

상세 설계서

머신러닝을 활용한 객체인식 레시피 추천 시스템 (냉장고를 부탁해)

Ver. 1.0

2019.5.14

한국외국어대학교 융복합소프트웨어학과

1팀(NUDGE)



상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

문서 정보

구 분	소 속	성 명	날 짜	서 명
작성자	한국외국어대학교	이민기	2019. 05.13	
	한국외국어대학교	윤형로	2019. 05.13	
	한국외국어대학교	김영주	2019. 05.13	
	한국외국어대학교	황선기	2019. 05.13	
	한국외국어대학교	표승수	2019. 05.13	
	한국외국어대학교	정기욱	2019. 05.13	
검토자	한국외국어대학교	윤형로	2019. 05.13	
승인자	한국외국어대학교	홍진표		

개정 이력

버전	작성자	개정일자	개정 내역	승인자
1.0	이민기	2019.05.13	상세 설계서 초안 작성	
	윤형로			
	김영주			
	황선기			
	표승수			
	정기욱			

목차

1. 개요	5
1.1 서비스 정의	5
1.2 문서의 목적	5
1.3 범위	5
1.4 관련 문서	6
2. 시스템 소개 및 구성	7
2.1 개발 동향	7
2.2 개발 동기 및 필요성	7
2.3 시스템 구성요소	8
2.3.1 시스템 구성도	8
2.3.2 시스템 소개	8
3. 머신러닝 모델, Application 및 Django 서버	9
3.1 머신 러닝	9
3.1.1 Create ML Frame Work를 사용한 데이터 모델 구축	9
3.2 Application	11
3.2.1 UI 구성	11
3.2.2 카메라 세션	11
3.2.3 데이터 인풋과 아웃풋	12
3.2.4 델리게이트 패턴 (Delegate Pattern)	13
3.2.5 멀티 스레드	15
3.2.6 실시간 객체 인식의 전체 흐름	17
3.2.7 사용자가 선택한 재료를 카운트	17
3.2.8 사용자가 선택한 식재료를 HTTP Request	19
3.2.9 POST를 통해 받은 Data 화면에 출력	20
3.3 사용자 레시피 요청 처리	20
3.3.1 요청 객체	20
3.3.2 응답 객체	20

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

3.3.3 요구 사항	20
3.4 사용자 재료 기반 레시피 선택	21
3.4.1 사용자의 재료 입력 및 DB Select	21
3.4.2 사용자 재료와 DB 재료 비교 코드	22
3.4.3 비교 결과를 바탕으로 추천 레시피 출력	22
3.4.4 추천 레시피 중 사용자가 선택한 레시피 출력	23
3.4.5 결과 예시	23
3.5 서버 레시피 응답 처리	24
4. Server	24
4.1 AWS 환경 설정	24
4.1.1 인스턴스 설정	24
4.1.2 보안 그룹 설정	25
4.2 서버 배포 과정	25
5. 데이터베이스	27
5.1 DB 설계	27
5.1.1 설계도	27
5.1.2 데이터베이스 정규화	27
5.1.3 사용자 정보 관리	27
5.1.4 레시피 관리	28
5.1.5 메뉴 정보 관리	28
5.2 Django	28
5.2.1 Django와 mysql 연결	28
5.2.2 JSON parsing algorithm	29
5.2.3 정제 과정	30
5.2.4 데이터베이스 Insert 과정	30
5.2.5 데이터 삽입 결과	30
6. 가상 시나리오	32
6.1 시나리오	32
6.2 Sequence 다이어그램	33
7. 프로젝트 일정	34

1. 개요

본 장에서는 머신러닝과 스마트폰 카메라를 이용한 냉장고 식재료 인식 및 레시피 추천 시스템인 '냉장고를 부탁해'에 대한 상세 설계의 총괄 개요를 제공한다.

1.1 서비스 정의

"냉장고를 부탁해"는 머신러닝 기법과 실시간 객체 인식을 기반으로, 스마트폰으로 실시간 촬영한 재료를 자동으로 판별하여 그에 맞는 레시피를 추천한다.

1.2 문서의 목적

- 본 문서는 머신러닝 기반 객체인식 레시피 추천 어플리케이션 개발 프로젝트 중 다양한 요구에 유연하게 대응하기 위한 시스템 상세설계를 명세하고 있다.
- 본 문서는 사용자, 기획팀, 프로젝트 관리자를 대상으로 한다.
- 본 문서를 바탕으로 고객의 요구사항을 명확하게 도출하여 향후 개발 과정에서 이를 반영하는데 그 목적이 있다. 따라서 본 문서는 요구정의서에서 요구한 상세 기술에 있어서 분석하여 명세한다.

1.3 범위

"냉장고를 부탁해" 서비스는 사용자가 스마트폰으로 냉장고 속 재료를 촬영하면 실시간으로 해당 재료의 이름을 파악할 수 있도록 한다. 이를 바탕으로, 비교 알고리즘을 통해 해당 재료로 만들 수 있는 추천 레시피를 사용자에게 제공한다. 또한, 재료 부족으로 아깝게 요리하지 못하는 추천 레시피를 소개하고, 부족한 재료를 어플리케이션 내에서 구매할 수 있도록 별도의 서비스를 제공한다.

본 프로젝트에서는 개발 진행에서 다음과 같은 범위를 둔다.

- 1) 카메라가 인식한 객체 구별 : 냉장고에서 인식한 객체들은 ML모델 내에 있는 데이터와 비교하여 정확도가 65% 이상일 경우 재료 리스트에 추가한다.
- 2) 재료 리스트 추가: 냉장고 안쪽에 있는 재료들을 인식하기 위해 카메라를 잠시 옆에 두고 재료를 꺼낸다. 그 후 카메라로 재료들을 인식하고 기존의 재료 리스트에 추가한다.
- 3) 레시피 추천 : 인식한 재료가 사용되는 레시피를 추천한다. 레시피는 미리 저장해둔 DB에서 참조한다.

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

- 4) 추가 자료 추천 : 몇 가지 자료를 추가하였을 시 새로운 요리가 가능할 경우, 해당 레시피를 분석하여 추가 자료 구매를 권유한다.

1.4 관련 문서

저자	문서 제목
김근영	ios 앱 개발을 위한 Swift 4
최은만	새로 쓴 소프트웨어 공학
Ramez Elmasri	데이터베이스 시스템 제 6판
최은만	UML로 배우는 시스템 분석 설계
오렐리앙 제롱	핸즈온 머신러닝(텐서플로를 활용한 머신러닝)

2. 시스템 소개 및 구성

2.1 개발 동향

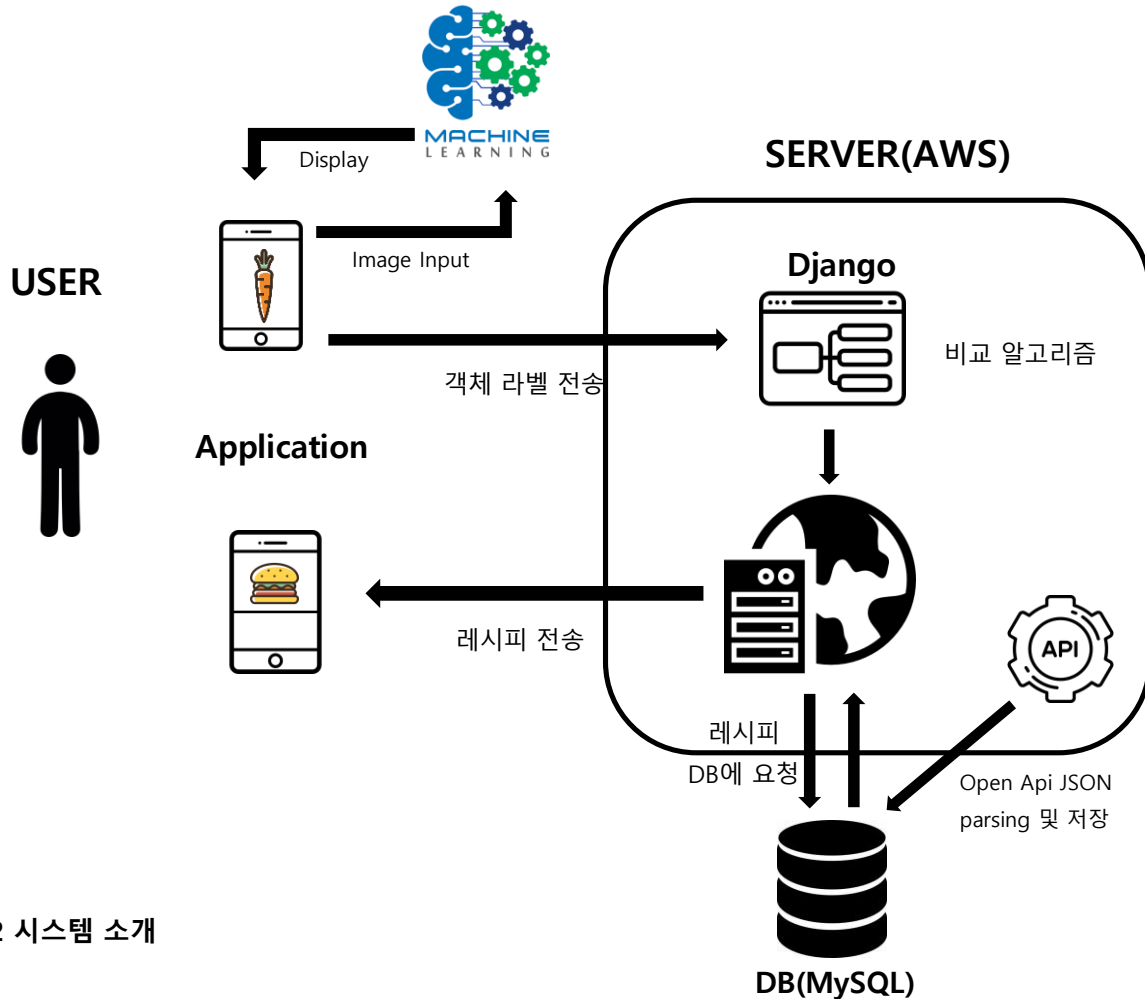
최근 "냉장고를 부탁해", "오늘 뭐 먹지" 등의 요리 예능이 많은 인기를 끌면서, 자취생부터 가정 주부까지 다양한 연령대의 사람들이 요리에 많은 관심을 보이고 있다. 또한, 이러한 예능에서 "집에서 쉽게 할 수 있는 요리"를 많이 선보이면서 사람들은 집에 있는 재료로 손쉽게 할 수 있는 요리 레시피를 많이 찾아보고 있다. 그로 인해 "만개의 레시피", "해먹남녀" 등의 레시피 관련 어플리케이션이 많은 인기를 끌고 있는 상황이다. 하지만 이들은 정해진 재료가 갖추어져야 요리를 할 수 있다는 한계점이 있다. 재료를 제대로 갖추지 못하거나 요리 실력이 부족한 사용자, 특히 자취생들은 수많은 레시피들이 존재함에도 불구하고 제대로 된 요리를 하지 못하는 경우가 많다. 따라서 본 프로젝트는 이러한 사용자를 타겟으로 하여, 머신러닝 기법을 이용한 객체 인식을 통해서 사용자가 가진 재료 내에서 더 편리하게 레시피를 추천할 수 있는 서비스를 제공하고자 한다.

