**종합설계 프로젝트 수행 보고서**

|  |  |
| --- | --- |
| **프로젝트명** | **딥 러닝을 사용한**  **학교 시설 실시간 모니터링 시스템** |
| **팀 번호** | **S2-12** |
| **문서제목** | **수행계획서()**  **2차발표 중간보고서()**  **3차발표 중간보고서( O )**  **최종결과보고서( )** |

**2019.2.24**

**팀원: 진성민 (팀장)**

**심영효**

**석철범**

**짠발람**

**지도교수 : 교수 (인)**

**지도교수 : 교수 (인)**

**목 차**

**Ⅰ. 서론** ----------------------------------------------------------------------------------**3**

1. 작품선정 배경 및 필요성 --------------------------------------------------------------**3**

2. 기존 연구/기술동향 분석 --------------------------------------------------------------**5**

3. 개발 목표 ----------------------------------------------------------------------------**6**

4. 팀 역할 분담 -------------------------------------------------------------------------**7**

5. 개발 일정 ----------------------------------------------------------------------------**8**

6. 개발 환경 ----------------------------------------------------------------------------**8**

**Ⅱ. 본론** ----------------------------------------------------------------------------------**9**

1. 개발 내용 ----------------------------------------------------------------------------**9**

2. 문제 및 해결방안 --------------------------------------------------------------------**10**

3. 시험시나리오 ------------------------------------------------------------------------**11**

4. 상세 설계 ---------------------------------------------------------------------------**16**

5. Proto Type 구현 ---------------------------------------------------------------------**88**

**Ⅰ. 서론**

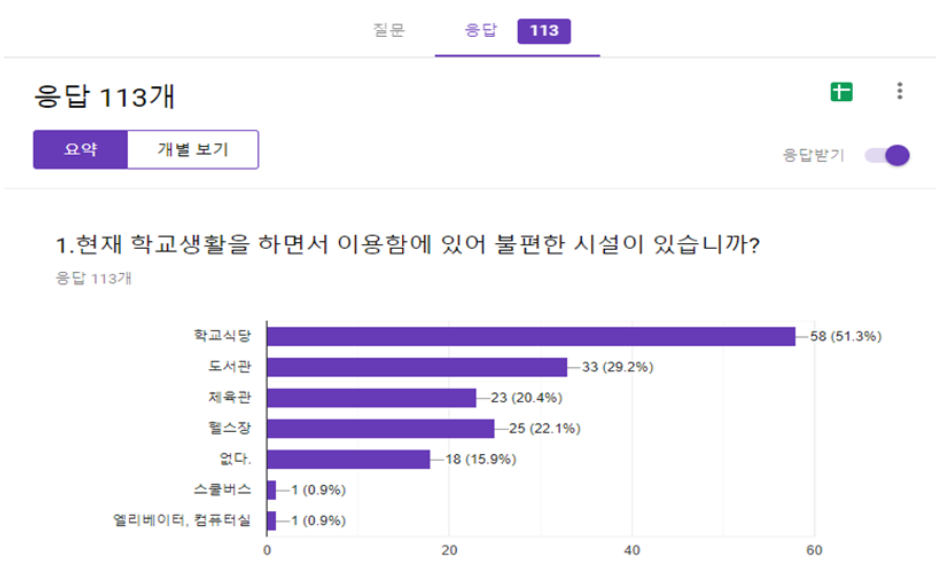
1. 작품선정 배경 및 필요성

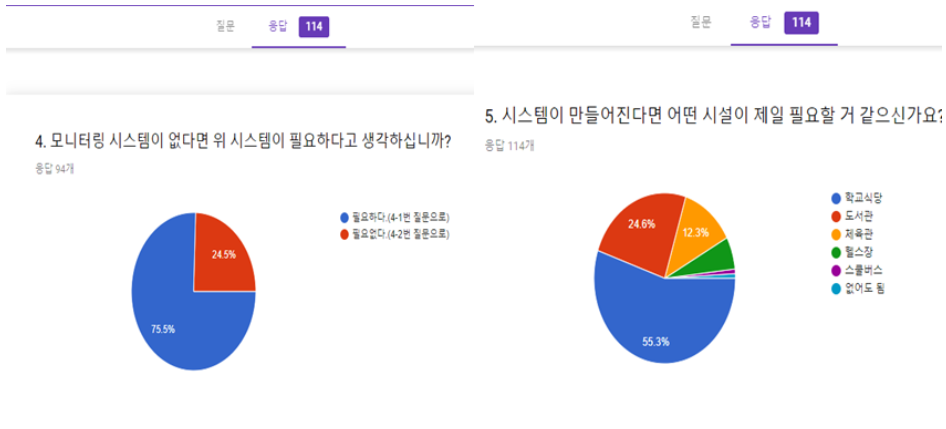
* 1. 작품선정 배경

|  |
| --- |
| KPU 학생, 교직원을 위한 학교 시설 시스템 시나리오  정보통신 서비스의 발전과 학생, 교직원이 다양하게 알고 싶은 학교 시설과 학교 활동 전반에 대하여 쉽고, 알고 싶은 욕망에 대하여 많은 변화를 맞이하여야 한다.   1. 신입생, 신입교직원의 경우 학교 지리를 잘 모르는 경우가 많다. 2. 학교시설 별로 인원을 실시간으로 파악할 수 없다. 3. 학교 시설을 이용하는데 걸리는 대기시간을 예측할 수 없다. 4. 위 문제를 해결할 수 있는 서비스가 없다. |

|  |  |
| --- | --- |
| **번호** | **핵심문제 타겟** |
| 1 | 학교 시설의 지리 문제 |
| 2 | 인원 실시간 파악 |
| 3 | 대기시간 예측 |
| 4 | 서비스 |

* 1. 필요성





통계조사1,2를 통해 제일 표본이 많은 학생 식당(tip, e동, 산업 융합관, 종합관)을 실시간으로 모니터링 할 수 있는 시스템을 만들기로 결정.

2. 기존 연구/기술동향 분석

|  |  |
| --- | --- |
| **기술** | **적용법** |
| 딥러닝 | - 딥러닝 연구에서 강조되는 데이터의 중요성을 인식하고 데이터의 노이즈를 제거하고 추가하는 다양한 영상처리 방법이 딥 러닝을 이용한 의학영상 분석의 성능 향상에 영향이 있는지에 대해 비교, 분석한 논문.  **본프로그램은 사람의 분포도를 영상 처리하는 과정에서 밝기와 명암을 뚜렷하게 만들 방법 중 Histogram equalization과 CLAHE을 이용할 것이고 영상의 화질 개선을 위해서 Wavelet transformation을 사용할 것이다.** |
|
|
|
|
|

|  |  |
| --- | --- |
| **기술** | **적용법** |
| 대기시간  예측 | - 환자의 대기시간을 실시간으로 모니터링하기 위한 오픈소스 기반의 시스템  - 빅데이터를 처리하기 위한 오픈 소스 시스템으로 다양한 하둡 서브프로젝트들로 구성된 하둡 에코시스템 활용.  - 아파치 스파크, 아파치 카프카 등을 이용한 시스템 구조 설계 및 상세 모듈 설계  - 분산 스트림 처리를 통해 고 성능과 안정성을 확보할 수 있다.  **본 시스템은 firebase 실시간 데이터베이스를 참조하여 시간 분석 예측 방법 고안하였음. 초기 임의의 시간 값은 베이지안 확률을 활용하여 구매자가 결제할 때 발생하는 음식, 개수, 결제시간, 조리시간 기반으로 구한다.** |
|
|
|
|
|

3. 개발 목표

3.1 딥 러닝을 활용한 현장 인원 분포도 예측

- yolo 딥 러닝을 이용한 실시간 모니터링 프로그램 개발

- Python을 활용하여 딥 러닝의 결과물을 실시간 그래프로 출력

3.2 실시간 대기시간 예측

- Test stub(주문결제시스템)으로부터 저장된 데이터베이스의 데이터들을 참조하여 머신 러닝 실행

- 베이지안 확률 기반으로 한 현재 대기 시간 예측

- 결과물을 통해 다른 음식점을 추천해주는 기능 추가

3.3 모바일 어플리케이션 개발

- 모바일 App 개발을 통해 3.1/3.2 기능들을 연동하여 사용자들의 지리적 번거로움 해소

- KPU Map을 제작하여 초보자에게 도움을 제공

- Chat bot 기능을 이식하여 도움말 서비스 제공

3.4 Chat bot

- Dialog Flow API를 활용하여 닫힌 대화식의 간단한 Chat bot을 구현하여 누구나 쉽게 모바일을

이용할 수 있도록 모바일 앱에 이식

- 기본 응답의 실패 시, 대체 응답으로 전환

4. 팀 역할 분담

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **팀원\_1 진성민** | **팀원\_2 심영효** | **팀원\_3 석철범** | **팀원\_3 짠발람** |
| 자료수집 | Yolo 딥러닝 자료 수집, 실시간 영상처리 수집, Raspberry PI 자료 수집, Firebase 자료 수집 | | | |
| 설 계 | DB설계(TestStub1~2, 딥러닝) | DB설계(TestStub3~4, 딥러닝) | - | - |
| Application설계(Dialog Flow) | - | Application설계(Dialog Flow, Web과 Raspberry PI 연동) | Application설계(Web과 Raspberry PI 연동) |
| OpenCV설계(yolo딥러닝, 설계) | Web설계(node.js) | OpenCV설계(Raspberry PI 연동) | Web설계(node.js) |
| - | Firebase설계(server, database연동) | - | Firebase설계(server, database연동) |
| Server설계(사용자) | Server설계(식당1~2) | Server설계(식당3~4) | Server설계(관리자) |
| 구현 | DB구현(TestStub3~4, 딥러닝) | DB구현(TestStub1~2, 딥러닝) | Application구현(Raspberry PI 연동 기능) | Application구현(Web 자체의 연동 기능) |
| OpenCV구현(Raspberry PI 연동) | Application구현(Dialog Flow) | OpenCV구현(yolo딥러닝) | Application구현(Dialog Flow) |
| Web구현(node.js) | Firebase구현(server, database연동) | Web구현(node.js) | Firebase구현(server, database연동) |
| Server구현(관리자) | Server구현(식당1~2) | Server구현(식당3~4) | Server구현(사용자) |
| 테스트 | v 장비 작동/제어 테스트  v 연동테스트  v 통합테스트  v 디버깅 | | | |

5. 개발 일정



6. 개발 환경

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **기술** | | | | | **Physical Element** |
| 코딩 | 네트워크 | WEB | DB | APP | Raspberry Pi B+ 3 |
| Java  Python | 리눅스 | Node.js | Firebase | Android | OpenCV,  Yolo Deep Learning |

**Ⅱ. 본론**

1. 개발 내용

1.1 딥 러닝을 활용한 현장 인원 분포도 예측

- OpenCV에서 PI 카메라를 사용하여 cv2 라이브러리를 import하여 Raspberry PI로 데이터를 가져온 후, 다시 Raspberry PI 내에서 yolo deep learning을 실행한다. 그 후, Raspberry PI 서버를 통해 해당 내용을 현장인원 분포도 DB에 저장한다.

- YOLO v3의 경우, PuTTY를 이용한 SSH를 통해 Raspberry PI에 설치한다.

- Python을 통해 Firebase에서 현장인원 분포도 DB의 자료를 가져와 Display에 출력할 경우, matplot 라이브러리를 설치하여 DB에 저장된 데이터를 그래프로 변환하여 결과를 출력하며, 별개로 모바일, 관리자 PC에서도 텍스트 형식으로 결과물을 출력한다.

\* Raspberry PI에서의 해당 내용 설정 방법

노트북에서 PuTTY를 이용하여 Raspberry PI와 노트북을 연결한다.

1.2 실시간 대기시간 예측

- 식당 DB의 실시간 주문 테이블의 데이터를 가져와 Raspberry PI에서 Python을 활용하여 베이지안 확률을 기반으로 머신 러닝을 실행한다.

- 대기시간 DB의 데이터를 통해 모바일 App에서 다른 음식점을 추천해주는 알고리즘을 구현한 후 해당 기능을 제공한다.

1.3 모바일 어플리케이션 개발

- 1.2의 두번째 내용에서 나온 음식점 추천, 교내 건물들의 위치와 그 내부의 식당들의 세부적인 위치(층 수, 건물 내부의 계단 등을 자세하게 표시)와 판매하는 메뉴 등을 적은 KPU맵 제공, 1.1의 기능인 실시간 인원 분포 등의 기능을 모바일 App를 통해 구현한다.

- 별도의 회원정보 DB를 운용하여 로그인 기능을 구현한다.

- 인원 분포도 예측 기능의 경우, 분포도 DB에 저장되어 있는 데이터를 사용자 서버를 통해 가져와, 다시 별도의 알고리즘을 제작한 후 실행 및 가공하여 출력한다.

1.4 Chat bot

- Dialog Flow를 활용하여 닫힌 대화식의 간단한 Chat bot 구현한다.

- Chat bot DB에 키워드에 대한 응답을 저장하고, 별도로 답변에 대한 성공, 실패 여부를 다른 테이블에 저장한다.

- 사용자의 만족도 조사를 통해 성공, 실패를 판별한다.

- 특정 기간마다, 실패 횟수가 많은 응답의 경우, 만족도 조사에서 올바른 응답이라고 제출된 데이터 중 비율이 높은 것으로 자동 업데이트 후, 응답하도록 재 설정한다.

