<PROJECT 최종보고서>

SURFI

Flutter 앱 개발 및 영상 정보 분석을 통한 설명문구 추천,

필터링 기능 개발

사공훈, 윤민수, 허남현

주한별 교수님 분반

서문길 (단비아이엔씨)

Table of Contents

A. Abstract

이 보고서는 창의적 통합설계 프로젝트로 개발 Flutter 기반의 장소 서핑 어플 'SURFI'에 대한 기말보고서입니다. 단비아이앤씨에서 런칭을 기획중인 SURFI 는 3 초짜리 장소 동영상을 서핑할 수 있는 기능을 제공하며, 추가적으로 AI 기능을 활용하여 동영상에 대한 설명문구를 추천하고, 부적절한 콘텐츠를 필터링하는 기능을 탑재하고 있습니다. 이 어플은 궁극적으로 사용자들이 맛집이나 관광지와 같은 핫플레이스를 쉽게 탐색할 수 있도록 돕는 것을 목표로 하고 있습니다. 이 보고서는 프로젝트 소개, 수행 목표 및 접근방법을 시작으로 Flutter 기반의 프론트엔드와 Firebase 기반의 백엔드, 마지막으로 Clip model 을 활용한 AI 기능의 개발 결과와 구현 방법에 대한 설명을 다루고 있습니다.

B. Introduction

이 보고서는 Flutter 기반의 장소 서핑 어플 개발 프로젝트에 대한 기말보고서입니다. 해당 프로젝트는 단비아이앤씨에서 제공한 서비스컨셉 및 기획안을 바탕으로, 기획안의 요구사항을 최대한 만족하는 어플을 시간 내에 개발하는 것을 목표로 했습니다. 이를 위해 중간 발표까지는 프론트엔드 개발과 AI 기능 개발을 중점적으로 수행했으며, 기말까지 백엔드 개발을 마무리하고 AI 기능을 어플에 탑재 완료했습니다.

이 프로젝트는 사용자들이 새로운 장소를 탐색할 수 있는 새로운 방법을 제공합니다. 기존의 인터넷 검색이나 다른 어플리케이션은 대부분 텍스트나 정적인 이미지로만 정보를 제공하므로, 사용자들이 실제 그 장소를 파악하기 어려웠습니다. 하지만 SURFI 는 실제 장소 동영상을 제공함으로써 보다 생생하고 구체적인 정보를 전달합니다. 사용자들이 자연스럽게 장소의 분위기와 상황을 파악함으로써 원하는 장소를 선택할 수 있도록 돕는 것이 이 프로젝트의 의의입니다.

물론 만족스러운 서비스를 만들기 위해서는 새로운 장소를 추천해줄 뿐만 아니라 사용자가 원하는 것이 명확할 때 그에 해당하는 콘텐츠를 제공할 수 있어야 합니다. 이를 만족하기 위해서는 검색 기능이 구현되어야 합니다. 검색은 기본적으로 찾고자 하는 것을 나타내는 query 와 query 에 해당하는 것을 선별하기 위한 label 이 필요합니다. 사용자는 동영상을 업로드할 때 다른 사용자들에게 자신의 콘텐츠를 노출시키기 위해 태그를 붙이는데, 서비스는 이 태그를 검색을 위한 label 로 사용할 수 있습니다. 따라서 사용자의 편의성을

위해 태그 추천 기능을 추가하고자 하고, 동영상이 태그 없이 업로드 되더라도 검색 가능하도록 자동 태깅 기능을 구현했습니다.

또한, 부적절한 콘텐츠는 사용자들에게 불편함을 느끼게 하고 서비스에 대한 반감과 부정적인 이미지를 각인시킬 수 있습니다. 하지만 부적절함은 정의가 어렵고 명확한 기준을 잡기 어려워 자동화된 알고리즘을 개발하기 어렵습니다. 따라서 많은 서비스들은 신고제도 뿐만 아니라 큰 비용을 들여 리뷰어들을 고용합니다. 이 프로젝트는 필터링 비용을 줄이면서 초기 서비스에 치명적일 수 있는 false filtering 을 최소화 하고자 당연한 true positive 만 필터링하는 간단한 알고리즘을 구현했습니다.

접근 방법으로는 Flutter, Firebase, OpenAI CLIP Model 를 활용하여 프론트엔드와 백엔드, AI 기능 개발을 수행했습니다. Flutter 를 통해 사용자 친화적인 UI/UX 를 구현하고 Firebase 를 통해 데이터베이스와 연동하여 필요한 정보를 저장 및 처리하며, CLIP model 을 통해 동영상 해시태그 추천 및 콘텐츠 필터링 기능을 구현했습니다.

이 보고서는 프로젝트 소개, 사전 연구, 목표 및 요구사항, 문제 정의 및 접근방법, 프로젝트 구조, 구현사항, 개발 방법, 개발 결과, 업무 분담, 스케줄표로 구성되어있습니다. 특히, 프론트엔드 개발 과정과 백엔드 개발 과정, AI 기능 구현에 대한 전반적인 이해를 돕고자 합니다.

C. Background Study

A. 관련 접근방법/기술 장단점 분석

앱 개발을 위해 프론트엔드로는 Google 에서 개발한 프레임워크 Flutter 를, 백엔드로는 Firebase 의 데이터베이스, 인증, 스토리지, 함수 기능을 사용했습니다. Flutter 의 장점으로는 우선, 크로스 플랫폼 개발이 가능해 하나의 코드베이스로 ios 와 Android 를 비롯한 다양한 플랫폼에서 앱을 개발할 수 있다는 점이 있습니다. 또한, Hot Reload 기능을 통해 코드 변경 후 바로 결과를 확인 할 수 있어 개발시간을 단축시킬 수 있고, GPU 를 사용해 빠른 렌더링과 다양한 애니메이션 기능을 사용할 수 있다는 장점도 있습니다. Firebase 의 장점으로는, 클라우드 서비스이기 때문에 서버 구축이 필요하지 않으며, 앱 개발을 단축시킬 수 있다는 점이 있습니다. 비록, 대규모 데이터

처리나 복잡한 서버 로직이 필요한 경우에는 부적합하지만, 이용자 수가 적은 단계에서는 큰 문제가 되지 않기 때문에 주어진 시간 안에 SURFI 의 빠른 개발을 위해서는 Flutter 와 Firebase 의 조합이 최선이라고 판단해 위와 같은 접근 방법을 선택하게 되었습니다.

자동 태깅과 필터링은 classification 으로 구현할 수 있습니다. Classification 은 supervised learning 의 일종으로 데이터가 들어왔을 때 데이터에 해당되는 라벨을 예측하는, dataset 에서 label set 으로 mapping 하는 함수를 찾는 것입니다. Image classification 은 deep learning 이 성공적으로 적용된 분야 중 하나로 활발히 연구되어 왔고 ImageNet 과 같은 벤치마크에서 높은 성능을 보여주는 ResNet, Vision Transformer 개발되었습니다.

하지만 함수로 classification 할 경우 label set 이 달라질 때 새로운 codomain 으로 대응시키는 함수를 다시 찾는 finetuning 이라는 과정이 필요하다는 단점이 있습니다. CLIP 은 image 에서 label 로 직접 mapping 하는 함수를 학습하지 않고, text-image pair 에 대해서 image 의 representation 과 text 의 representation 이 joint embedding space 에서 cosine similarity 가 maximize 되도록 image 와 text encoder 를 학습시키는 방법을 사용합니다. 더욱 효과적인 학습을 위해 text-image pair 의 거리를 좁히는 것 뿐만 아니라 batch 에 속한 다른 text / image 의 embedding 과는 embedding space 에서 거리가 멀어지도록 하는 contrastive learning framework 을 사용합니다. CLIP 은 text 와 image encoder 가 분리되어 있어 임의의 text-image 의 similarity 측정할 수 있고, finetuning 없이 text-image 에 대해 추론할 있습니다.

CLIP 은 인터넷에서 크롤링된 4 억개의 text-image pair 에 대해서 contrastive learning framework 으로 학습되어 zero-shot image classification, few-shot learning task 에서 좋은 성능을 보여주고 있습니다. 더불어 CLIP 의 representation 은 image classification 뿐만 아니라 DALLE-2 의 기반이 되어 text-conditioned image generation 에서도 좋은 성과를 보여줬습니다. CLIP 의 representation 이 다양하게 적용 가능하다는 것이 밝혀진 만큼 SURFI 의 콘텐츠에 대해서도 좋은 성능을 보여줄 것으로 생각되어 CLIP 을 사용하게 되었습니다.

B. 프로젝트 개발환경

앱 개발의 개발환경은 다음과 같습니다. 프레임워크로는 Flutter 를 선택해 Dart 언어를 기반으로 프론트엔드를 개발하고 있습니다. 백엔드로는 Firebase 에서 제공하는 서비스를 이용했으며, ios, Android 를 Target platform 으로 개발했습니다. Github 을 통해 협업했으며, 단비아이앤씨에서 제공한 서비스컨셉과 기획안, 그리고 SURFI 와 UI 가 유사한 TikTok 을 참고해 개발했습니다. CLIP 은 Pytorch 로 모델을 로딩해서 Flask 프레임워크로 서빙하고 있습니다.

D. Goal/Problem & Requirements

이 프로젝트의 목표는 3 초 짜리 장소 동영상을 유저들이 쉽게 탐색할 수 있는 어플리케이션을 개발하는 것입니다. 이를 위해 Flutter 와 Firebase 를 사용하여 프론트엔드와 백엔드를 개발하고, AI 기능을 탑재하여 유저들의 사용 경험을 향상시키고자 합니다. 요구사항은 다음과 같습니다.

- 유저들은 Google 회원가입 및 로그인을 할 수 있어야 한다.
- 어플리케이션에 등록된 장소들의 3 초짜리 동영상을 유저들이 쉽게 탐색할 수 있어야 한다.
- 유저들은 동영상 촬영기능을 통해 주변 장소를 3 초간 촬영하고, 주소와 해시태그를 포함해 서핑포인트를 생성할 수 있어야 한다.
- 유저들은 어플리케이션에 등록된 장소들을 거리순, 시간순으로 상하좌우 스와이핑을 통해 탐색할 수 있어야 한다. 또한, 편의를 위해 나침반 모드, 럭키 모드 등의 기능이 제공되어야 한다.
- 댓글 기능을 통해 해당 영상에 대한 코멘트를 남길 수 있어야 한다.
- 검색 기능을 통해 유저 이름, 해시태그 등을 입력하고 조건에 맞는 서핑포인트만 탐색할 수 있어야 한다.
- 프로필 화면에서 유저 프로필을 탐색하고, 편집할 수 있어야 한다.

태깅과 필터링을 구현하기 위해서는 SURFI 에 업로드된 동영상에 대해 classification 을 할 수 있어야 합니다. Video 는 시간과 공간에 대해 연속적인 image 의 수열로 정의할 수 있으며, 이는 image 에 비해 변화에 대한 정보가 추가된 것으로 생각할 수 있습니다. Video classification 을 하기 위해서는 각 프레임에 대한 계산 뿐만 아니라 프레임 간의 dependency 에 대한 계산까지 해야 합니다. Image classification 의 복잡도를 O(X)라고 할 때, temporal kernel size 가 k 인 temporal convolution 을 사용하면 복잡도가 O((n-k+1)*X)로

증가합니다. 한번에 처리해야하는 데이터의 양도 증가함에 따라 공간 복잡도도 증가해 계산 자원에 부담을 가하게 되고 latency 에 영향을 줍니다. 하지만 SURFI 는 동영상의 길이가 3 초로 제한되어 있어 공간에 대한 큰 변화가 생길 확률이 낮습니다. 따라서 하나의 프레임을 샘플링해서 그 프레임에 대해 앞서 언급된 CLIP 으로 image classification 하는 방법으로 자동 태깅을 구현하기로 결정했습니다. 자동 태깅을 위해서는 다음과 같은 기능들이 요구됩니다.

- 동영상을 잘 나타내는 안정적인 프레임을 빠르게 샘플링하는 알고리즘이 필요하다.
- SURFI 에 업로드될 콘텐츠를 잘 표현하며 CLIP 의 예측 정확도가 높은 label set 을 만들어야 한다.
- 샘플링된 프레임에 대해 빠르게 추론할 수 있도록 효율적인 serving framework 가 필요하다.

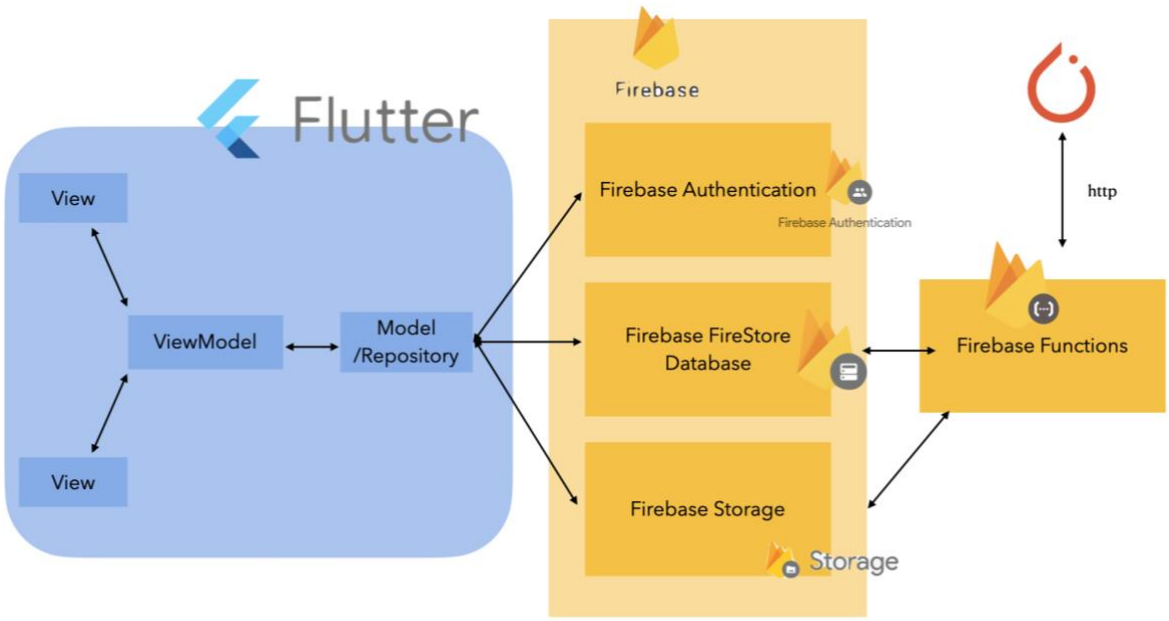
E. Approach

[프로젝트 수행 목표 또는 문제 정의 및 접근방법]