2.2 개발 동기 및 필요성

최근 "쉽게 할 수 있는 요리"에 대한 관심이 급증하면서 요리 레시피 관련 어플리케이션도 많은 인기를 끌고 있다. 하지만 대부분의 어플리케이션은 단순히 여러 레시피를 보여주거나 인기 레시피를 추천하는 정도에만 머물러 있다. 또한, 해당 레시피의 재료가 없다면 다른 가능한 레시피를 일일이 찾아봐야 하는 불편함도 존재한다. 따라서 본 프로젝트는 최근 유행하는 머신러닝 기법을 이용하여 냉장고 속 재료를 촬영하기만 하면, 자동으로 재료를 인식하고 이에 맞게 레시피를 추천하는 서비스를 제공하고자 한다. 또한, 부족한 재료의 구매를 즉각적으로 배송 업체와 연결하여, 사용자들이 부족한 재료를 손쉽게 구매할 수 있도록 한다. 최종적으로, 이러한 기능을 통해서 사용자가 더욱 편리하고 효율적으로 요리할 수 있도록 도움을 주고자 한다.

2.3 시스템 구성요소

2.3.1 시스템 구성도



2.3.2 시스템 소개

USER	'냉장고를 부탁해' 서비스를 받는 대상이다. 스마트폰 카메라를 통해 인식한 객체의 라벨을 서버로 전송한다
Application	<p>: 스마트폰에서 운용되는 Application으로 각종 구성요소와 Interface를 담당하고 있다. 카메라를 이용하여 재료를 찍고, Delegate Multi-Thread는 머신러닝으로 재료를 인식하고 디스플레이해준다.</p> <p>인식된 재료를 서버에 객체와 라벨로 보내준다. 그리고 레시피를 서버로부터 받는다.</p>
Server	: 비교 알고리즘 기반으로 냉장고에 있는 재료에 적합한 레시피를 DB에 요청하고 받는다. 또한 식품안전처에서 제공받은 JSON 타입 Open Api 를 parsing하여 DB에 저장시킨다.

DB(MySQL)

Server에서 운용되는 Database로 Server와 정보를 주고 받을 수 있다. 테이블은 레시피 테이블, 마켓테이블, 유저테이블

3. 머신러닝 모델, Application 및 Django 서버

3.1머신 러닝

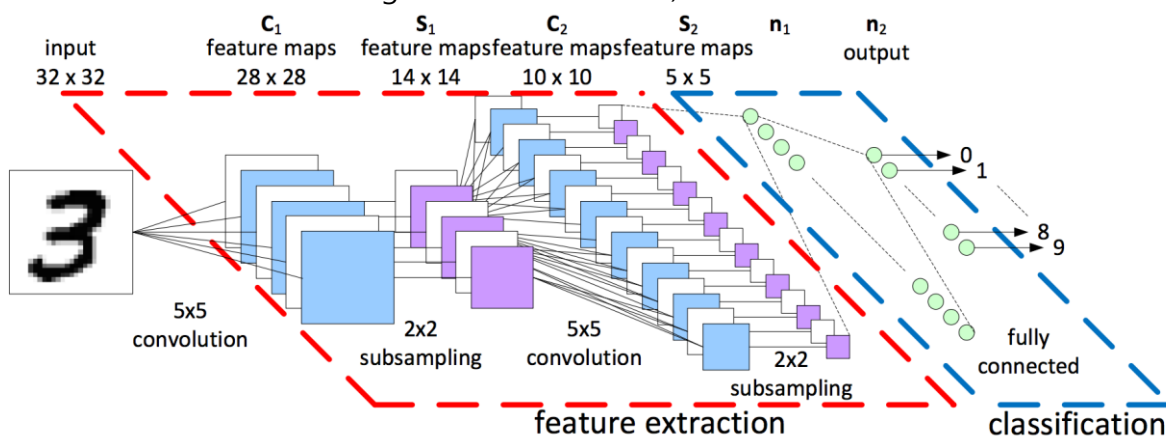
3.1.1 Create ML Frame Work를 사용한 데이터 모델 구축

Real time Object Classification : Mac OS에서 제공되는 Pre-Trained Model(scenePrint, Apple에서는 Pre-trained Model로 부르지 않고 Feature Extractor라 부른다.

왜냐하면 일반적인 pre-trained 모델은 이미지를 특징 벡터로 전환한 후 classification에서 사용하기 때문이다. (Apple uses the phrase "feature extractor" instead of the more common "pre-trained model". This is because the common layers of the pre-trained model convert an image into a feature vector that is good for classification tasks.)

출처: <https://www.learnopencv.com/how-to-train-a-deep-learning-based-image-classifier-in-macos/>

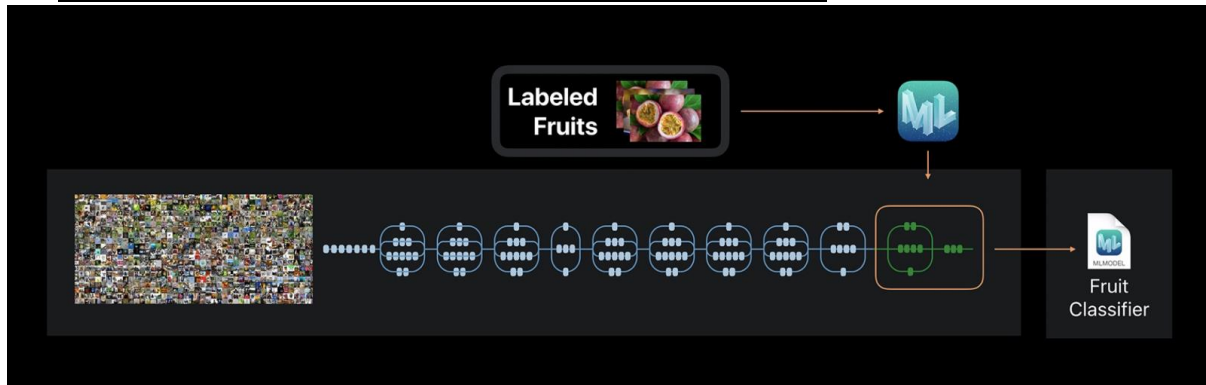
1억장 이상의 트레이닝 된 이미지 데이터가 있으며 이미지 데이터는 모두 특징 벡터로 전환되어 있어 transfer learning을 하는데 유리하다.)



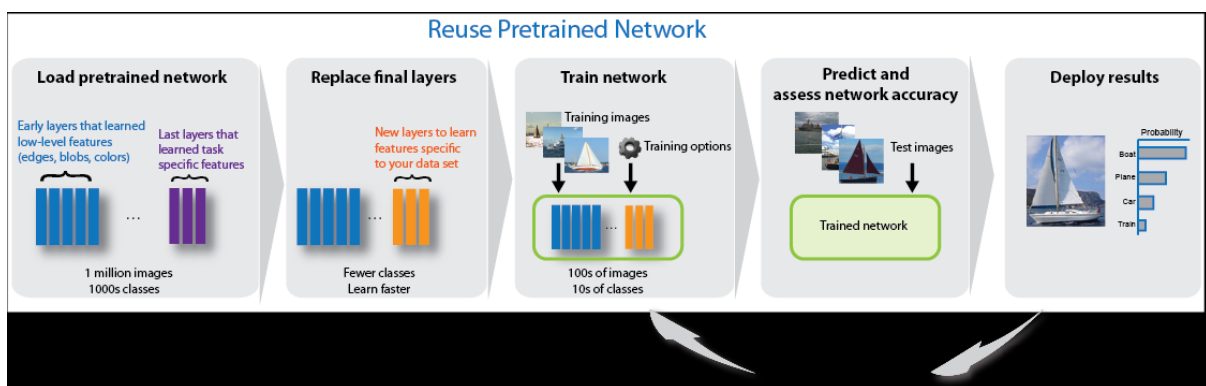
출처 : https://www.kernix.com/blog/a-toy-convolutional-neural-network-for-image-classification-with-keras_p14

즉, pre-trained 모델을 사용하는 것은 앞서 학습된 모델의 feature extraction을 이용해 마지막 layer를 학습시키는 것이다.

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스



출처 : <https://medium.com/eliiza-ai/wwdc-2018-apple-announces-create-ml-976c30a80192>



(Network에서 Pre-trained 모델을 사용해서 새로운 모델을 학습시키는 과정)

(출처 : <https://www.mathworks.com/help/deeplearning/examples/transfer-learning-using-alexnet.html?sessionid=108d986e091575f5d4b0d64853a8>)

→위의 그림에서는 네트워크에서 pre-trained 모델을 로드 받은 뒤, 마지막 레이어 (Last layers)를 사용자가 입력한 데이터 셋으로 바꾼 뒤 학습시키는 것을 설명하고 있다.