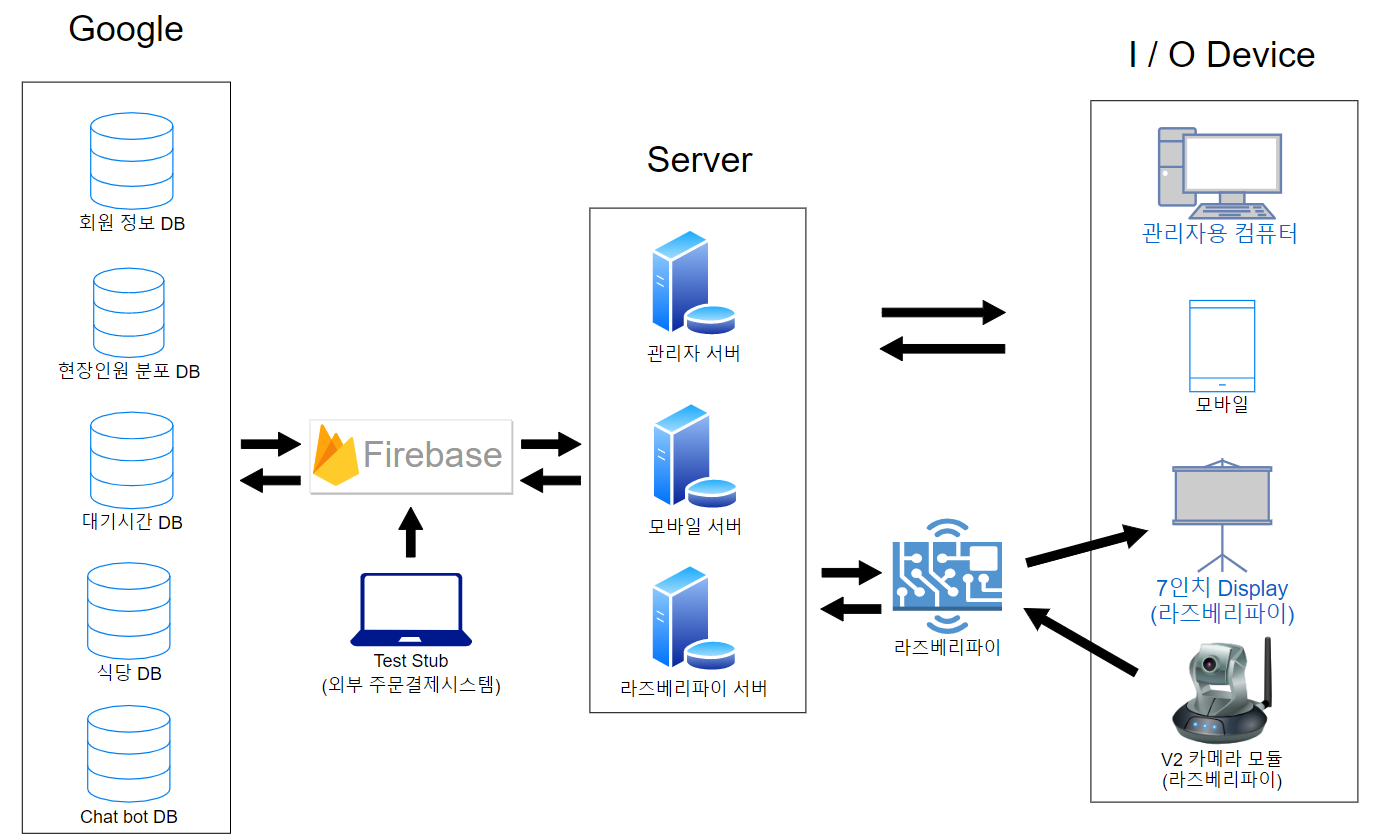
1. 문제 및 해결방안

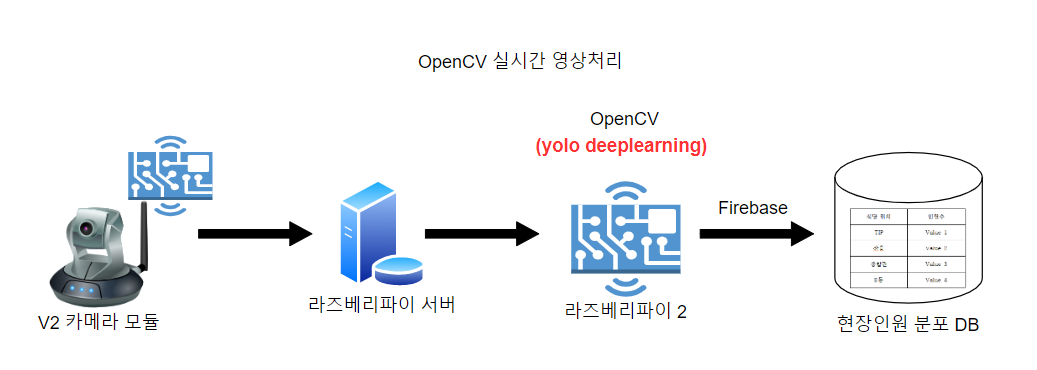
|  |  |  |  |
| --- | --- | --- | --- |
| **문제** | **How** | **기능** | **기대효과** |
| **식당에서 인원분포를 명확히 파악하기 힘들다.** | OpenCV와 Yolo deep learning을 사용하여 실시간으로 영상처리 | - 실시간 인원 분포 확인기능  - 현재 주문 현황 확인기능 | 실시간으로 식당의 인원 및 주문현황을 확인 가능 |
| **식당에서 대기시간을 예측할 수 없다.** | 베이지안 알고리즘을 이용하여 실시간으로 대기시간, 현 인원을 예측 | - 대기시간 예측기능  - 음식점 추천 기능 | 주문한 음식에 대한 대기 시간 예측 가능 |
| **신입생, 신입교직원의 경우 학교 지리를 잘 모르는 경우가 많다.** | Google GPS API 이용 | - kpu 맵 제공 및 표시 기능 | 신입생의 학교 지리 파악 가능­ |
| **초보자가 어플을 사용할 경우 어떠한 기능이 있는지 파악하지 못할 수 있다.** | Dialog Flow 이용 | - Chat bot기능 | 어플의 사용성을 극대화 |

