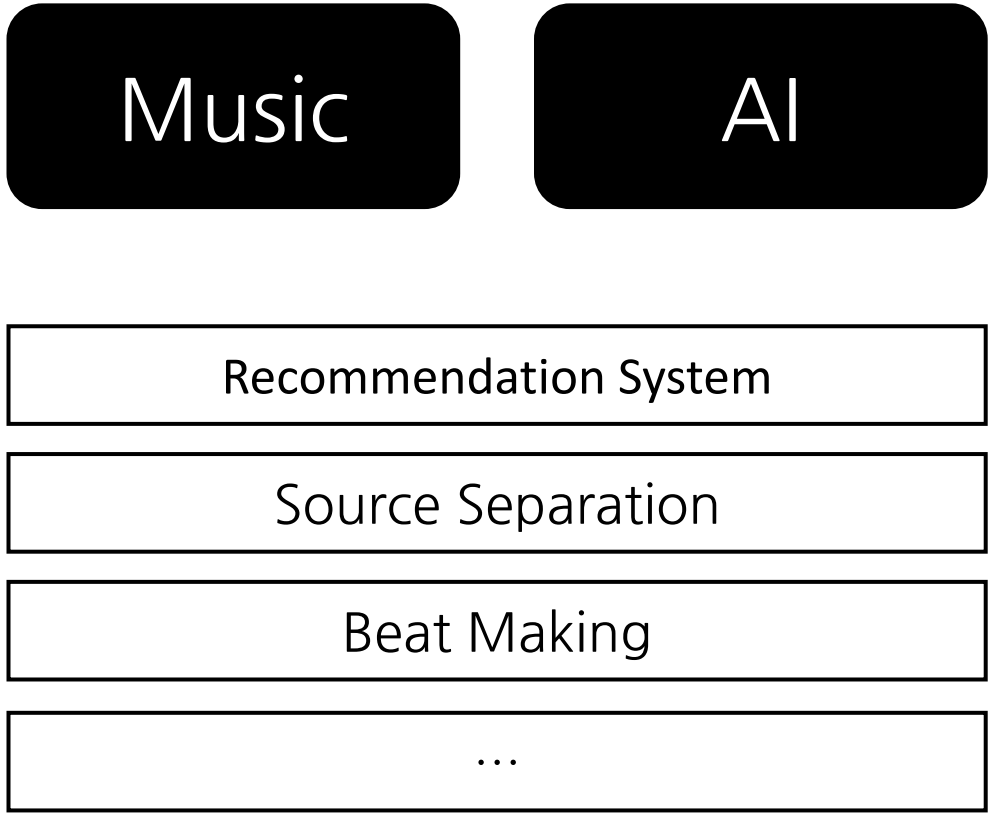
HAI AI PROJECT

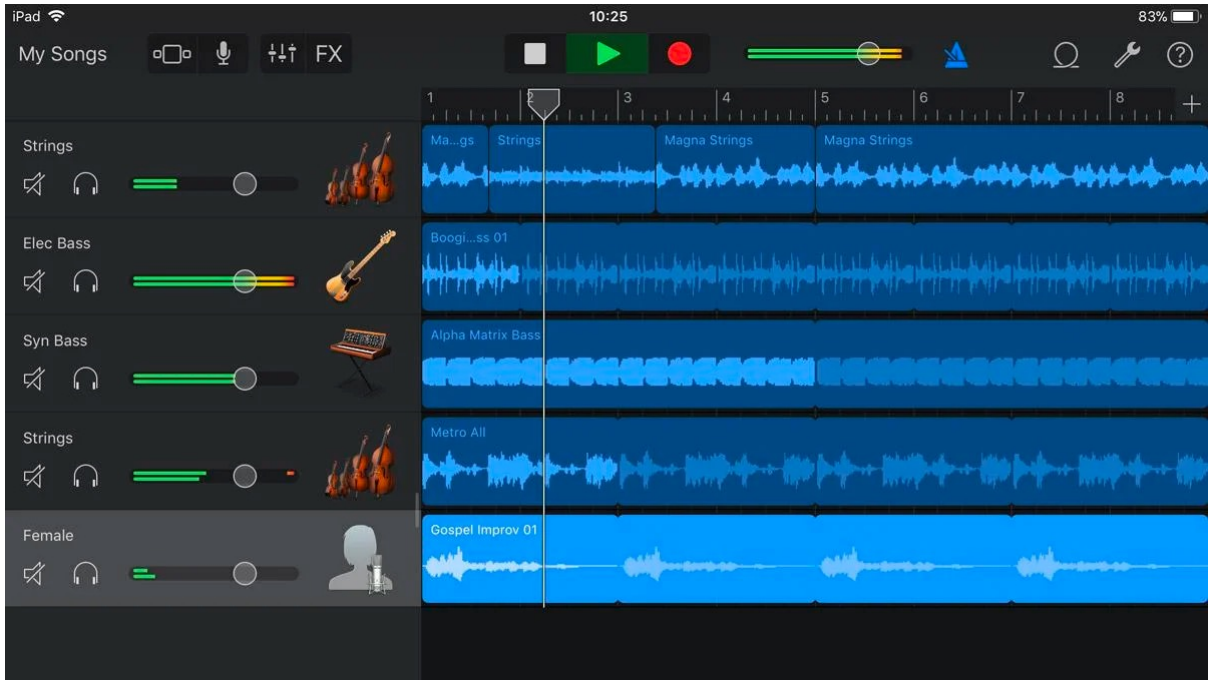
Music Team

배경


Music
AI
Recommendation System
Source Separation
Beat Making
…

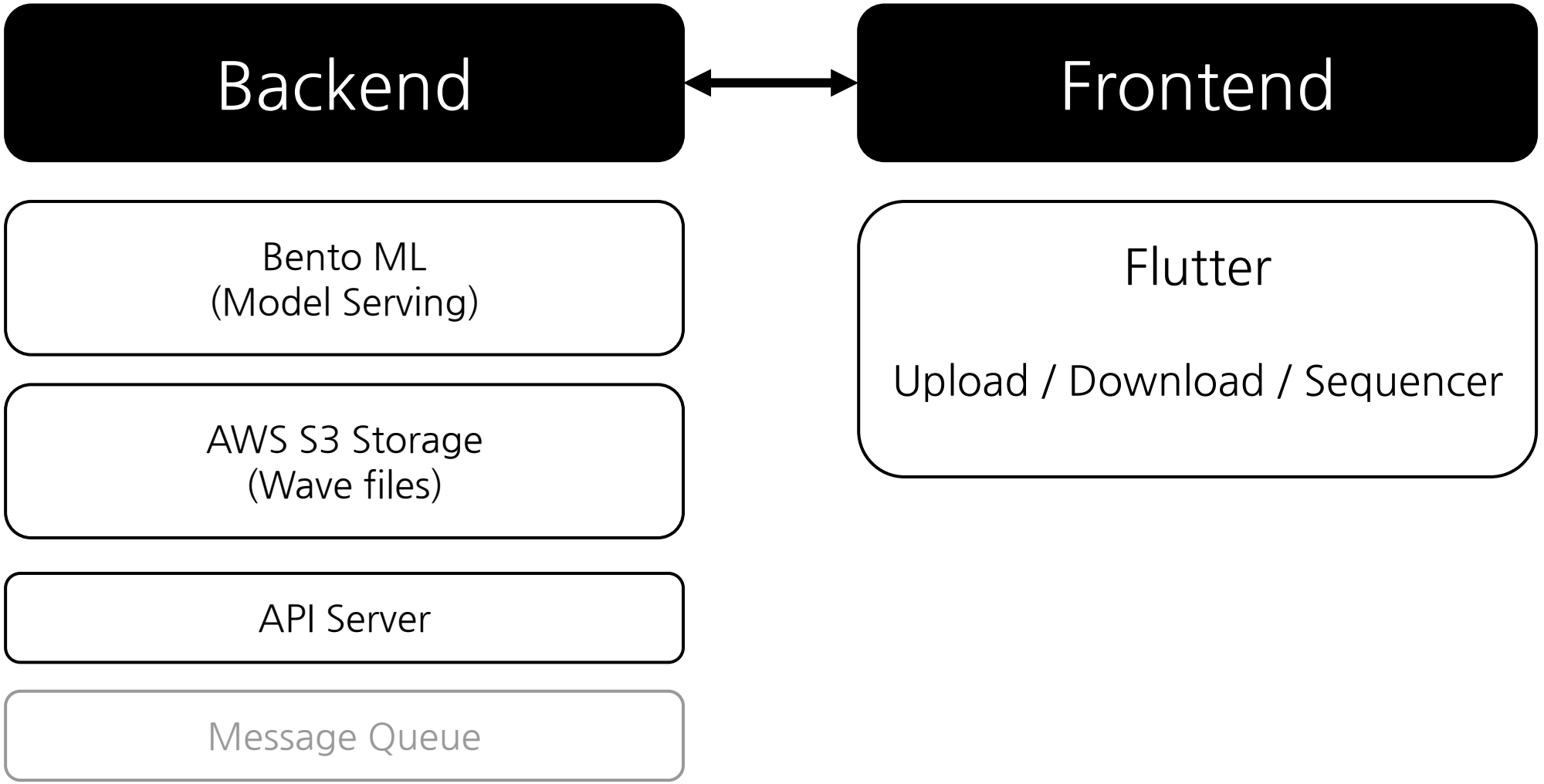
배경

배경

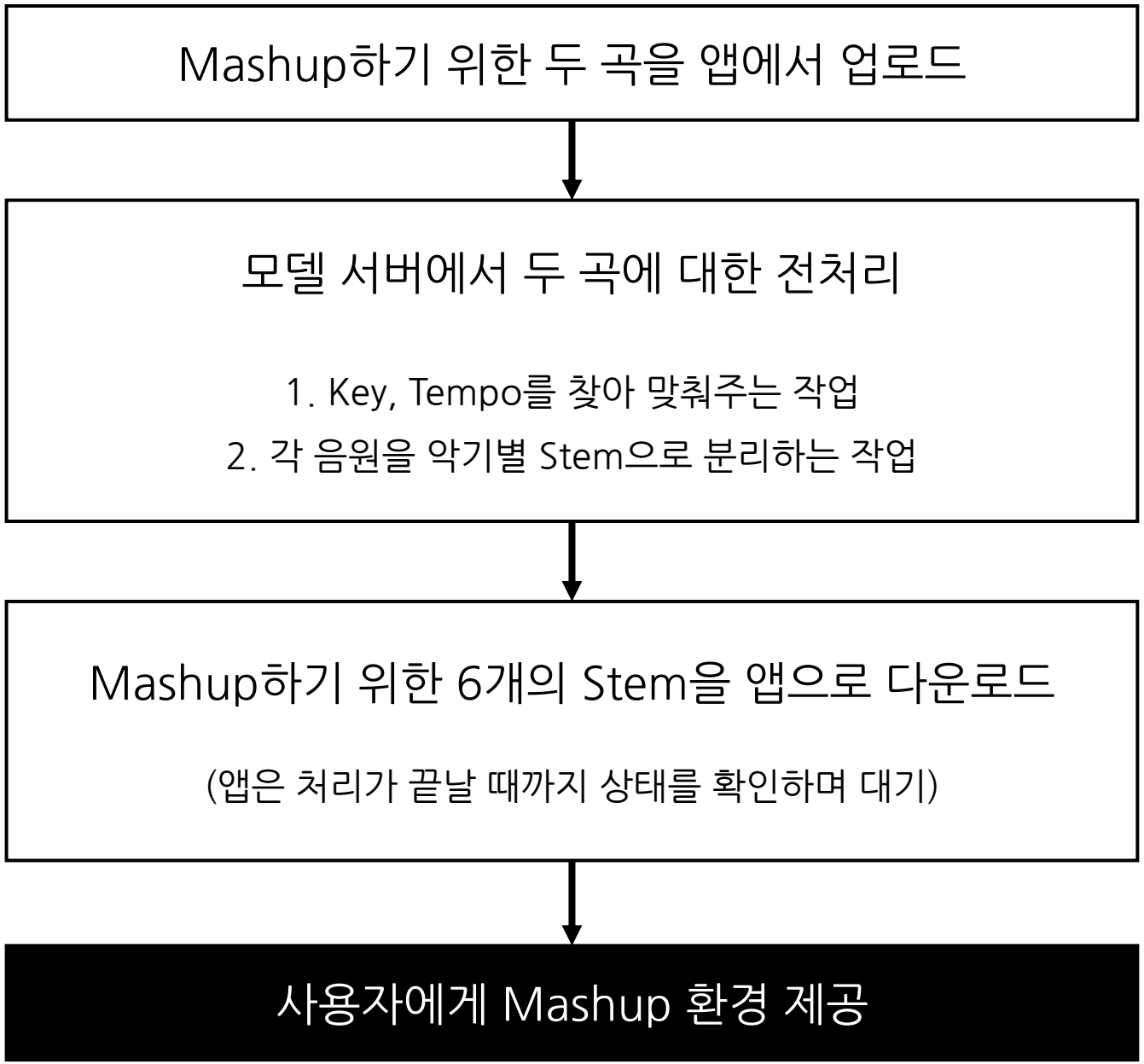


- Source Separation
- Key Detection
- Beat Tracking
- ML Serving
- Mobile App

구조


Backend
Frontend
Bento ML
(Model Serving)
AWS S3 Storage
(Wave files)
API Server
Message Queue
Flutter
Upload / Download / Sequencer

구조


Mashup하기 위한 두 곡을 앱에서 업로드
모델 서버에서 두 곡에 대한 전처리
1. Key, Tempo를 찾아 맞춰주는 작업
2. 각 음원을 악기별 Stem으로 분리하는 작업
Mashup하기 위한 6개의 Stem을 앱으로 다운로드
(앱은 처리가 끝날 때까지 상태를 확인하며 대기)
사용자에게 Mashup 환경 제공

Model

Model Server Pipeline

INPUT
하나의 WAV File URL

파일 유효성 검사

음원 분석
Loading / Beat Tracking / Key Detection

음원 처리
Pitch Shift / Time Stretch / Source Separation

OUTPUT
곡에 대한 정보 json

Model Server Pipeline

파일 유효성 검사

존재하는 파일인지 확인 (없다면 예외 발생)

지원하는 확장자인지 확인

wav, mp3만 일단 지원

mp3인 경우 wav로 변환
(라이브러리에서의 호환을 위함)

pydub.AudioSegment 이용

Model Server Pipeline