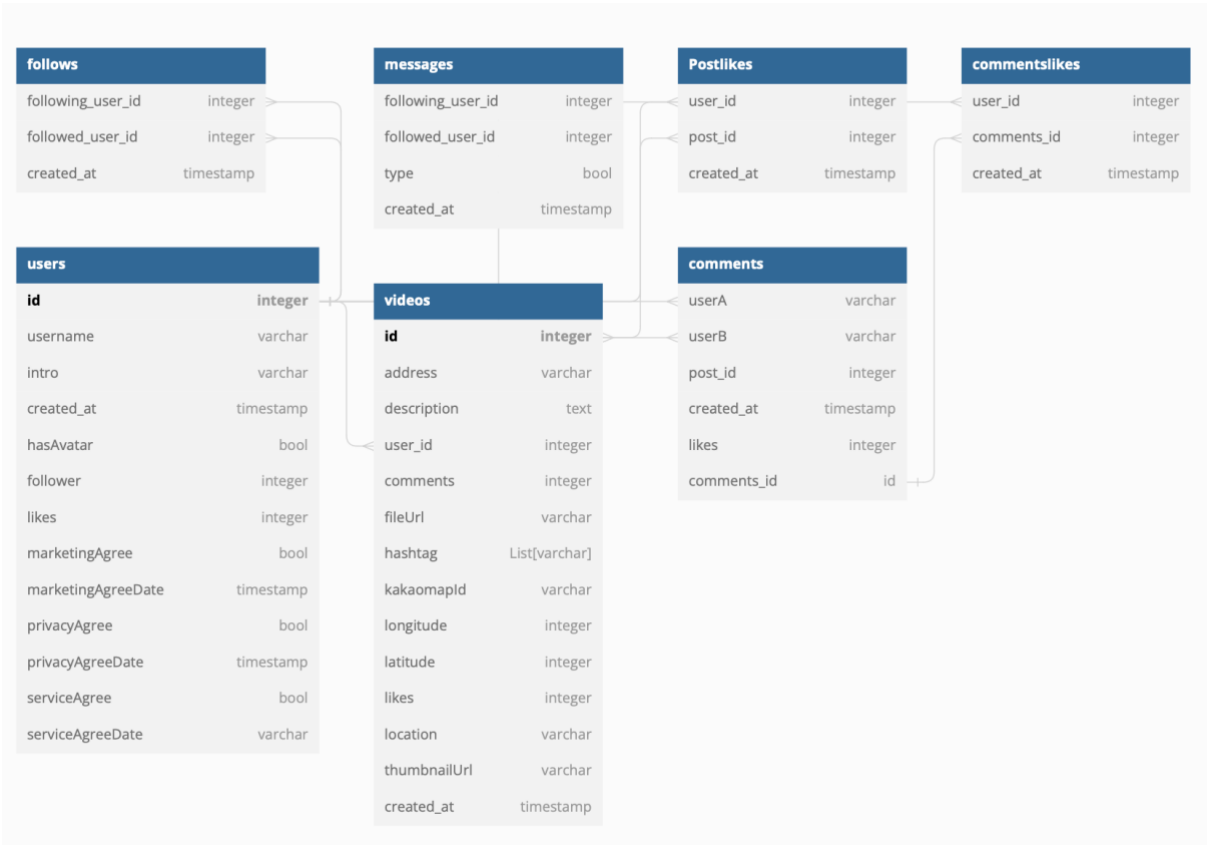
이번 프로젝트는 사용자들에게 공간을 탐색하는 새로운 방법을 만들어내는 것을 목표로 하고 있습니다. 기존의 경우 검색에서 상위 노출되는 장소만 확인하거나 일부 사진을 통해서 정보를 얻었고 이러한 정보들은 상당히 제한적이고 정보를 얻는데 많은 노력을 들여야 했습니다. 그러나, 이 프로젝트 서피에서는 공간을 탐색하는 일이 간단하고 직관적일 뿐더라, 사용자들에게 미처 몰랐던 우리 주변의 장소들을 알아가는 즐거움을 주는 것을 목표로 하고 있습니다. 회사에서는 기획안이 있었으나 개발은 되지 않은 상태였기 때문에, 좋은 아이디어의 빠른 구현을 위해서 Flutter 와 Firebase 조합으로 앱을 개발하고 있습니다. 저희는 앱 개발을 하면서 유저들이 만족할 수 있는 경험을 선사하기 위해서 다양하고 직관적인 UI 를 부드럽게 구현하는 것을 목표하고 있습니다. 또 동영상을 다루는 앱인 만큼 서버에 부하가 많이 가게 되고 이는 많은 비용이 드는 것과 함께 사용자 경험을 해치는 요인이 되기도 합니다. 그렇기 때문에 캐시와 효율적인 db 와 storage 관리를 통해서 서버에 부하를 최대한 줄이는 것이 목표이기도 합니다.

F. Project Architecture

A. Architecture Diagram



<Architecture Diagram>



<DB Diagram>

B. Architecture Description

플러터 앱 개발의 경우 MVVM 아키텍쳐를 기반으로 개발하고 있습니다. MVVM 은 Model-View-ViewModel 의 약자로 UI(View)를 비즈니스 로직이나 백엔드 로직(Model)로 분리하는 아키텍쳐 패턴입니다. ViewModel 은 UI 의 변화에 따라 데이터의 전환을 담당합니다. 상태 관리의 경우 riverpod 을 사용하고 있습니다.

백엔드의 경우에는 Firebase 를 사용하고 있는데, 여기서 사용하고 있는 서비스는 크게 네가지 입니다.
첫 번째는 firebase authentication 으로, 사용자 인증의 여러가지 방식을 지원하는데 이를 통해서 구글 로그인과 회원가입을 구현하고 있습니다. 두번째는 Cloud Firestore 으로 유연하고 확장 가능한 NoSQL 클라우드 데이터베이스를 사용해 클라이언트 및 서버 측 개발에 사용되는 데이터를 저장하고 동기화하고 있습니다. 세번째 Firebase Storage 를 통해서 Firebase 용 Cloud Storage 는 썸네일과 같은 사진, 그리고 유저가 촬영한 동영상을 저장하기 위한 용도로 사용하고 있습니다. 마지막으로 firebase functions 는 Firebase 기능과 HTTPS 요청에 의해 트리거되는 이벤트에 응답하여 백엔드 코드를 자동으로 실행할 수 있는 서버리스 프레임워크입니다.

위의 DB 다이어그램에서는 SQL 에서 사용되는 Entitiy-Relationship 을 표현하였지만, nosql document 환경에서는 이와 다른 설계를 가지기 때문에 위의 데이터 베이스 다이어그램은 개념적인 모형입니다. 실제 프로젝트에서는 nosql 환경에 최적화하기 위해서 denormalize 작업이 되어있습니다. Denormalize 는 데이터베이 스키마 디자인의 한 방법으로 데이터의 중복성을 허용하고 여러 테이블에 분산되어 있는 데이터를 단일 문서로 결합함으로써 쿼리의 성능을 향상시키는 방법입니다. 관계형 데이터베이스에서는 정규화라는 프로세스를 사용하여 데이터를 중복없이 분리하지만, Nosql 데이터베이스에서는 중복된 데이터가 쿼리 성능을 향상시키는 데 유용한 경우가 많습니다.

저희 프로젝트의 예시로는 관계형 데이터 베이스에서는 비디오에 대한 정보와 유저에 대한 정보를 다른 테이블에 저장할 수 있습니다. 하지만, NOSQL 데이터베이스에서는 유저의 정보와 유저가 촬영한 동영상을 하나의 문서로 결합하여 저장할 수 있습니다. 이렇게 하면 쿼리에서 데이터를 가져오는 데 필요한 JOIN 연산을 피할 수 있습니다. 가령 유저 프로필에서는 유저가 촬영한 동영상 목록을 Gridview 로 나타내는데 이때 빠르게 데이터를 받아올 수 있습니다.

G. Implementation Spec

A. Input/Output Interface

Repo

Auth Repo	signOut()
	googleSignIn()
User Repo	createProfile()
	findProfile()
	uploadAvatar()
	updateUser()
	fetchThumbnail()
	toggleFollowUser()
	toggleAlarm()
Video Repo	uploadVideoFile()
	saveVideo()
	fetchVideos()
	likeVideo()
	isLikeVideo()
Comment Repo	addComment()
	deleteComment()
	fetchComments()
	likeComments()

자세한 내용은 부록을 참고해주세요

B. Inter Module Communication Interface

ViewModel

UsersViewModel	createAccount() onAvatarUpload()
	registerProfile()
	updateProfile()
	updateAgreement()
profileViewModel	fetchThumbnail()

avatarViewModel	uploadAvatar()
socialAuthViewModel	googleSignup()
	googleSignIn()
commentsViewModel	fetchComments()
	uploadComment()
	Refresh()
	deleteComment()
uploadVideoViewModel	uploadVideo
videoPostViewModel	toggleLikeVideo()
TimelineViewModel	fetchNextPage()
	refresh()

자세한 내용은 부록을 참고해주세요

C. Modules

Model

ThumbnailModel
UserProfileModel
CommentModel
VideoModel

H. Solution

A. Implementations Details

저희 앱을 구현한 핵심 코드에 대해서 설명하겠습니다.

히어탭에서는 영상을 가로축과 세로축으로, 상하로 스크롤할 경우 멀어지는 영상이 나와야하고, 좌우로 스크롤할 경우 같은 장소의 다른 영상이 나와야합니다. 이를 구현하기 위해서 먼저 현재 db 에 현재 거리부터 가까운 장소들을 탐색합니다. 장소들에 대한 1 차원 배열을 구한 다음, 그 다음 각 장소에 대해서 영상들을 가져와서 2 차원 배열을 얻습니다. 이를 장소를 세로축으로, 그리고 각각의 영상들을 가로축으로 정렬합니다.

나침반 모드에서는 휴대폰이 향하는 방향에 있는 영상만을 보여줍니다. Here 탭에서 장소들에 대한 목록을 가지고 있고 이를 기반으로 휴대폰에 내장된 나침반과 휴대폰의 위치, 그리고 각 장소들의 위치를 통해서 방향을 계산합니다. 우선 유저가 향하는 방향을

4 분할 해서 동서남북으로 나누고, 해당 방향에 존재하는 영상들을 보여줍니다. 그리고 유저가 향하는 방향이 달라진다면, 가령 북에서 동으로 바뀌게 된다면 보여주는 영상을 다르게 해줍니다. 유저가 향하는 방향은 90 도를 경계로 하였습니다.

오버뷰 모드에서는 히어탭의 2 차원 배열을 나침반 모드와 같이 휴대폰이 향하는 방향에 맞게 보여주는 것으로 수정해서 전체적인 배열을 한 번에 볼 수 있게 하였습니다. 이때는 썸네일만 보여주고 썸네일을 누를 경우 동영상이 재생되도록 하였습니다. 럭키모드에서는 영상을 랜덤으로 재생되도록 하였고, 검색 기능에서는 원하는 유저나 해시태그로 db 에서 제한에서 보여줍니다.

키워드 추출과 콘텐츠 필터링 부분에서는 로직이 크게 frame sampling 과 모델 추론으로 나눌 수 있습니다. 최종 구현된 frame sampler 의 로직은 전 프레임 대비 absolute pixel difference 가 가장 낮은 프레임과 무작위로 샘플링된 프레임 5 개를 추출해 총 6 개의 프레임을 뽑습니다. 키워드 추천 관점에서 프레임에 움직임과 같은 Noise 가 있을 경우 예측성능이 낮아질 가능성이 있어 하나의 안정적인 프레임을 뽑는게 목표였지만, 반대로 움직임이 많은 폭력적인 동영상의 경우 안정적인 프레임에 폭력 장면이 포함되지 않는다는 단점이 있었습니다. GPU 로 추론할 경우 batch 로 묶은 6 개의 프레임을 처리하는 것과 1 개의 프레임을 처리하는 latency 는 실질적으로 동일하기 때문에 추론 시간에 변화 없이 키워드 예측과 필터링을 위한 프레임을 뽑도록 구현했습니다.

Algorithm 1 Sampling Frames from a Video

```
procedure SAMPLE_FRAMES(video_filepath)
    Initialize VideoCapture with video_filepath as vidcap
    Get total frame count of video as total_frames
    Generate sorted list of 5 random integers within range of total_frames as random_frames_idx
    Initialize prev_image, current_image, min_diff_image to None
    Initialize min_diff to ∞
    Initialize returning_frames as an empty list
    for i in range of total_frames do
        Read the frame into current_image from vidcap
        if not successful in reading frame then
            break the loop
        end if
        if prev_image is not None then
            Compute absolute difference between prev_image and current_image as diff
            if diff is less than min_diff then
                Assign diff to min_diff and current_image to min_diff_image
            end if
        end if
        if i is in random_frames_idx then
            Append current_image to returning_frames
        end if
        Assign current_image to prev_image
    end for
    Insert min_diff_image at the beginning of returning_frames
    Release the video file in vidcap
    return returning_frames
end procedure
```

모델 추론은 필터링을 위한 추론과 키워드 예측을 위한 추론으로 나뉘어져 있습니다. 처음에는 image embedding 과 필터링 레이블 text embedding 의 cosine similarity 를 구해 softmax 를 취하고, 필터링 레이블의 확률이 threshold 를 넘지 않으면 image embedding 과 키워드 레이블 text embedding 의 cosine similarity 를 계산합니다. Image embedding 은 한번만 계산하고 text embedding 은 바뀌지 않기 때문에 저장해 놓을 수 있어 두번의 추론은 dot product 를 두 번 계산하는 것과 동일합니다. 폭력적인 동영상을 필터링하는 키워드는 'This photo contains violence'와 'violence'를 사용했고, 선정적인 동영상을 필터링 하는 키워드는 임의의 야한 동영상 제목 두개를 가져와서 예측하고 있습니다.
추론은 현재 flask server 에서 계산하고 결과를 반환하고 있습니다. Flutter 앱에 모델을 직접 로딩해서 추론하는 것을 시도했지만, Flutter 에서 영상처리를 하기 위해서는 동영상의 각 프레임을 모두 storage 에 저장한 후에 각 프레임을 다시 메모리에 로딩하는 방법이 유일해 비효율적이라고 판단했습니다.
추론 결과에 부적절한 키워드가 포함된 경우, JSON 의 flag key 가 1 로 설정되어 사용자에게 키워드가 추천되지 않도록 구현되어 있습니다. 또한, flag 된 동영상을 사용자가 업로드할 경우 warning 문구를 표시하는 것으로 구현되어 있습니다.

B. Implementations Issues

앱을 구현하면서 어려웠던 점은 두 가지였습니다. 첫 번째는 어떻게 같은 장소를 구분할 것이냐, 두 번째는 어떻게 거리순으로 영상을 정렬할 것이냐였습니다. 첫 번째로 동영상의 장소를 구분하기 위해서 저희는 촬영한 영상의 주소를 단일한 채널을 통해서 정하도록 했습니다. 동영상 촬영시 카카오맵 api 와 연결하여서 카카오맵에 있는 위치로 저장하였습니다. 장점으로는 카카오맵에 있는 주소를 사용해서 똑같은 장소에는 똑같은 위치가 들어가고, 장소의 구획기준 문제도 해결할 수 있었습니다. 가령, 장소를 얼마나 세분화해야할지 예를 들어 301 동 퀴즈노스, 탁구장, 1 층 복도, 104 호 강의실 등등 얼마나 세부적으로 장소를 올려야하는지 고민이 되는데 이 기준은 카카오맵에 올라와있는지 여부로 결정하기로 했습니다. 단점으로는 카카오맵에 없는 장소의 경우에는 장소를 올릴 수가 없었습니다. 가령 서울대학교 폐수영장의 경우 카카오맵에 등재되지 않아서 장소로 올릴 수 없었고 인근 장소였던 지진관측소로 올릴 수 밖에 없었습니다.

두번째는 어떻게 거리순으로 영상 정렬에 관한 문제입니다. 저희 앱에서는 사용자의 현재 위치에 따라서 거리순으로 영상을 정렬해서 보여주어야 하는데, 영상을 거리순으로 정렬하기 위해서는 모든 Db 를 훑어야 하는 비효율성이 일어납니다. 이 문제를 해결하기 위해서 저희는 지오해시를 사용하여 해결했습니다. Geohash 는 지리적 위치를 문자와 숫자의 짧은 문자열로 인코딩하는 공개 도메인 지오코드 시스템입니다. Geohash 는 지구 표면을 2 차원 그리드로 나누고각 그리드 셀에 고유한 문자열을 할당합니다. Geohash 의 길이를 늘리면 그리드 셀의 크기가 작아지고 위치의 정확도가 높아집니다. 이를 이용해서 유저의 현재위치가 속한 지오해시의 앞자리 5 자리가 일치하는 위치의 영상만을 우선적으로 fetch 하기로 했습니다. Db 에도 {geohash 앞자리 5 자리}/{geohash 뒷자리 4 자리}로 영상을 저장해서 geohash 앞자리 5 자리가 일치하는 영상을 우선적으로 확인해서 범위를 좁혀서 영상을 정렬하였습니다. 또한 나침반 기능이나 오버뷰 기능에서도 이와 비슷한 접근법으로 문제를 해결했습니다.