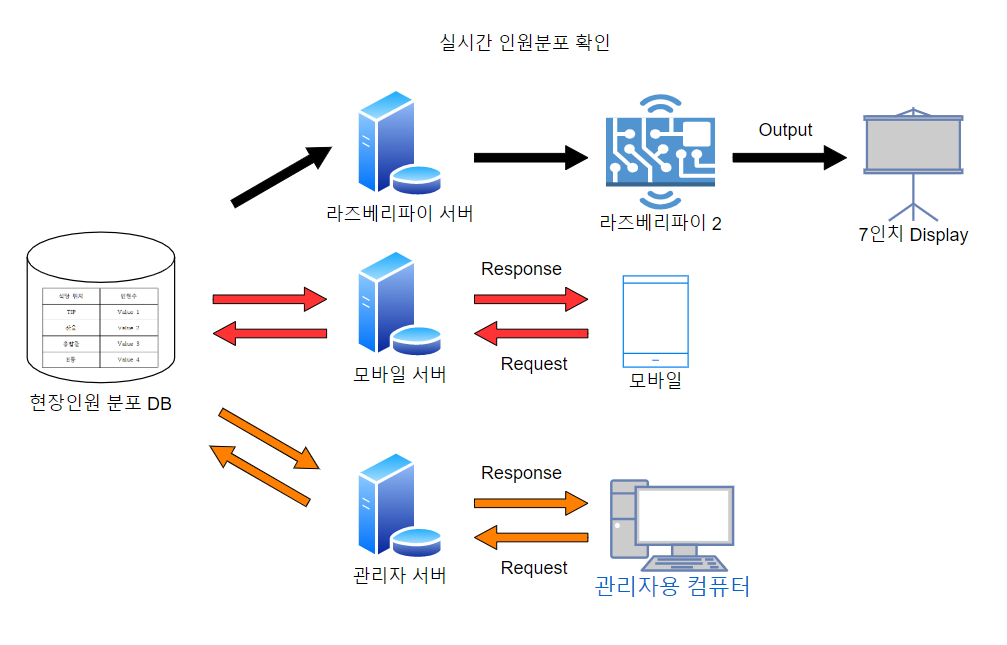
1. 시험시나리오

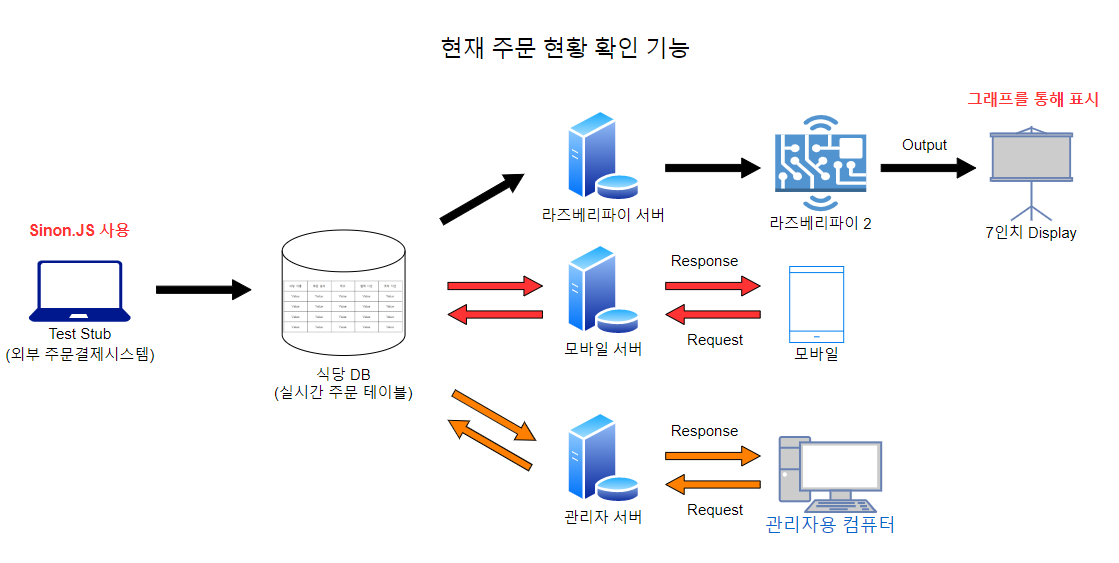
3.1 시스템 구성도와 흐름

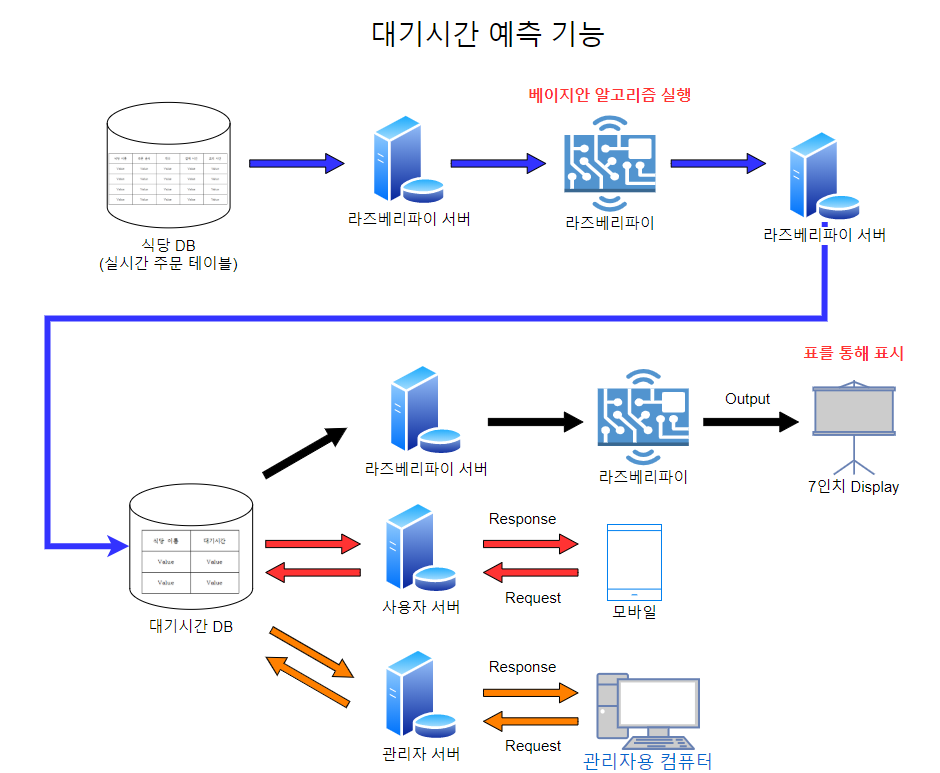
**시스템 전체 구성도**

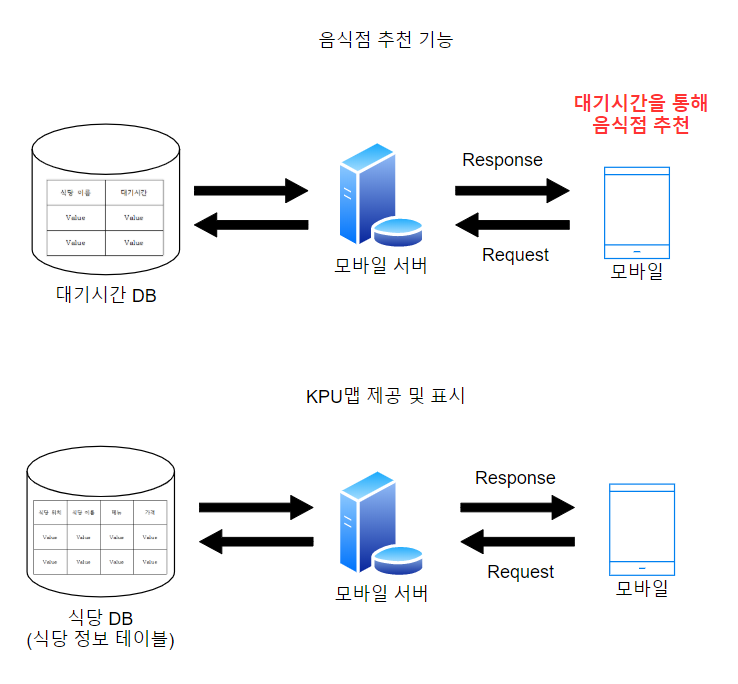


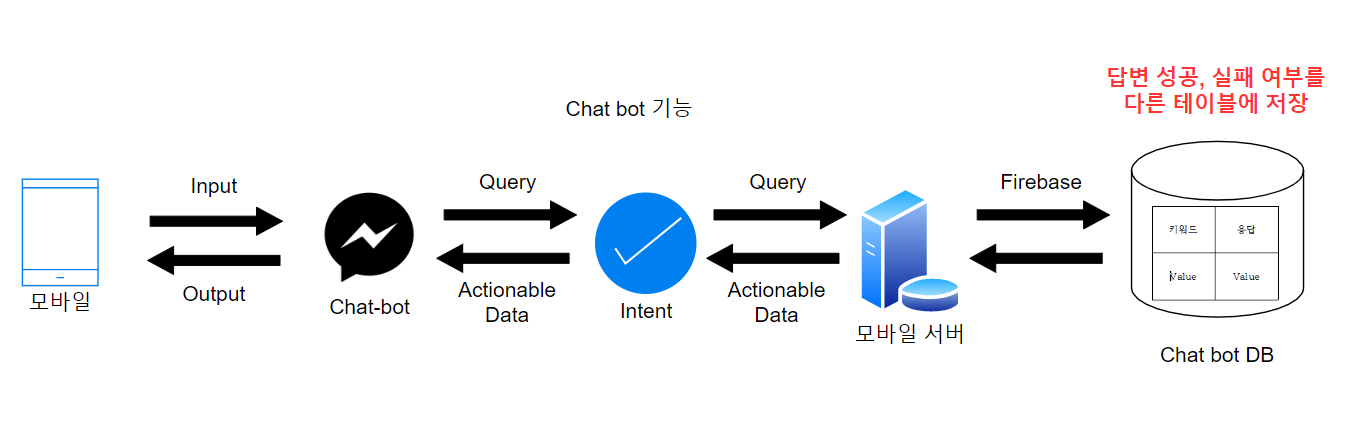






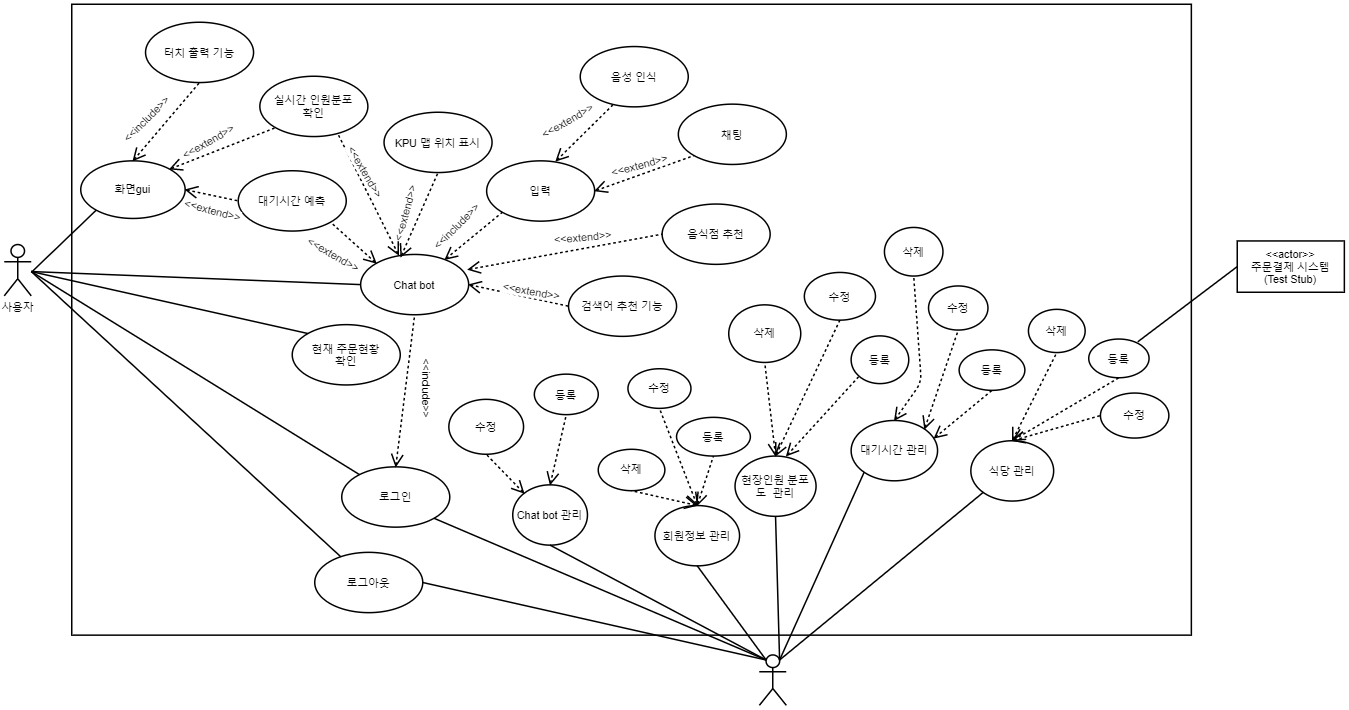






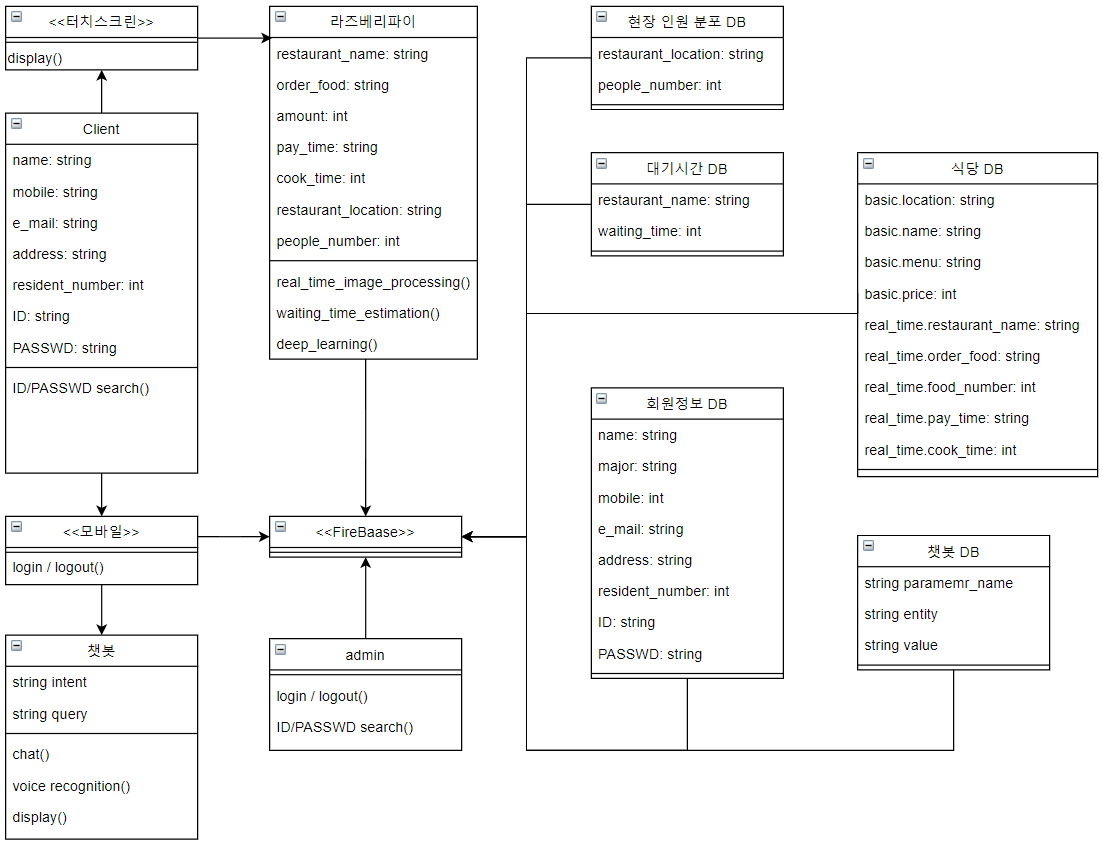
3.2 다이어그램

**유스케이스 다이어그램**



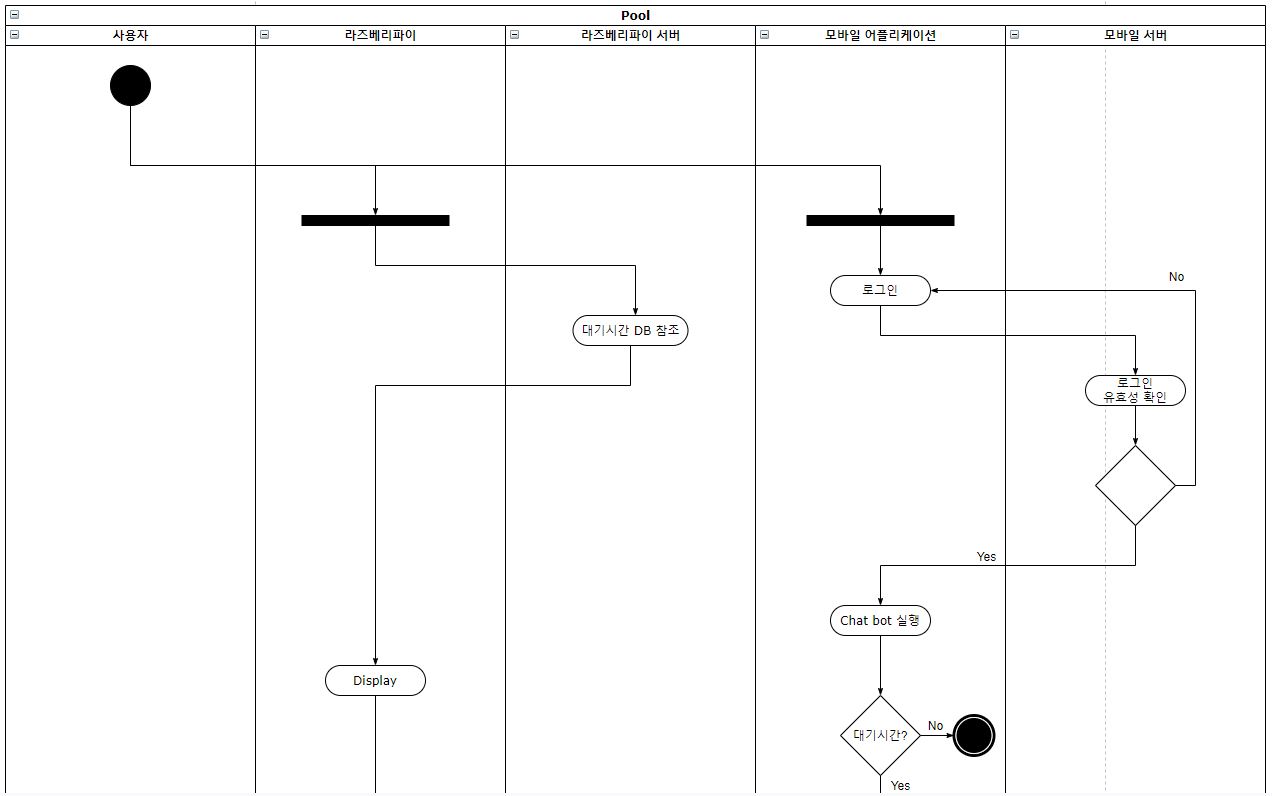
1. 상세 설계

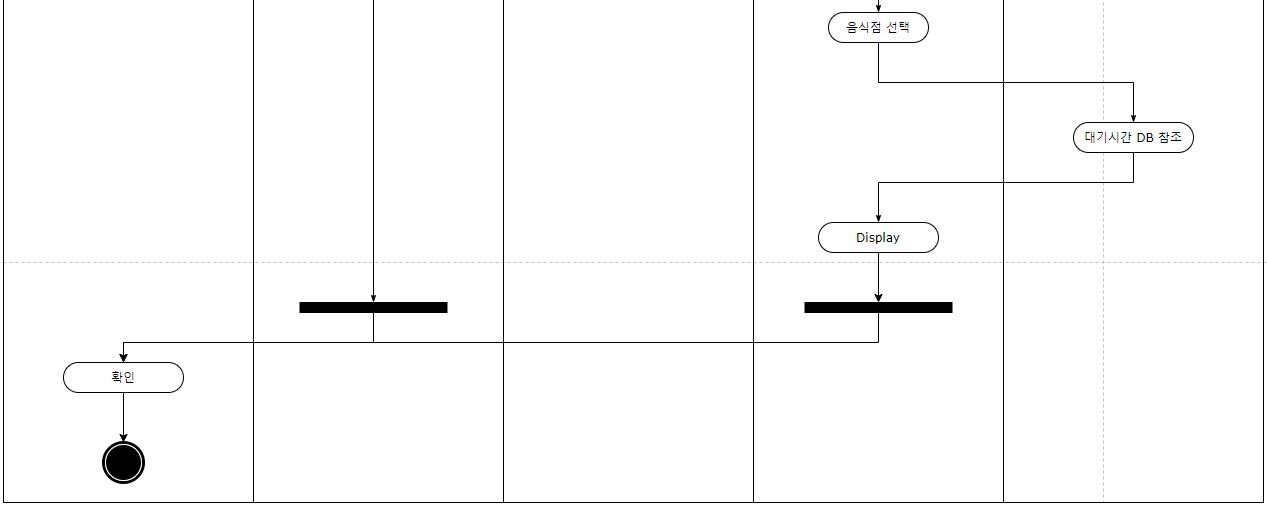
4.1 클래스 다이어그램



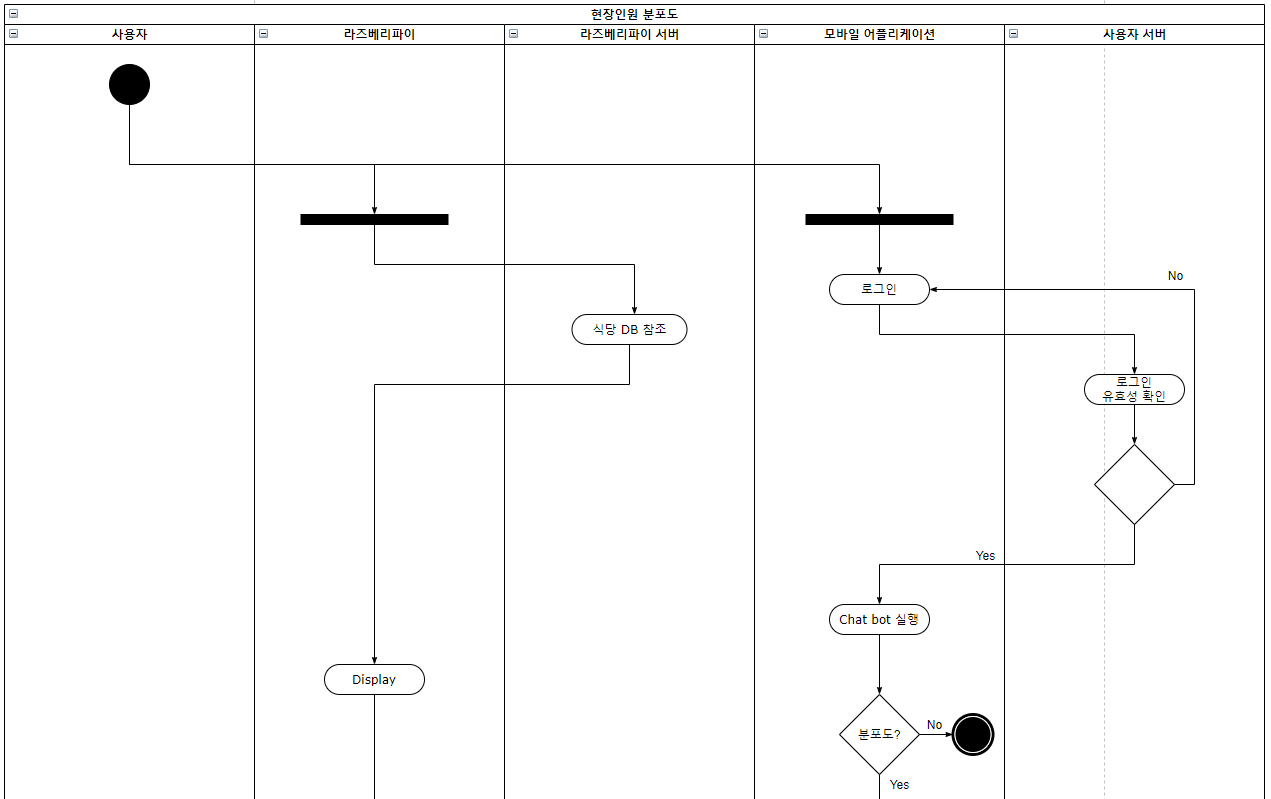
4.2 활동 다이어그램

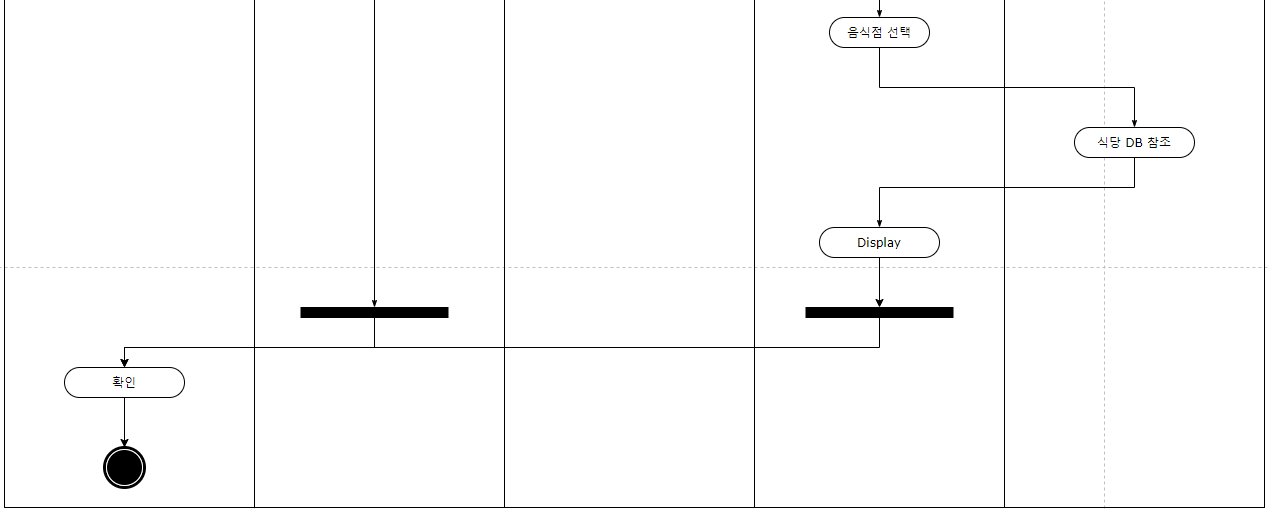
**대기 시간 예측 활동 다이어그램**





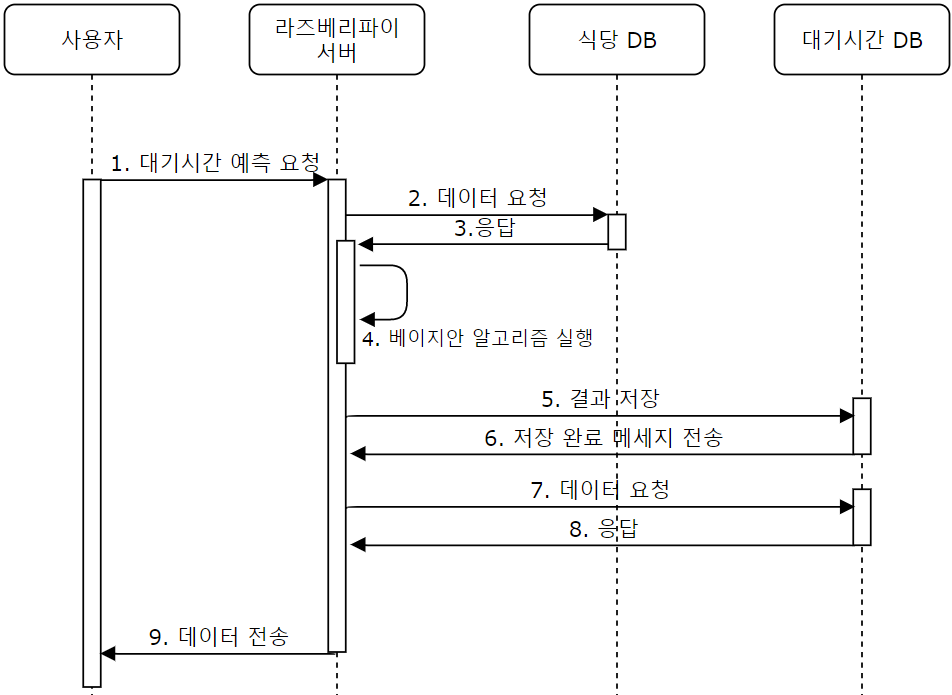
**현재 인원 분포도 활동 다이어그램**



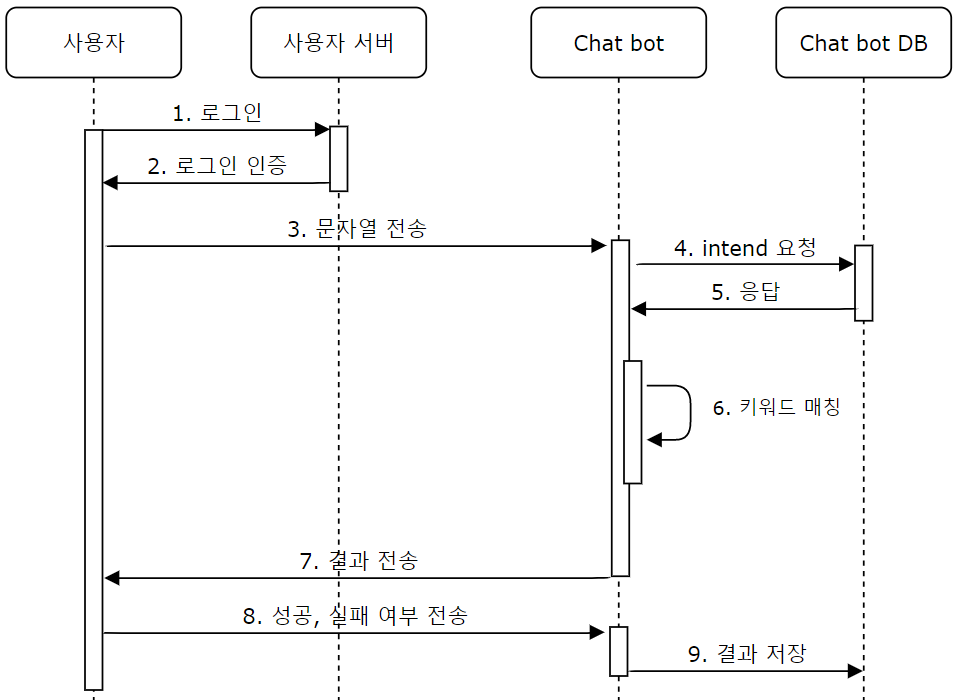