음원 분석 - 기본 정보 로딩

- librosa 이용

```
self.y, self.sr = librosa.load(self.wav_file)
self.bpm = librosa.beat.tempo(self.y, self.audio_sr)[0]
```

- length, sample rate, bpm 등 기본 정보
- 음원의 각 샘플을 배열(numpy array)로 가져옴 (length * sample rate 개 샘플)
- librosa를 이용하는 전처리 과정에 필요
- 좀더 구체적으로 필요한 정보를 찾아내기 위해서 분석할 수 있음 (현재는 없는 기능)
    - 구간 정보 (pre-chorus, chorus, post-chorus, bridge or solo, intro, outro, etc.)
    - 키 혹은 템포의 변화
- 이 프로젝트에서는 한 박자 구간의 길이와 키만 알면 됨

Model Server Pipeline

음원 분석 - Beat Tracking

- madmom 이용

```
proc_tempo = madmom.features.beats.DBNBeatTrackingProcessor (look_ahead=0.4, fps=100)
act = madmom.features.beats.RNNBeatProcessor()(self.wav_file)
self.beats = proc_tempo(act)

self.num_beats = len(beats)
self.first_beat = beats[0]
self.last_beat = beats[-1]
self.beat_length = (self.beats[-1] - self.beats[0]) / (self.num_beat - 1)
```

- 기존에 생각하던 first_beat는 첫번째 드럼 비트가 나오는 지점이었으나 madmom으로 분석한 정보는 들리지 않는 시점까지 고려한 실제 첫 박자 지점이었음
- 이를 해결하기 위해서는 drum 추출 후 첫 박자 지점을 다시 찾는 등의 방식 필요
- 50초짜리 wave file로 테스트했을 때 6.67초 정도에 분석완료