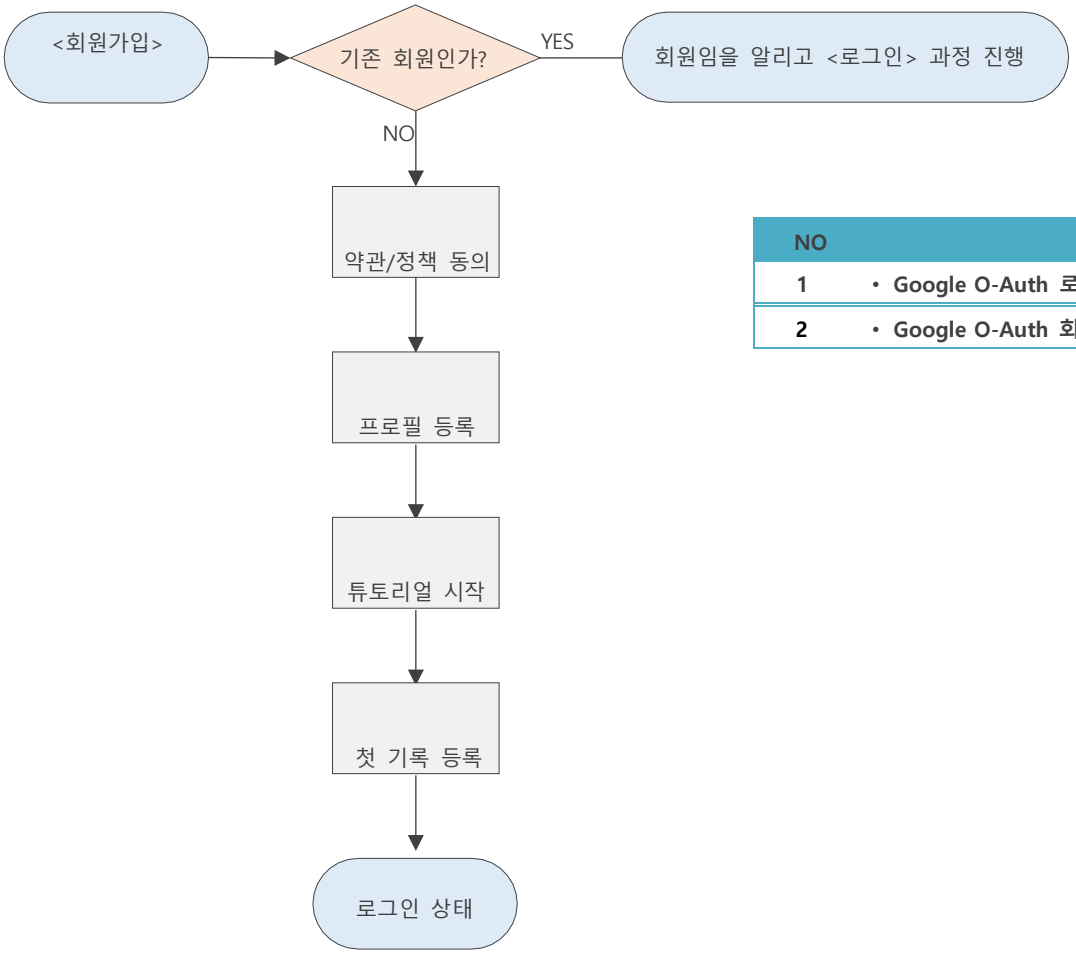
<서울대 인근 지역의 지오해시>

현재 모델 추론의 가장 큰 한계점은 network latency 입니다. GPU 자원을 사용하면 총 계산 시간은 평균 0.7 초 걸리지만, 사용 가능한 GCP GPU 자원의 물리적 거리가 멀어 전체 키워드 추출 로직은 평균 5 초 소요됩니다. 추론 서버가 멀리 있을 경우 network latency 가 전체 시간의 85%정도를 차지 하는 것으로 확인되었고, 반대로 가까운 곳에 위치한 CPU 서버를 사용할 시 network latency 가 크게 줄어드는 것을 확인할 수 있습니다. 서비스를 출시할 경우 국내에 위치한 GPU 서버를 사용하면 안정적으로 빠른 추론 속도를 제공할 수 있을 것이라고 생각합니다.

I. Results

앱	SURFI	화면 위치	LOGIN 화면
화면 기능	Firebase Authetication을 이용한 Google 로그인. 회원가입		

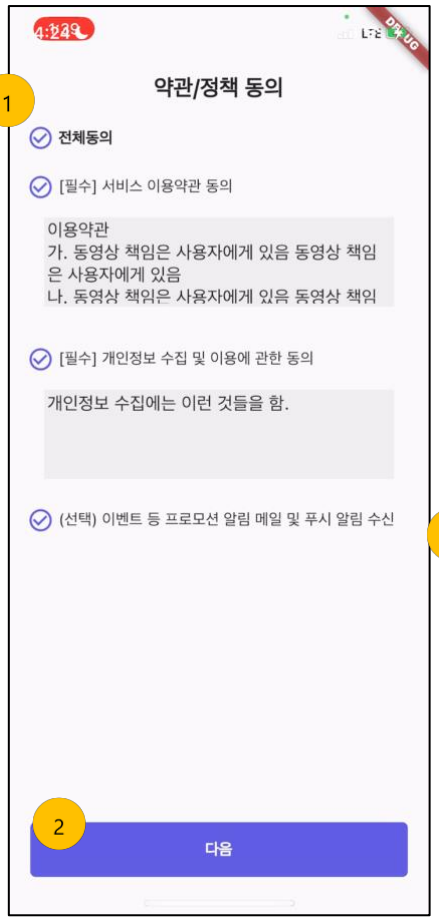
01-로그인



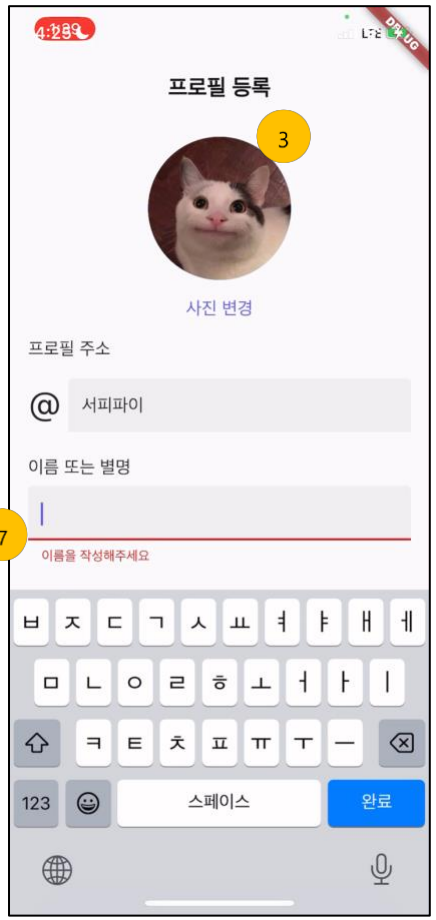
NO	설명
1	• Google O-Auth 로그인
2	• Google O-Auth 회원 가입 진행 -> 약관 동의화면 이동

앱	SURFI	화면 위치	LOGIN 화면
화면 기능	약관/정책동의 후 프로필 등록 완료		

02-약관/정책동의



03-프로필등록-입력중



03-프로필등록-입력완료

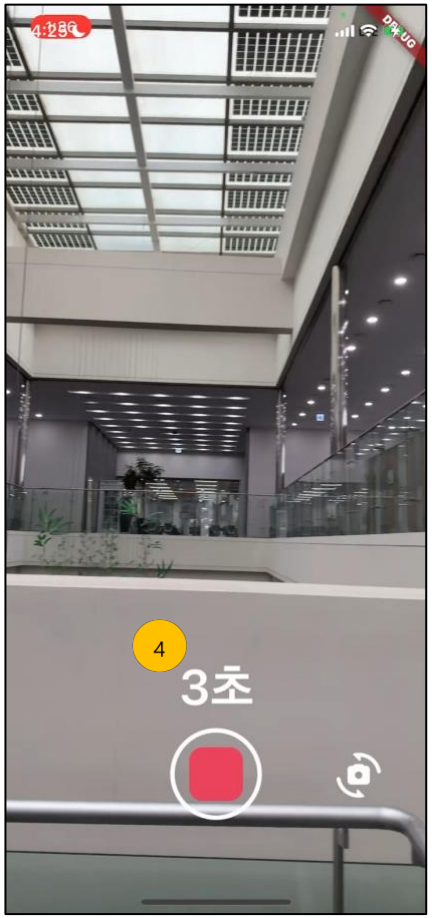
NO	설명
1	• Checkbox 모두 초기에 uncheck된 상태. 전체동의 박스를 통해 전체 Check 상태로 변경 가능. 필수 항목들이 모두 체크되면 '다음'버튼 클릭 가능.
2	• 클릭 시, 프로필 등록 화면으로 이동.
3	• 사진 변경을 눌러 폰 사진(갤러리)을 등록할 수 있음.
4	• 프로필 주소 입력란
5	• 이름 또는 별명 입력 란
6	• 자기소개 입력 란.
7	• Validator를 통해 요구조건 추가 설명
8	• 클릭 시 튜토리얼 화면으로 이동.

앱	SURFI	화면 위치	Tutorial 화면
화면 기능	튜토리얼 과정을 통해 첫 회원가입 시, 서핑포인트 생성을 해볼 수 있음.		

04-튜토리얼 시작

05-첫 촬영(준비)

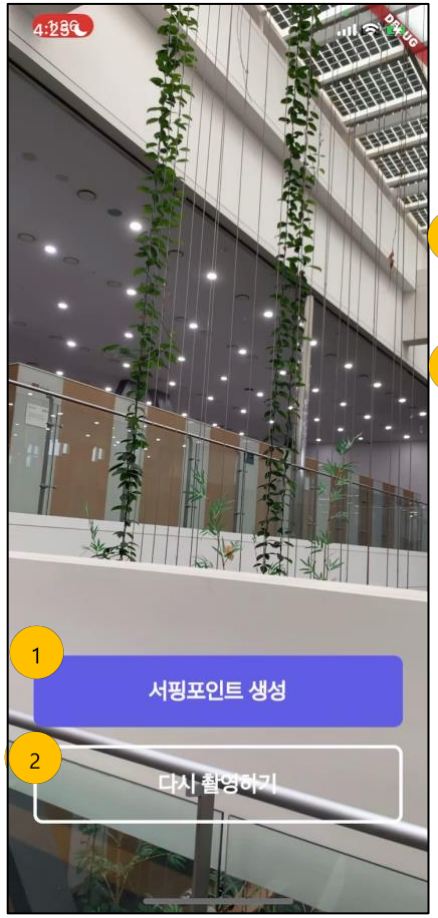
05-첫 촬영



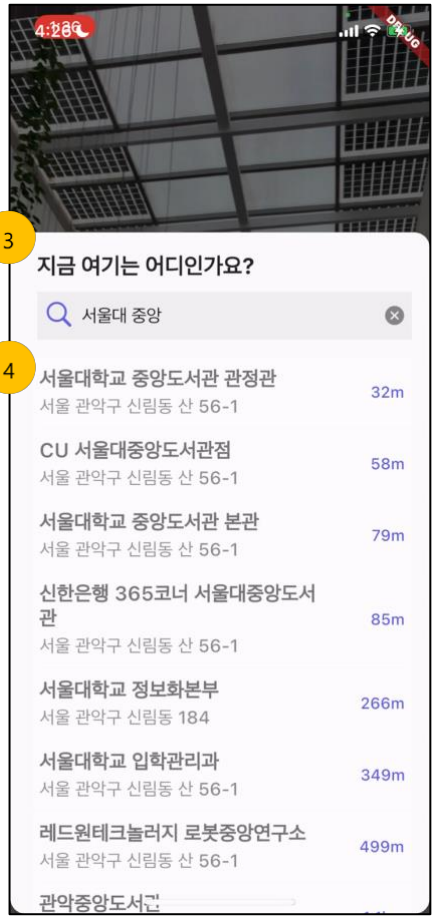
NO	설명
1	• 튜토리얼 시작 시 카메라가 켜지며, 현재위치를 확인할 수 있음. 클릭 시 촬영준비화면으로 이동
2	• 클릭 시, 촬영이 시작되며 녹화 중임을 직관적으로 알 수 있게 버튼 모양이 자연스럽게 변함.
3	• 클릭 시 selfi mode로 전환
4	• 1초, 2초, 3초 순서로 촬영시간을 알려주며 3초 도달 시 자동으로 촬영 종료

앱	SURFI	화면 위치	Tutorial 화면
화면 기능	촬영한 영상을 확인 후, 관련된 주소 및 설명문구를 입력할 수 있음.		

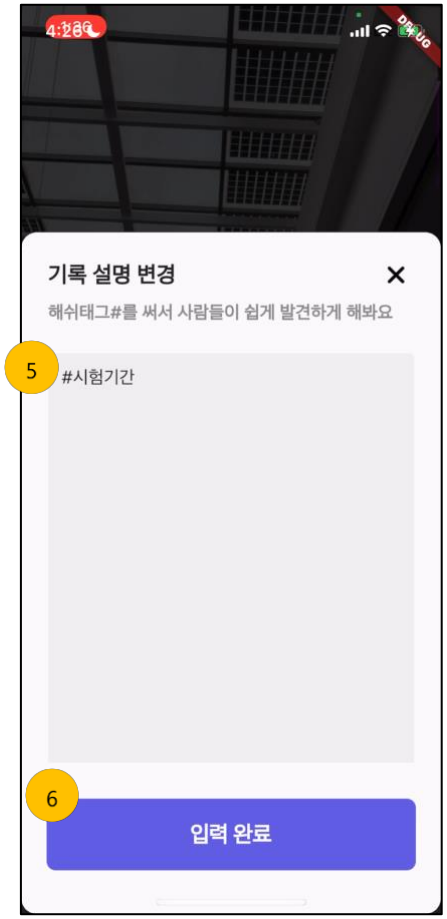
06-영상 확인 화면



07-주소 입력



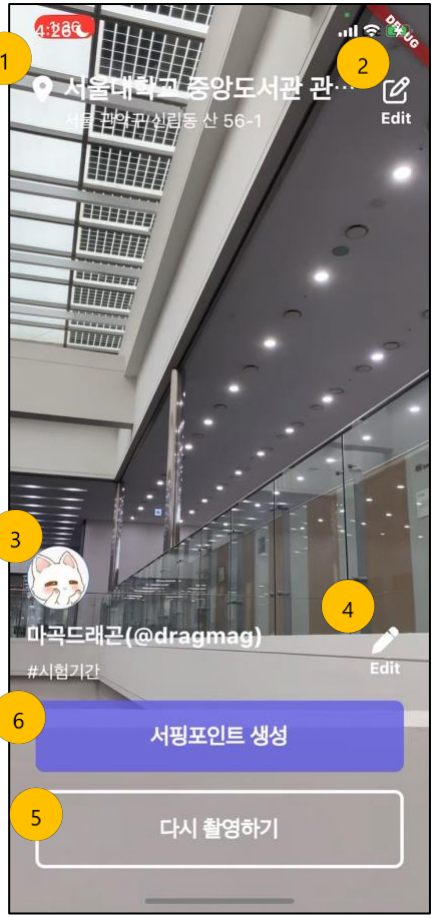
08-설명문구 입력



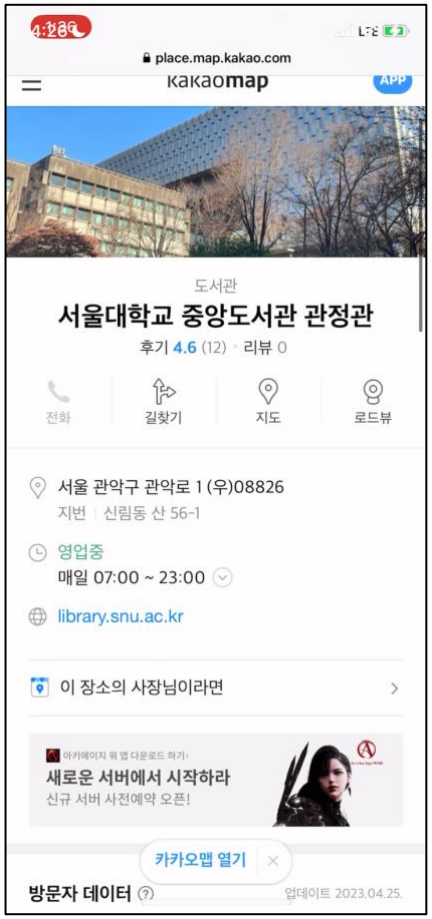
NO	설명
1	• 촬영한 영상이 계속 반복됨. 이를 확인하고, 클릭 시 주소 입력 화면이 팝업.
2	• 재촬영을 원할 시, 클릭하면 동영상 촬영화면으로 다시 이동하며 재촬영 가능.
3	• 카카오 로컬 API를 이용해 구현. 장소명을 입력하면 현재 위치를 기반으로 거리순으로 관련 장소들의 검색결과를 보여줌.
4	• 주소 클릭 시, 설명문구 입력 화면으로 이동
5	• 해쉬태그 등을 이용해 영상에 대한 설명문구 입력
6	• 클릭 시 입력한 정보들을 바탕으로 콘텐츠 확인 화면으로 이동.