4.3 순차 다이어그램

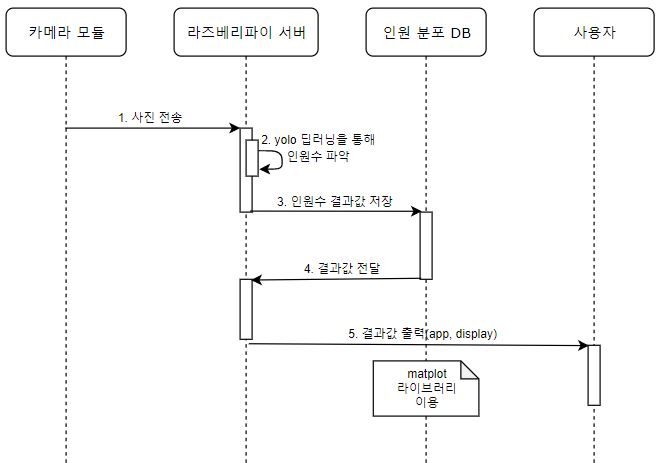
**대기 시간 예측 순차 다이어그램**



**Chat bot 순차 다이어그램**



**실시간 인원 분포 확인 다이어그램**

****

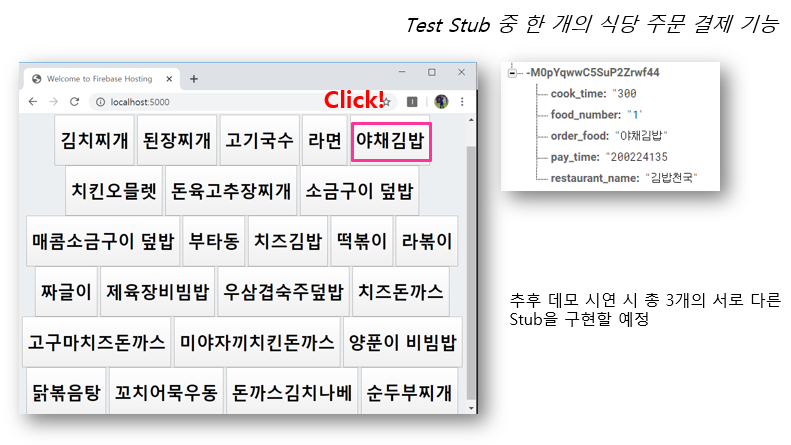
4.4 모듈 상세 설계

**관리자 웹 어플리케이션**



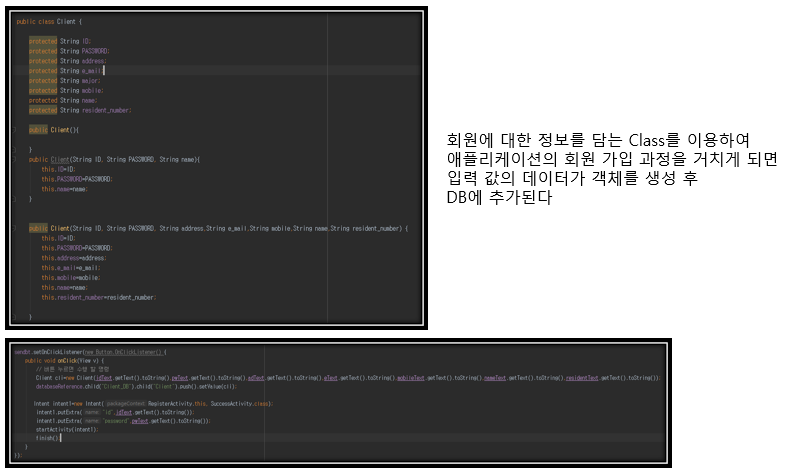


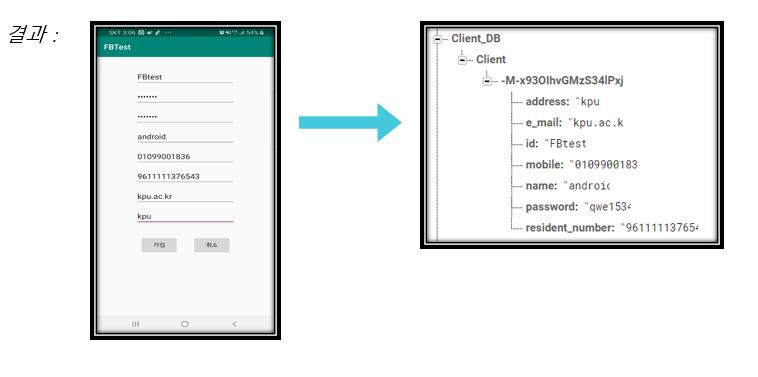
**테스트 스텁**



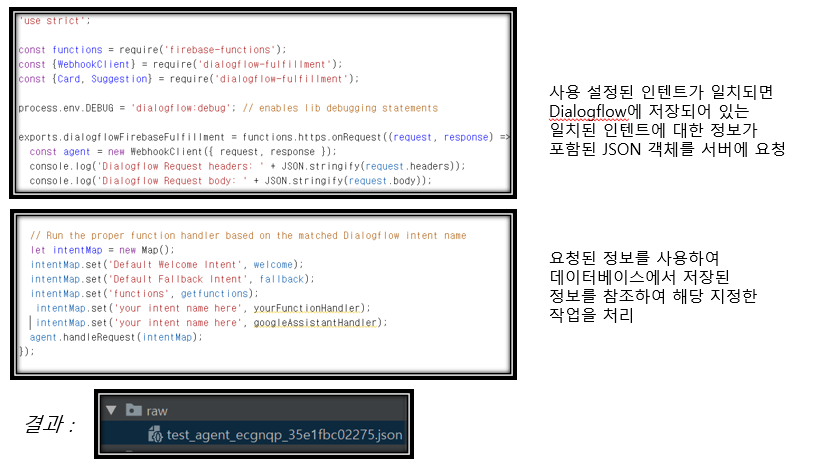
**모바일 어플리케이션**





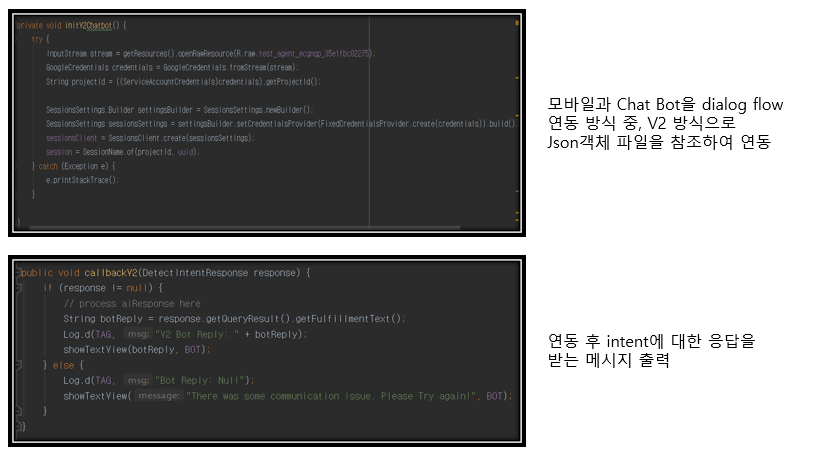


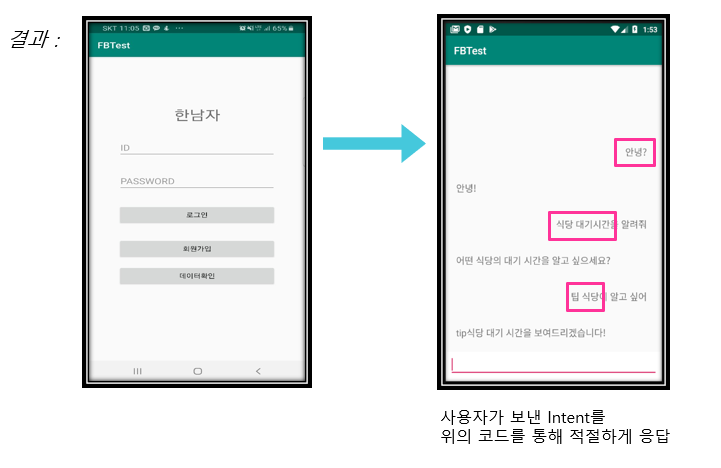
**Chat bot**



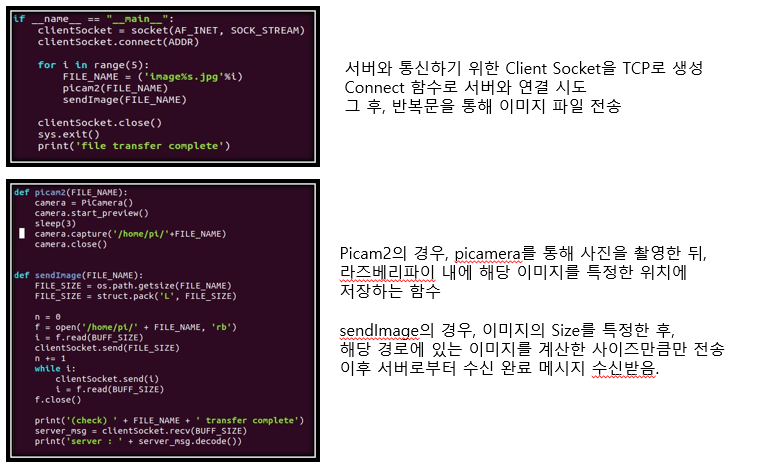


**모바일 & Chat bot 연동**

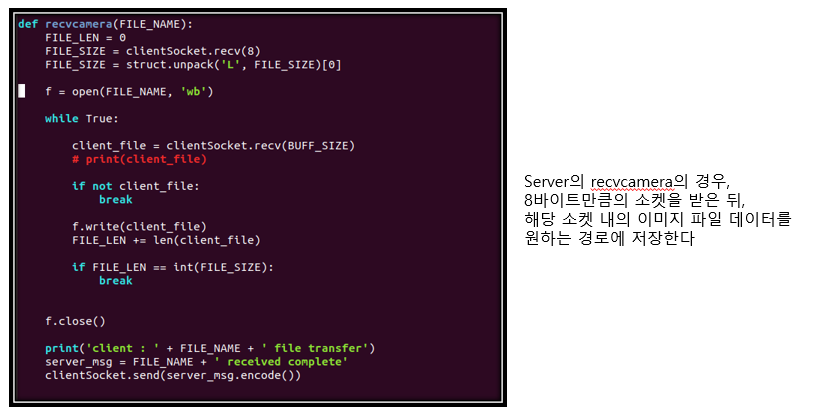


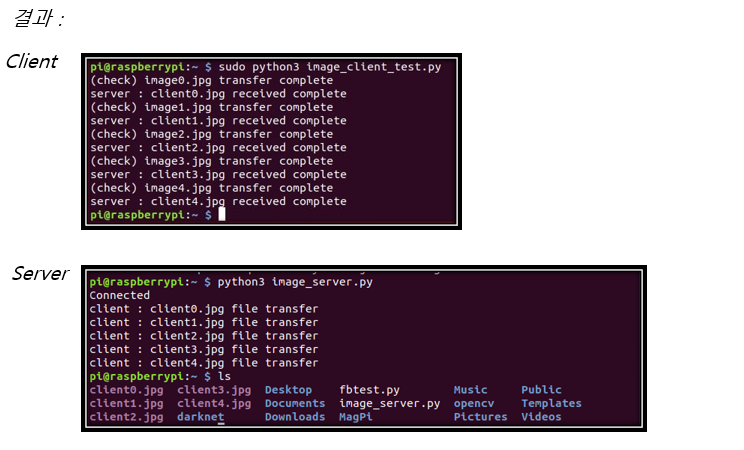


**라즈베리파이(Client)**



**라즈베리파이(Server)**





4.5 앱 스토리보드

**History**

|  |  |  |