Model Server Pipeline

음원 분석 - Key Detection

- madmom 이용

```
proc_key = madmom.features.key.CNNKeyRecognitionProcessor()
key_pred = proc_key(self.wav_file)
self.key_label = madmom.features.key.key_prediction_to_label(self.key_pred)
self.key_label_num = BeatTracker.KEYs[self.key_label]
```

- 키는 처음부터 끝까지 하나라고 가정
- 12개 근음과, 2가지 스케일(Major, Minor)로 분석
- 음계 상 Major 키는 Minor 키와 3도 차이나므로 해당 값을 고려하여 키를 0~11까지의 값으로 변환하여 상대적 키 차이 분석이 용이하도록 함
- 평균율에서 키의 1도차이는 주파수의 12^(1/12) = 1.05946309436배 차이임

Model Server Pipeline

음원 처리

두 곡에 대한 분석이 끝나면 각 곡의 정보를 확인하여 음원을 처리하는 과정

- 두 곡의 Tempo, Key 를 평균값 혹은 한쪽에 맞춤
- Tempo는 빠른 쪽에, Key는 평균 키의 버림 값을 이용

Model Server Pipeline

음원 처리 - Pitch Shift

- librosa 이용

```
y_shifted = librosa.effects.pitch_shift(y, sr, n_steps=key_to_shift)
```

- 키 차이에 대한 정보
  - tone (온음: 2키 차이)
  - semitone (반음: 1키 차이)
  - octave (옥타브: 12키 차이)
- bins_per_octave 인자를 이용하여 step의 정도를 조절할 수 있음
  - 기본값: 12 (1옥타브를 12개의 step으로 쪼갬 = step 차이가 key 차이와 동일)

Model Server Pipeline

음원 처리 - Time Stretch

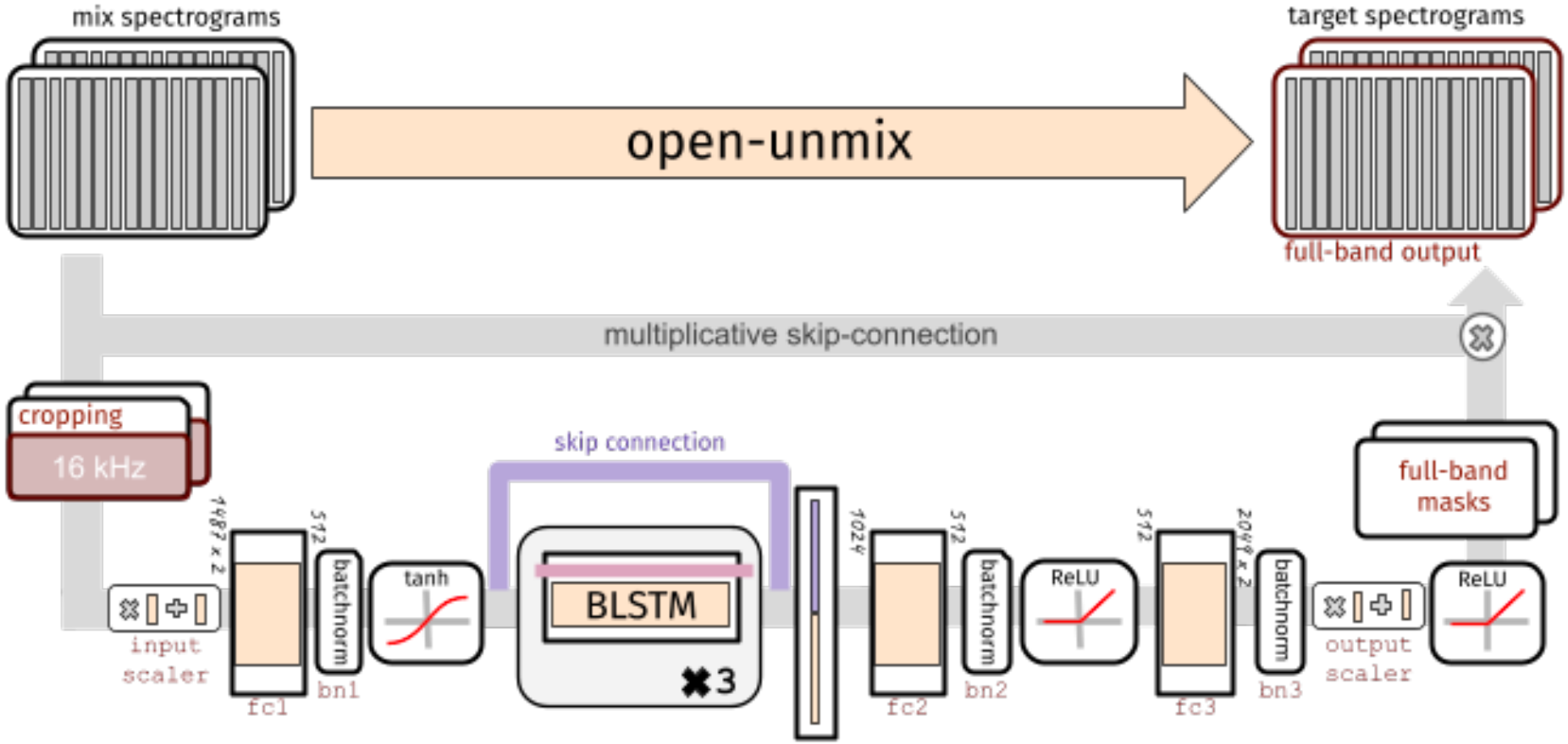
- librosa 이용

```
y_fast = librosa.effects.time_stretch(y, rate=stretch_rate)
```

- stretch_rate = bpm_fast / bpm_slow 를 이용해 느린 템포의 곡을 빠른 템포의 곡에 맞춤

Model Server Pipeline

음원 처리 - Source Separation



Model Server Pipeline

음원 처리 - Source Separation

- open-unmix 이용

```
audio, rate = stempeg.read_stems(
    self.wave_file,
    sample_rate=44100,
    start=self.start,
    duration=self.stop-self.start
)
estimates = predict.separate(
    audio=torch.as_tensor(audio).float(),
    rate=44100,
    device=device,
)
```

- vocal, bass, drums, other 로 분리된 각 stem을 얻을 수 있음
- 모든 처리가 끝나면 s3에 각 wave file을 업로드함

