목록

종류 위치
5.짧은_명_02.mp3 앞
뒤

어 그리고 이제 악 처방

도출된 키워드 하이라이팅

**추출한 상위 4개 종류
짧고/긴 특징 별 각각 2개씩 제공**

명	흑
티지는 땅	짧은 빠
딱딱한 땅	물방울 뿍
밝은 빠	우아한 뾰통
개입용 빠	딥풀렁 뾰통

사리랑	희
등장 사리랑	희익
밝은 사랑	희
도래미파 사랑	후욱
도시라솔 사랑	후옹

구간 선택 > 효과음을 클릭
해당 효과음이 적용된 구간의 오디오재생

#2 선택해서 들어보고
수정할 수 있도록 하는 것

선택된 오디오 표시

효과음 추천 키워드 목록

추가된 효과음

종류 위치
5.짧은_명_02.mp3 앞
뒤

어 그리고 이제 악 처방

선택된 오디오 표시

오디오가 추가되는 시점선택 : 키워드의 시작/끝

07 효과음 추가 구현방법

#1 과정 자동화

: 어떤 키워드에 어떤 효과음을 추가할지 결정

효과음 추가시점 결정

Text script Processing

STT 후 추출한 script에서
효과음을 추가할 키워드 및 재생 타이밍 도출

Google Cloud Speech API : 단어별 시작시간, 끝시간 제공

: 키워드 앞/뒤 선택 세분화

What is STT

Speech-to-Text : 사람이 말하는 음성 언어를 컴퓨터가 해석해 그 내용을 텍스트로 전환하는 처리

키워드 결정

연결/강조/숫자

: 영상의 주제와 별개로 범용적으로 사용

'연결/강조/숫자' 키워드 도출과정

1. 유튜브 정보전달영상 21개 STT
2. 품사태깅을 활용하여 연결/강조 키워드 추출(접속사, 부사 - 79단어)
3. 키워드로 적합하지 않은 (ex.퍽퍽) 단어를 제외한 52개 단어선정
4. 52개 단어를 연결(20단어), 강조(32개)로 구분지어 초기 키워드셋 정의
5. 변형가능한 형태와 유사한 의미를 가지는 단어를 추가하여 최종적인 키워드셋 정의

효과음 종류 결정

띠링/뽁/휙/별가루

자주 사용된 효과음 상위 4개 추출

뽁(19.6%)
띠링(12.5%)
휙(7.1%)
뽁(7.1%)
별가루(2.7%)

뽁(23.4%)
띠링(12.8%)
휙(8.5%)
별가루(5.3%)
띵,후욱(3.2%)

뽁(18.1%)
띠링(10.6%)
후욱(9.6%)
휙(6.4%)
별가루(3.2%)

연결효과음분포

강조효과음분포

숫자연결음분포

#2 현 상태 이해를 위한 적절한 피드백

효과음&비디오 동시재생 구현

: 실제 효과음이 어떻게 추가될 지 확인



선택한 효과음과 시간값 매칭 후
효과음과 원본 영상의 오디오 파일을 합쳐서
하나의 오디오 파일로 변환

```
def effect_export(id, video_name, effect_list, time_list):
    input_path = os.path.join(INPUT_FOLDER, id, video_name)
    output_path = os.path.join(OUTPUT_FOLDER, id, f"output_{id}.mp4")

    time = time_list
    effect = effect_list
    # print(effect)

    input_video = ffmpeg.input(input_path)
    added_audio = input_video.audio

    for i in range(len(time)):
        # print(effect[i])
        a = ffmpeg.input(os.path.join(EFFECT_FOLDER, effect[i])).audio.filter("adelay", f"{time[i]}:{time[i]}")
        added_audio = ffmpeg.filter(added_audio, a).amix()

    (
        ffmpeg
        .concat(input_video, added_audio, v=1, a=1)
        .output(output_path)
        .run(overwrite_output=True)
    )

    return {
        'status': 'COMPLETE',
        'src': get_srt(output_path),
        'msg': 'exporting...'
    }
```

What is ffmpeg

디지털 음성 스트림과 영상 스트림에 대해서
다양한 종류의 형태로 기록하고 변환하는 라이브러리

KoNLPy(코엔엘파이)
한국어 정보처리를 위한 파이썬 패키지
형태소 분석 및 품사태깅을 지원

품사태깅(POS tagging)
주어진 텍스트 각 토큰에
적절한 품사 태그를 붙이는 작업

08 병렬 처리

“비디오 변환 시 대기시간이 오래 걸리는 문제점 발생” >>> 병렬처리

#1 과정 자동화

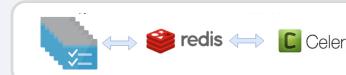
효율적인 요청처리를 위한 celery task 비동기처리

What is Celery

Celery는 분산 메세지 전달에 기반한 비동기 작업 큐
별도로 실행 중인 Worker Process가 Broker로부터 Message를 전달 받아
작업을 대신 수행해 주는 라이브러리