| --- | --- | --- |
| **버전** | **변경날짜** | **변경사유** |
| 1.0.0 | 2020-02-25 | 신규 작성 |
| 1.0.1 | 2020-04-13 | 보기 편한 로그인 기능으로 변경 |
| 1.0.2 | 2020-04-14 | 회원가입 기능& 패스워드 찾기 추가 |
| 1.0.3 | 2020-04-15 | 로그아웃 기능 추가 |

**서비스 기능 목록**

-로그인 기능

-회원 가입 기능

-패스워드 찾기 기능

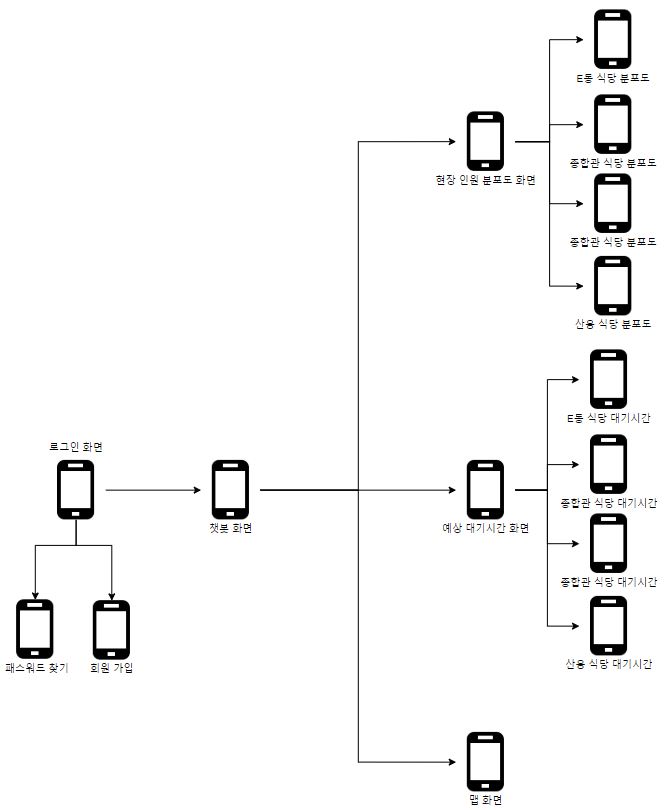
-챗봇 서비스 기능

-맵 제공 기능

-각 식당 별 예상 대기 시간 기능

-각 식당 별 현장인원분포도 기능

**화면흐름도**



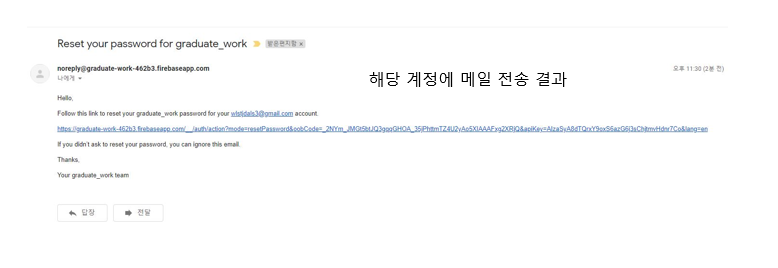
**스토리 보드**

- 로그인 화면

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  | 1. 어플리케이션 이름표시 2. 아이디, 비밀번호 입력 3. 회원가입 4. 비밀번호 찾기 5. 로그인 실패 시 에러 유형 출력 |

- 패스워드 찾기

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  | 1. 비밀번호를 찾기 위해 이메일 입력 2. 해당 이메일로 메일을 보내 비밀번호 재전송 3. 메일 전송 성공/실패 시 메시지 출력 |