▼ Model Evaluation Parameters

Name	Type	Flexibility	Description
▼ Inputs			
image	Image (Color 299 x 299)	299... x 299...	Input image to be classified
▼ Outputs			
classLabelProbs	Dictionary (String → Double)		Probability of each category
classLabel	String		Most likely image category

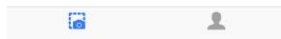
(실제로 만든 ML모델을 사용해서 이미지를 인식할 때 input과 output)

ML 모델을 사용하여 이미지를 인식 할 때 인풋(input)인 이미지(image)를 이용하여 연산한 결과 (output) 는 String(물체의 이름) 과 Double(정확도)를 반환해준다.

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

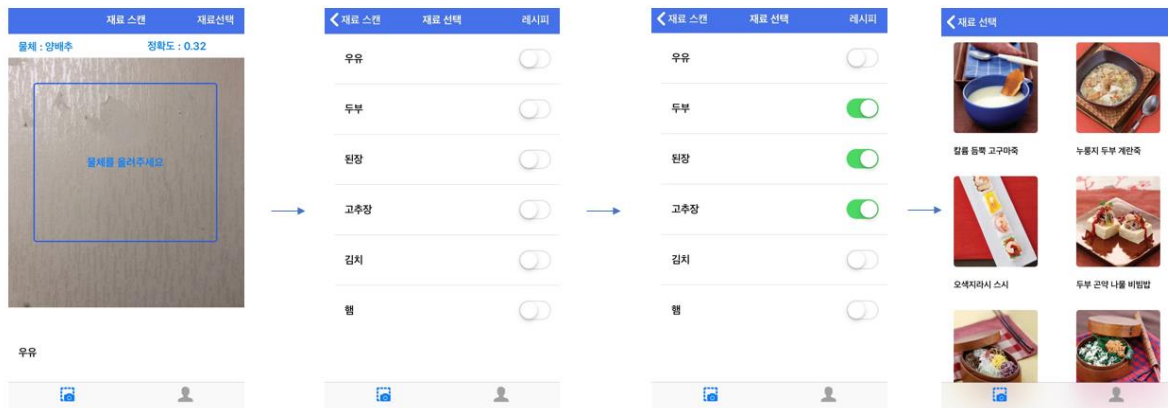


김치



3.2 Application

3.2.1 UI 구성



- 사용자가 물체를 스캔한다.
- 인식된 물체는 일정 정확도 (65%)가 넘으면 아래 표시된다.
- 재료를 모두 스캔하면 재료선택 화면으로 넘어간다.
- 스캔 된 재료가 표시되고 1~5개까지 반드시 선택하여야 한다.
- 서버에서 비교알고리즘을 통해 요리가 추천된다.

3.2.2 카메라 세션

- 객체를 인식하기 위하여 동영상을 실행해야 한다.
- 객체 인식을 위한 캡처 세션 (디바이스의 캡처 활동을 관리하고 디바이스로부터 데이터 흐름의 관리하는 객체) 인스턴스를 할당한다.



상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

- 캡처를 위한 디바이스(카메라 - 동영상)을 정하고 세션에 추가해준다.
- 캡처 세션을 시작해준다.

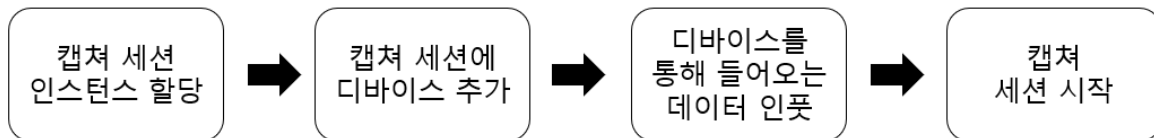
```
// 카메라 시작
// 카메라 동영상을 위한 AVCaptureSession 인스턴스 생성
// An object that manages capture activity and coordinates the flow of data from input devices to capture outputs.
let captureSession = AVCaptureSession()
// 사진 수준의 품질을 전송한다.
captureSession.sessionPreset = .photo

// 인풋을 주는 디바이스를 디폴트로하고 ( 폰의 카메라, 동영상으로 설정 )
guard let captureDevice =
    AVCaptureDevice.default(for: .video) else { return }

// 위에서 정한 디바이스가 보내주는 카메라의 인풋
guard let input = try? AVCaptureDeviceInput(device: captureDevice) else { return }

// 인스턴스에 인풋을 추가해준다.
captureSession.addInput(input)

// AVCaptureSession을 시작한다.
captureSession.startRunning()
```



- 데이터 인풋은 1초에 30장의 이미지를 찍는 동영상 인풋이다.

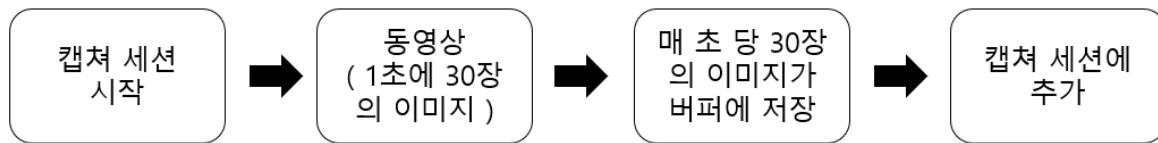
3.2.3 데이터 인풋과 아웃풋

- 디바이스는 카메라를 통해 객체를 캡처 한다.
- 캡처 된 객체는 동영상(stream)에서 이미지(image) 단위(30프레임 : 1초에 30장의 이미지)로 인식되어 인풋이 된다.
- 인풋은 버퍼에 저장되고 머신 러닝 모델을 사용해 (라벨, 정확도)의 아웃풋으로 표현 된다.

```
// A capture output that records video and provides access to video frames for processing.
// 비디오가 제공하고 있는 데이터 아웃풋
let dataOutput = AVCaptureVideoDataOutput()
// 비디오가 제공하는 아웃풋을 모니터링하기 위해 샘플 버퍼를 할당해준다.
// 데이터 아웃풋을 모니터링 되기 위하여 DispatchQueue큐에 저장된다.
dataOutput.setSampleBufferDelegate(self, queue: DispatchQueue(label: "videoQueue"))

captureSession.addOutput(dataOutput)
```

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스



3.2.4 델리게이트 패턴 (Delegate Pattern)

- 하나의 객체가 모든 일을 처리하는 것이 아니라 해야 할 일을 다른 객체에게 위임하는 것.
- 특정 이벤트가 발생했을 때의 처리를 다른 객체에 코드를 작성해주는 패턴.
- 즉, A 객체의 일을 B 객체에게 위임해 일을 처리 시킴.
- 이벤트가 발생하면 이벤트에 대한 처리를 객체에 위임하기 때문에 코드의 재사용성이 높고 한 객체에서 모든 것을 관리해 줄 필요가 없으므로 한 객체의 부담이 줄어 듦(코드가 짧아짐)
- A객체 : 카메라가 객체를 인식하여 프레임 단위(이미지)를 Queue 버퍼에 저장하는 객체
- B객체 : 델리게이트 객체, Queue버퍼에 저장하는 이벤트가 발생하면 Queue버퍼에 있는 데이터를 queueing하여 (Input -> Output)으로 변환해주는 대리인 객체
- 델리게이트 패턴을 사용한 이유 : A와 B의 일을 모두 A에서 처리해 줄 수도 있지만, A의 일을 B에 위임하여 이벤트가 발생할 때마다 실행해주게 하면 A객체의 부담이 줄어든다.(A객체는 계속해서 동영상을 찍고 30장의 이미지를 버퍼에 저장 해야 하는데 30장의 저장된 이미지에 대한 처리까지 함수를 호출하여 실시하면 A객체에 부담이 커짐. 따라서 이미지에 대한 처리를 B 객체에 위임하여 연속된 흐름의 작업을 두 객체가 실시하도록 만들.



상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

```
// 새로운 캡처 이미지 아웃풋이 버퍼에 저장되면 호출되는 델리게이트 메소드
func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection:
    AVCaptureConnection) {

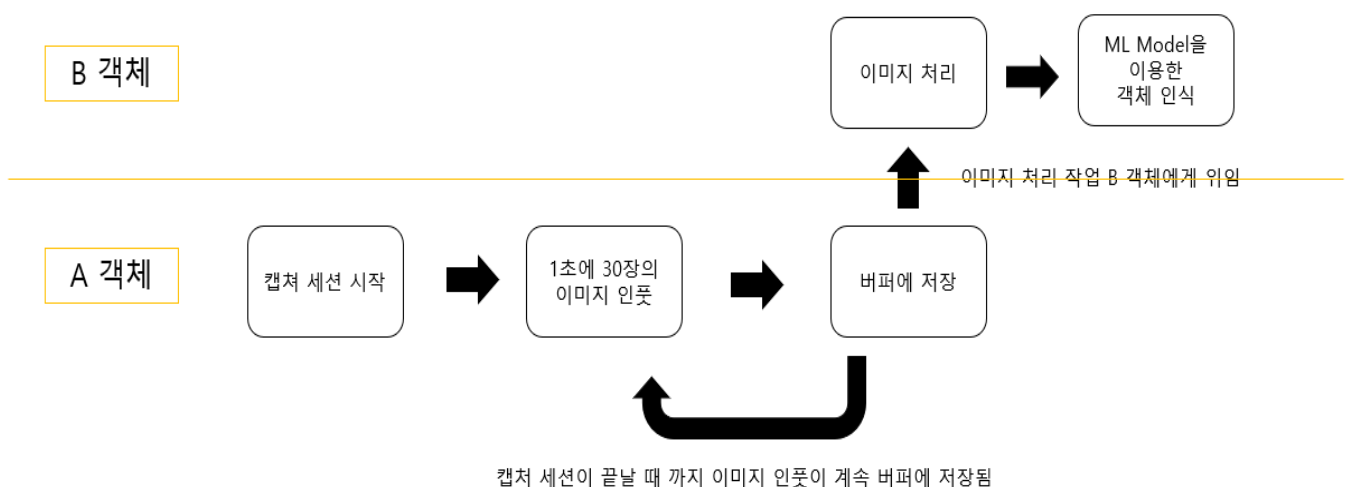
    // Returns a sample buffer's CVPixelBuffer of media data.
    // 매개변수로 받은 샘플버퍼를 리턴해준다.
    guard let pixelBuffer: CVPixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer) else {
        return
    }
    // 사용할 MLModel을 결정해준다.
    guard let model = try? VNCoreMLModel(for: Advanced().model) else { return }

    //An image analysis request that uses a Core ML model to process images.
    // ML model을 사용한 이미지의 분석 요청
    let request = VNCoreMLRequest(model: model) { (finishedReq, err) in
        // 에러 체크
        // print(finishedReq.results)

        // 분류된 이미지의 결과를 출력해주기 위해 results에 분석된 이미지 결과 할당
        guard let results = finishedReq.results as? [VNClassificationObservation] else { return }

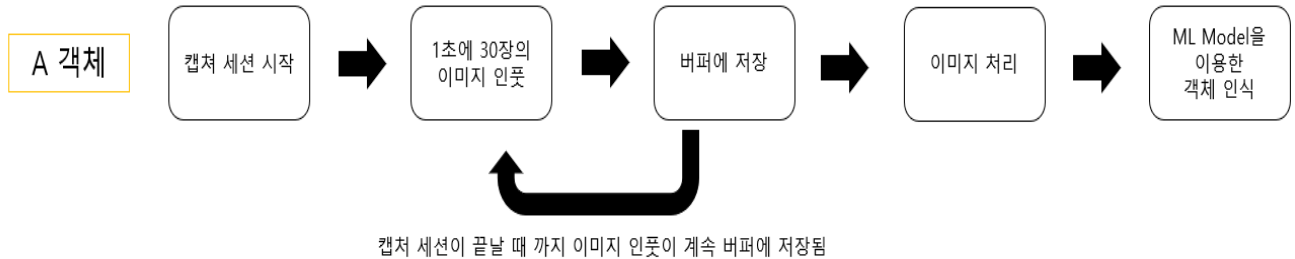
        guard let firstObservation = results.first else { return } ⚠ Value 'firstObservation' was defined but never used
    }
    // 이미지 분석 요청을 Array로 받아 관리한다.
    try? VNImageRequestHandler(cvPixelBuffer: pixelBuffer, options: [ : ]).perform([request])
}
```