Model Server Pipeline

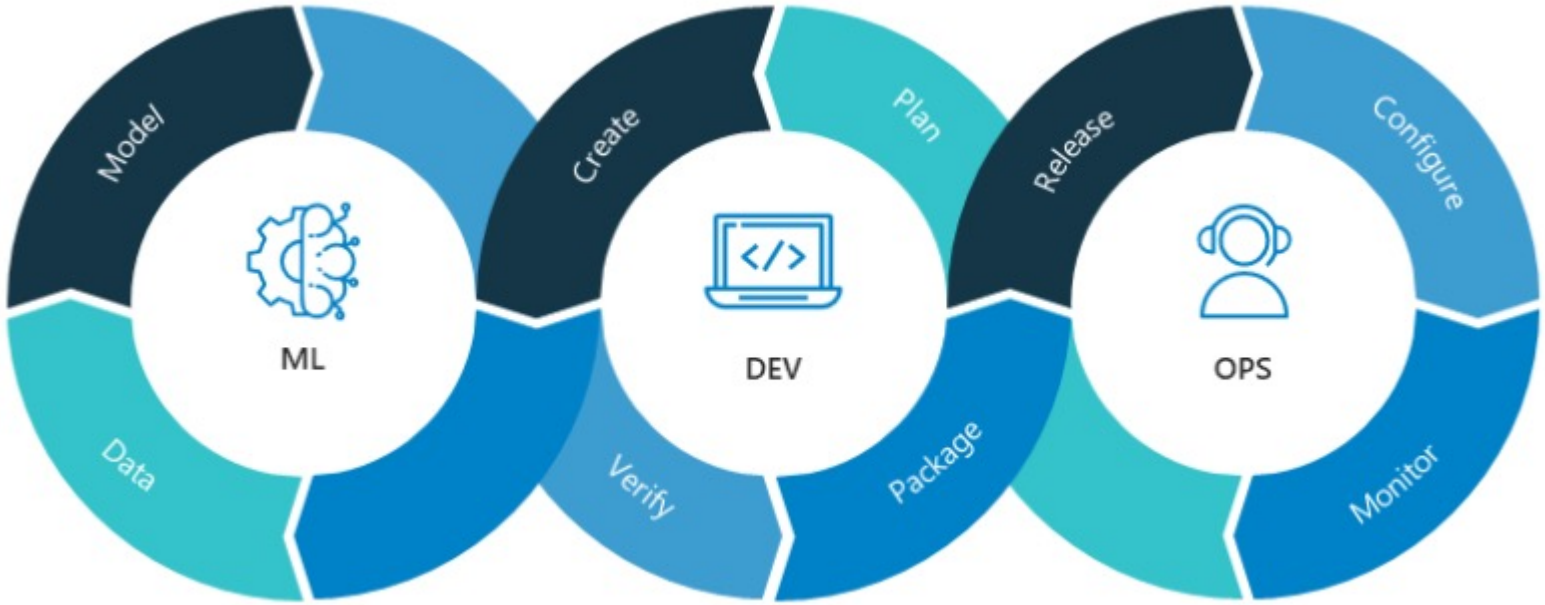
OUTPUT: 처리 완료된 곡에 대한 정보

S3에 Stem Files 업로드


```
{
    "title": "mix1",
    "key_label": "F major",
    "key_label_num": 8,
    "bpm": 108,
    "first_beat": 0.41,
    "last_beat": 49.85,
    "beat_length": 0.5555056179775282,
    "length": 50000,
    "url": "https://s3.ap-northeast-2.amazonaws.com/mix-music-analysis/mix1.wav",
}
```

Backend

MLOps



ML Ops = ML + Dev + Ops

Orchestration

- Reproducibility

MLOps - Cortex

MLOps - Cortex

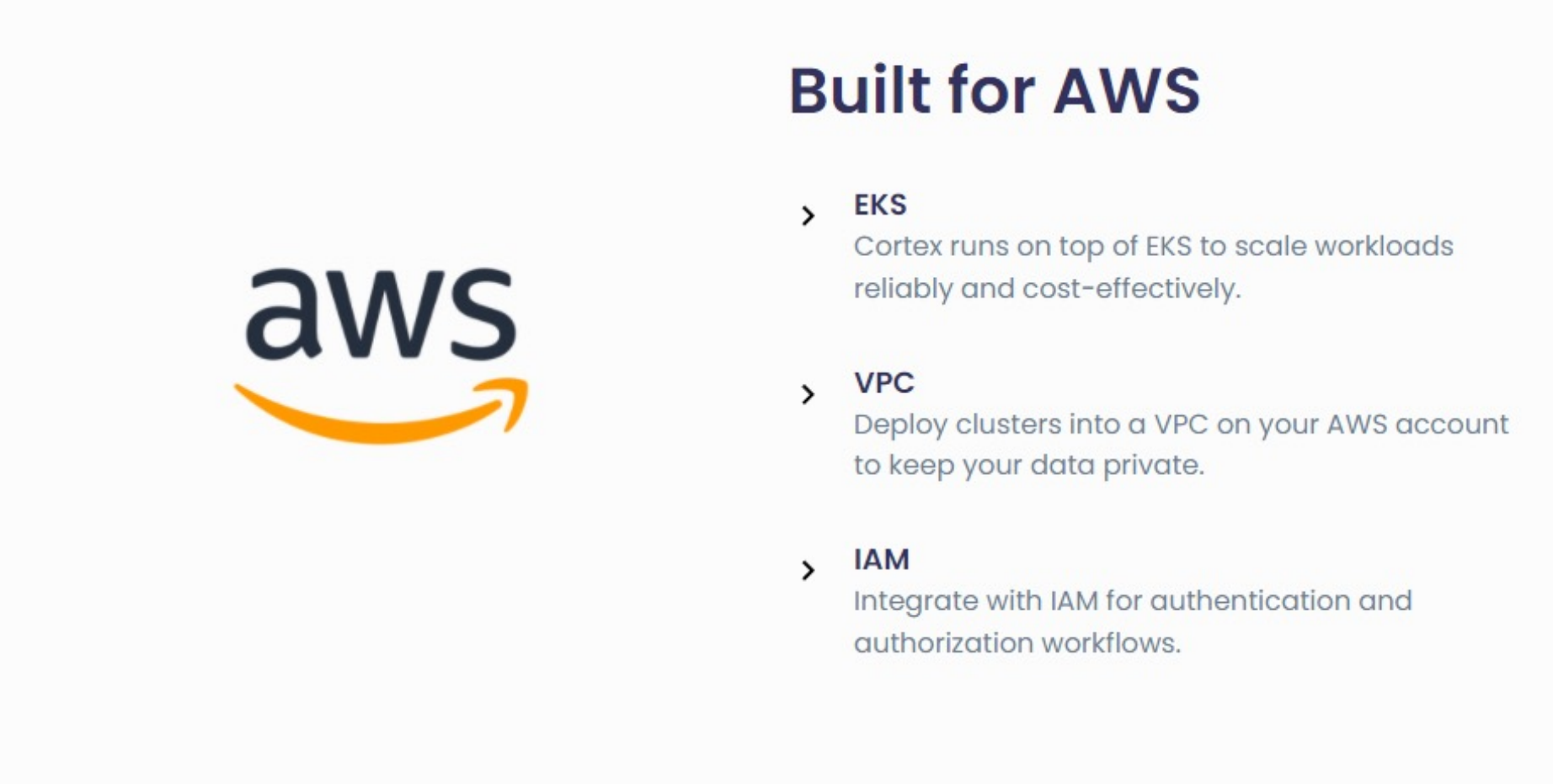
```yaml
# cluster.yaml

- name: production
  region: us-east-1
  node_groups:
    - instance_type: c5.xlarge
      max_instances: 10
      spot: true
    - instance_type: g4dn.xlarge
      max_instances: 20
      spot: true
```

Automated cluster management

- **Cluster autoscaling**
  Elastically scale clusters with CPU and GPU instances.
- **Spot instances**
  Run workloads on spot instances with automated on-demand backups.
- **Environments**
  Create multiple clusters with different configurations.

MLOps - Cortex



MLOps - BentoML

An Open Source Platform for

Machine-Learning Model Serving and Deployment

MLOps - BentoML

MLOps - BentoML

IrisClassifier 20200422120234_51B039 OAS3

/docs.json

To get a client SDK, copy all content from docs and paste into editor.swagger.io then click the tab **Generate Client** and choose the language.

infra

- POST /feedback
- GET /healthz
- GET /metrics

app

POST /predict

BentoService API

Parameters Cancel

No parameters

Request body required — application/json

```
[
  [5.1, 3.5, 1.4, 0.2],
  [5.0, 3.5, 1.6, 0.3],
  [4.9, 4.5, 1.2, 0.2]
]
```

Execute | Clear

Responses

Curl

```
curl -X POST "http://127.0.0.1:5000/predict" -H "accept: */*" -H "Content-Type: application/json" -d "
[[5.1,3.5,1.4,0.2],[5,3.5,1.6,0.3],[4.9,4.5,1.2,0.2]]"
```

Request URL

```
http://127.0.0.1:5000/predict
```

Server response

Code	Details
200	Response body: `[ 0, 0, 0 ]` Download

Response headers

```
content-length: 9
content-type: application/json
date: Wed, 06 May 2020 00:08:59 GMT
request_id: cc0257bd-5299-439e-9f38-27cc731f887c
server: Werkzeug/0.16.0 Python/3.7.5
```