앱	SURFI	화면 위치	Tutorial 화면
화면 기능	콘텐츠 확인 후, 동영상, 주소, 설명문구를 다시 한번 수정할 수 있음. 최종 확인 후 서버에 업로드 함으로써 튜토리얼 완료.		

09-콘텐츠 확인 화면



10-카카오맵 상세정보

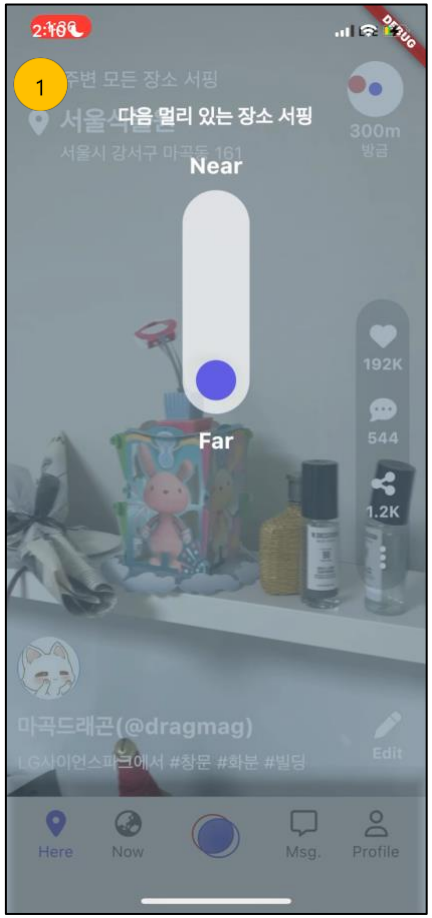


11-서핑포인트 업로드 화면

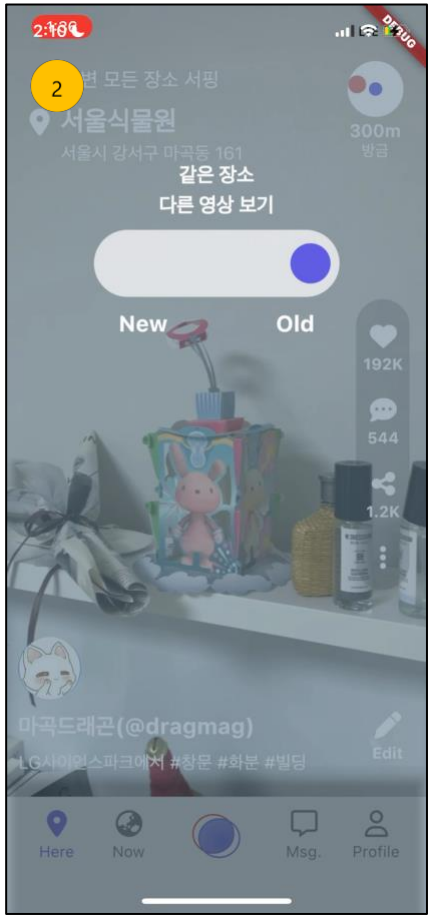
NO	설명
1	• 입력했던 정보에 맞는 장소명, 주소 표시. 클릭 시 카카오맵 상제정보 링크로 연결됨.
2	• 클릭 시, 주소 수정 가능.
3	• 유저의 프로필, 이름 표시. 콘텐츠 설명문구 표시.
4	• 클릭 시, 설명문구 수정 가능.
5	• 클릭 시, 재촬영을 통해 동영상 수정 가능.
6	• 클릭 시, 콘텐츠가 서버에 업로드 됨.
7	• 클릭 시, 튜토리얼 과정이 완료되고 홈 화면으로 이동.

앱	SURFI	화면 위치	Splash 화면
화면 기능	회원 가입 후, 홈 화면을 처음으로 접속한 유저들에 한해 스플래쉬 스크린으로 앱의 기본 기능 설명..		

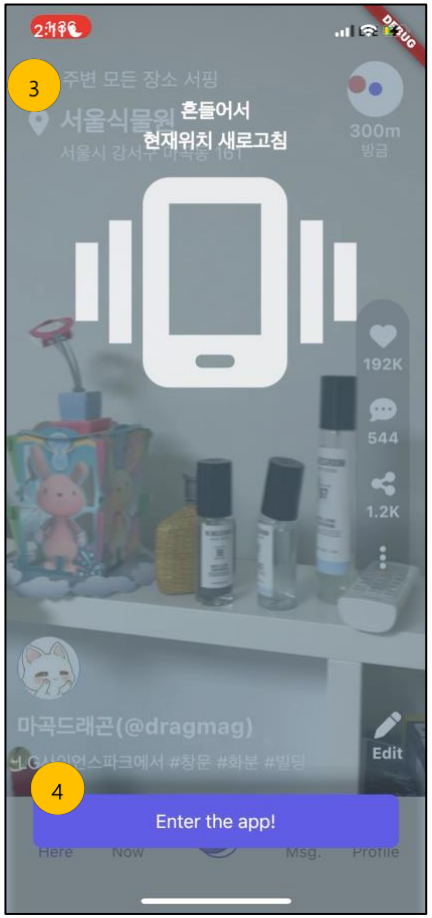
12-스플래쉬 화면(1)



12-스플래쉬 화면(2)



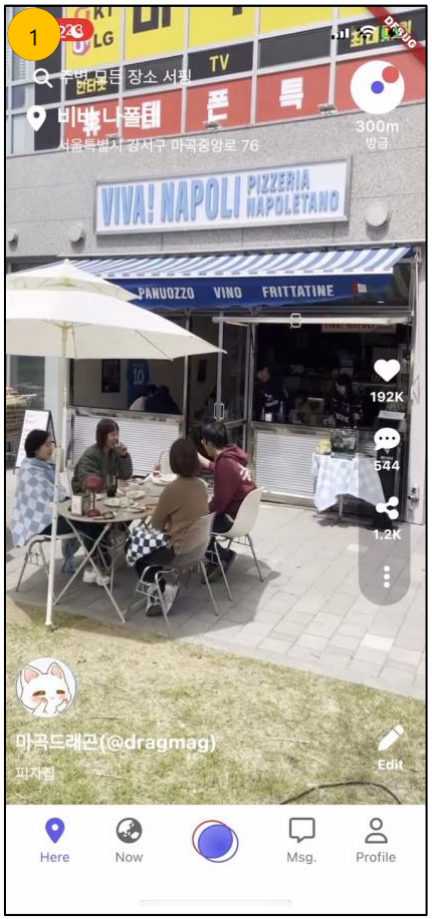
12-스플래쉬 화면(3)



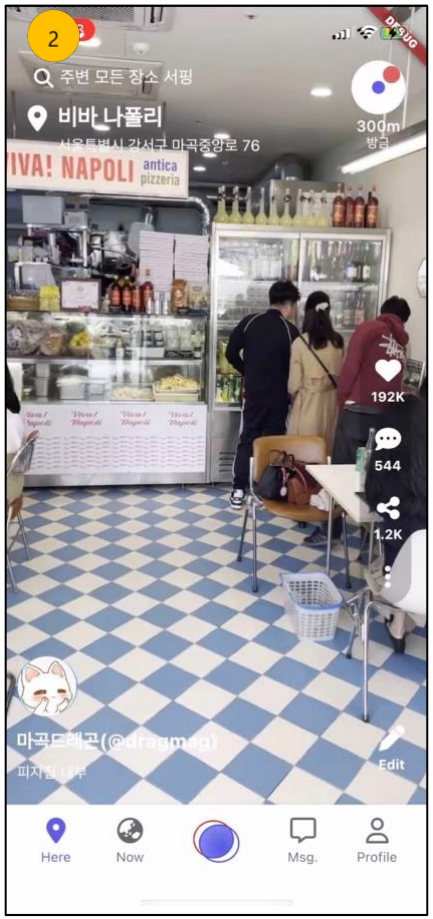
NO	설명
1	• 상하 스와이핑을 통해 거리순으로 장소를 서핑 할 수 있음을 알려주는 화면. 왼쪽으로 스와이핑 하면 스플래쉬 화면(2)으로 이동.
2	• 좌우 스와이핑을 통해 시간순으로 같은 장소를 서핑 할 수 있음을 알려주는 화면. 왼쪽으로 스와이핑 하면 스플래쉬 화면(3)으로 이동.
3	• 폰을 흔들어서 현재위치를 새로고침 할 수 있음을 알려주는 화면.
4	• 클릭 시, 스플래쉬 화면이 종료되고 홈 화면으로 진입..

앱	SURFI	화면 위치	홈 화면(Here Tab)
화면 기능	홈 화면에서 좌우 스와이핑을 통해, 같은 장소의 영상을 시간순으로 스와이핑 가능		

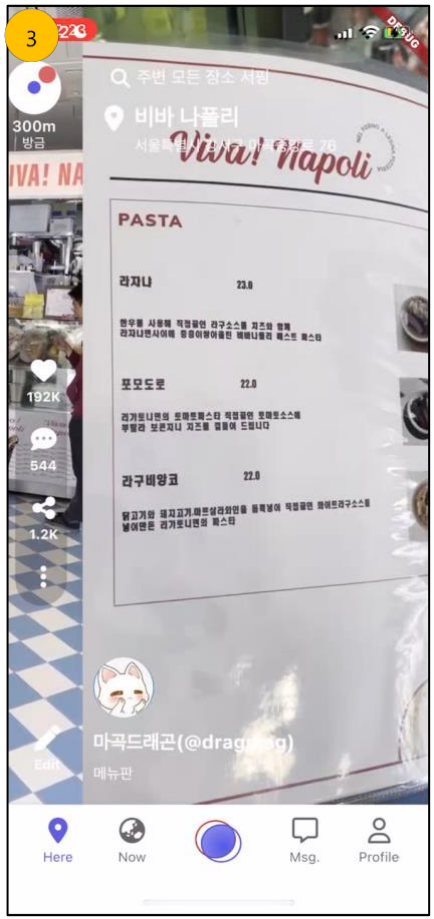
13-홈 화면



13-홈 화면(좌우스와이핑)



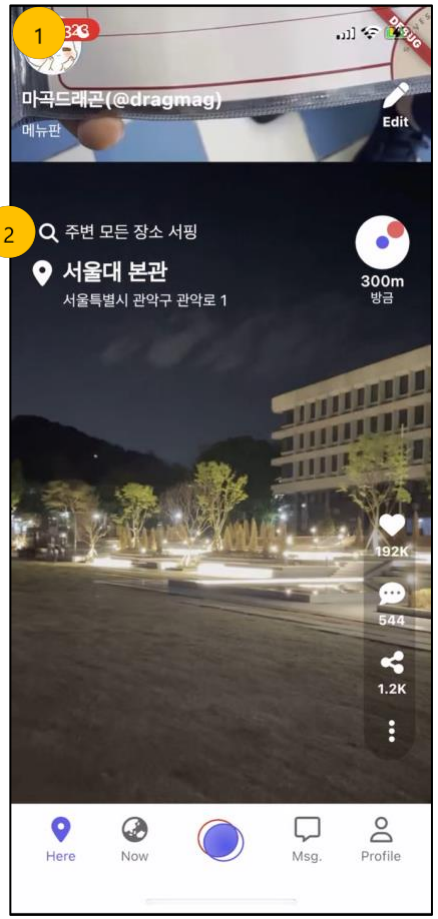
13-홈 화면(좌우스와이핑 중)



NO	설명
1	• 홈 화면에서 좌우 스와이핑을 통해, 같은 장소의 영상을 시간순으로 스와이핑 가능. 좌측으로 스와이핑 시 2번 화면으로 이동
2	• 좌측으로 스와이핑 시 3번 화면으로 이동
3	• 좌측으로 스와이핑 중인 화면

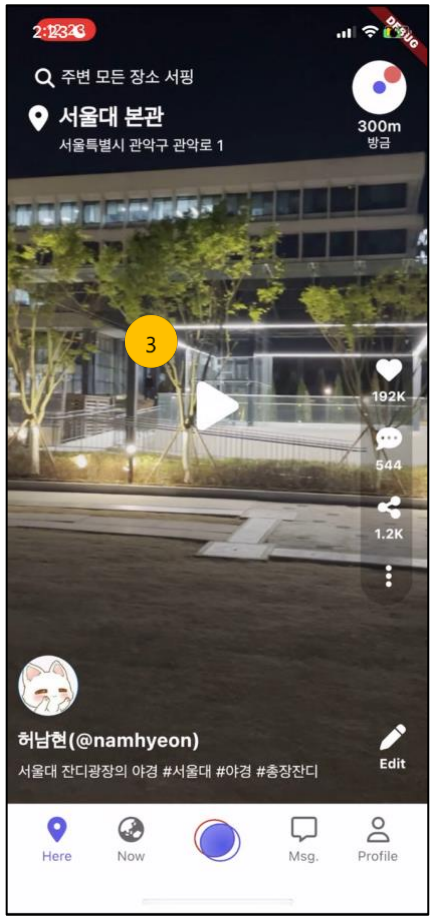
앱	SURFI	화면 위치	홈 화면(Here Tab)
화면 기능	홈 화면에서 상하 스와이핑을 통해, 가까운 장소의 영상을 거리순으로 스와이핑 가능		

13-홈 화면



14-카카오맵 주소

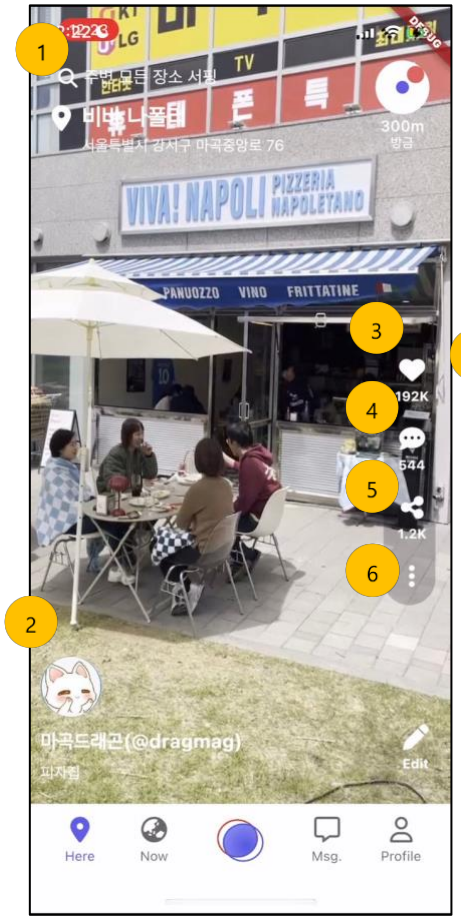
13-홈 화면(영상 정지)



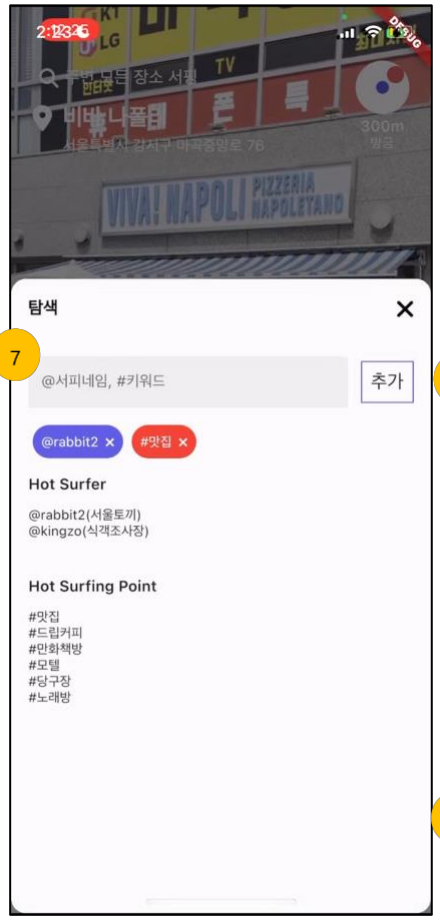
NO	설명
1	• 홈 화면에서 상하 스와이핑을 통해, 가까운 장소의 영상을 거리순으로 스와이핑 가능. 상하 스와이핑 도중의 모습.
2	• 좌상단의 주소를 클릭 시, 카카오맵 링크로 연결되어 상세위치를 알 수 있음.
3	• 영상 클릭 시, 화면이 정지되고 재생버튼이 나타남. 재클릭시, 영상 재생됨.