- 위의 메소드는 A객체가 동영상의 30프레임 이미지를 버퍼에 담아주는 이벤트가 발생 할 때 실행이 되며 Machine Learning 모델을 이용해 이미지를 (String(물체이름) , String(정확도))로 반환해주는 메소드
- 즉, 카메라가 찍으면 모든 인풋은 버퍼에 저장되고 버퍼에 저장되는 이벤트가 발생하는 순간 위의 메소드가 호출되어 아웃풋으로 반환한 뒤 다시 다른 버퍼에 담아준다.



- A 객체가 모든 일을 처리할 수 있지만 다음과 같은 흐름이 되어 한 객체가 많은 일을 처리해야 하기 때문에 비효율적이다.

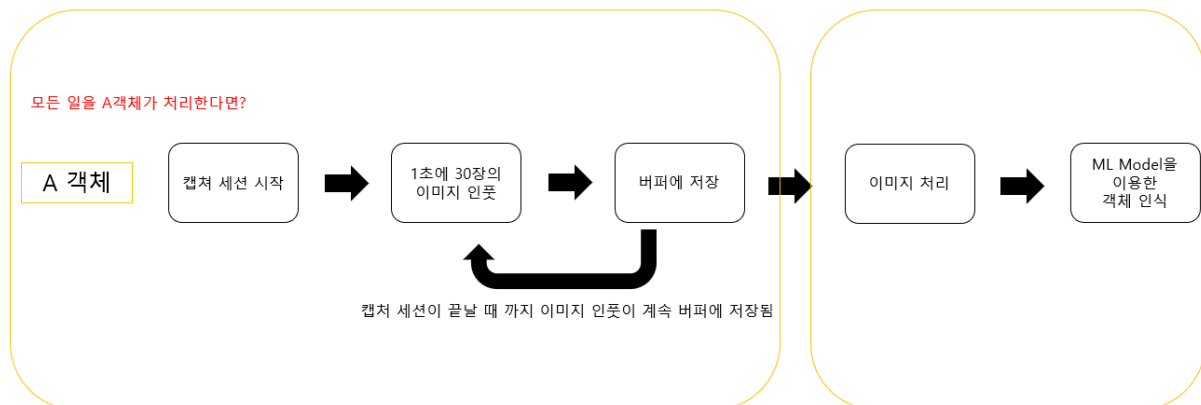
모든 일을 A객체가 처리한다면?



- 즉, 굳이 델리게이트 패턴을 사용한 이유는 두 객체에서 처리하면 하나의 작업 흐름에 있는 두 가지 일을 각 객체가 하나씩 맡아서 할 수 있으므로 효율적이기 때문이다.

첫 번째 일

두 번째 일 -> 다른 객체에 위임하면 효율적!



3.2.5 멀티 스레드

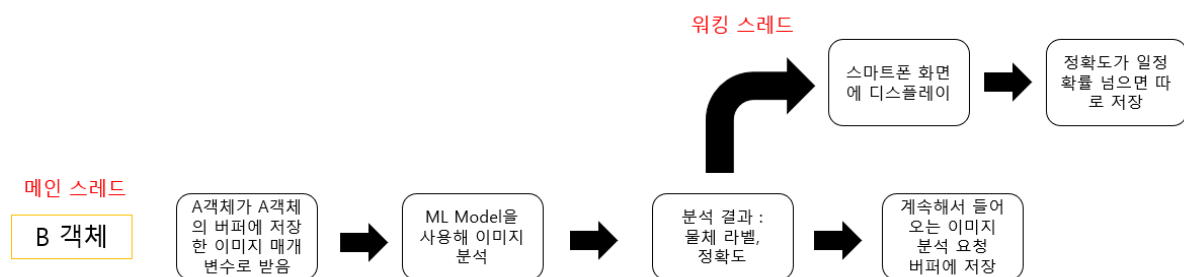
- 캡처 세션의 인풋(1초에 30장의 이미지)가 버퍼에 저장되면 델리게이트 객체가 실행되는데, 델리게이트 객체는 멀티 스레드로 실행한다.
- 메인 스레드 (첫 번째 스레드) : A 객체에 있는 이미지를 B 객체로 받아와서 물체이름, 정확도 로 변환해주는 작업을 진행한 뒤 이미지 분석 요청을 관리하는 버퍼 (물체이름, 정확도)에 큐잉 시켜주는 역할.
- 워킹 스레드 (두 번째 스레드) : 메인 스레드가 버퍼에 넣은 (물체이름 , 정확도)를 큐잉 하여 정확도를 분석 (65% 이상 이면 새로운 배열에 저장)하고 화면에 디스플레이 해준다.

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

```
// 물체 인식이 65%가 넘으면 지정한 배열에 추가해주고 결과값을 표시해주는 비동기 스레드
DispatchQueue.main.async {
    if (firstObservation.confidence > 0.65)
    {
        if ( !self.hasIngredient.contains(String(firstObservation.identifier)))
        {
            self.hasIngredient.append(firstObservation.identifier)
            self.Str = self.Str + "\(self.ingredient[firstObservation.identifier]!) "
        }
    }
    self.object.text = "물체 : \(self.ingredient[firstObservation.identifier]!)"
    self.confidence.text = "정확도 : \(round(firstObservation.confidence*100)/100)"
    //round(avgTemp*100)/100
    self.output.text = self.Str
}
```

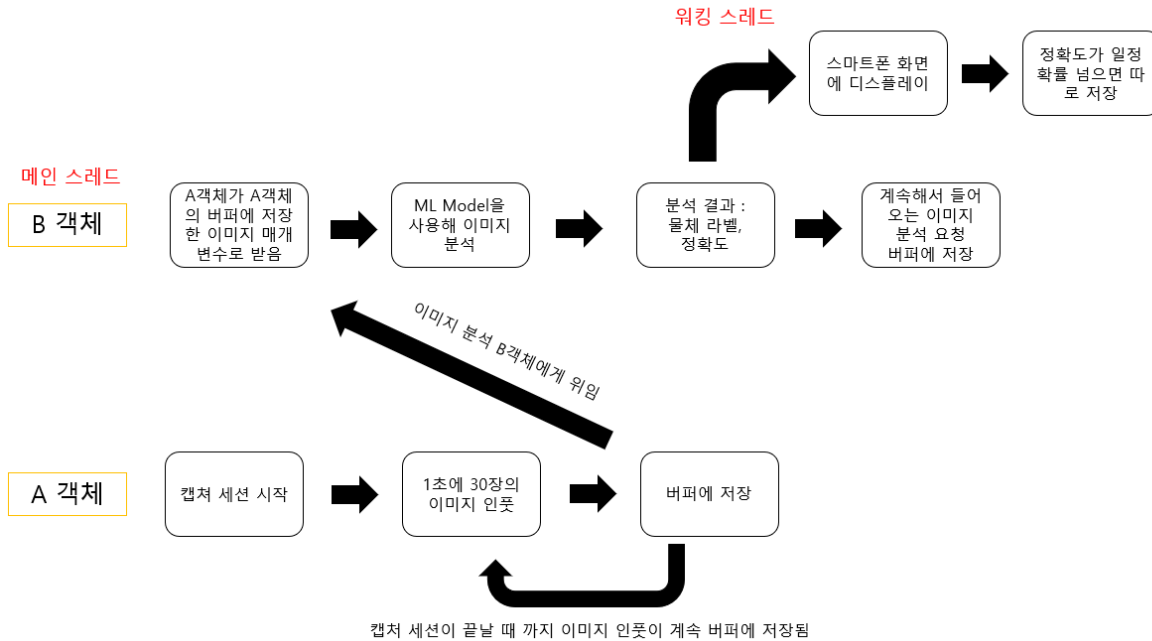
(메인스레드가 저장한 버퍼에 있는 첫번째 값을 큐잉 하여 계산해준다. 버퍼에 값은 메인 스레드에서 계속 갱신되기 때문에 워킹 스레드의 첫번째 값도 계속해서 갱신된다.)