Responses

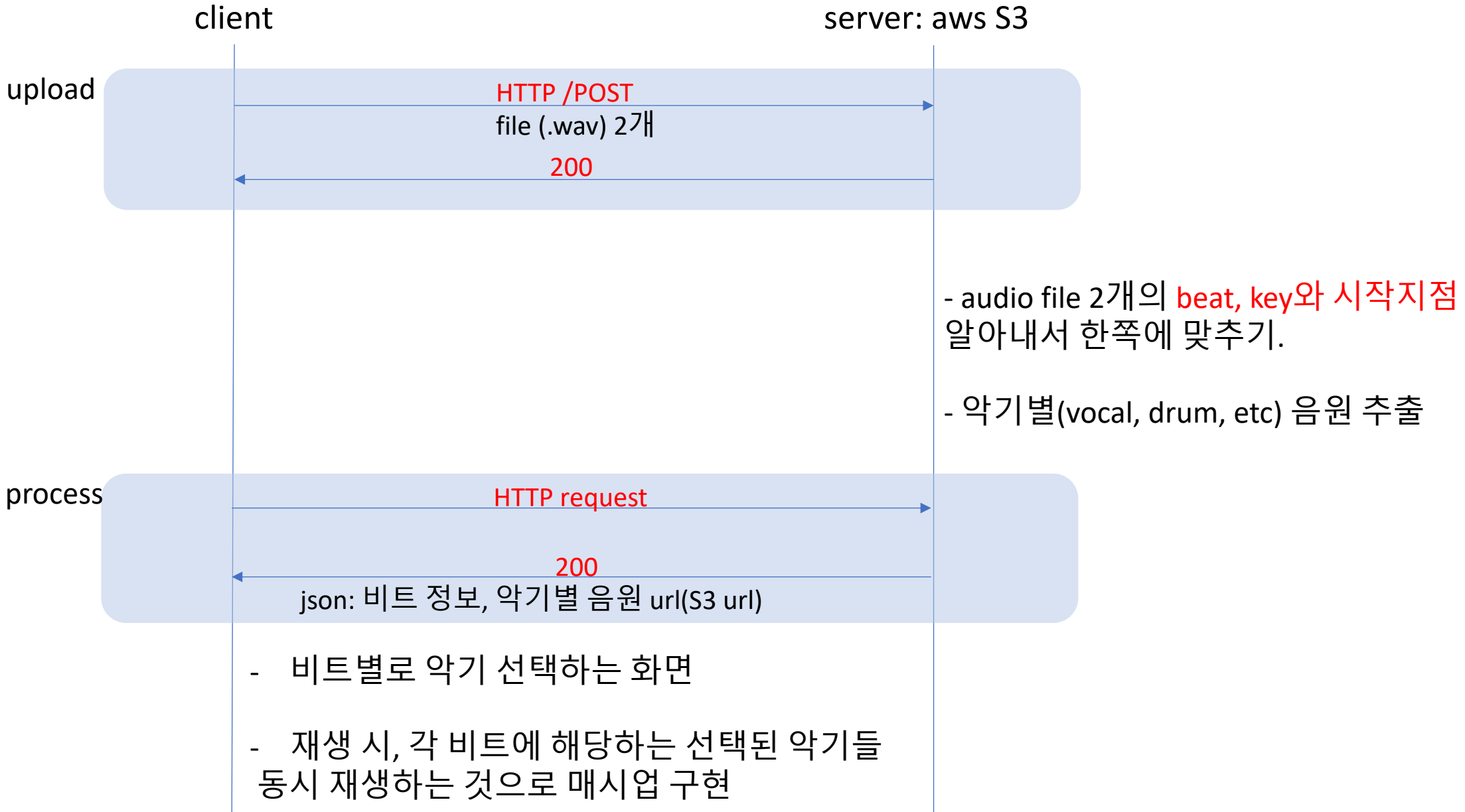
Code	Description	Links
200	success	*No links*

MLOps - Flask

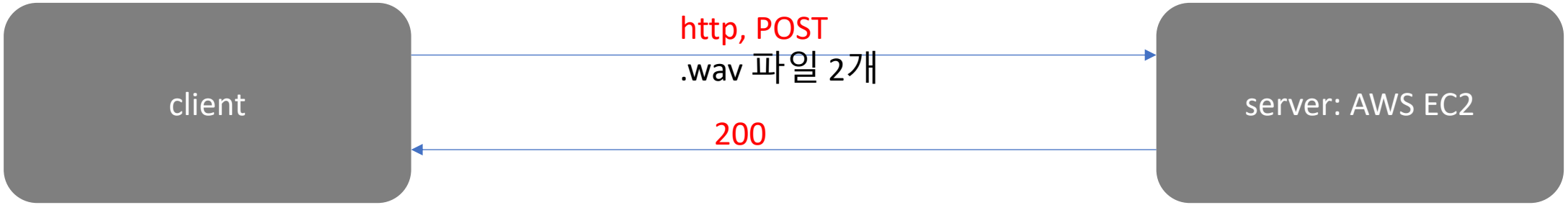
API Server

Frontend

how it works

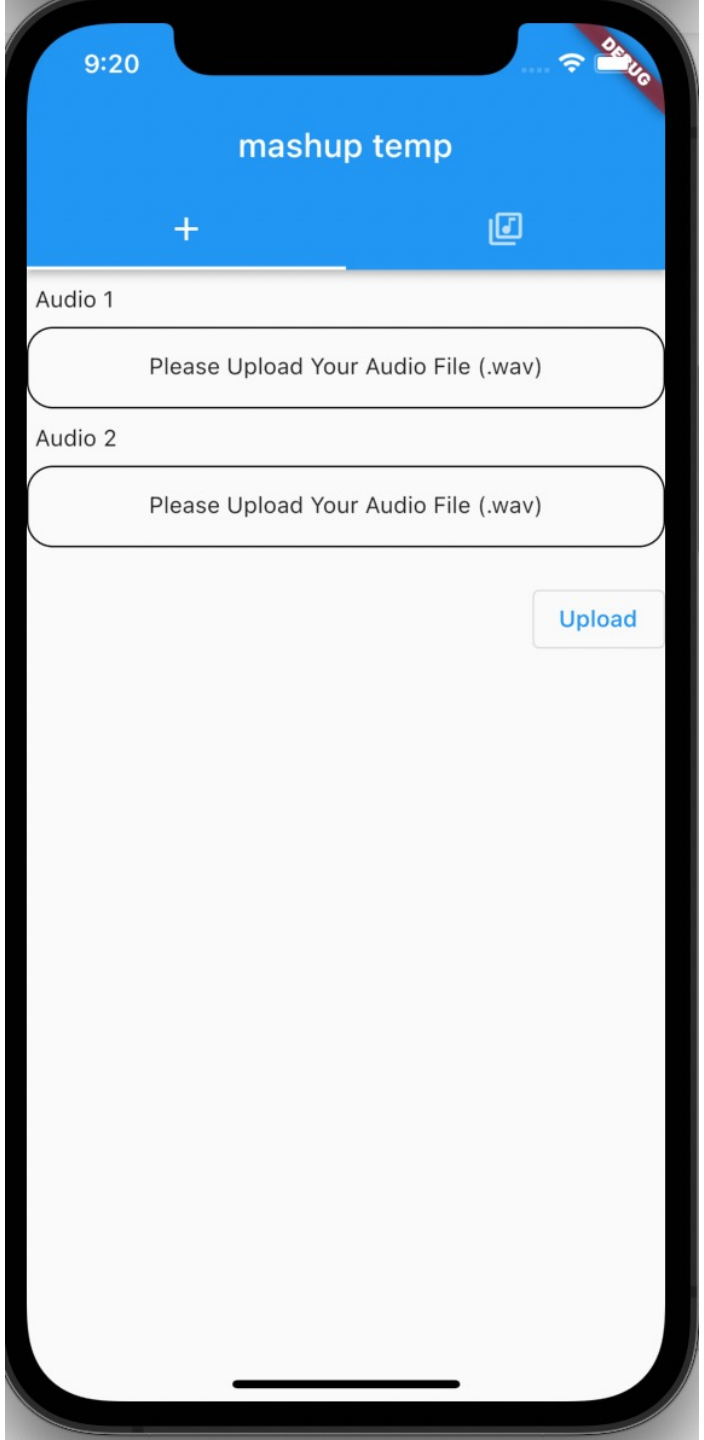

client
server: aws S3
upload
HTTP /POST
file (.wav) 2개
200
- audio file 2개의 beat, key와 시작지점 알아내서 한쪽에 맞추기.
- 악기별(vocal, drum, etc) 음원 추출
process
HTTP request
200
json: 비트 정보, 악기별 음원 url(S3 url)
- 비트별로 악기 선택하는 화면
- 재생 시, 각 비트에 해당하는 선택된 악기들 동시 재생하는 것으로 매시업 구현