앱	SURFI	화면 위치	홈 화면(Here Tab)
화면 기능	홈 화면의 키워드 검색 기능, 댓글 기능		

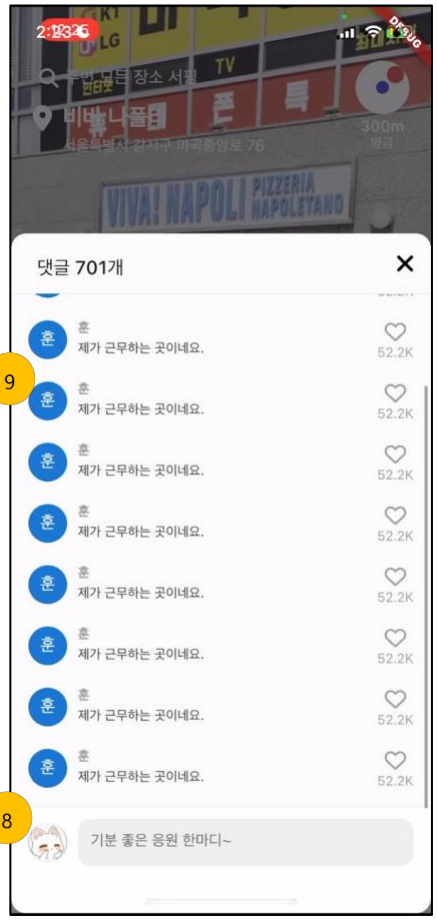
13-홈 화면



15-키워드 검색 서핑



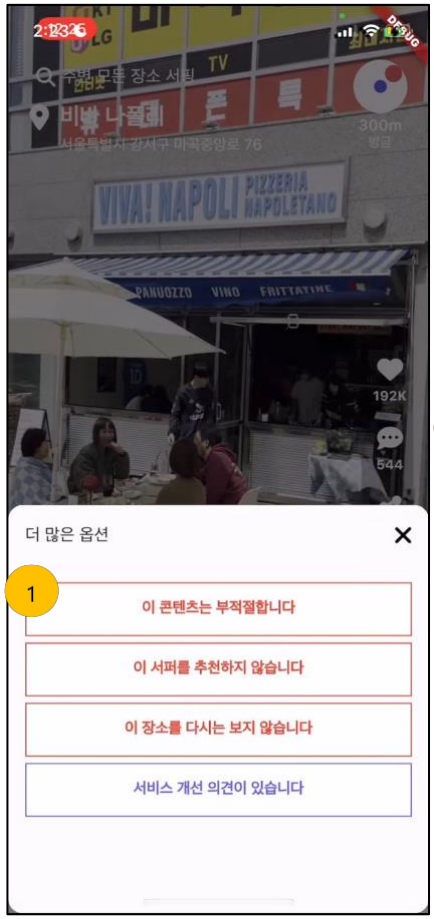
16-댓글 화면



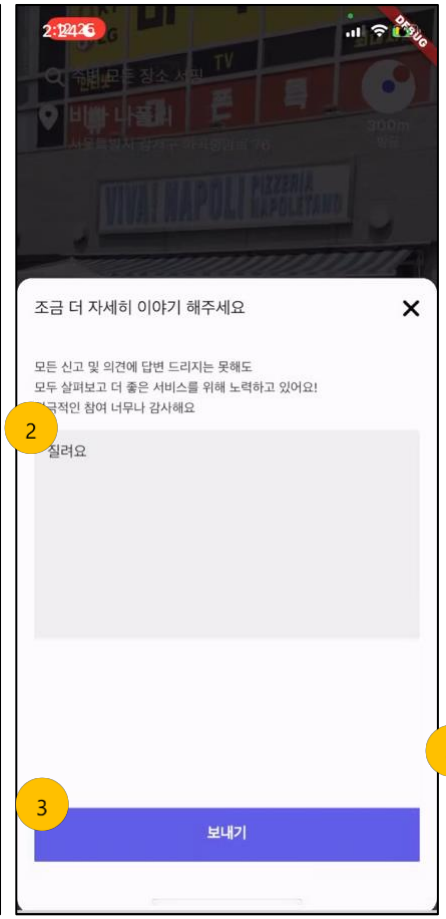
NO	설명
1	• **'주변 모든 장소 서핑' 클릭 시 15번 화면으로 이동**
2	• 영상을 업로드한 유저의 프로필 정보와 영상에 대한 설명문구 표시
3	• 클릭 시, 영상 좋아요
4	• 클릭 시, 16번 화면으로 이동
5	• 클릭 시, 다른 사람과 영상 공유 가능(현재 UI 미구현)
6	• 클릭 시, 17번 화면으로 이동(다음 페이지)
7	• 유저 이름, 혹은 키워드를 입력해서 키워드에 해당하는 영상들만 필터링해 서핑 가능
8	• 영상에 대한 코멘트를 달 수 있음
9	• 하트 클릭 시 댓글 좋아요가 가능하며, 영상의 주인인 경우 댓글을 좌우로 스와이핑 해 삭제 가능

앱	SURFI	화면 위치	홈 화면(Here Tab)
화면 기능	부적절한 콘텐츠의 경우, 신고버튼을 통해 신고 가능		

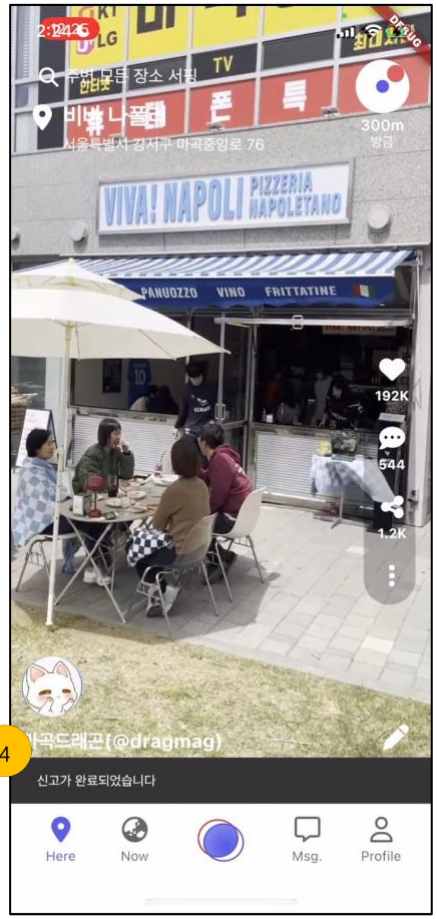
17-콘텐츠 신고 화면



17-콘텐츠 신고 화면



17-콘텐츠 신고 화면

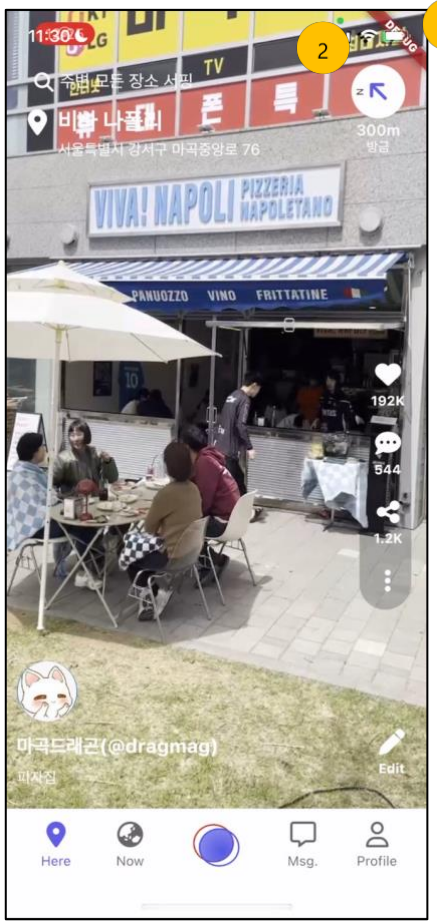


NO	설명
1	• 해당 영상에 대한 불편 사항을 가장 옵션을 통해 해결 가능. 특히, 부적절한 콘텐츠임을 클릭하면 신고 화면으로 이동
2	• 부적절한 콘텐츠에 대해 코멘트를 달 수 있고, 추후 관리측에서 해당 내용을 확인
3	• 클릭 시, 신고 완료
4	• 신고가 완료되었음을 푸시 알람을 통해 알려줌

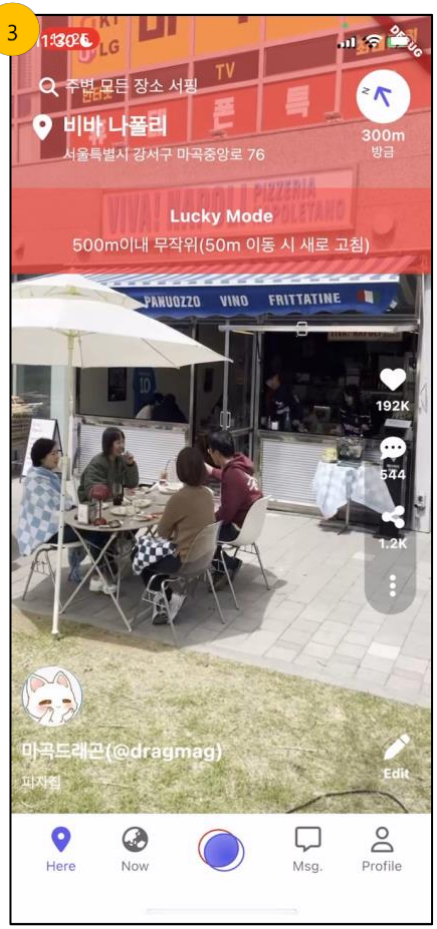
앱	SURFI	화면 위치	홈 화면(Here Tab)
화면 기능	나침반 & Lucky 모드로 전환하여 해당 모드의 기능을 사용할 수 있음		

13-홈 화면

13-홈 화면(나침반 모드)



13-홈 화면(럭키 모드)

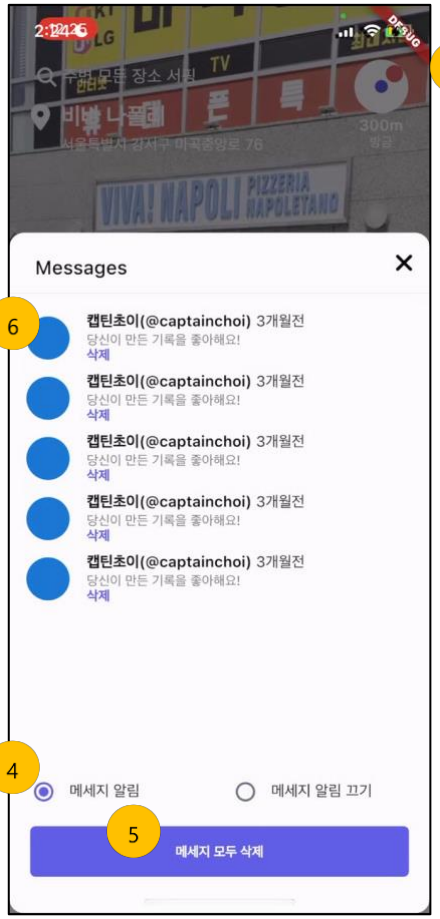


NO	설명
1	• 현재 보여지고 있는 장소의 거리와 위치를 표시. 클릭 시, 나침반 모드로 변경
2	• 핸드폰이 가리키고 있는 방향에 해당하는 영상들만 필터링해서 보여줌.
3	• 홈 화면에서 핸드폰을 흔들면 럭키모드로 진입. 럭기모드에서는 500미터 이내 장소의 영상들을 무작위로 보여줌.

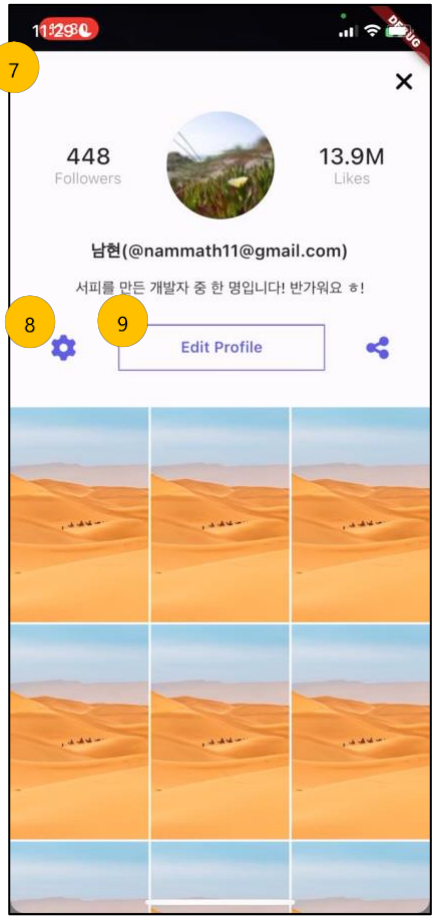
앱	SURFI	화면 위치	홈 화면(Here Tab, Msg Tab, Profile Tab)
화면 기능	화면 하단의 탭들을 통해 Now, Msg, Profile 화면으로 전환 가능		

13-홈 화면

18-Msg 화면



19-프로필 화면

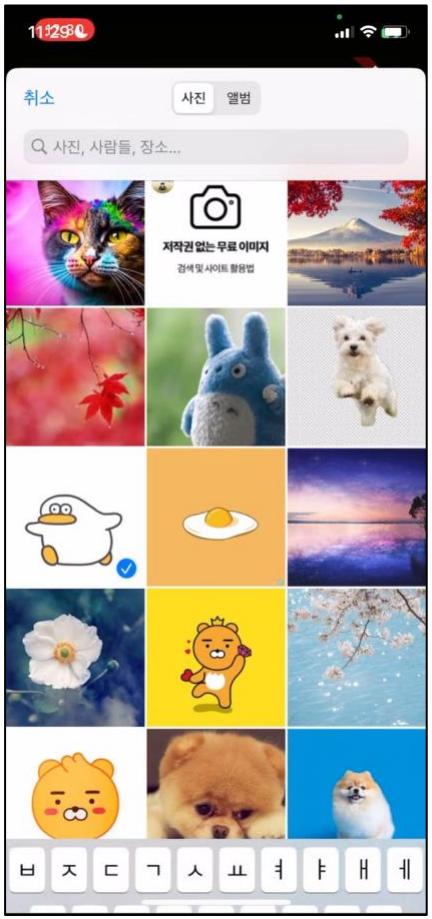


NO	설명
1	• 클릭 시, Now 탭으로 이동하며 UI는 Here tab과 유사하나 전세계에서 가장 최근에 올라온 영상들을 보여줄 예정.
2	• 클릭 시, 메시지 화면으로 이동하며, 유저에게 온 여러 메시지들(좋아요 알림 등)을 확인 가능
3	• 클릭 시, 프로필 화면으로 이동.
4	• 체크박스를 통해 메시지 알림 on/off 가능
5	• 클릭 시, 메시지 모두 삭제
6	• 삭제 버튼을 통해 메시지 개별 삭제 가능
7	• 프로필 화면으로 팔로워 수, 좋아요 수, 프로필 정보 등 표시. 특히, 하단에는 해당 유저가 업로드한 장소들의 썸네일을 표시할 예정
8	• 클릭 시, 환경설정 화면으로 이동
9	• 클릭 시, 프로필 편집 화면으로 이동

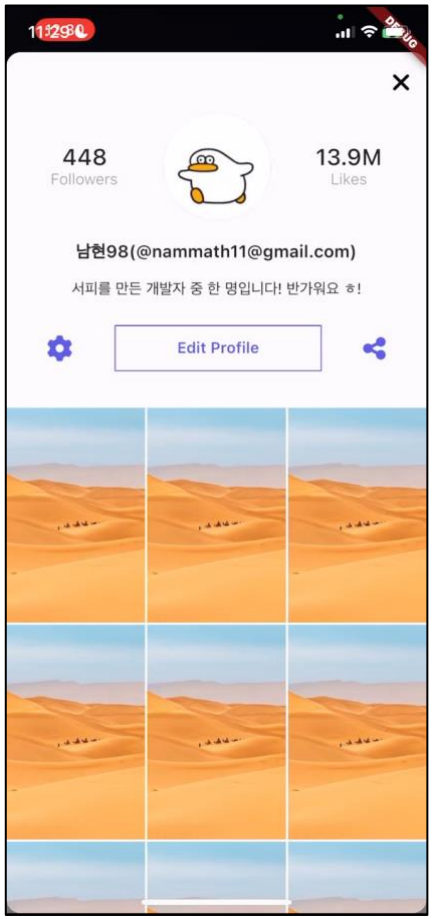
앱	SURFI	화면 위치	프로필 편집 화면
화면 기능	프로필 사진, 이름, 자기소개를 수정할 수 있음		