- 멀티 스레드 이점 : 첫번째 스레드가 계속해서 이미지를 분석해야 하고 새로운 요청이 계속 들어오기 때문에 (카메라가 계속해서 동영상을 찍고 이미지를 버퍼에 저장하기 때문에 델리게이트 객체가 계속해서 호출된다.) 하나의 스레드를 가지고 연속적인 흐름으로 코드를 작성하면 작업이 더더 질 수 있다. 하지만 두번째 스레드를 만들어 버퍼에 있는 값이 저장되자마자 꺼내어 분석하고 디스플레이하는 작업을 따로 해주면 Concurrent 프로그램을 작성할 수 있다.



- 멀티 스레드를 사용한 이유: B 객체는 계속해서 요청을 받는데, 처리 속도보다 요청이 들어오는 속도가 느릴 경우를 대비하여 요청을 버퍼에 큐로 저장한 뒤 처리해주고 있다. 따라서, 메인 스레드가 머신 러닝 모델을 사용하여 이미지를 분석하는 동시에 새로운 스레드를 만들어 화면에 출력하는 작업과 정확도를 분석하는 작업을 동시에 실행시켜줄 필요가 있다. 동시에 실행시켜주지 않는다면 메인 스레드의 흐름이 길어지게 되고 B 객체에 들어오는 요청이 처리되지 못한 채 쌓일 수 있기 때문이다.

3.2.6 실시간 객체 인식의 전체 흐름



3.2.7 사용자가 선택한 재료를 카운트

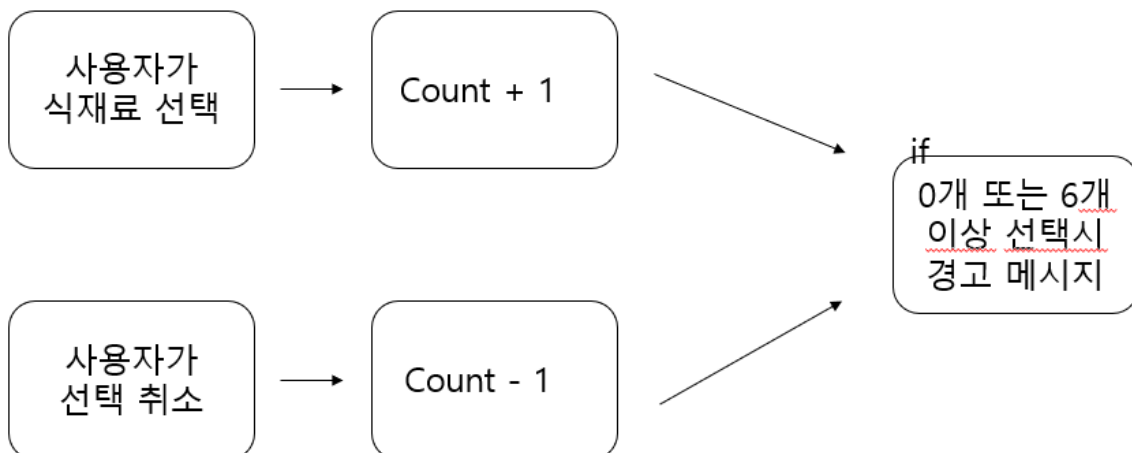
사용자가 두번째 화면에서 원하는 재료를 선택하면 Count가 올라가고 선택을 취소하면 Count가 내려가도록 설계
즉, 사용자가 선택한 재료의 개수만큼 카운트를 센다.



상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
  
    let row = self.hasIngredient[indexPath.row]  
    // 테이블 뷰의 셀을 재사용 할 수 있도록 dequeue Reusable Cell로 만들어 준다.  
    // 이렇게 되면 한번 사용한 셀을 재사용 할 수 있기 때문에 계속해서 새로운 셀을 만들어줄 필요가 없다.  
    let cell = tableView.dequeueReusableCell(withIdentifier: "IngredientCell") as! IngredientCell  
  
    // 재료의 이름을 라벨에, 스위치를 이용해 모든 값을 선택하지 않은 것으로 설정한다.  
    cell.ingredient.text = self.ingredient[row]  
    cell.want.isOn = false  
  
    // 사용자가 switch를 선택 할 때마다 호출되는 메소드  
    cell.want.addTarget(self, action: #selector(switched(_:)), for: .valueChanged)  
  
    return cell  
}  
  
// 사용자가 선택을 하면 Count 값을 +1 해주고 사용자가 선택을 취소하면 -1 해준다.  
// swtich의 초기값은 0이다.]  
@objc func switched(_ sender: UISwitch){  
    if ( sender.isOn == true ) {  
        switchCounter += 1  
    } else {  
        switchCounter -= 1  
    }  
  
    print(switchCounter)  
}
```

→위의 함수에서 사용자가 선택을 하면 switchCounter가 ++1이되고 사용자가 선택을 하지 않으면 switchCounter가 --1이 된다.



3.2.8 사용자가 선택한 식재료를 HTTP Request

```
func requestHttpPost(_ sender: Any){
    let json = ["재료1":self.hasIngredient[0],
               "재료2":self.hasIngredient[1],
               "재료3":self.hasIngredient[2],
               "재료4":self.hasIngredient[3],
               "재료5":self.hasIngredient[4]]

    // url 주소 할당
    guard let url = URL(string: "ec2-54-180-113-7.ap-northeast-2.compute.amazonaws.com") else { return }

    // http 설정 배열을 JSON 파일로 만들어준다.
    var request = URLRequest(url: url)
    request.httpMethod = "POST"
    request.addValue("application/json", forHTTPHeaderField: "Content-Type")
    guard let httpBody = try? JSONSerialization.data(withJSONObject: json, options: []) else { return }

    request.httpBody = httpBody

    let session = URLSession.shared
    session.dataTask(with: request) { (data, response, error) in
        if let response = response {
            print(response)
        }
        if let data = data {
            do {
                let json = try JSONSerialization.jsonObject(with: data, options: [])
                print(json)
            } catch {
                print(error)
            }
        }
    }.resume()
}
```

- 사용자가 선택한 식재료를 JSON형식으로 만들기전 JSON에 들어갈 내용을 dictionary로 만든다.
- URL주소를 할당 받고 POST방식과 헤더 부분을 할당해준다.
- 미리 dictionary로 만들어 두었던 JSON의 내용부분을 JSONSerialization함수를 통해 JSON객체로 만들고 Http body부분에 넣어준다.
- URLSession 메소드를 통해 앞서 할당 받았던 URL로 HTTP 메시지를 보내준다.
- 서버로부터 받은 데이터는 다시 JSONSerialization을 통해 JSON객체에서 Native로 바뀌준 후 저장해준다.



3.2.9 POST를 통해 받은 Data 화면에 출력

- JSON을 JSONSerialization.jsonObject함수를 통해 Native Data로 바꾼다.
- Native Data로 바꾼 데이터를 이용해 화면에 출력한다.

```

override func viewDidLoad() {
    // API 호출을 위한 URI 생성
    let url = "http://openapi.foodsafetykorea.go.kr/api/6d82f3c09e2f4568b124/COOKRCP01/json/1/10"
    let apiURI: URL! = URL(string: url)

    // REST API를 호출
    let apidata = try! Data(contentsOf: apiURI)

    do {
        let apiDictionary = try JSONSerialization.jsonObject(with: apidata, options: []) as! NSDictionary

        let cookrcp = apiDictionary["COOKRCP01"] as! NSDictionary
        let row = cookrcp["row"] as! NSArray

        for food in row {
            let f = food as! NSDictionary

            self.foodImage.append(f["ATT_FILE_NO_MAIN"] as! String)
            self.foodName.append(f["RCP_NM"] as! String)
        }
    } catch { }
}
  
```

3.3 사용자 레시피 요청 처리

사용자로부터 레시피 요청을 받았을 때 레시피를 응답할 수 있어야 한다.

3.3.1 요청 객체

request.data로서 json데이터를 포함하여 아무 데이터나 다룰 수 있고 이를 파이썬 데이터로 변환해야 한다. 사용자와 재료의 정보가 담겨서 온다.

3.3.2 응답 객체

return Response(data)이며 클라이언트가 요청한 형태로 콘텐츠를 렌더링 한다. 여기서는 json으로 데이터를 변환 후 전달한다. 즉 레시피가 넘어간다.

3.3.3 요구 사항

- 사용자의 요청 객체에 담겨 있는 재료를 새로운 리스트에 담는다.
- 데이터베이스에 저장되어 있는 모든 레시피 정보를 불러와 객체에 담는다.
- 이를 함수로 정의하여 요청이 들어올 때 마다 응답할 수 있도록 해야 한다.
- 다음과 같이 비교알고리즘을 사용하여 최적의 레시피 메뉴 이름을 선택한다.



상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

3.3.4 사용자 요청 설계

사용자가 보낸 json 파일은 다음과 같다.

```
json = [{"재료1":self.hasIngredient[0],  
        "재료2":self.hasIngredient[1],  
        "재료3":self.hasIngredient[2],  
        "재료4":self.hasIngredient[3],  
        "재료5":self.hasIngredient[4]}
```

이를 요청 객체인 request.data에 담겨 있고 이를 파이썬 native data로 변환해주어야 한다.