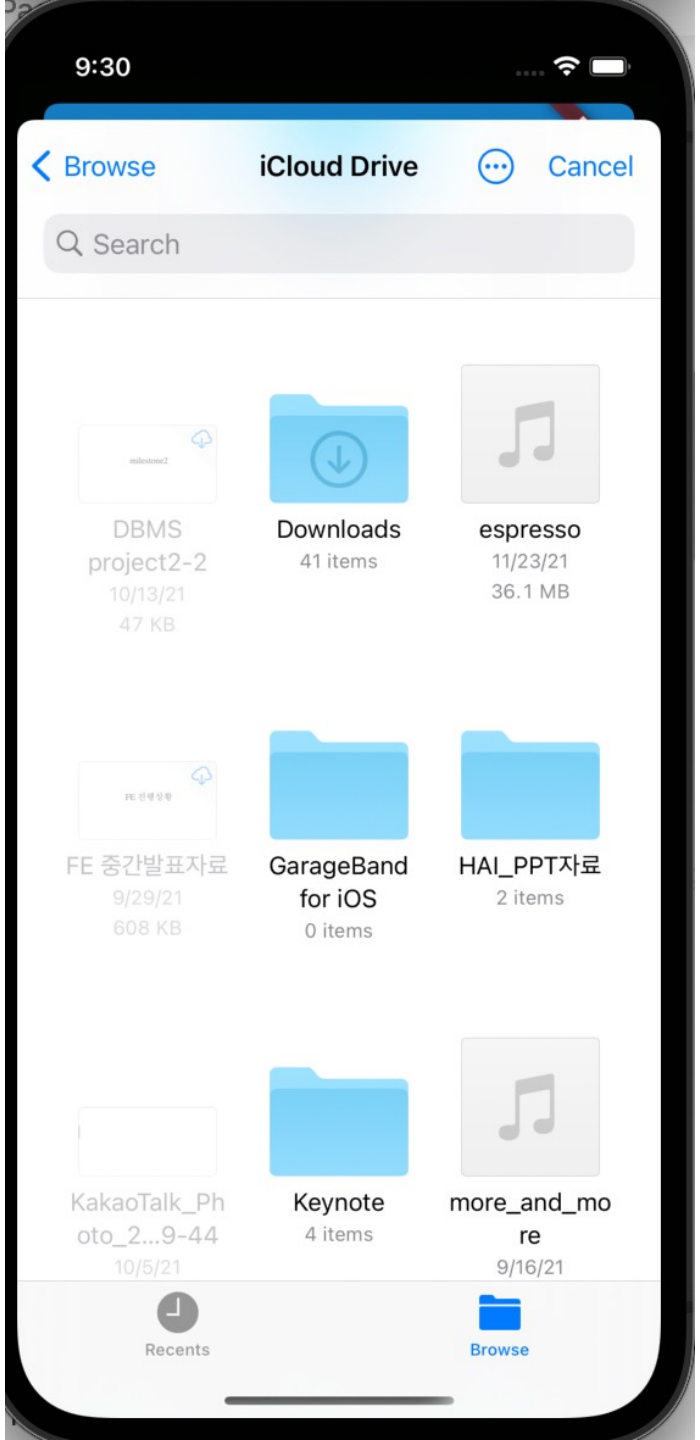
work flow - upload



구현화면 - upload

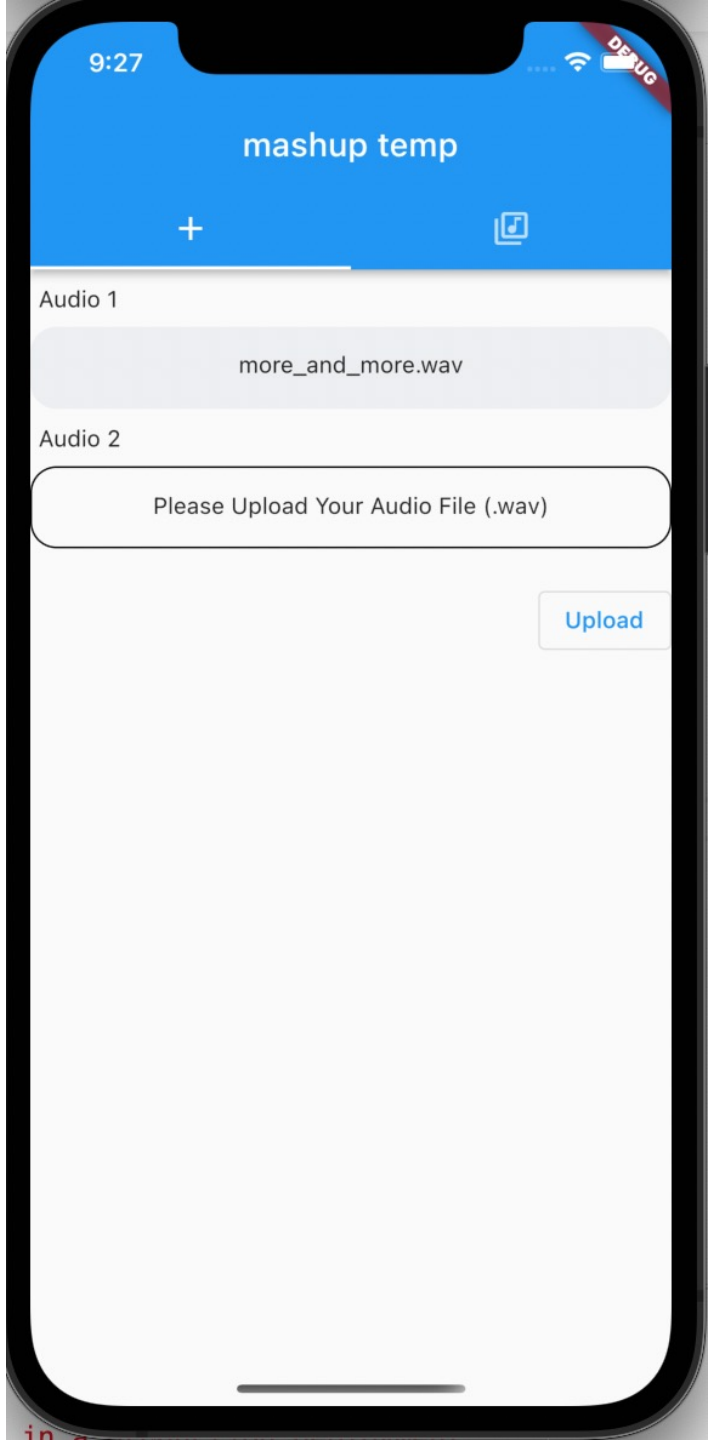
- Upload 버튼을 눌러, wav 파일 선택

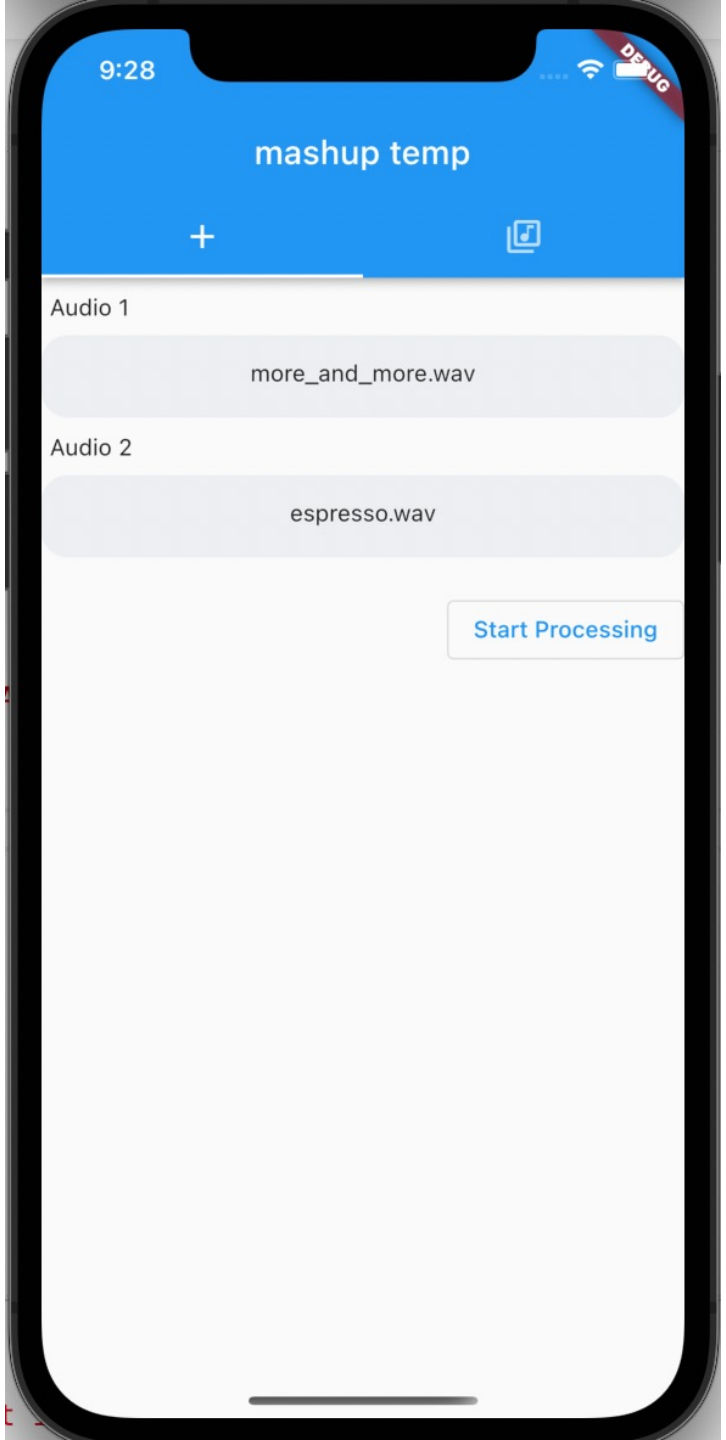




구현화면 - upload

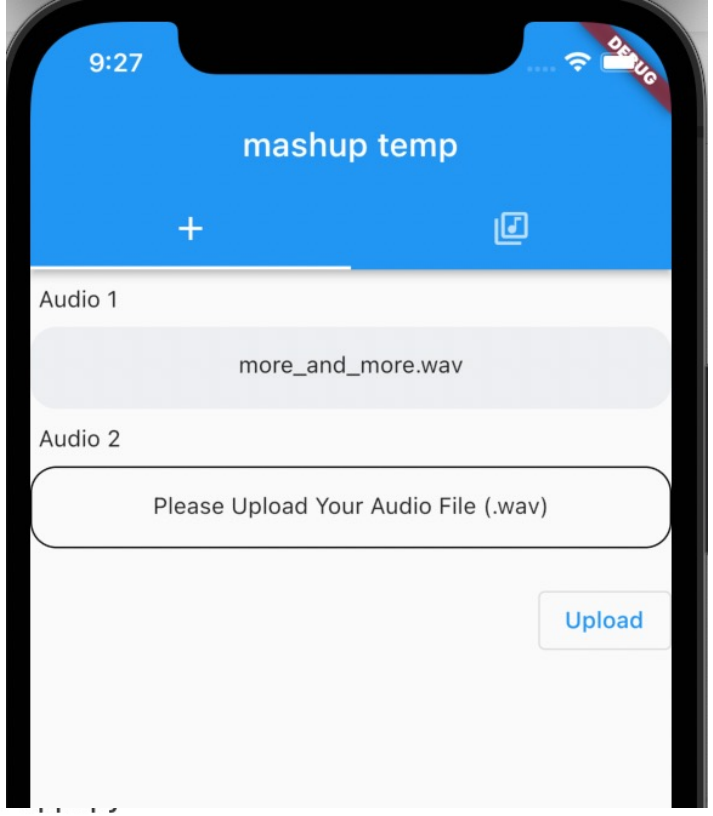
- 2개의 파일이 업로드되면
Start Processing 버튼으로 바뀐다.

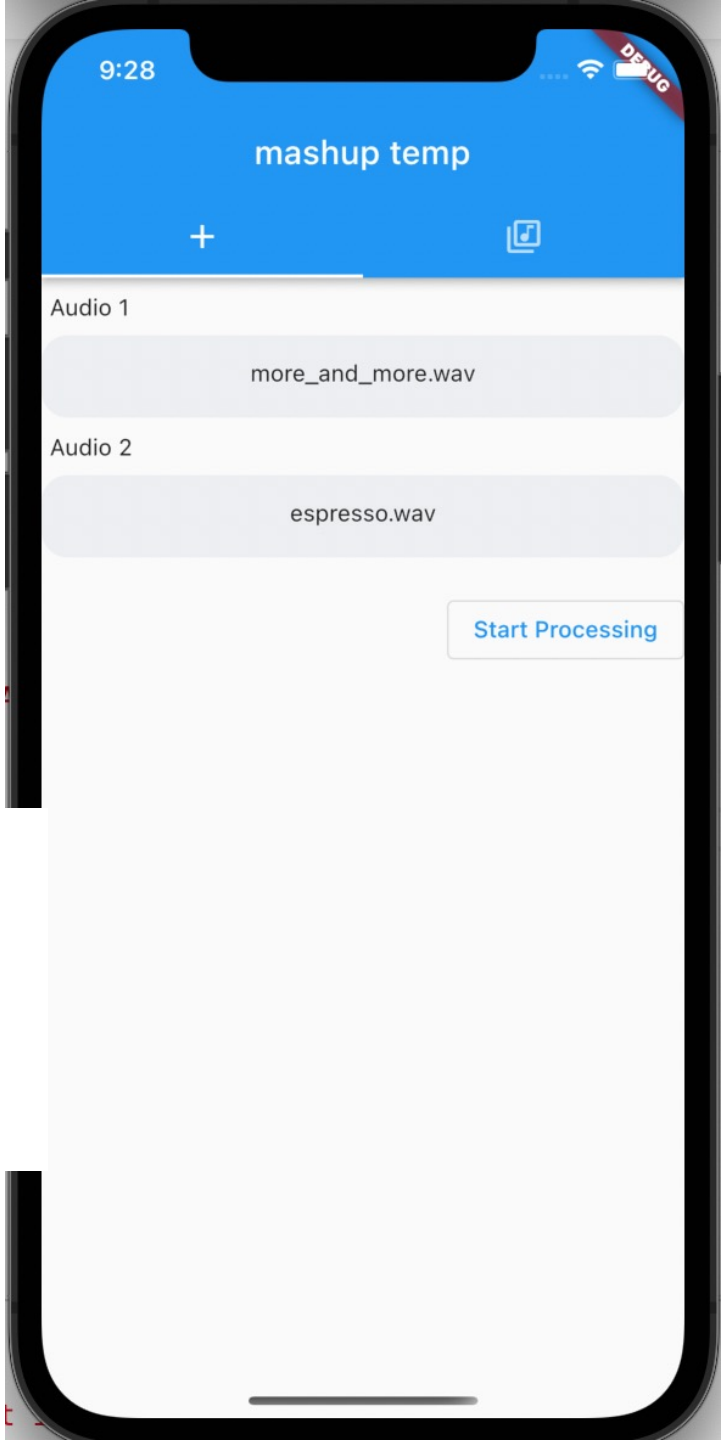




구현화면 - upload