- 회원가입

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  | 1. 이메일&비밀번호  입력 텍스트창 2. 회원 가입 완료 버튼  처리 3. 사용자의 잘못 클릭했을 경우를 위해 하단부에 로그인 화면 링크 |

- 로그아웃

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  |  |

- 챗봇 화면

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  | 1. 로그인 성공 시 chat bot 화면 이동 2. 로그아웃 버튼 (추후 navigation bar을 통해 수정) 3. 사용자의 입력 값 (인텐트)에 맞는  chatbot의 응답 |

- 현장 인원 분포도

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  |  |

- 예상 대기 시간

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  |  |

- 맵 화면

|  |  |  |
| --- | --- | --- |
| 화면 | 화면 상세 보기 | 설명 |
|  |  |  |

4.6 상세 설계 API

**안드로이드**

|  |  |
| --- | --- |
| Sign\_up() | |
| 분류 | 사용자 회원가입 |
| 기능 | Dialogflow chatbot 접속 권한 획득 |
| 리턴 값 | void |
| 소스코드 | private void userLogin(){  String email = editTextEmail.getText().toString().trim();  String password = editTextPassword.getText().toString().trim();  if(TextUtils.isEmpty(email)){  Toast.makeText(this, "email을 입력해 주세요.", Toast.LENGTH\_SHORT).show();  return;  }  if(TextUtils.isEmpty(password)){  Toast.makeText(this, "password를 입력해 주세요.", Toast.LENGTH\_SHORT).show();  return;  }  progressDialog.setMessage("로그인중입니다. 잠시 기다려 주세요...");  progressDialog.show();  //logging in the user  firebaseAuth.signInWithEmailAndPassword(email, password)  .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  @Override  public void onComplete(@NonNull Task<AuthResult> task) {  progressDialog.dismiss();  if(task.isSuccessful()) {  finish();  startActivity(new Intent(getApplicationContext(), ChatbotActivity.class));  } else {  Toast.makeText(getApplicationContext(), "로그인 실패!!", Toast.LENGTH\_LONG).show();  textviewMessage.setText("로그인 실패 유형\n - id와 papssword를 확인해주세요!!\n ");  }  }  });  } |
| 설명 | 실재 존재하는 계정 중에서(naver, google) 원하는 아이디와 별도의 Chat bot 로그인 전용 비밀번호를 입력 받아 Fire base 의 회원 DB에 해당 내용을 저장합니다. |

|  |  |
| --- | --- |
| Login() | |
| 분류 | 사용자 로그인 |
| 기능 | Dialogflow chatbot 접속 권한 획득 시도 |
| 리턴 값 | void |
| 소스코드 | private void registerUser(){  //사용자가 입력하는 email, password를 가져온다.  String email = editTextEmail.getText().toString().trim();  String password = editTextPassword.getText().toString().trim();  //email과 password가 비었는지 아닌지를 체크 한다.  if(TextUtils.isEmpty(email)){  Toast.makeText(this, "Email을 입력해 주세요.", Toast.LENGTH\_SHORT).show();  return;  }  if(TextUtils.isEmpty(password)){  Toast.makeText(this, "Password를 입력해 주세요.", Toast.LENGTH\_SHORT).show();  return;  }  //email과 password가 제대로 입력되어 있다면 계속 진행된다.  progressDialog.setMessage("등록중입니다. 기다려 주세요...");  progressDialog.show();  //creating a new user  firebaseAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  @Override  public void onComplete(@NonNull Task<AuthResult> task) {  if(task.isSuccessful()){  finish();  startActivity(new Intent(getApplicationContext(), ChatbotActivity.class));  } else {  //에러발생시  textviewMessage.setText("에러유형\n - 이미 등록된 이메일 \n -암호 최소 6자리 이상 \n - 서버에러");  Toast.makeText(RegisterActivity.this, "등록 에러!", Toast.LENGTH\_SHORT).show();  }  progressDialog.dismiss();  }  });  } |
| 설명 | 아이디와 비밀번호를 입력하면 Fire base의 DB에서 해당 내용과 일치하는 데이터가 있는지 확인한 후 존재한다면 사용자 권한을 승인한다. |

|  |  |
| --- | --- |
| Search\_Password() | |
| 분류 | 비밀번호 찾기 |
| 기능 | 특정 계정의 비밀번호 찾기 |
| 리턴 값 | void |
| 소스코드 | String emailAddress = editTextUserEmail.getText().toString().trim();  firebaseAuth.sendPasswordResetEmail(emailAddress)  .addOnCompleteListener(new OnCompleteListener<Void>() {  @Override  public void onComplete(@NonNull Task<Void> task) {  if(task.isSuccessful()){  Toast.makeText(FindActivity.this, "이메일을 보냈습니다.", Toast.LENGTH\_LONG).show();  finish();  startActivity(new Intent(getApplicationContext(), MainActivity.class));  } else {  Toast.makeText(FindActivity.this, "메일 보내기 실패!", Toast.LENGTH\_LONG).show();  }  progressDialog.dismiss();  }  }); |
| 설명 | 실재 존재하는 계정이므로 이메일을 입력 받아 해당 이메일로 Chat bot 계정 전용 비밀번호를 송신한다. |

|  |  |
| --- | --- |
| Logout() | |
| 분류 | 사용자 로그아웃 |
| 기능 | Dialogflow chatbot 접속 권한 반환 |
| 리턴 값 | void |
| 소스코드 | buttonLogout = (Button) findViewById(R.id.buttonLogout);  //initializing firebase authentication object  firebaseAuth = FirebaseAuth.getInstance();  //유저가 로그인 하지 않은 상태라면 null 상태이고 이 액티비티를 종료하고 로그인 액티비티를 연다.  if(firebaseAuth.getCurrentUser() == null) {  finish();  startActivity(new Intent(this, MainActivity.class));  }  //유저가 있다면, null이 아니면 계속 진행  FirebaseUser user = firebaseAuth.getCurrentUser();  //logout button event  buttonLogout.setOnClickListener(this);  //스크롤뷰  final ScrollView scrollview = findViewById(R.id.chatScrollView);  scrollview.post(() -> scrollview.fullScroll(ScrollView.FOCUS\_DOWN));  chatLayout = findViewById(R.id.chatLayout);  ImageView sendBtn = findViewById(R.id.sendBtn);  sendBtn.setOnClickListener(this::sendMessage);  queryEditText = findViewById(R.id.queryEditText);  queryEditText.setOnKeyListener((view, keyCode, event) -> {  if (event.getAction() == KeyEvent.ACTION\_DOWN) {  switch (keyCode) {  case KeyEvent.KEYCODE\_DPAD\_CENTER:  case KeyEvent.KEYCODE\_ENTER:  sendMessage(sendBtn);  return true;  default:  break;  }  }  return false;  });  public void onClick(View v) {  if (v == buttonLogout) {  firebaseAuth.signOut();  finish();  startActivity(new Intent(this,MainActivity.class));  }  } |
| 설명 | 유저가 이미 로그인 되어있는 상태일 경우, 사용자 권한을 해제하고 로그인 창으로 돌아간다. |

|  |  |
| --- | --- |
| initChatbot() | |
| 분류 | 챗봇 연동 모듈 / 함수 |
| 기능 | Dialogflow chatbot 연동 설정 및 초기화 |
| 리턴 값 | void |
| 소스코드 | final AIConfiguration config=  new AIConfiguration("4e77176662a94f89a6d6f7ba2b4b729a",  AIConfiguration.SupportedLanguages.Korean,AIConfiguration.RecognitionEngine.System);  aiDataService = new AIDataService(this,config);  customAIServiceContext = AIServiceContextBuilder.buildFromSessionId(uuid);  aiRequest= new AIRequest(); |
| 설명 | V1 패키징 방식으로 챗 봇 프로그램을 연동 및 default 상태로 초기화한다. |

|  |  |
| --- | --- |
| initV2Chatbot() | |
| 분류 | 챗봇 연동 모듈 / 함수 |
| 기능 | Dialogflow chatbot 연동 V2 방식 설정 및 초기화 |
| 리턴 값 | void |
| 소스코드 | try {  InputStream stream = getResources().openRawResource(R.raw.test\_agent\_ecgnqp\_35e1fbc02275);  GoogleCredentials credentials = GoogleCredentials.fromStream(stream);  String projectId = ((ServiceAccountCredentials)credentials).getProjectId();  SessionsSettings.Builder settingsBuilder = SessionsSettings.newBuilder();  SessionsSettings sessionsSettings =  settingsBuilder.setCredentialsProvider(FixedCredentialsProvider.create(credentials)).build();  sessionsClient = SessionsClient.create(sessionsSettings);  session = SessionName.of(projectId, uuid);  } catch (Exception e) {  e.printStackTrace();  } |
| 설명 | V2 패키징 방식으로 챗 봇 프로그램을 연동 및 default 상태로 초기화한다. |

|  |  |
| --- | --- |
| sendMessage(View view) | |
| 분류 | 쿼리 모듈 / 함수 |
| 기능 | chatbot에게 사용자의 인텐트 전송 |
| 리턴 값 | void |
| 소스코드 | String msg = queryEditText.getText().toString();  if (msg.trim().isEmpty()) {  Toast.makeText(ChatbotActivity.this, "Please enter your query!", Toast.LENGTH\_LONG).show();  } else {  showTextView(msg, USER);  queryEditText.setText("");  // Android client  // aiRequest.setQuery(msg);  //RequestTask requestTask = new RequestTask(MainActivity.this, aiDataService, customAIServiceContext);  // requestTask.execute(aiRequest);  // Java V2  QueryInput queryInput = QueryInput.newBuilder().setText(TextInput.newBuilder().  setText(msg).setLanguageCode("ko-KR")).build();  new RequestJavaV2Task(ChatbotActivity.this, session, sessionsClient, queryInput).execute();  } |
| 설명 | Chat bot 프로그램에게 메시지를 보낸다. |

|  |  |
| --- | --- |
| callback(AIResponse aiReponse) | |
| 분류 | 인텐트 응답 모듈 / 함수 |
| 기능 | 사용자의 인텐트에 대한 응답 |
| 리턴 값 | void |
| 소스코드 | if (aiResponse != null) {  // process aiResponse here  String botReply = aiResponse.getResult().getFulfillment().getSpeech();  Log.d(TAG, "Bot Reply: " + botReply);  showTextView(botReply, BOT);  } else {  Log.d(TAG, "Bot Reply: Null");  showTextView("There was some communication issue. Please Try again!", BOT);  } |
| 설명 | 보내진 메시지를 통해 사전에 정의해둔 response에 따라 출력할 메시지를 정한다. |

|  |  |
| --- | --- |
| callbackV2(AIResponse aiReponse) | |
| 분류 | 인텐트 응답 모듈 / 함수 |
| 기능 | Dialogflow와 연동하여 응답을 보여줄 V2방식 |
| 리턴 값 | void |
| 소스코드 | if (response != null) {  // process aiResponse here  String botReply = response.getQueryResult().getFulfillmentText();  Log.d(TAG, "V2 Bot Reply: " + botReply);  showTextView(botReply, BOT);  } else {  Log.d(TAG, "Bot Reply: Null");  showTextView("There was some communication issue. Please Try again!", BOT);  } |
| 설명 | V2 패키징 방식으로 보내진 메시지를 통해 사전에 정의해둔 response에 따라 출력할 메세지를 정한다. |

|  |  |
| --- | --- |
| showTextView(String message, int type) | |
| 분류 | 레이아웃 텍스트 설정 모듈 / 함수 |
| 기능 | 사용자는 EditText를 사용하여 Android 앱에서 쿼리를 보낼 수 있으며 응답은 TextView에 표시됩니다. |
| 리턴 값 | void |
| 소스코드 | FrameLayout layout;  switch (type) {  case USER:  layout = getUserLayout();  break;  case BOT:  layout = getBotLayout();  break;  default:  layout = getBotLayout();  break;  }  layout.setFocusableInTouchMode(true);  chatLayout.addView(layout); // move focus to text view to automatically make it scroll up if softfocus  TextView tv = layout.findViewById(R.id.chatMsg);  tv.setText(message);  layout.requestFocus();  queryEditText.requestFocus(); // change focus back to edit text to continue typing |
| 설명 | 사용자의 메시지에 대해 가져온 응답을 TextView를 통해 출력한다. |

|  |  |
| --- | --- |
| showWaitTime(String message, int type) | |
| 분류 | 레이아웃 텍스트 설정 모듈 / 함수 |
| 기능 | Firebase에 저장되어 있는 대기 시간을 불러와 표로 출력합니다. |
| 리턴 값 | void |
| 소스코드 | private FirebaseDatabase mDatabase;  private DatabaseReference mReference;  private ChildEventListener mChild;  TextView tv1;  TextView tv2;  TextView tv3;  TextView tv4;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_waiting\_time);  initDatabase();  tv1=(TextView)findViewById(R.id.Edong);  tv2=(TextView)findViewById(R.id.tip);  tv3=(TextView)findViewById(R.id.Olive);  tv4=(TextView)findViewById(R.id.sanyong);  mDatabase=FirebaseDatabase.getInstance();  mReference=mDatabase.getReference("Waiting\_time\_DB");  mReference.addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  for (DataSnapshot myData : dataSnapshot.getChildren()) { ////values에 데이터를 담는 과정  String time1= dataSnapshot.child("Edong").child("wait\_time").getValue(String.class);  String time2= dataSnapshot.child("Kpu").child("wait\_time").getValue(String.class);  String time3= dataSnapshot.child("Olive").child("wait\_time").getValue(String.class);  String time4= dataSnapshot.child("Sanyung").child("wait\_time").getValue(String.class);  Log.e(time1,"error");  Log.e(time2,"error");  Log.e(time3,"error");  Log.e(time4,"error");  tv1.setText(time1);  tv2.setText(time2);  tv3.setText(time3);  tv4.setText(time4);  }  }  @Override  public void onCancelled(@NonNull DatabaseError databaseError) {  }    });  }  private void initDatabase() {  mDatabase = FirebaseDatabase.getInstance();  mReference = mDatabase.getReference("Waiting\_time\_DB");  mChild = new ChildEventListener() {  @Override  public void onChildAdded(DataSnapshot dataSnapshot, String s) {  }  @Override  public void onChildChanged(DataSnapshot dataSnapshot, String s) {  }  @Override  public void onChildRemoved(DataSnapshot dataSnapshot) {  }  @Override  public void onChildMoved(DataSnapshot dataSnapshot, String s) {  }  @Override  public void onCancelled(DatabaseError databaseError) {  }  };  mReference.addChildEventListener(mChild);  } |
| 설명 | Firebase에 저장되어 있는 대기 시간을 불러와 표로 출력합니다. |