```
- def Find_menu(request):  
    import io  
    from rest_framework.parsers import JSONParser  
    stream = io.StringIO(request.data) # json 데이터  
    data = JSONParser().parse(stream)  
    user_ingredient = list()  
    user_ingredient.append(data)
```

- 사용자가 재료를 보내면서 레시피를 요청하면 Find_menu 함수를 호출하게 된다.
- request.data를 rest_framework.parsers에서 제공하는 JSONParser를 사용하여 파싱한다.
- user_ingredient에서 담긴 재료들을 기반으로 하여 비교알고리즘을 실행하면 된다.

3.4 사용자 재료 기반 레시피 선택

3.4.1 사용자의 재료 입력 및 DB Select

```
1 import pymysql.cursors  
2 import time  
3  
4 start = time.time() # 시간 측정 시작  
5  
6 # 시나리오 1: 사용자 재료 수신, DB와 비교, 추천 레시피 전달  
7 user_ingredient = ["우유", "달걀", "닭", "두부"] # 사용자로부터 입력 받은 재료 예시  
8  
9 mydb = pymysql.connect(  
10     host='localhost',  
11     user='root',  
12     password='12345678',  
13     db='food',  
14     charset='utf8mb4'  
15 )  
16  
17 try:  
18     with mydb.cursor() as cursor:  
19         sql = 'SELECT Recipe_name, Ingredients FROM myapp_food'  
20         cursor.execute(sql)  
21         recipe_DB = cursor.fetchall()  
22         # print(result)  
23  
24 finally:  
25     mydb.close()
```

사용자가 보낸 "우유, 달걀, 닭, 두부" 라는 재료를, deserialize를 통해서 user_ingredient 라는

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

파이썬 리스트에 넣은 상태라고 가정한다. Django에서 DB에 접근하여 SELECT *을 통해 recipe_DB 리스트에 레시피 정보를 전부 받아온다.

3.4.2 사용자 재료와 DB 재료 비교 코드

```
cntnum = [0 for _ in range(1000)] # 사용자 재료와 DB재료의 매치 갯수

for i in range(0, 1000):
    cntnum[i] = 0
    for j in range(0, len(user_ingredient)):
        comparing = recipe_DB[i][1].find(user_ingredient[j])
        # find() 를 통해서 사용자의 재료가 DB 재료에 매칭 되는 지 확인(매칭되면 >=0, 매칭 되지 않으면 -1)
        if comparing >= 0:
            cntnum[i] += 1 # 매칭 되었을 때 갯수 ++
```

Find() 함수를 통해서 DB의 Ingredient 애트리뷰트에 있는 재료 정보와 사용자가 서버로 전송한 재료 정보를 비교한다. Find()가 true이면 >=0이고, false이면 =-1 이므로, if comparing >=0을 통해서 재료가 매칭되는지 일일이 확인한다.

3.4.3 비교 결과를 바탕으로 추천 레시피 출력

```
maxi = 0 # 최대 매칭 레시피를 찾기 위한 변수
reci_num = 1 # 결과 레시피 번호

for m in range(0, 1000): # 최대 매칭 레시피의 매칭 재료 개수 설정(maxi의 최신화)
    if cntnum[m] > maxi:
        maxi = cntnum[m]
        # print(maxi)

print("사용자 재료: %s" % (user_ingredient), "\n")

for m in range(0, 1000): # maxi에 해당하는 최대 매칭 레시피 모두 출력
    if cntnum[m] == maxi:
        print("추천 레시피%d: %s" % (reci_num, recipe_DB[m][0]))
        reci_num += 1
```

Cntnum 리스트는 사용자의 재료와 DB 재료가 몇 개나 매칭되는지를 저장한다. 첫 번째 for문은 cntnum 리스트의 최대값을 maxi에 넣는 과정이다. 두 번째 for문은 1000개의 레시피 중 maxi 값만큼 매칭되는 모든 추천 레시피를 출력하는 과정이다.

3.4.4 추천 레시피 중 사용자가 선택한 레시피 출력

```
# 시나리오 2: 사용자가 추천 레시피 중 하나 선택, DB에서 해당 레시피 정보 출력, 선택 레시피 정보 다시 전달
user_choice = ["닭고기김치찌개"]_# 사용자가 닭고기김치찌개를 선택함

try:
    with mydb.cursor() as cursor:
        sql = 'SELECT * FROM myapp_food WHERE Recipe_name = %s'
        cursor.execute(sql, user_choice)
        Rec_recipe = cursor.fetchone()

    finally:
        mydb.close()

    print("\n-----")
    print("\n선택 요리: %s" % Rec_recipe[1])
    print("\n필요 재료: %s" % Rec_recipe[2])
    print("\n조리 방법: %s" % Rec_recipe[3].strip())

    print("\n소요 시간: %s 초" % (round((time.time() - start), 4)))_# 시간 측정 끝
```

사용자가 "닭고기김치찌개"를 선택했다고 가정한다. "SELECT * FROM myapp_food WHERE Recipe_name = %s" 를 이용해 사용자가 선택한 메뉴를 DB에서 가져온다. Recipe_name 자체가 Primary Key이기 때문에 별도의 구별 방법이 없이도 가능하다.

3.4.5 결과 예시

```
사용자 재료: ['우유', '달걀', '닭', '두부']

추천 레시피1: 단호박 닭고기 들깨탕
추천 레시피2: 닭고기김치찌개
추천 레시피3: 나가사키부대찌개
추천 레시피4: 닭고기스테이크
추천 레시피5: 소금물에 찐 시금치 두부선
추천 레시피6: 두부 크림 브릴레
추천 레시피7: 삼색 닭 만둣국
추천 레시피8: 크림닭
```

사용자의 재료에 따라 추천 레시피 8개가 출력되었다.



상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

선택 요리: 닭고기김치찌개

필요 재료:

재료 닭가슴살(60g), 애호박(30g), 미나리(20g), 청양고추(3g), 두부(60g)
김치(120g), 참기름(20g), 느타리버섯(10g), 달걀(50g)
육수 다시마(5g), 무(40g), 대파(5g), 알파(10g), 마른 고추(3g), 멸치(10g)
마늘(10g), 참주(10g), 물(300g)

조리 방법:

1. 육수 재료를 넣고 중간 불로 끓인 뒤
체에 걸러 육수를 만든다.
2. 닭가슴살은 채 썰 뒤 물에 넣고
삶는다.
3. 애호박은 반달 썰고, 미나리는 5cm
길이로 썰고, 청양고추는 잘게 썰 뒤
찬물에 담가 씨를 빼고, 두부는 한입
크기로 썰어 물에 담가둔다.
4. 김치를 한입 크기로 썰 뒤 참기름에
볶다가 육수를 붓는다.
5. 두부, 애호박, 느타리버섯, 미나리,
닭가슴살을 넣어 끓여오르면
달걀물과 청양고추를 넣고 조금
더 끓여 마무리한다.

소요 시간: 0.0439 초

추천 레시피 중 사용자가 선택한 1개의 레시피에 대한 정보가 출력되었다. 소요 시간은 약 0.04초이다.

3.5 서버 레시피 응답 처리

```
from rest_framework.renderers import JSONRenderer
from rest_framework.response import Response

json = JSONRenderer().render(Rec_recipe)
return Response(json)
```

- 이 레시피를 다시 json 파일로 콘텐츠 타입을 만든다.
- json으로 변환한 객체를 HttpResponse 형태로 반환해야 한다.
- Rest API가 제공해주는 from rest_framework.response import Response을 사용하여 요청
이나 응답을 명시적으로 연결하지 않아도 원하는 형태로 응답 객체를 렌더링 해줍니다.

4. Server

4.1 AWS 환경 설정

4.1.1 인스턴스 설정

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스
인스턴스: i-098aa5f28d1850d1a (new server) 퍼블릭 DNS: ec2-54-180-113-7.ap-northeast-2.compute.amazonaws.com

설명	상태 검사	모니터링	태그
인스턴스 ID	i-098aa5f28d1850d1a	퍼블릭 DNS(IPv4)	ec2-54-180-113-7.ap-northeast-2.compute.amazonaws.com
인스턴스 상태	running	IPv4 퍼블릭 IP	54.180.113.7
인스턴스 유형	t2.micro	IPv6 IP	-
탄력적 IP	-	프라이빗 DNS	-
가용 영역	ap-northeast-2c	프라이빗 IP	-
보안 그룹	EC2 Security Group_190513. 인바운드 규칙 보기. 아웃바운드 규칙 보기	보조 프라이빗 IP	-
예약된 이벤트	예약된 이벤트 없음	VPC ID	vpc-a77284cc
AMI ID	ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20190212 (ami-067c32f3d5b9ace91)	서브넷 ID	subnet-12194e5e
플랫폼	-	네트워크 인터페이스	eth0
IAM 역할	-	소스/대상 확인	예
키 페어 이름	ubuntu_190513	T2/T3 무제한	비활성
소유자	977304418090	EBS 최적	아니요
시작 시간	2019년 5월 13일 오전 1시 8분 46초 UTC+9(18시간)	루트 디바이스 유형	ebs

[Table 2] 인스턴스 설정

AWS 에서 할당받은 Public DNS 주소를 통해 App 과 http 통신을 하는데 사용한다.

4.1.2 보안 그룹 설정

보안 그룹: sg-0df5c9b8b8bfc58abc

설명

인바운드

아웃바운드

태그

편집

유형 ⓘ	프로토콜 ⓘ	포트 범위 ⓘ	소스 ⓘ	설명 ⓘ
HTTP	TCP	80	0.0.0.0/0	http default port
HTTP	TCP	80	::/0	http default port
사용자 지정 TCP 규칙	TCP	8000	0.0.0.0/0	django default por...
사용자 지정 TCP 규칙	TCP	8000	::/0	django default por...
SSH	TCP	22	220.120.112.2/32	my server

[Table 3] 보안 그룹 설정

외부에서 접근을 허용하기 위해 80 번 포트를 개방한다. django runserver 을 위해 8000 번 포트 또한 개방한다.

4.2 서버 배포 과정



자료 출처 : <https://www.vndeveloper.com/django-behind-uwsgi-nginx-centos-7/>

Nginx는 서버 앞단에서 사용자의 request를 가장 먼저 처리하는 웹서버이다. 가장 널리

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

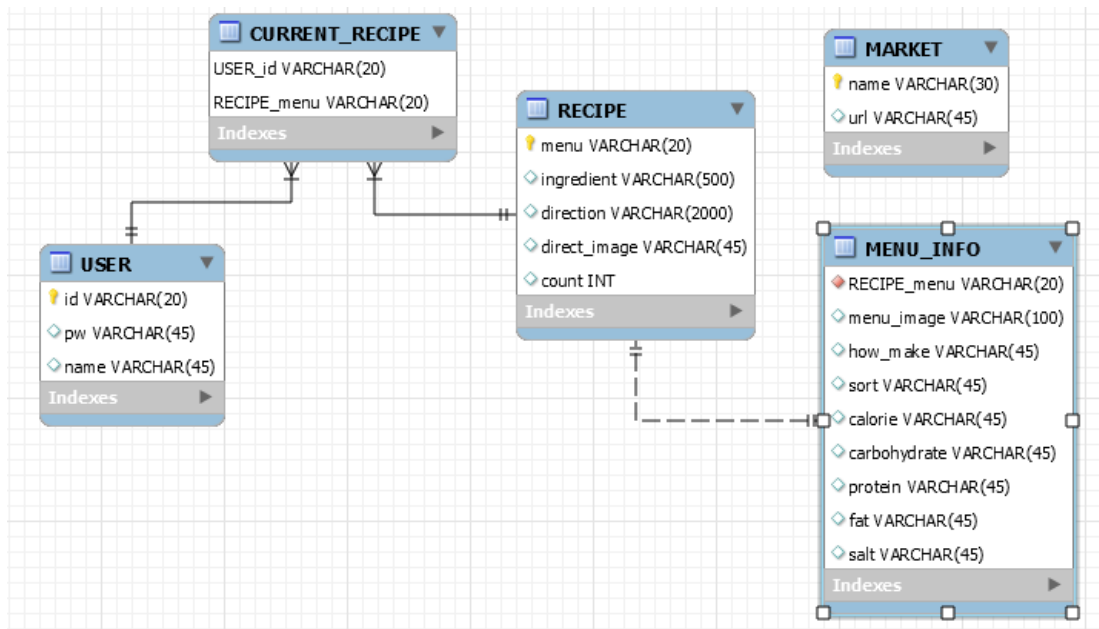
사용되는 apache와 같은 역할이다.

WSGI(Web Server Gateway Interface)는 웹 서버(Nginx)와 웹 애플리케이션(Django)를 이어주는 역할(인터페이스)을 한다. 웹 서버로 들어온 요청을 Python언어로 만들어진 웹 애플리케이션과 소통할 수 있도록 해주는 역할을 한다. uWSGI는 WSGI규격으로 만들어진 서버 중 하나이며 Django 프로젝트를 배포할 때 가장 일반적으로 사용된다.

5. 데이터베이스

5.1 DB 설계

5.1.1 설계도



- 5개의 테이블
- current_recipe은 user와 recipe의 다 대 다 관계에서 파생된 weak 테이블
- market에는 사용자에게 보내줄 마켓 url
- menu_info와 recipe는 일대 일 관계

5.1.2 데이터베이스 정규화

- weak테이블을 제외한 모든 테이블은 하나의 키가 다른 애트리뷰트들을 모두 결정하는 스키마를 가지고 있으므로, 정규화를 거칠 필요 없이, bcnf에 속해있음.

5.1.3 사용자 정보 관리