20-프로필 편집 화면

20-프로필 편집 화면



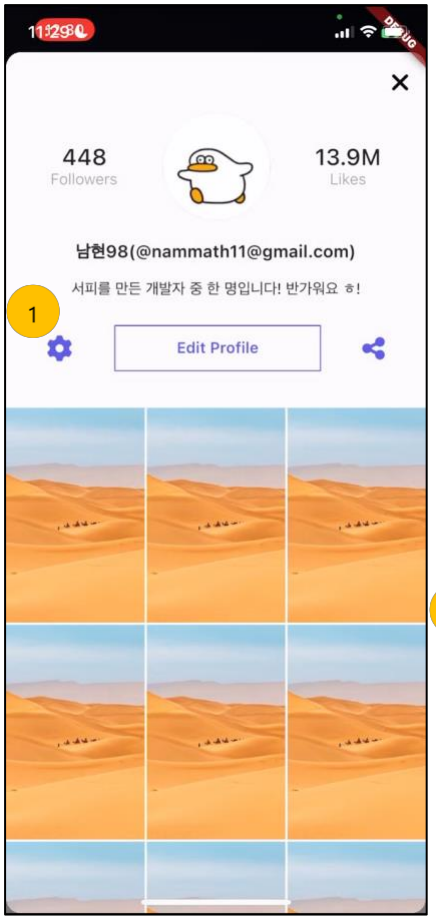
19-프로필 화면(편집 완료)



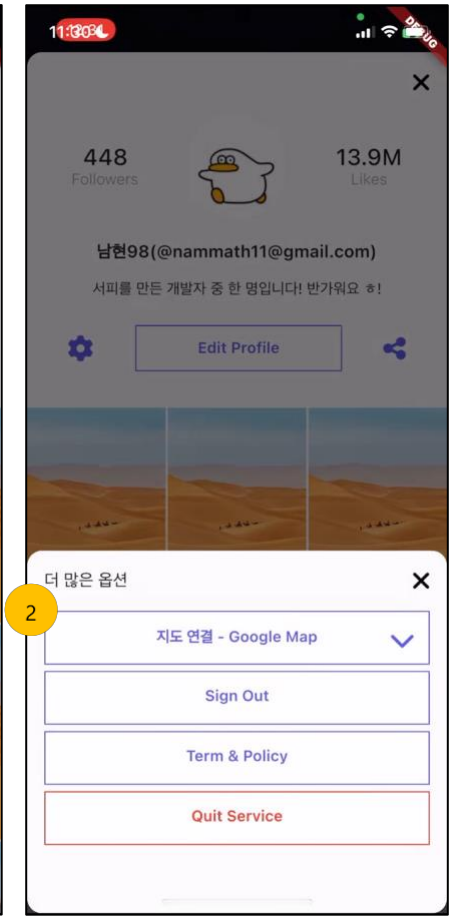
NO	설명
1	• 프로필 편집 화면. 프로필 사진, 이름, 자기소개를 수정할 수 있음
2	• 클릭 시, 사진 갤러리로 이동해서 프로필 사진 선택 가능
3	• 클릭 시, 프로필 편집이 완료되며 변경사항이 저장됨

앱	SURFI	화면 위치	환경설정 화면
화면 기능	환경 설정 화면에서 여러 옵션들을 설정하고, 로그아웃 할 수 있음.		

19-프로필 화면



21-환경설정 화면



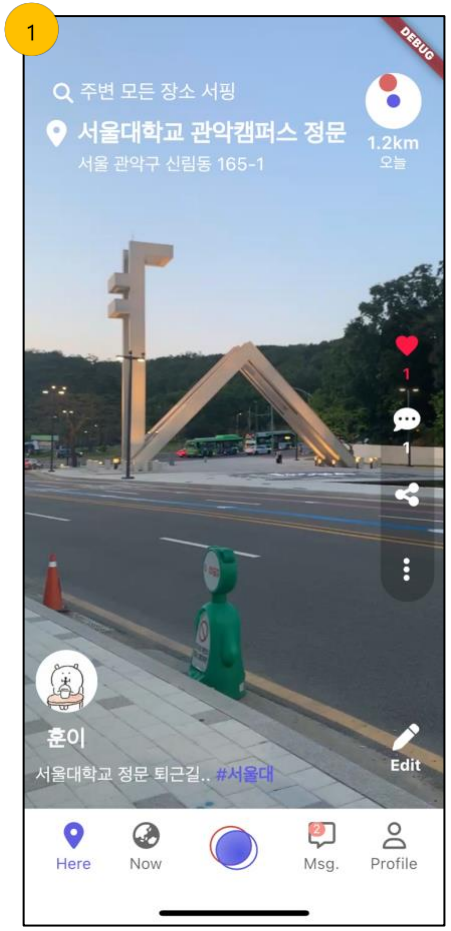
01-로그인 화면

NO	설명
1	• 클릭 시, 환경설정 화면으로 이동
2	• 환경설정 옵션들. • 지도 연결을 통해 Google, Kakao, Naver Map 중 선택 가능 • Sign Out 클릭 시, 로그아웃 되며 로그인 화면으로 이동 • Term & Policy 클릭 시, 관련 문서로 이동

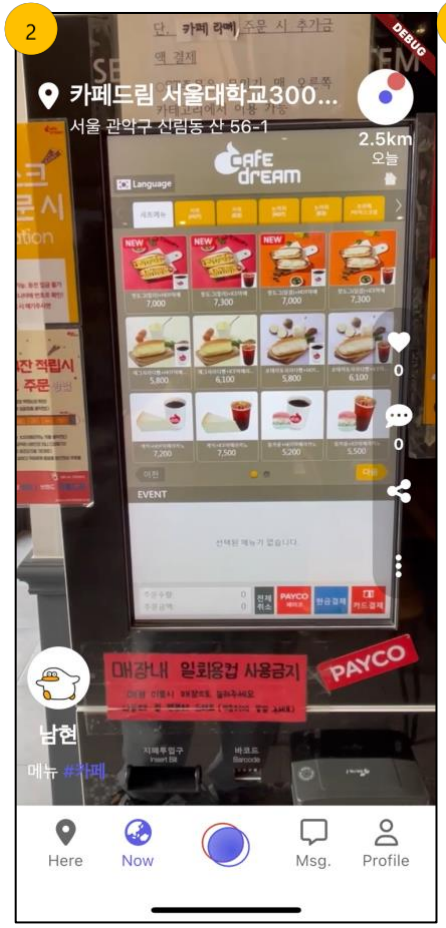
Result : 현재 페이지부터는 중간 발표 이후 완성된 기능들로, 실제 Firebase의 데이터베이스와 연결되어 작동하는 모습을 보여준다.

앱	SURFI	화면 위치	홈 화면(Here Tab, Now Tab, Lucky Mode)
화면 기능	Firebase database와 연결하고, 서울대 주변의 영상 수백여 개를 직접 촬명 및 업로드해서 Here Tab, Now Tab, Lucky Mode가 작동하도록 구현했다.		

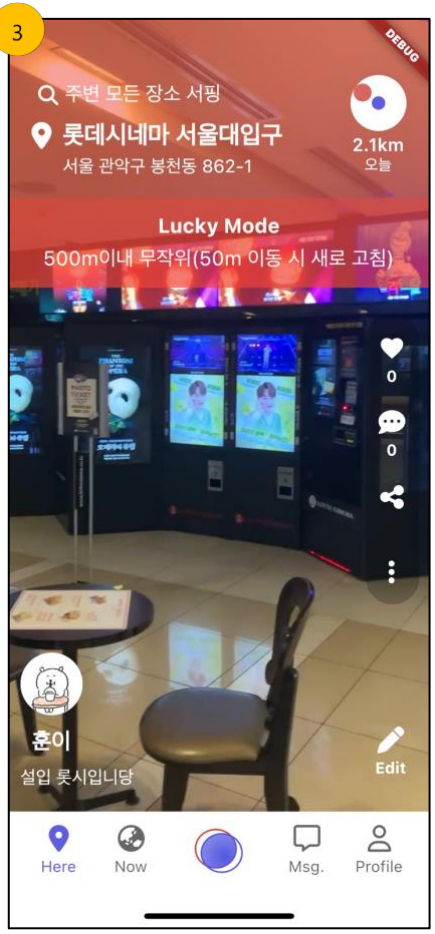
22-Here Tab



23-Now Tab



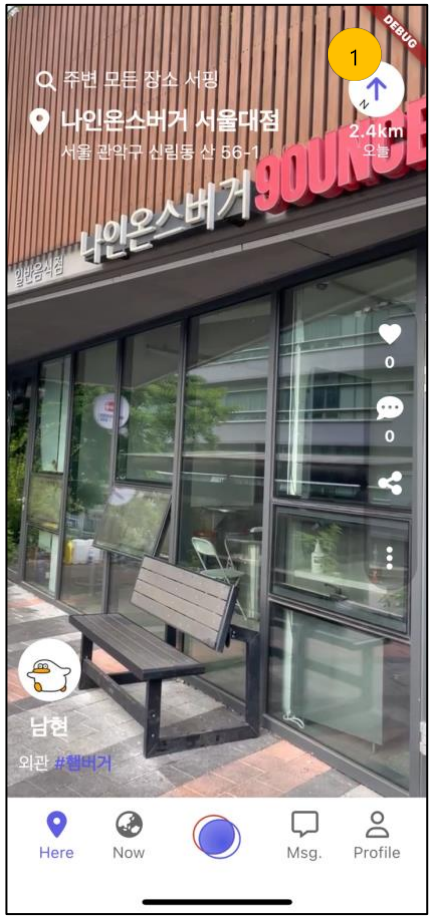
24-Lucky Mode



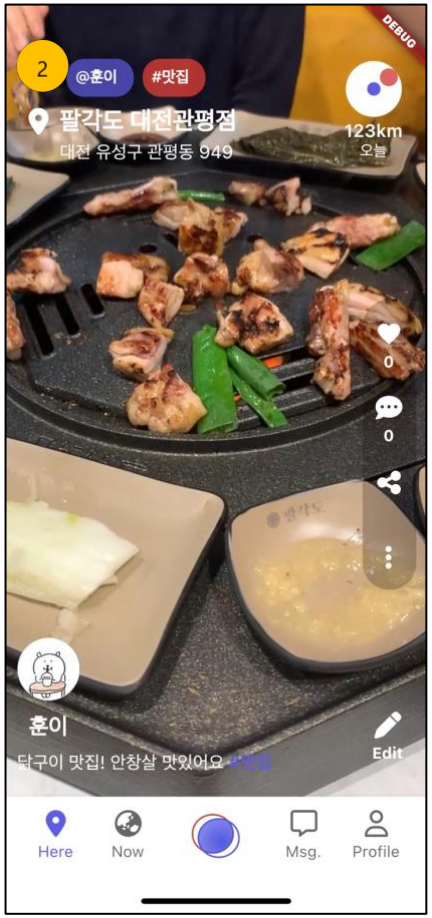
NO	설명
1	• -현재 위치 기준, 거리 순서대로 가까운 영상들을 보여줌 • 좌우 스크롤 : 같은 장소 다른 영상 • 상하 스크롤 : 다음으로 가까운 다른 장소 • GeoHash를 이용해 구현
2	• 최근에 업로드 된 영상들을 시간 순으로 보여줌 • 좋아요, 댓글, 공유, 신고, 삭제 기능 구현 완료
3	• 휴대폰 흔들 시 진입 • 반경 5km이내의 영상들을 랜덤으로 보여줌 • 3초마다 새로운 영상 보여줌

앱	SURFI	화면 위치	홈 화면(나침반 모드, 검색 기능, 오버뷰 모드)
화면 기능	Firebase database와 연결해 나침반모드, 검색 기능, 오버뷰 모드를 구현했다.		

25-나침반모드



26-검색 기능

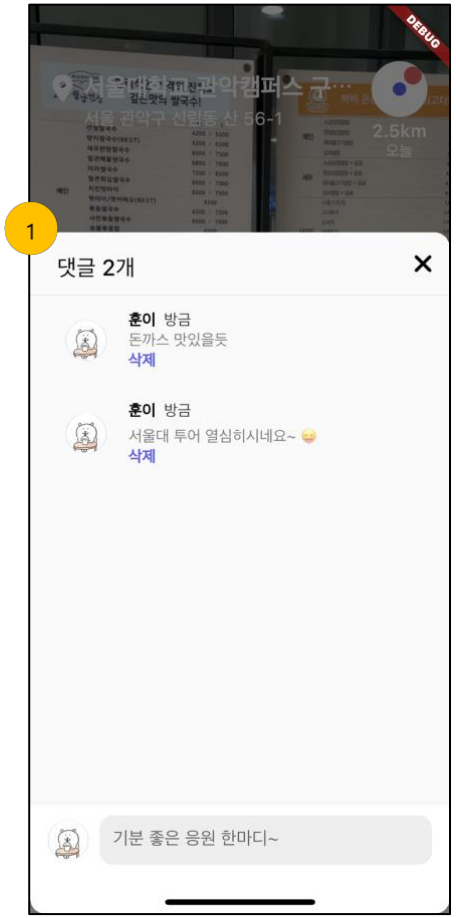


27-오버뷰 모드

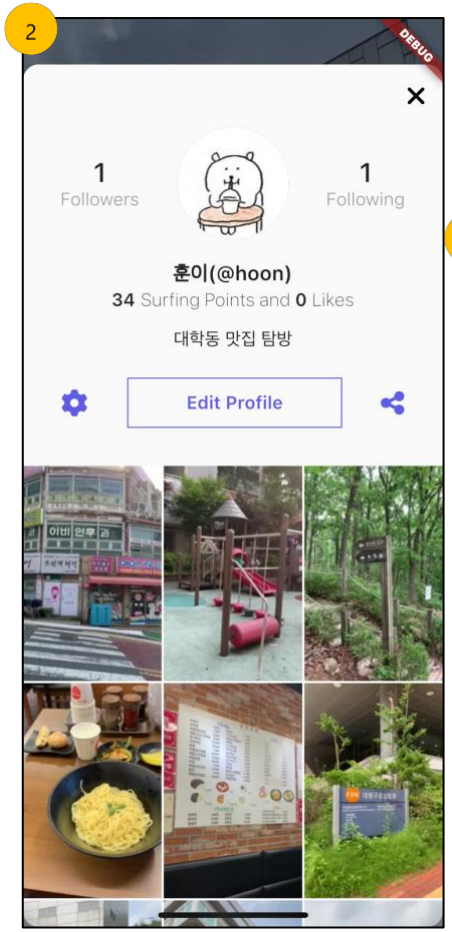
NO	설명
1	• 우측 상단의 나침반 버튼 클릭 시 진입 • 현재 위치와 영상 속 장소의 위도와 경도를 이용해 휴대폰이 가리키는 방향의 영상 필터링 • Geolocator와 FlutterCompass로 구현
2	• 검색을 통해 Hot Surfer, HashTag를 검색 • 보고싶은 유저와 해시태그에 해당하는 영상만 필터링 • 현재 '@훈이'의 영상 중 '#맛집' 해시태그가 붙은 영상만 보여줌
3	• Here Tab에서 줌아웃을 통해 진입 • 거리순으로, 휴대폰이 가리키는 방향에 있는 장소들의 영상을 보여줌 • 한 눈에 주변 장소를 살펴볼 수 있는 기능

앱	SURFI	화면 위치	댓글 기능, 프로필 화면, 알림 기능
화면 기능	Firebase database와 연결해 댓글 기능, 프로필 화면, 알림 기능을 구현했다.		