핵심 TASK & 비동기처리 TASK

: 시스템에서 접근하는 자동화



1. 업로드

오디오 추출

2. 무음 제거

@split

무음구간 결정

3. 효과음 추가

효과음
종류, 타이밍 결정

@process

STT 추출

@export

무음제거

@export

효과음추가

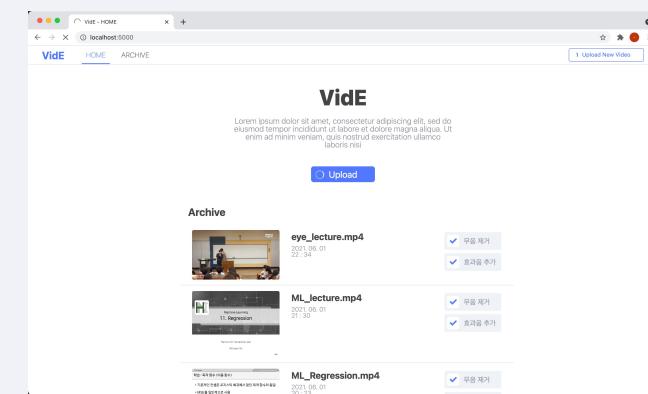
redis pubsub
개별 비디오 진행상황 페이지에서 대기 중

작업 완료 시 server sent event : 자동으로 상황 업데이트

#2 현 상태 이해를 위한 적절한 피드백

아카이브

이전 작업물 목록과 진행상태



상태 별 버튼

진행 중

무음 제거

효과음 추가

작업완료

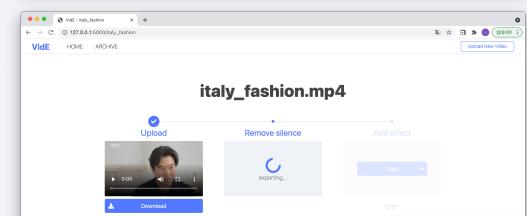
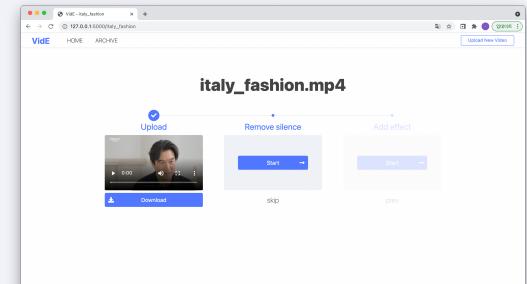
무음 제거