```

CREATE TABLE IF NOT EXISTS `food`.`USER` (
  `id` VARCHAR(20) NOT NULL,
  `pw` VARCHAR(45) NULL,
  `name` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
    
```



5.1.4 레시피 관리

```
CREATE TABLE IF NOT EXISTS `food`.`RECIPE` (  
  `menu` VARCHAR(20) NOT NULL,  
  `ingredient` VARCHAR(500) NULL,  
  `direction` VARCHAR(2000) NULL,  
  `direct_image` VARCHAR(45) NULL,  
  `count` INT NULL,  
  PRIMARY KEY (`menu`))  
ENGINE = InnoDB;
```

5.1.5 메뉴 정보 관리

```
CREATE TABLE IF NOT EXISTS `food`.`MENU_INFO` (  
  `RECIPE_menu` VARCHAR(20) NOT NULL,  
  `menu_image` VARCHAR(100) NULL,  
  `how_make` VARCHAR(45) NULL,  
  `sort` VARCHAR(45) NULL,  
  `calorie` VARCHAR(45) NULL,  
  `carbohydrate` VARCHAR(45) NULL,  
  `protein` VARCHAR(45) NULL,  
  `fat` VARCHAR(45) NULL,  
  `salt` VARCHAR(45) NULL,  
  INDEX `fk_MENU_INFO_RECIPE1_idx` (`RECIPE_menu` ASC) VISIBLE,  
  CONSTRAINT `fk_MENU_INFO_RECIPE1`  
    FOREIGN KEY (`RECIPE_menu`)  
    REFERENCES `food`.`RECIPE` (`menu`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE)  
ENGINE = InnoDB;
```

5.2 Django

5.2.1 Django와 mysql 연결



상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'food',
        'USER': 'root',
        'PASSWORD': '*****',
        'HOST': 'localhost',
        'PORT': ""
    }
}
```

(venv) C:\Users\user\PycharmProjects\RESTFUL\restful>python manage.py inspectdb > models.py

```
class MenuInfo(models.Model):
    recipe_menu = models.ForeignKey('Recipe', models.DO_NOTHING, db_column='RECIPE_menu')
    how_make = models.CharField(max_length=45, blank=True, null=True)
    sort = models.CharField(max_length=45, blank=True, null=True)
    calorie = models.CharField(max_length=45, blank=True, null=True)
    carbohydrate = models.CharField(max_length=45, blank=True, null=True)
    protein = models.CharField(max_length=45, blank=True, null=True)
    fat = models.CharField(max_length=45, blank=True, null=True)
    salt = models.CharField(max_length=45, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'menu_info'
```

Setting.py에 연결되어 있는 데이터베이스에 대한 models.py의 내용을 가져온다.

5.2.2 JSON parsing algorithm

```
import pymysql.cursors
import urllib
import json
import pandas as pd
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

FoodIdListURL = "http://openapi.foodsafetykorea.go.kr/api/6d82f3c09e2f4568b124/COOKRCP01/json/1/10"

FoodIdPage = urllib.request.urlopen(FoodIdListURL)
FoodIdData = json.loads(FoodIdPage.read())
FoodIdData

FoodIDDF = pd.DataFrame()
FoodIDDF = FoodIDDF.append(
    {
        "RCP_NM": "", "RCP_PARTS_DTLS": "", "MANUAL01": "", "MANUAL02": "", "MANUAL03": "", "MANUAL04": ""
    },
    ignore_index = True)
FoodIDDF

num = len((FoodIdData["COOKRCP01"]["row"]))

mydict = {}
direction = ''
for i in range(0,num):
    FoodIDDF.ix[i,"RCP_NM"] = FoodIdData["COOKRCP01"]["row"][i]["RCP_NM"]
```

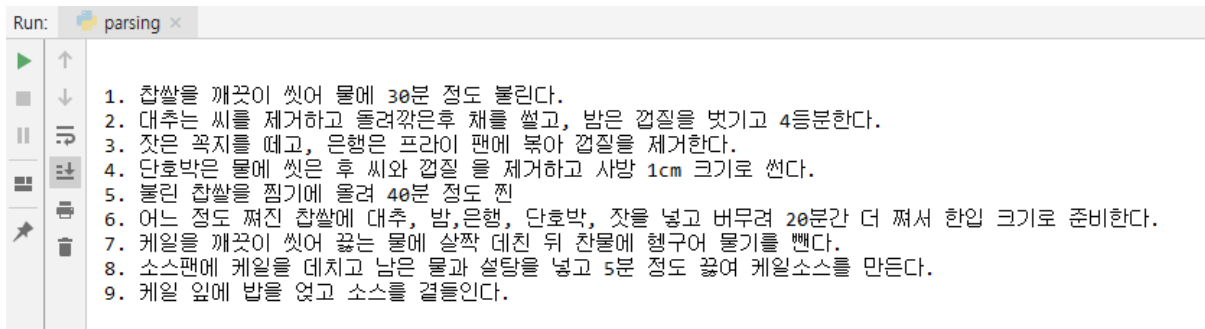


상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

5.2.3 정제 과정

```
if FoodIDDF.ix[i, "MANUAL01"] != '':
    if FoodIDDF.ix[i, "MANUAL01"][-1] != '.':
        FoodIDDF.ix[i, "MANUAL01"] = FoodIDDF.ix[i, "MANUAL01 ~ 20"][: -1]
```

Run: parsing x



1. 찹쌀을 깨끗이 씻어 물에 30분 정도 불린다.
2. 대추는 씨를 제거하고 돌려깎은후 채를 썰고, 밤은 껍질을 벗기고 4등분한다.
3. 잣은 꼭지를 떼고, 은행은 프라이 팬에 볶아 껍질을 제거한다.
4. 단호박은 물에 씻은 후 씨와 껍질 을 제거하고 사방 1cm 크기로 썬다.
5. 불린 찹쌀을 찜기에 올려 40분 정도 찐
6. 어느 정도 찌진 찹쌀에 대추, 밤, 은행, 단호박, 잣을 넣고 버무려 20분간 더 찌서 한입 크기로 준비한다.
7. 케일을 깨끗이 씻어 끓는 물에 살짝 데친 뒤 찬물에 헹구어 물기를 뺀다.
8. 소스팬에 케일을 데치고 남은 물과 설탕을 넣고 5분 정도 끓여 케일소스를 만든다.
9. 케일 앞에 밥을 얹고 소스를 곁들인다.