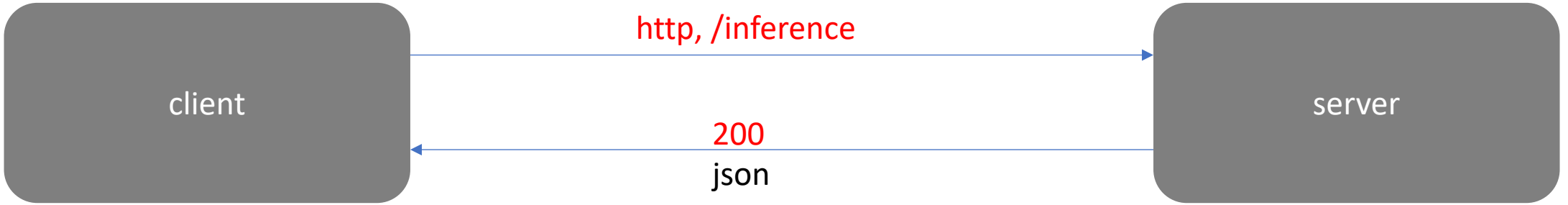
- 2개의 파일이 업로드되면
Start Processing 버튼으로 바뀐다.






```
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.31.45.80:5000/ (Press CTRL+C to quit)
58.230.143.202 - - [30/Nov/2021 12:27:33] "POST /uploadFile HTTP/1.1" 200 -
58.230.143.202 - - [30/Nov/2021 12:28:04] "POST /uploadFile HTTP/1.1" 200 -
```