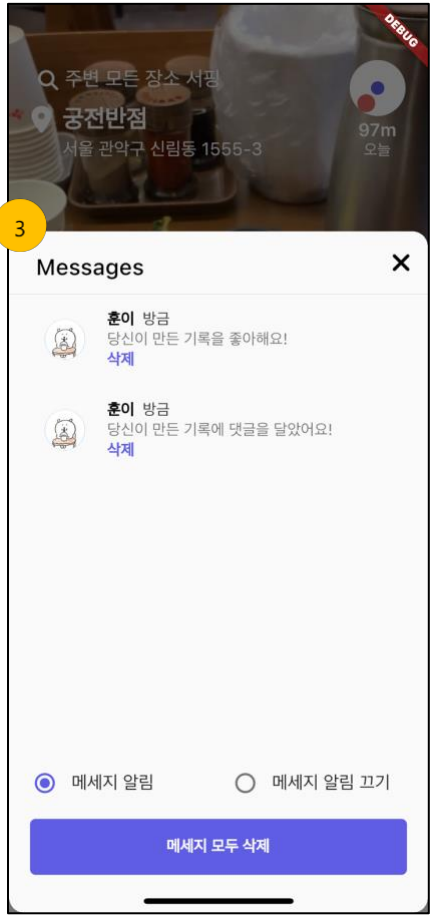
28-댓글 기능



29-프로필 화면



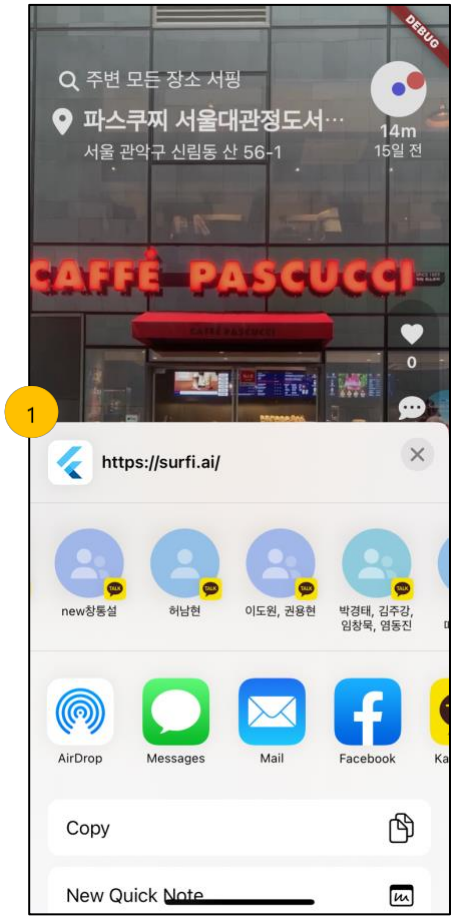
30-알림 기능



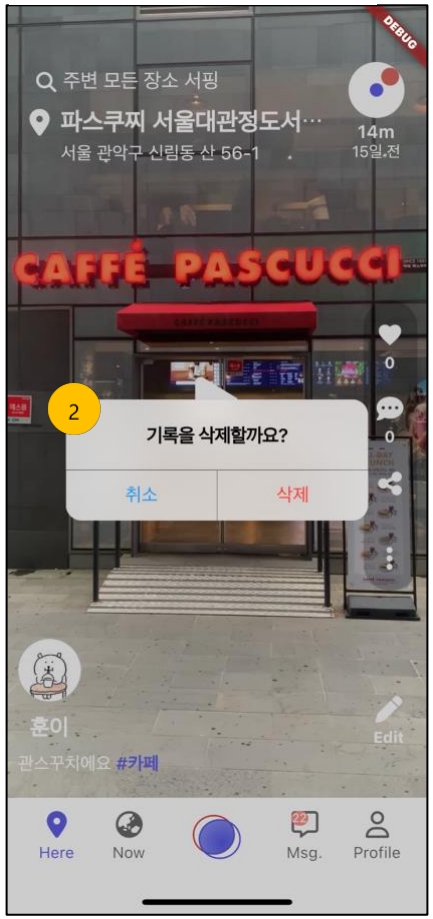
NO	설명
1	• 본인 혹은 다른 유저가 영상에 대한 댓글을 남길 수 있는 기능
2	• 프로필 화면으로, 현재까지 업로드한 영상의 개수, 받은 좋아요 수가 표시됨 • 자신 혹은 다른 유저가 업로드한 영상을 한 눈에 볼 수 있음 • 타 유저를 팔로우 할 수 있으며, 다른 유저의 프로필 또한 볼 수 있음
3	• 다른 유저가 자신의 기록을 좋아요 하거나, 댓글을 달 경우, 또는 자신을 팔로우할 경우 알람이 오는 기능 • 알람을 클릭할 경우 다른 유저가 좋아요한 영상이 팝업됨 • 삭제 버튼 혹은 '메시지 모두 삭제' 버튼을 클릭해 알람 삭제 가능

앱	SURFI	화면 위치	홈 화면(공유 기능, 기록 삭제 기능)
화면 기능	Share_plus를 이용해 공유 기능을 구현하고, 본인이 업로드한 영상의 경우 Database에서 관련 정보를 삭제할 수 있도록 구현했다.		

31-공유 기능



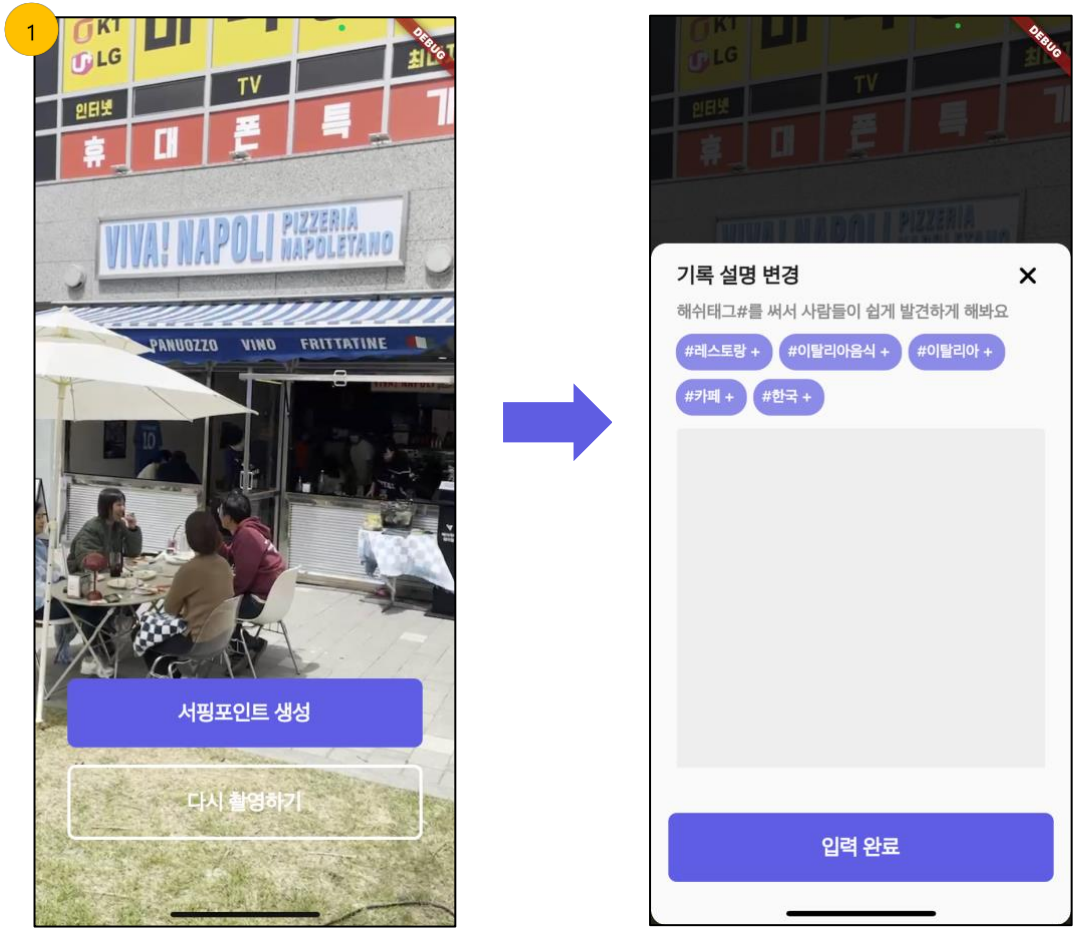
32-영상 삭제 기능



NO	설명
1	• 다른 유저가 영상을 볼 수 있도록 여러가지 SNS 서비스를 통해 공유할 수 있는 기능. • 현재는 앱 출시 전이기 때문에 SURFI의 랜딩페이지 링크를 공유하도록 구현 • Share_plus 플러그인 사용해 구현
2	• 본인이 올린 영상일 경우에만 우측 하단 'Edit' 버튼이 보임 • 'Edit' 버튼 클릭 시, 기록을 삭제할 지 묻는 팝업이 뜨며 '삭제' 클릭 시 Database에서 관련 정보들이 삭제됨

앱	SURFI	화면 위치	동영상 업로드 화면(Hash Tag Recommendation)
화면 기능	Clip Model을 활용해 '기록 설명 변경' 화면에서 동영상과 관련된 해시태그들을 추천해주는 기능을 구현했다.		

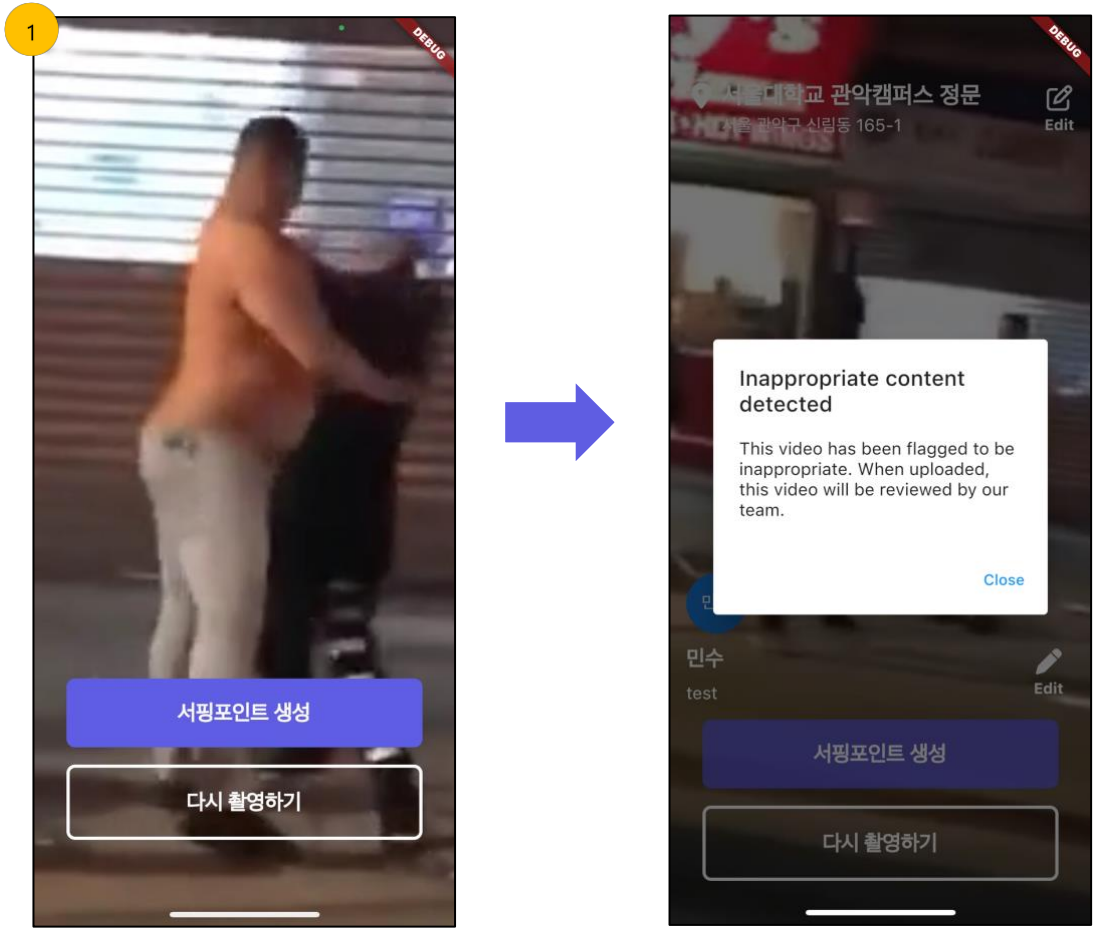
33-Hash Tag Recommendation



NO	설명
1	• '서핑 포인트 생성' 버튼을 클릭하고, 주소를 입력하면 그 시간 동안 서버에서 Clip Model이 관련 해시태그들의 정보를 추출하고 이를 반환한다 • 해당 화면의 경우 '#레스토랑', '#이탈리아음식' '#이탈리아', '#카페', '#한국' 등 영상과 어울리는 태그들을 적절히 추천했음을 알 수 있다.

앱	SURFI	화면 위치	동영상 업로드 화면(컨텐츠 필터링)
화면 기능	Clip Model을 활용해 '기록 설명 변경' 화면에서 동영상이 불건전할 경우 경고 메시지를 띄우는 기능을 구현했다..		

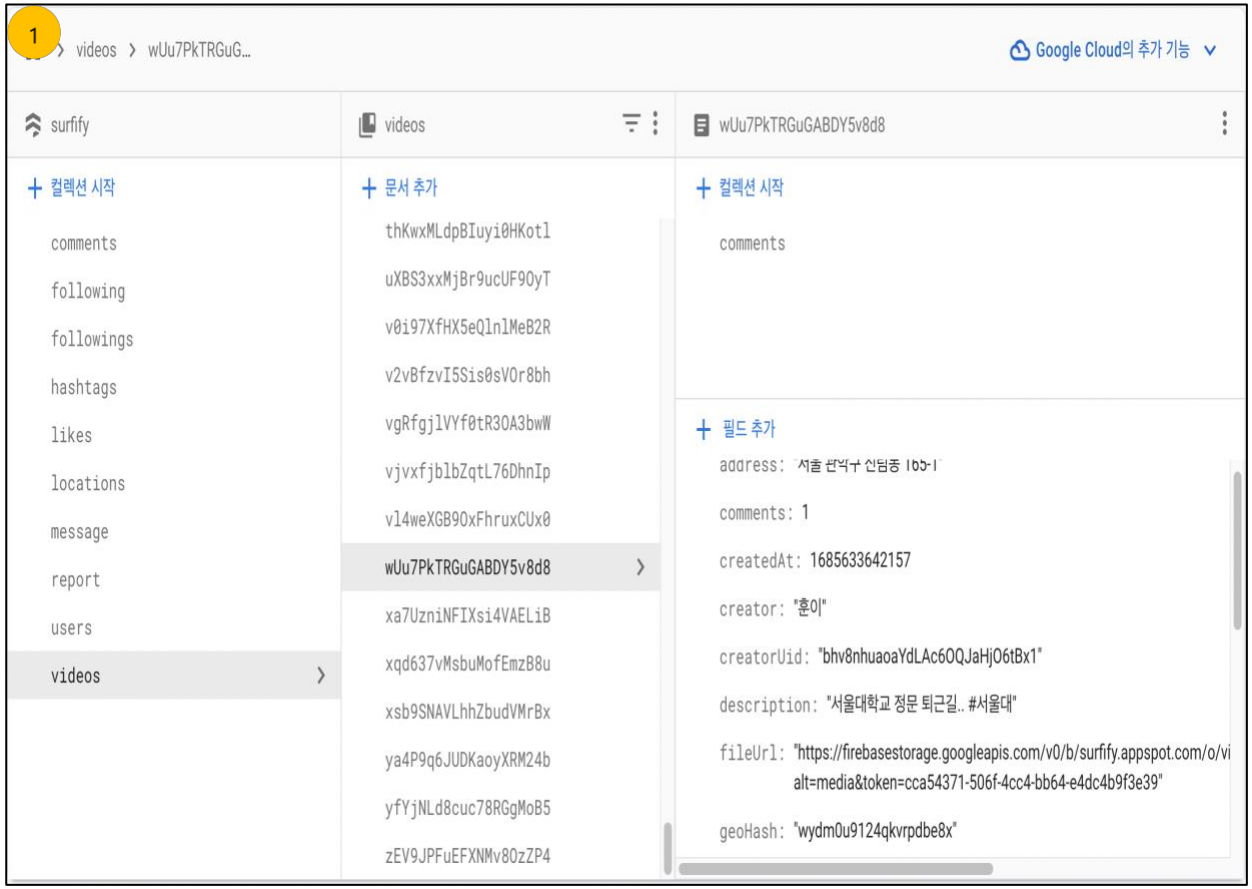
34-선정적, 폭력적 컨텐츠 필터링



NO	설명
1	• '서핑 포인트 생성' 버튼을 클릭하고, 주소를 입력하면 그 시간 동안 서버에서 Clip Model이 관련 텍스트를 추출하고, 추출 내용에 'Violence', 'This photo contains violence' 등이 있을 경우 필터링한다 • 해당 화면의 경우 폭력적인 컨텐츠이기 때문에 경고 메시지가 띄워지는 것을 볼 수 있다. 유저에게 해당 영상을 업로드할 경우 관리자에게 리뷰될 수 있음을 알려주는 경고이다.