정제결과

5.2.4 데이터베이스 Insert 과정

```
mydb = pymysql.connect(
    host="localhost",
    user="root",
    passwd="*****",
    db="food",
    charset='utf8mb4',
)

try:
    with mydb.cursor() as cursor:
        sql = "INSERT INTO recipe (menu, Ingredient, direction) " \
              "VALUES (%s, %s, %s)" \
              "SELECT * FROM recipe"
        for j in range(0, num):
            val = (FoodIDDF.ix[j, "RCP_NM"], FoodIDDF.ix[j, "RCP_PARTS_DTLS"], mydict[j])
            cursor.execute(sql, val)
        mydb.commit()

finally:
    mydb.close()
```

5.2.5 데이터 삽입 결과

상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

Query 1 SQL File 3* SQL File 4*

```

1 • use food;
2 • select * from recipe;

```

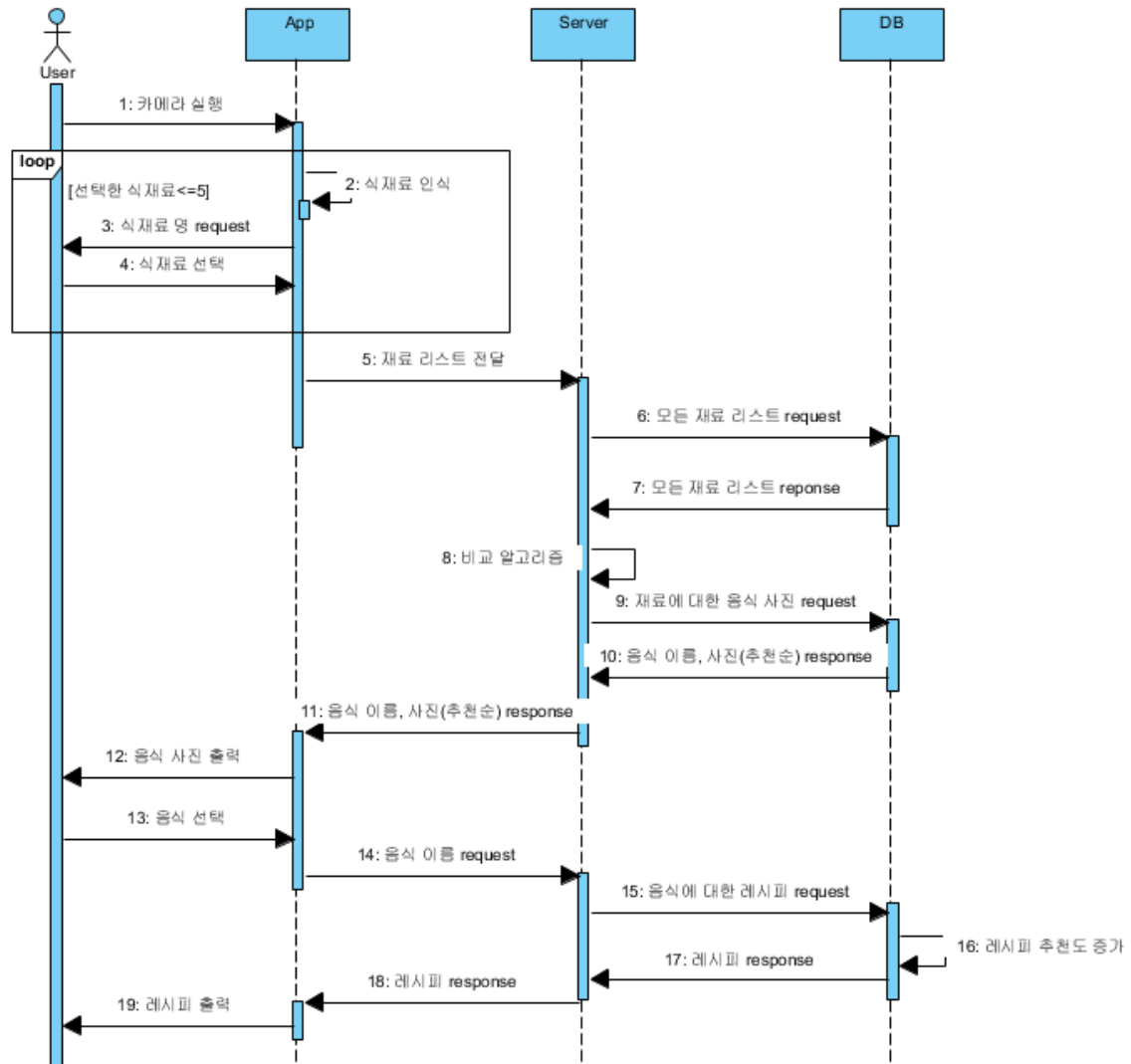
menu	ingredient	direction
▶ 누룽지 두부 저탄죽	제스준비 애호박 30g(1/6개), 표고버섯 20g(2개), 당근 5g(3*2*1cm) 누룽지...	1. 깨끗이 씻어 손질한 애호박, 당근과 기둥을 편 표고버섯을 잘게 다지듯이 썰는다. 2. 누룽지는 1cm 정도로 잘게 부숴준다. 3. 냄비에 참기름을 두르고 달여놓...
단호박 케일밥	단호박찜밥 찜밥 100g, 쌀 30g(3개), 은행 20g(8알), 잣 10g(10알), 단호박...	1. 찜밥을 깨끗이 씻어 물에 30분 정도 불린다. 2. 대추는 씨를 제거하고 물려와서로 자를 잘고, 밥은 겉집을 벗기고 4등분한다. 3. 잣은 꼭지를 떼고, 은행은...
두부 근약 나물 비빔밥	두부 근약잡곡밥 두부 110g(1/4개), 완두콩 15g, 현미밥 3g, 잡곡 3g, 실근약...	1. 고사리는 상태에 따라 2~5시간 정도 불린 후 30분 정도 삶아 찬물에 헹구어 물기를 쏙 빼준다. 2. 콩나물은 다듬어서 끓는 물에 데친 후 건져 식힌다. 3. 불린 표...
새우 두부 계란찜	새우두부계란찜 연두부 75g(3/4개), 약태살새우 20g(5마리), 달걀 30g(1/2...	1. 손질된 새우를 끓는 물에 데쳐 건진다. 2. 연두부, 달걀, 상크림, 설탕에 녹인 우유를 섞어 냄비에 넣고 간 뒤 새우(1)를 함께 섞어 그릇에 담는다. 3. 시금치...
오색영양밥 롤	영양밥 찰밥 100g, 흑미30g, 바지락 50g(16개), 강낭콩 15g(1큰술), 완두...	1. 바지락은 소금물에 20~30분정도 담근 후, 2~3번 헹구어 해감하고 바지락이 살짝 벌어질 때까지 삶는다. 2. 불린 찰밥과 흑미를 섞고 바지락 삶은 물을 넣...
오색리사시 스키	초밥 밥 210g(1공기) 배합초 식초 20g(1/4큰술), 설탕 10g(2작은술), 소금...	1. 배합초는 중탕에서 설탕이 녹을 때까지 저어가면서 녹인다. 2. 뜨거운 밥에 배합초를 넣어서 밥알이 으개지지 않게 고루 저어가면서 밥을 채운 정도로 식...
저염 간장을 이용한 닭가장 비빔밥	흑미밥 찰 90g, 강은 쌀 10g 닭가슴살 삶기 닭가슴살 40g(1/3개), 황계수...	1. 삶은 30분 정도 불린 후 물을 1:1로 하여 밥을 짓는다. 2. 숙주는 끓는 소금물에 넣어 삶아 죽으면 찬물에 헹구어 물기를 빼준다. 3. 고사리는 상태에 따라 2~...
취나물 비빔밥 & 청국장 찜장	취나물밥 밥 210g(1공기), 다진 쇠고기 20g(1/4큰술), 취나물 20g(8줄기), ...	1. 끓는 소금물에 취나물을 데쳐 찬물에 헹구어 물기를 짜고 1.5cm 정도로 잘라 참깨와 참깨에 무친다. 2. 쇠고기를 곱게 다지고 다진 양파와 마늘, 간장, ...
탈출 뚝배기 구구마	구구마죽 구구마 100g(2/3개), 설탕 2g(1/3작은술), 달걀가루 3g(2/3작은...	1. 구구마는 깨끗이 씻어서 겉집을 벗기고 4cm 정도로 잘라준다. 2. 찜기에 구구마를 넣고 20~30분 정도 삶아 주고, 불면다나 채를 이용하여 잘 으개어 곱게...
해물 채소장을 넣은 굴회 밥밥	잡곡밥 잡곡밥 140g(2/3공기) 굴회밥 굴회 30g, 소금 약간 해물채소장 조...	1. 잡곡을 깨끗하게 씻고 물에 30분 정도 불려 물을 1:1로 하여 밥이 끓으면 약한 불로 끓이고 물을 들여 고슬고슬한 밥을 지어 한입 크기로 뭉쳐 놓는다. 2. ...

6.가상 시나리오

6.1 시나리오

민기는 학교 앞에서 혼자 지내고 있는 자취생이다. 그의 냉장고에는 넣어둔지 오래된 재료들이 가득 쌓여있다. 그는 이 재료들을 이용해서 할 수 있는 요리가 없을까 고민했다. 그래서 "냉장고를 부탁해" 어플리케이션을 이용하기로 했다. 그의 냉장고에는 재료들이 많이 있었지만 어떻게 사용해야할지 잘 모르기 때문에, 사진의 객체 인식을 통해 재료를 판별하는 "냉장고를 부탁해"가 사용하기 매우 적합하였다. 그는 먼저 어플리케이션에 로그인을 하였다. 그리고 촬영을 위해서 어플리케이션의 카메라 접근을 허용하였다. 이 후, 스마트폰을 이용해 냉장고 속 재료를 촬영하였다. "냉장고를 부탁해"는 객체 인식을 통해 민기의 냉장고 재료를 인식하고, 그에 맞는 레시피를 추천하였다. 민기는 그 중에서 가장 마음에 드는 "돼지고기 김치찌개"를 선택해서, 냉장고 속 재료를 이용해 요리하였다. 그리고 "계란"을 추가로 구매하면 "장조림"을 할 수 있다는 어플리케이션의 추천을 받는 민기는, 화면 아래에 있는 링크를 통해 "계란"을 즉시에서 구매할 수 있었다.

6.2 Sequence 다이어그램





상세 설계서: 머신러닝을 활용한 객체인식 레시피 추천 '냉장고를 부탁해' 서비스

7. 프로젝트 일정

구분	구현 내용		4월				5월				6월			
			1주	2주	3주	4주	1주	2주	3주	4주	1주	2주	3주	4주
계획	아이디어 구상 및 기초 설계													
사전준비	데이터 수집 및 정제	데이터 수집												
		데이터 정제												
설계	DB설계 및 서버 구축	DB설계												
		서버 구 축												
개발	어플리케이션 개발	UI제작												
		프로그램 연동												
테스트	서버 연동 및 테스트	서버 연 동 및 프 로그램 디버깅												
종료	작품전시회 준비 및 최종보고서 작성													