work flow - process

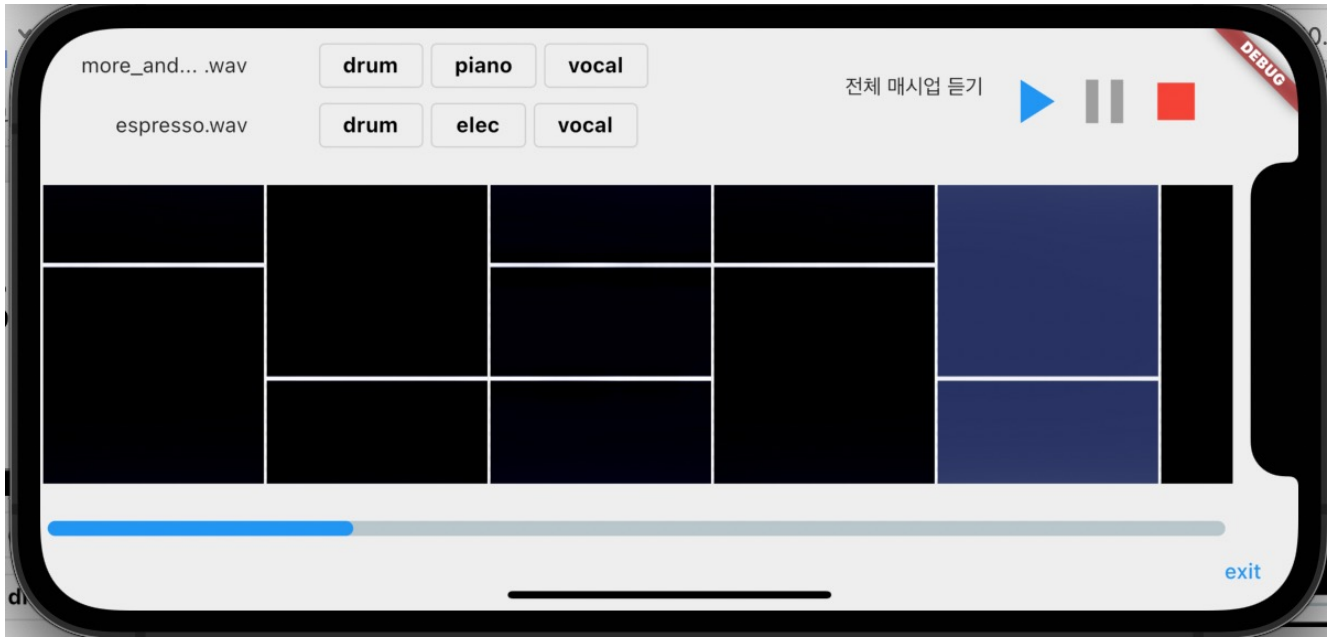


json 예시

```
{
  'first_beat': 0.12,
  'last_beat': 60.0,
  'num_beat': 129,
  's3_url': "https://test-soomin-bucket.s3.ap-northeast-2.amazonaws.com/audios/"

}
```

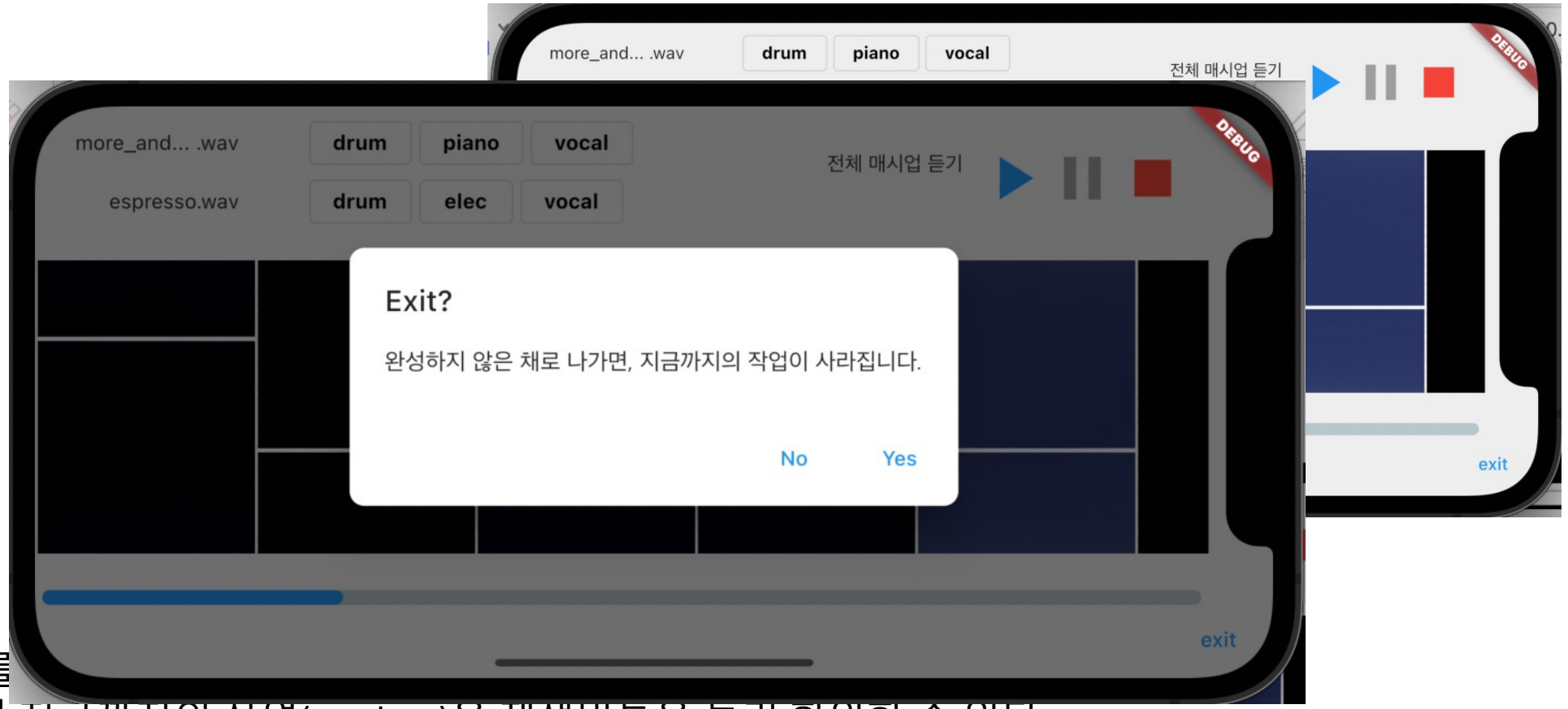
구현화면 - process



- 트랙 안의 비트를 tap한 후, 악기 버튼을 눌러 해당 비트의 악기를 선택한다.
- 작업을 수행하며 지금까지의 작업(mashup)을 재생버튼을 눌러 확인할 수 있다.

progress bar가 마지막에 도달하면(모든 비트의 악기를 결정하면) 다운로드 화면으로 이동한다.

구현화면 - process



- 트랙 안의 비트를
- 작업을 수행하며 지금까지의 작업(mashup)을 재생버튼을 눌러 확인할 수 있다.

progress bar가 마지막에 도달하면(모든 비트의 악기를 결정하면) 다운로드 화면으로 이동한다.

구현화면 - download

- mashup 완료된 곡들을 S3에 저장하고, 다운로드할 수 있게 하려 했으나,
- 분리된 음원 파일들을 합치는 로직 구현x

- 현재 다운로드는 불가능.

Simulation