앱	SURFI	화면 위치	Firebase Database
화면 기능	Firebase Database로, 앱에서 필요한 정보들을 가져오기 위한 데이터베이스이다.		

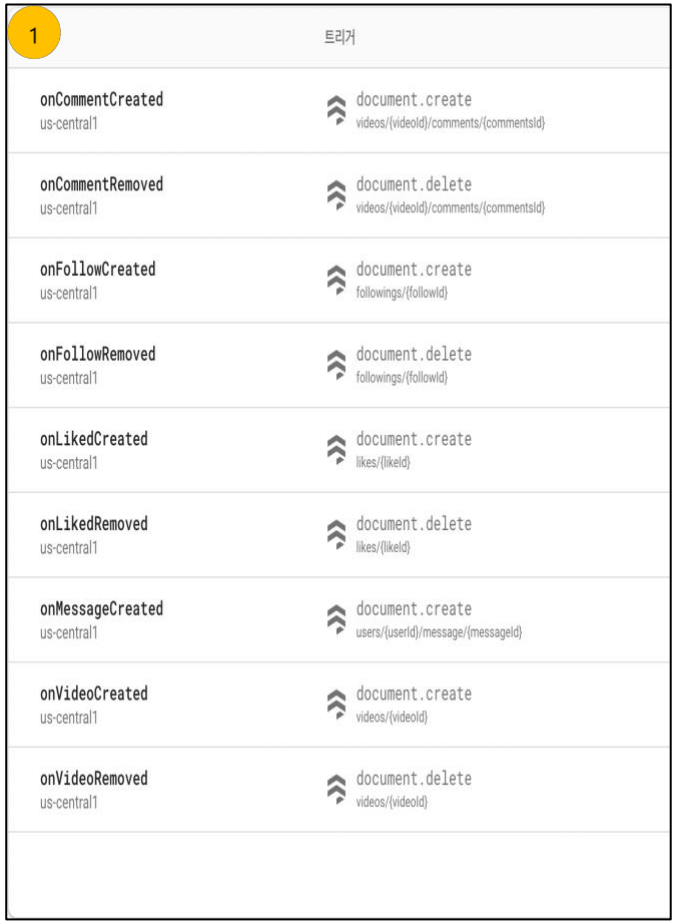
35-Firebase Database



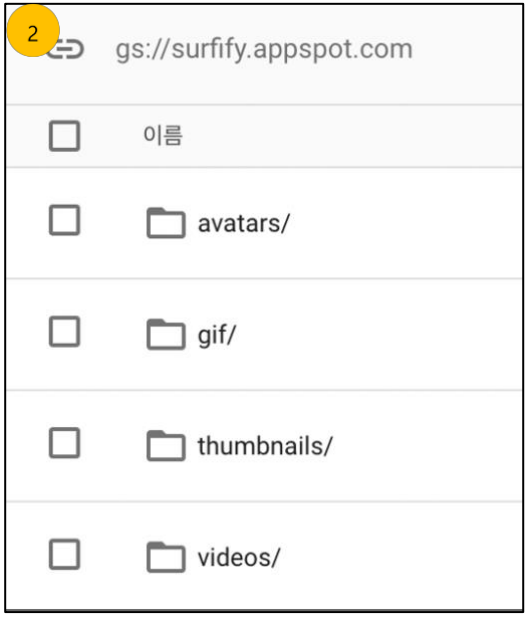
NO	설명
1	• Videos: 해당 컬렉션에는 업로드한 동영상들의 address, comments, createdAt, creator, creatorUid, description, fileUrl, hashtag, id, kakaomapId, latitude, likes, location, longitude, thumbnailUrl, title의 정보가 담겨있다. • Users: 해당 컬렉션에는 각 유저별로 follwers, followings, message, videos 정보들이 담겨있다. • Report: 부적절한 컨텐츠를 신고할 경우 해당 컬렉션에 신고 메시지가 저장된다 • Location: 해당 컬렉션에는 GeoHash 정보가 담겨있고, 각 지오해시 안에 해당하는 videos 컬렉션이 저장된다. HereTab 기능을 구현하기 위한 컬렉션 • Hashtags: 해당 컬렉션에는 유저들이 그동안 업로드한 해시태그들이 담겨있고, 해시태그 안에 해당하는 videos 컬렉션이 저장되어 있다. 검색 기능을 구현하기 위한 컬렉션

앱	SURirel	화면 위치	Firebase Functions & Storage
화면 기능	Firebase Functions는 앱에서 댓글, 좋아요, 팔로우, 메시지, 영상 업로드 등의 기능을 사용할 때 Database를 업데이트한다. Firebase Storage는 프로필, 동영상, 영상 썸네일을 저장한다.		

36-Firebase Functions



37-Firebase Storage



NO	설명
1	• 각 함수들은 앱에서 댓글이 생성 및 삭제되었을 때, 다른 유저를 팔로우 및 팔로우 취소 했을 때, 영상을 좋아요 및 좋아요 취소 했을 때, 영상을 업로드 및 삭제 했을 때 호출되며 Firebase Database에서 관련된 정보들을 수정한다.
2	• Firebase Storage에는 각 유저들의 프로필, 동영상의 썸네일, 동영상 파일들을 저장한다.

A. Experiments

폭력적 컨텐츠 필터링 성능을 Real Life Violence Situations Dataset 에서 벤치마킹을 진행했습니다. 필터링 label set 은 Instagram 의 인기 hashtag 150 개, filtering label 4 개로, 임의로 넣은 label 6 개로 총 160 개의 label 을 사용했습니다. 동영상에서 추출된 6 개의 프레임에 대해 "violence", "This photo contains violence", 그리고 야한 동영상 제목 2 개가 추출된 경우 필터링 된 것으로 정의했습니다. 총 1000 개의 폭력적 동영상 중 856 개가 flag 되어 85.6%의 성공률을 확인할 수 있었고, 폭력적이지 않은 동영상 951 개 중 53 개가 flag 되어 5.6% 정도의 폭력적이지 않은 동영상이 필터링 될 것으로 기대할 수 있습니다.
선정적인 데이터는 윤리적으로 대량을 수집할 수 있는 방법 혹은 데이터셋이 존재하지 않아 벤치마크를 진행하지 못했습니다. 10 개의 소량의 사진으로 성능을 확인해본 결과 야한 동영상의 경우 모두 필터링 되는 것을 확인할 수 있었습니다.

B. Result Analysis and Discussion

Real Life Violence Situations Dataset 에서 필터링 되지 않은 14.4%는 대부분 스포츠 경기에서 발생하는 폭력적인 동영상이 포함되어 있었습니다. 미식축구, 럭비, 아이스하키 경기에서 발생하는 접촉, 싸움은 폭력적인 동영상으로 분류할 수 있지만 사회적 통념에 어긋나 필터링 되어야하는 콘텐츠인지에 대한 논의는 필요한 것으로 보입니다.
더불어 선정적인 동영상의 경우 폭력적인 동영상과는 달리 "This photo contains nudity,", "This photo contains pornography"와 같은 레이블로 효과적으로 필터링 되지 않은 것을 확인할 수 있었습니다. CLIP 은 인터넷에 존재하는 이미지와 이미지에 붙은 caption 으로 학습된 모델로, image embedding 과 cosine similarity 가 높은 text embedding 을 구하기 위해서는 실제 인터넷에서 이미지와 함께 발견될 text 를 label 로 사용하는 것이 효과적입니다. 따라서 야한 동영상을 필터링 하는 경우 실제 야한 동영상의 제목을 사용하는 것이 필터링 성능을 향상 시켰고, 선정적이지 않은 동영상에도 더욱 예측되지 않는 것을 볼 수 있었습니다.
또한, filtering label set 과 keyword label set 을 나뉘어 예측하는 것이 효과적인 것을 확인할 수 있었습니다. 한번에 filtering label set + keyword label set 의 text embedding 과 image embedding 의 cosine similarity 를 구하면 accuracy 가

전반적으로 낮아지는 것을 확인할 수 있었습니다. 한번에 예측하는 label set 의 크기가 커질 수록 비교하는 embedding 의 수가 증가하기 때문에 spurious 하게 다른 text 와 높은 유사도 나온다고 해석할 수 있습니다. CLIP 은 zero-shot classification 에서도 좋은 성능을 보여줄 만큼 robust 한 embedding 을 구하지만, filtering 과 키워드 예측 같이 다른 task 에 대해서 CLIP 을 사용할 경우 independent 한 label set 을 사용하는 것이 성능과 flexibility 측면에서 이점이 있다고 볼 수 있습니다.

J. Division & Assignment of Work

항목	담당자
프론트엔드, 앱개발	사공훈, 허남현
백엔드	사공훈, 허남현, 윤민수
Ai feature	윤민수

K. Schedule

내용	3 월			4 월				5 월				6 월	
	2 주	3 주	4 주	1 주	2 주	3 주	4 주	1 주	2 주	3 주	4 주	1 주	2 주
로그인/회원가입	허남현	허남현											
메인탭/홈	허남현	사공훈											
메시지/프로필		허남현	허남현										
동영상 촬영			사공훈	사공훈									
위치 태그/ 수정					사공훈	사공훈							
럭키모드/나침반					허남현	허남현							
파이어베이스 회원/동영상업로드							허남현						
댓글, 좋아요, 유저 프로필 백엔드								허남현					
거리순으로 정렬									허남현				
다이나믹 링크									사공훈				
태그와 프로필로 동영상 검색										사공훈			
나침반 모드/럭키모드										허남현			
팔로우 / 유저 알림/신고											허남현/사공훈		

코드 리팩토링/테스팅												허남현, 사공훈	
Clip 논문	윤민수												
FrameSampling + Clip Inference		윤민수											
Benchmarking			윤민수										
Label Engineering				윤민수	윤민수								
Frame Sampling test						윤민수	윤민수						
Label Engineering								윤민수					
Torchserve								윤민수					
Backend									윤민수	윤민수			

L. Conclusion

이상의 과정을 거쳐서 저희는 프로젝트를 성공적으로 완수할 수 있었습니다. 플러터를 통해서 크로스앱 플랫폼 앱개발을 마쳤고, 태깅과 필터링 모델 또한 원활하게 작동하는 것을 확인할 수 있었습니다. 회사에서도 결과물을 확인하고 만족하셨고, 기획안의 모든 내용을 개발해주어서 기대 이상이라고 평가해주셨습니다. 회사의 멋진 아이디어 기획안을 가지고 세상에 없던 앱을 만들면서 많은 것을 배울 수 있었습니다.

◆ [Appendix] Detailed Implementation Spec

A. Module Name

a. Repo

Authentication_repo 의 input/output interface

```
Future<void> signOut()
```

로그아웃합니다

```
Future<UserCredential> googleSignIn()
```

구글회원가입합니다.

User repo 의 input/output interface

```
Future<void> createProfile(UserProfileModel profile)
```

프로필을 만듭니다

```
Future<Map<String, dynamic>?> findProfile(String uid)
```

프로필을 찾습니다

```
Future<void> uploadAvatar(File file, String fileName)
```

프로필사진을 업로드합니다

```
Future<void> updateUser(String uid, Map<String, dynamic> data)
```

유저의 데이터를 받아서 유저 정보를 수정합니다

```
Future<QuerySnapshot<Map<String, dynamic>>> fetchThumbnail(String uid)
```

유저가 촬영한 동영상의 썸네일을 받아옵니다.

Video repo 의 input/output interface

```
UploadTask uploadVideoFile(File video, String uid)
```

동영상 파일과 유저의 uid 를 인자로 받아서 동영상을 storage 에 업로드합니다.

```
Future<void> saveVideo(VideoModel data)
```

Videomodel 을 인자로 받아서 db 에 동영상 관련 정보를 저장합니다.

```
Future<QuerySnapshot<Map<String, dynamic>>> fetchVideos({
    int? lastItemCreatedAt,
  })
```

Timeline 에서 비디오를 fetch 합니다. 한번에 모든 비디오를 가져오면 부하가 심하기 때문에 유저가 마지막 비디오를 보고 있을 때 추가적으로 2 개씩 영상을 받아옵니다.

```
Future<void> likeVideo(String videoId, String userId)
```

유저 아이디와 비디오 아이디를 받아서 동영상 좋아요를 합니다

Comments repo 의 input/output interface

```
Future<void> addComment(CommentModel commentModel)
```

댓글을 답니다.

```
Future<void> deleteComment(CommentModel commentModel)
```

댓글을 삭제합니다

```
Future<QuerySnapshot<Map<String, dynamic>>> fetchComments({
    String? videoId,
  })
```

댓글을 불러옵니다

b. ViewModel

socialAuthViewModel

```
Future<void> googleSignUp(BuildContext context)
```

구글 회원가입 합니다.

```
Future<void> googleSignIn(BuildContext context)
```

구글 로그인을 합니다.

AvatarViewModel

```
Future<void> uploadAvatar(File file)
```

프로필 사진을 업로드 합니다

ProfileViewModel

```
Future<List<ThumbnailModel>> _fetchThumbnail(String arg)
```

썸네일을 fetch 합니다

UsersViewModel

```
Future<void> createAccount(UserCredential credential)
```

Fireauth 에서 받은 UserCredential 을 통해서 계정을 만듭니다

```
Future<void> onAvatarUpload()
```

유저가 프로필 사진이 업로드 되었음을 업데이트합니다.

```
Future<void> registerProfile({
    required String? profileAddress,
    required String? name,
    required String? intro,
  })
```

프로필 등록시 프로필과 이름, 자기소개를 등록합니다.

```
Future<void> updateProfile({
    required String? name,
    required String? intro,
  })
```

프로필 이름과 자기소개를 수정합니다.

```
Future<void> updateAgreement({
    required bool? serviceAgree,
```

```
    required bool? privacyAgree,
    required bool? marketingAgree,
  })
```

각종 동의사항을 업데이트 합니다.

CommentsViewModel

```
Future<List<CommentModel>> _fetchComments({
    String? videoId,
  })
```

비디오 아이디를 인자로 받아서 해당 비디오의 댓글을 fetch 합니다

```
Future<void> uploadComment({
    String? videoId,
    String? comment,
  })
```

유저정보와 현재 시간 정보를 읽어서 비디오 레포에서 addComment 함수를 호출해서 해당 비디오 댓글을 업로드 합니다.

```
Future<void> refresh(String arg) async
```

댓글이 달리거나 삭제될때 다시 댓글들을 fetch 합니다.

```
Future<void> deleteComment(CommentModel comment)
```

해당 댓글을 삭제합니다

TimelineViewModel

```
Future<List<VideoModel>> _fetchVideos({
    int? lastItemCreatedAt,
  })
```

본 영상의 촬영시점을 인자로 받아서 그 다음에 오는 동영상을 Fetch 합니다.

```
Future<void> fetchNextPage()
```

다음 페이지를 fetch 하도록 합니다

```
Future<void> refresh()
```

영상이 업로드되거나 삭제될시 새로 fetch 합니다

UploadVideoModel

```
Future<void> uploadVideo(
    File video,
    String location,
    String address,
    double longitude,
    double latitude,
    String description,
    String url,
    BuildContext context,
  )
```

비디오와 장소, 주소, 위도, 경도, 설명, 카카오지도 url 을 인자로 받아서 videoRepo 의 saveVideo 를 호출합니다.

VideoPostViewModel

```
Future<void> toggleLikeVideo() async
```

유저정보를 읽어서 유저 레포에서 likeVideo 를 호출합니다.

Tagging

```
def sample_frame(video_filepath):
```

TemporaryDirectory 에 저장된 비디오의 경로를 받아 min diff frame 을 구하고 numpy.ndarray 로 반환합니다.

```
def extract_keyword(image, labels):
```

CLIP Model, CLIP Processor 를 loading 하고 이미지에 대해 추론하고 결과를 List[Tuple(Label, Probability)]로 반환합니다.

```
@app.route('/predict_video', methods=['POST'])
def predict_video():
```

HTTP Post request 를 받아 동영상을 TemporaryDirectory 에 MP4 로 저장합니다. Extract keyword 를 호출해서 결과를 JSON 으로 반환합니다.