효과음 추가

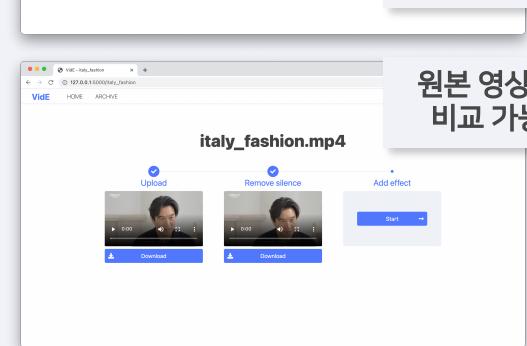
작업 전

페이지 이동

진행상황



진행상황 표시



원본 영상과 비교 가능

09 Result

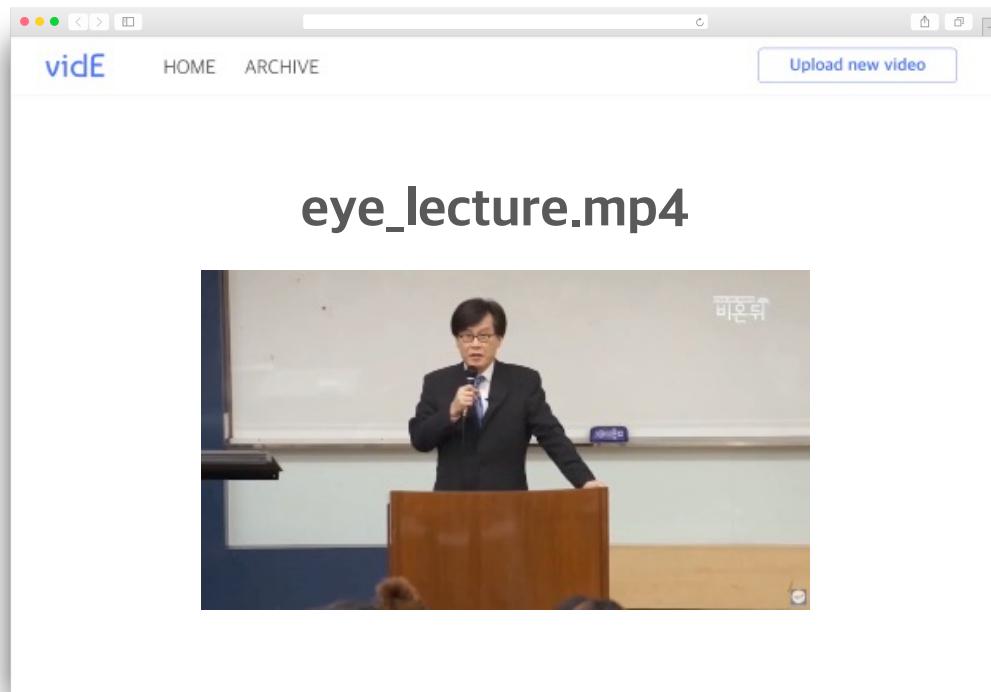
Changes & Effects

영상 길이 변화 (무음제거)

4 분 36초

2분 39초

총 1분 57초 단축 (약 42%)



▲ 비디(VidE) 시연영상
유튜브 정보전달영상 사용(라식 라섹 수술 눈 건강 이태원 안과 '이태원 원장' 의학개론 특강 | 의학채널 비온뒤)
<https://www.youtube.com/watch?v=u-Db1Vel4H0>

효과음 추가 위치

그 (0분 4초)

먼저 (0분 22초)

또 (1분 01초)

그래서 (0분 14초)

굉장히 (0분 58초)

모든, 또한, 그만큼, 16년, 이렇게, 그리고, 그런 등 키워드에 적용

The Time Required

대기 시간
2분 59초

+

조작 시간
6분 14초

=

소요 시간
9분 13초

10 사용자평가 및 결론

User Feedback

: 사용자 피드백을 통해 도출한 애플리케이션 성과



**“저는 영상 편집이 처음이었는데도 사용하기 어렵지 않았고
빠르게 업무를 처리할 수 있었어요”**

코로나 19로 인해 발표를 비대면으로 진행하게 돼서
처음으로 영상 편집을 접하게 되었어요.
마냥 복잡하고 어려울 줄 알았는데, VidE를 사용하니 흐름에 따라
어렵지 않게 영상을 편집할 수 있었습니다.
제가 한 영상을 작업하는 동안 추가적으로
다른 작업도 진행할 수 있었어서 좋았어요.



**“배속 기능의 한계를 극복하고
사용자 맞춤 커스터마이징을 할 수 있다는 점이 좋았어요.”**

평소에 강의를 들을 때 배속을 해서 들으면 말이 빨라질 때
여러 번 그 구간을 반복해서 듣거나
배속 정도를 바꾸어야 한다는 점이 불편했어요.
배속 기능과 다르게 VidE는 말 자체의 속도를 빠르게 하는 것이 아니라
불필요한 구간을 줄여서 영상을 컴팩트하게 만들 수 있다는 점과
삭제할 구간을 내가 원하는 정도로 섬세하게 설정할 수 있다는 점이
인상 깊었습니다.

창의성

- 01 **효과적 전달을 위해 배속이 아닌
무음제거를 통해 영상에 속도감 부여**
- 02 **효과음추가 키워드 라이브러리 사전 정의**
- 03 **STT를 활용하여 오디오 추가 과정 간소화**
- 04 **사용자가 원하는 위치에 효과음 추가**
- 05 **User Friendly UI 설계**

완결성

- 01 **원본 영상 input에서부터 무음제거와
효과음추가를 거쳐 편집된 영상 output까지
시스템의 완결된 flow 구현 (백 + 프론트)**
- 02 **각 페이지/단계별 세부 기능 구현**
- 03 **애플리케이션이 자동으로 해주는 부분,
사용자가 입력을 진행해야 하는 부분을
고려하여 구현**
- 04 **페이지 이동 간 비동기처리를 통해
페이지를 벗어나도 해당 작업이
유지될 수 있도록 구현**

Future Research