|  |  |
| --- | --- |
| getUserLayout() | |
| 분류 | 화면 출력 모듈 / 함수 |
| 기능 | 사용자에게 동적으로 실시간 레이아웃을 띄움 |
| 리턴 값 | FrameLayout 객체 |
| 소스코드 | LayoutInflater inflater = LayoutInflater.from(ChatbotActivity.this);  return (FrameLayout) inflater.inflate(R.layout.user\_msg\_layout, null); |
| 설명 | 사용자가 메시지를 입력할 때마다 동적으로 그에 반응한다. |

|  |  |
| --- | --- |
| doInBackground | |
| 분류 | 백그라운드 모듈 / 함수 |
| 기능 | 챗봇과의 통신을 비동기 방식으로 처리.  사용자의 요청 쿼리가 챗봇에 요청으로 전송되고 응답이 캡처되는 AsyncTask에서 수행 |
| 리턴 값 | DectectIntentResponse 객체 |
| 소스코드 | protected AIResponse doInBackground(AIRequest... aiRequests) {  final AIRequest request = aiRequests[0];  try {  return aiDataService.request(request, customAIServiceContext);  } catch (AIServiceException e) {  e.printStackTrace();  }  return null;  }  @Override  protected void onPostExecute(AIResponse aiResponse) {  ((ChatbotActivity)activity).callback(aiResponse);  } |
| 설명 | 구글에서 자체적으로 저장되어 있는 메시지에 따른 반응 리스트들을 비동기 방식으로 통신한다. |

|  |  |
| --- | --- |
| onPostExecute(DectectIntentResponse response) | |
| 분류 | 인텐트 감지 모듈/ 함수 |
| 기능 | 백그라운드 작업이 끝나면 호출이 되고 callbackV2()를 호출 |
| 리턴 값 | void |
| 소스코드 | protected void onPostExecute(AIResponse aiResponse) {  ((ChatbotActivity)activity).callback(aiResponse); |
| 설명 | callbackV2()를 호출한다. |

|  |  |
| --- | --- |
| Costume\_label\_class(DectectIntentResponse response) | |
| 분류 | 인텐트 감지 모듈/ 함수 |
| 기능 | 그래프의 x라벨(float형)을 스트링 값으로 변환 |
| 리턴 값 | void |
| 소스코드 | public class MyXAxisValueFormatter extends ValueFormatter {  private String[] mValues = new String[] {};  private int mValueCount = 0;  //`setValues`를 사용하여 축 레이블을 설정할 경우  public MyXAxisValueFormatter() {  }  // 특정 `setValues`를 사용하여 축 레이블을 설정할 경우  public MyXAxisValueFormatter(String[] values) {  if (values != null)  setValues(values);  }  //x축의 라벨을 지정하는 생성자  public MyXAxisValueFormatter(Collection<String> values) {  if (values != null)  setValues(values.toArray(new String[values.size()]));  }  @Override  public String getFormattedValue(float value) {  int index = Math.round(value);  if (index < 0 || index >= mValueCount || index != (int)value)  return "";  return mValues[index];  }  public String[] getValues()  {  return mValues;  }  public void setValues(String[] values)  {  if (values == null)  values = new String[] {};  this.mValues = values;  this.mValueCount = values.length;  }  } |
| 설명 | 플로트형인 그래프의 x라벨을 string형으로 변환한다. |

|  |  |
| --- | --- |
| Show\_graph(DectectIntentResponse response) | |
| 분류 | 화면 출력 모듈 / 함수 |
| 기능 | 특정 값들을 통한 그래프 출력 |
| 리턴 값 | void |
| 소스코드 | public class Distribution extends AppCompatActivity{  private FirebaseDatabase mDatabase;  private DatabaseReference mReference;  private ChildEventListener mChild;  LineChart lineChart; //Temp\_linechart  TextView tv;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_distribution);  initDatabase();  lineChart=(LineChart)findViewById(R.id.mpChart);  // tv=(TextView)findViewById(R.id.cntid);  mDatabase=FirebaseDatabase.getInstance();  mReference=mDatabase.getReference("Distribution\_DB");  // mReference.child("Kpu").addValueEventListener(new ValueEventListener() {  mReference.addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  List<Entry> dataVals = new ArrayList<>(); //차트 데이터 셋에 담겨질 데이터  ArrayList<String> labels = new ArrayList<String>();  labels.add("TIP");  labels.add("TIP");  labels.add("E동");  labels.add("Olive");  labels.add("산융");  //float i=0;  for (DataSnapshot myData : dataSnapshot.getChildren()) { ////values에 데이터를 담는 과정  //i = i + 1;  Integer cnt= dataSnapshot.child("Kpu").child("people\_number").getValue(Integer.class);  Integer cnt2= dataSnapshot.child("Edong").child("people\_number").getValue(Integer.class);  Integer cnt3= dataSnapshot.child("Olive").child("people\_number").getValue(Integer.class);  Integer cnt4= dataSnapshot.child("Sanyung").child("people\_number").getValue(Integer.class);  Float SensorValue = Float.valueOf(cnt).floatValue();  Float SensorValue2 = Float.valueOf(cnt2).floatValue();  Float SensorValue3 = Float.valueOf(cnt3).floatValue();  Float SensorValue4 = Float.valueOf(cnt4).floatValue();  dataVals.add(new Entry(1f,SensorValue));  dataVals.add(new Entry(2f,SensorValue2));  dataVals.add(new Entry(3f,SensorValue3));  dataVals.add(new Entry(4f,SensorValue4));  // tv.setText("현재 인원:"+cnt);  }  Collections.sort(dataVals, new EntryXComparator());  final LineDataSet lineDataSet = new LineDataSet(dataVals, "인원 수"); //LineDataSet 선언  LineData data = new LineData(lineDataSet); //LineDataSet을 담는 그릇 여러개의 라인 데이터 삽입  lineDataSet.setColors(ColorTemplate.COLORFUL\_COLORS);  //x축 string으로 변환  XAxis xAxis = lineChart.getXAxis();  xAxis.setValueFormatter(new MyXAxisValueFormatter(labels)); //MyXAxisValueFormatter 커스텀 디자인 클래스 호출  //출력  lineChart.setData(data);  lineChart.animateY(5000);  lineChart.notifyDataSetChanged();  lineChart.invalidate();  }  @Override  public void onCancelled(DatabaseError databaseError) {  Toast.makeText(Distribution.this, "Fail to load post", Toast.LENGTH\_SHORT).show();  }  });  }  private void initDatabase() {  mDatabase = FirebaseDatabase.getInstance();  mReference = mDatabase.getReference("Distribution\_DB");  mChild = new ChildEventListener() {  @Override  public void onChildAdded(DataSnapshot dataSnapshot, String s) {  }  @Override  public void onChildChanged(DataSnapshot dataSnapshot, String s) {  }  @Override  public void onChildRemoved(DataSnapshot dataSnapshot) {  }  @Override  public void onChildMoved(DataSnapshot dataSnapshot, String s) {  }  @Override  public void onCancelled(DatabaseError databaseError) {  }  };  mReference.addChildEventListener(mChild);  } |
| 설명 | Firebase DB에서 불러온 각 식당 별 현재 인원 상황을 그래프를 통하여 출력한다. |

**Wep App**

|  |  |
| --- | --- |
| Client\_Insert() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB에 데이터 삽입 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var Client\_DB\_ID = document.getElementById("Client\_DB\_ID").value;  var Client\_DB\_PASSWORD = document.getElementById("Client\_DB\_PASSWORD").value;  var Client\_DB\_address = document.getElementById("Client\_DB\_address").value;  var Client\_DB\_e\_mail = document.getElementById("Client\_DB\_e\_mail").value;  var Client\_DB\_mobile = document.getElementById("Client\_DB\_mobile").value;  var Client\_DB\_name = document.getElementById("Client\_DB\_name").value;  var Client\_DB\_resident\_number = document.getElementById("Client\_DB\_resident\_number").value;  database.ref('Client\_DB/Client/').push({  ID: Client\_DB\_ID, PASSWORD: Client\_DB\_PASSWORD,  address: Client\_DB\_address, e\_mail: Client\_DB\_e\_mail,  mobile: Client\_DB\_mobile, name: Client\_DB\_name, resident\_number: Client\_DB\_resident\_number  });  Client\_load(); |
| 설명 | Client\_DB의 목록에 맞게 입력받은 데이터들을 Firebase의 해당 DB에 추가한다. |

|  |  |
| --- | --- |
| Distribution\_Insert() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB에 데이터 삽입 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var Distribution\_DB\_people\_number =  document.getElementById("Distribution\_DB\_people\_number").value;  var Distribution\_DB\_restaurant\_location =  document.getElementById("Distribution\_DB\_restaurant\_location").value;  database.ref('Distribution\_DB/distribution/').push({  people\_number: Distribution\_DB\_people\_number, restaurant\_location:  Distribution\_DB\_restaurant\_location  });  Distribution\_load(); |
| 설명 | Distribution\_DB의 목록에 맞게 입력받은 데이터들을 Firebase의 해당 DB에 추가한다. |

|  |  |
| --- | --- |
| Restaurant\_Basic\_Insert() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB에 데이터 삽입 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var Restaurant\_DB\_Basic\_location = document.getElementById("Restaurant\_DB\_Basic\_location").value;  var Restaurant\_DB\_Basic\_menu = document.getElementById("Restaurant\_DB\_Basic\_menu").value;  var Restaurant\_DB\_Basic\_name = document.getElementById("Restaurant\_DB\_Basic\_name").value;  var Restaurant\_DB\_Basic\_price = document.getElementById("Restaurant\_DB\_Basic\_price").value;  database.ref('Restaurant\_DB/Basic/').push({  location: Restaurant\_DB\_Basic\_location, menu: Restaurant\_DB\_Basic\_menu,  name: Restaurant\_DB\_Basic\_name, price: Restaurant\_DB\_Basic\_price  });  Restaurant\_load(); |
| 설명 | Restaurant\_Basic\_DB의 목록에 맞게 입력받은 데이터들을 Firebase의 해당 DB에 추가한다. |

|  |  |
| --- | --- |
| Restaurant\_Real\_Time\_Insert() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB에 데이터 삽입 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var Restaurant\_DB\_Real\_time\_cook\_time =  document.getElementById("Restaurant\_DB\_Real\_time\_cook\_time").value;  var Restaurant\_DB\_Real\_time\_food\_number =  document.getElementById("Restaurant\_DB\_Real\_time\_food\_number").value;  var Restaurant\_DB\_Real\_time\_order\_food =  document.getElementById("Restaurant\_DB\_Real\_time\_order\_food").value;  var Restaurant\_DB\_Real\_time\_pay\_time =  document.getElementById("Restaurant\_DB\_Real\_time\_pay\_time").value;  var Restaurant\_DB\_Real\_time\_restaurant\_name =  document.getElementById("Restaurant\_DB\_Real\_time\_restaurant\_name").value;  database.ref('Restaurant\_DB/Real\_time/').push({  cook\_time: Restaurant\_DB\_Real\_time\_cook\_time, food\_number:  Restaurant\_DB\_Real\_time\_food\_number,  order\_food: Restaurant\_DB\_Real\_time\_order\_food, pay\_time: Restaurant\_DB\_Real\_time\_pay\_time,  restaurant\_name: Restaurant\_DB\_Real\_time\_restaurant\_name  });  Restaurant\_load(); |
| 설명 | Restaurant\_Real\_Time\_DB의 목록에 맞게 입력받은 데이터들을 Firebase의 해당 DB에 추가한다. |

|  |  |
| --- | --- |
| Waiting\_Time\_Insert() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB에 데이터 삽입 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var Waiting\_time\_DB\_restaurant\_name =  document.getElementById("Waiting\_time\_DB\_restaurant\_name").value;  var Wainting\_time\_DB\_waiting\_time =  document.getElementById("Wainting\_time\_DB\_waiting\_time").value;  database.ref('Waiting\_time\_DB/Waiting\_time/').push({  restaurant\_name: Waiting\_time\_DB\_restaurant\_name, waiting\_time: Wainting\_time\_DB\_waiting\_time  });  Waiting\_time\_load(); |
| 설명 | Restaurant\_Real\_Time\_DB의 목록에 맞게 입력받은 데이터들을 Firebase의 해당 DB에 추가한다. |

|  |  |
| --- | --- |
| Client\_Load() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB 데이터 값 가져오기 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var result = document.getElementById("result");  //var dataRef = database.ref("Client\_DB\_ID");  var client\_box = document.getElementsByClassName("client\_box")[0];  var distribution\_box = document.getElementsByClassName("distribution\_box")[0];  var waiting\_time\_box = document.getElementsByClassName("waiting\_time\_box")[0];  var restaurant\_box = document.getElementsByClassName("restaurant\_box")[0];  client\_box.style.display = 'block';  distribution\_box.style.display = 'none';  waiting\_time\_box.style.display = 'none';  restaurant\_box.style.display = 'none';  firebase.database().ref('Client\_DB/Client/').orderByKey().once('value', function(snapshot) {  //var rows = [];  snapshot.forEach(function(childSnapshot) {  var childKey = childSnapshot.key;  var childData = childSnapshot.val();  //rows.push(childKey);  //alert(childData.ID);  //result.value += 'Key: ' + childKey + '\n';  result.value += 'ID: ' + childData.ID + '\n';  result.value += 'PASSWORD: ' + childData.PASSWORD + '\n';  result.value += 'address: ' + childData.address + '\n';  result.value += 'e\_mail: ' + childData.e\_mail + '\n';  result.value += 'mobile: ' + childData.mobile + '\n';  result.value += 'name: ' + childData.name + '\n';  result.value += 'resident\_number: ' + childData.resident\_number + '\n\n';  });  });  result.value = ""; |
| 설명 | Firebase에서 Client\_DB에 있는 모든 리스트들을 형식에 맞게 불러온다. |

|  |  |
| --- | --- |
| Distribution\_Load() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB 데이터 값 가져오기 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var result = document.getElementById("result");  var client\_box = document.getElementsByClassName("client\_box")[0];  var distribution\_box = document.getElementsByClassName("distribution\_box")[0];  var waiting\_time\_box = document.getElementsByClassName("waiting\_time\_box")[0];  var restaurant\_box = document.getElementsByClassName("restaurant\_box")[0];  client\_box.style.display = 'none';  distribution\_box.style.display = 'block';  waiting\_time\_box.style.display = 'none';  restaurant\_box.style.display = 'none';  firebase.database().ref('Distribution\_DB/distribution/').orderByKey().once('value', function(snapshot) {  //var rows = [];  snapshot.forEach(function(childSnapshot) {  var childKey = childSnapshot.key;  var childData = childSnapshot.val();  result.value += 'people\_number: ' + childData.people\_number + '\n';  result.value += 'restaurant\_location: ' + childData.restaurant\_location + '\n\n';  });  });  result.value = ""; |
| 설명 | Firebase에서 Distribution\_DB에 있는 모든 리스트들을 형식에 맞게 불러온다. |

|  |  |
| --- | --- |
| Restaurant\_Load() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB 데이터 값 가져오기 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var result = document.getElementById("result");  var client\_box = document.getElementsByClassName("client\_box")[0];  var distribution\_box = document.getElementsByClassName("distribution\_box")[0];  var waiting\_time\_box = document.getElementsByClassName("waiting\_time\_box")[0];  var restaurant\_box = document.getElementsByClassName("restaurant\_box")[0];  client\_box.style.display = 'none';  distribution\_box.style.display = 'none';  waiting\_time\_box.style.display = 'none';  restaurant\_box.style.display = 'block';  firebase.database().ref('Restaurant\_DB/Basic/').orderByKey().once('value', function(snapshot) {  result.value += '\n' + 'Basic' + '\n\n';  //var rows = [];  snapshot.forEach(function(childSnapshot) {  var childKey = childSnapshot.key;  var childData = childSnapshot.val();  result.value += 'location: ' + childData.location + '\n';  result.value += 'menu: ' + childData.menu + '\n';  result.value += 'name: ' + childData.name + '\n';  result.value += 'price: ' + childData.price + '\n\n';  });  });  firebase.database().ref('Restaurant\_DB/Real\_time/').orderByKey().once('value', function(snapshot) {  result.value += '\n' + 'Real\_time' + '\n\n';  //var rows = [];  snapshot.forEach(function(childSnapshot) {  var childKey = childSnapshot.key;  var childData = childSnapshot.val();  result.value += 'cook\_time: ' + childData.cook\_time + '\n';  result.value += 'food\_number: ' + childData.food\_number + '\n';  result.value += 'order\_food: ' + childData.order\_food + '\n';  result.value += 'pay\_time: ' + childData.pay\_time + '\n';  result.value += 'restaurant\_name: ' + childData.restaurant\_name + '\n\n';  });  });  result.value = ""; |
| 설명 | Firebase에서 Restaurant\_DB에 있는 모든 리스트들을 형식에 맞게 불러온다. |

|  |  |
| --- | --- |
| Waiting\_time\_Load() | |
| 분류 | DB 모듈/함수 |
| 기능 | DB 데이터 값 가져오기 |
| 리턴 값 | void |
| 소스코드 | var database = firebase.database();  var result = document.getElementById("result");  var client\_box = document.getElementsByClassName("client\_box")[0];  var distribution\_box = document.getElementsByClassName("distribution\_box")[0];  var waiting\_time\_box = document.getElementsByClassName("waiting\_time\_box")[0];  var restaurant\_box = document.getElementsByClassName("restaurant\_box")[0];  client\_box.style.display = 'none';  distribution\_box.style.display = 'none';  waiting\_time\_box.style.display = 'block';  restaurant\_box.style.display = 'none';  firebase.database().ref('Waiting\_time\_DB/Waiting\_time/').orderByKey().once('value', function(snapshot) {  //var rows = [];  snapshot.forEach(function(childSnapshot) {  var childKey = childSnapshot.key;  var childData = childSnapshot.val();  result.value += 'restaurant\_name: ' + childData.restaurant\_name + '\n';  result.value += 'waiting\_time: ' + childData.waiting\_time + '\n\n';  });  });  result.value = ""; |
| 설명 | Firebase에서 Waiting\_time\_DB에 있는 모든 리스트들을 형식에 맞게 불러온다. |

**라즈베리파이 (PI카메라 연동)**

|  |  |
| --- | --- |
| Video\_record() | |
| 분류 | 카메라 모듈/함수 |
| 기능 | 동영상 촬영 및 사진 캡쳐 |
| 리턴 값 | void |
| 소스코드 | def recordVideo():  with picamera.PiCamera() as camera:  camera.resolution = (320, 240)  camera.start\_preview()  sleep(20)  camera.start\_recording('경로’)  camera.capture('경로’)  camera.wait\_recording(1)  camera.stop\_recording()  camera.stop\_preview()  def record(name1, name2):  print(name1)  recordVideo()  print(name2) |
| 설명 | 시간을 초 단위로 설정하고, 설정한 시간만큼 Picamera를 통해 h264 형식으로 비디오를 촬영한다. 촬영 중 특정 시간에 1 프레임을 캡쳐한다 |

|  |  |
| --- | --- |
| Face\_Count(File\_Name) | |
| 분류 | 라즈베리파이 모듈/함수 |
| 기능 | 송신받은 동영상을 욜로 딥러닝을 이용하여 인원수 구하기 |
| 리턴 값 | Int |
| 소스코드 | def deeplearning(name1, name2):  print(name1)  inputs = tf.placeholder(tf.float32, [None, 416, 416, 3])  model = nets.YOLOv3COCO(inputs, nets.Darknet19)  # model = nets.YOLOv2(inputs, nets.Darknet19)  # frame=cv2.imread("D://pyworks//yolo//truck.jpg",1)  # classes={'0':'person','1':'bicycle','2':'car','3':'bike','5':'bus','7':'truck'}  classes = {'0': 'person'}  # list\_of\_classes=[0,1,2,3,5,7]  list\_of\_classes = [0]  with tf.Session() as sess:  sess.run(model.pretrained())  # "D://pyworks//yolo//videoplayback.mp4"  # cap=cv2.imread('test.jpg',cv2.IMREAD\_COLOR)  cap = cv2.imread('image.JPG', cv2.IMREAD\_COLOR)  img = cv2.resize(cap, (416, 416))  image = np.array(img).reshape(-1, 416, 416, 3)  start\_time = time.time()  preds = sess.run(model.preds, {inputs: model.preprocess(image)})  print(time.time() - start\_time)  boxes = model.get\_boxes(preds, image.shape[1:3])  cv2.namedWindow('image', cv2.WINDOW\_NORMAL)  cv2.resizeWindow('image', 700, 700)  # print("--- %s seconds ---" % (time.time() - start\_time))  boxes1 = np.array(boxes)  for j in list\_of\_classes:  count = 0  if str(j) in classes:  lab = classes[str(j)]  if len(boxes1) != 0:  for i in range(len(boxes1[j])):  box = boxes1[j][i]  if boxes1[j][i][4] >= .40:  count += 1  cv2.rectangle(img, (box[0], box[1]), (box[2], box[3]), (0, 255, 0), 1)  cv2.putText(img, lab, (box[0], box[1]), cv2.FONT\_HERSHEY\_SIMPLEX, .5, (0, 0, 255),  lineType=cv2.LINE\_AA)  print(lab, ": ", count)  ref3.set({'people\_number': count})  cv2.imshow("IMAGE", img)  cv2.destroyAllWindows()  print(name2) |
| 설명 | 촬영한 동영상 파일을 욜로 딥러닝을 통해 원하는 프레임 단위로 영상에 찍힌 사람의 수를 세어 count에 저장 및 firebase의 Distribution\_DB의 people\_number를 업데이트 한다. |

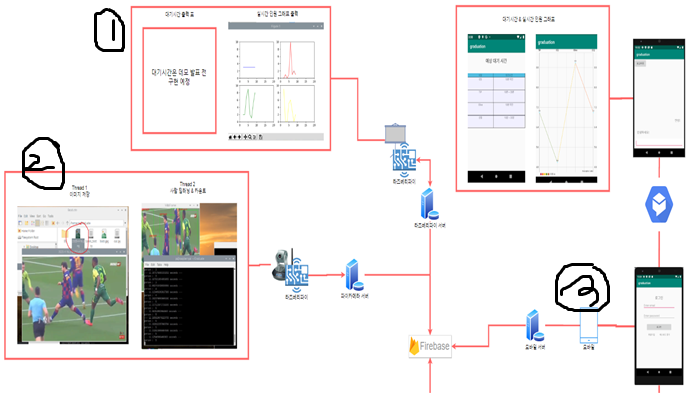
|  |  |
| --- | --- |
| Firebase\_init() | |
| 분류 | 라즈베리파이 모듈/함수 |
| 기능 | Firebase와 연동 및 초기화 |
| 리턴 값 | void |
| 소스코드 | cred = credentials.Certificate('myKey.json')  firebase\_admin.initialize\_app(cred, {  'databaseURL': 'https://graduate-work-462b3.firebaseio.com'  })  ref1 = db.reference('Distribution\_DB/Edong')  ref2 = db.reference('Distribution\_DB/Sanyung')  ref3 = db.reference('Distribution\_DB/Kpu')  ref4 = db.reference('Distribution\_DB/Olive') |
| 설명 | 라즈베리파이와 firebase의 DB를 연동한다 |

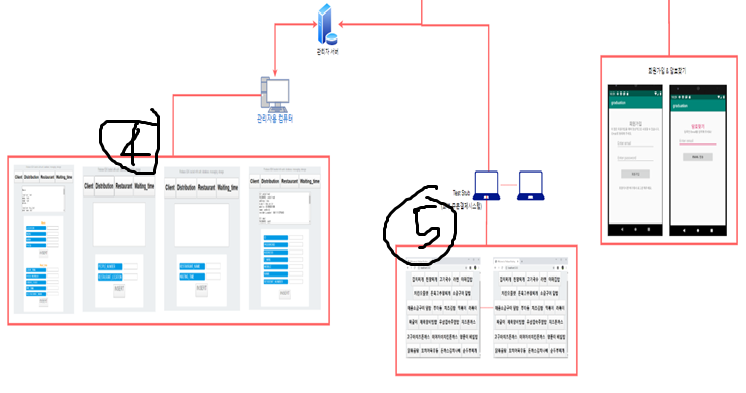
**라즈베리파이(디스플레이)**

|  |  |
| --- | --- |
| Show\_graph() | |
| 분류 | 라즈베리파이 모듈/함수 |
| 기능 | 예측 대기시간, 현장인원 분포도 그래프로 표시 |
| 리턴 값 | void |
| 소스코드 | firebase = firebase.FirebaseApplication("https://graduate-work-462b3.firebaseio.com/", None)  result = firebase.get('Distribution\_DB/Kpu', None)  fig = plt.figure() # figure(도표) 생성  ax = plt.subplot(221, xlim=(0, 20), ylim=(0, 10))  ax1 = plt.subplot(222, xlim=(0, 20), ylim=(0, 10))  ax2 = plt.subplot(223, xlim=(0, 20), ylim=(0, 10))  ax3 = plt.subplot(224, xlim=(0, 20), ylim=(0, 10))  ax.set\_title('TIP')  ax1.set\_title('Olive')  ax2.set\_title('Sanyung')  ax3.set\_title('Edong')  ax.set\_ylabel('the number of persons');  ax1.set\_ylabel('the number of persons');  ax2.set\_ylabel('the number of persons');  ax3.set\_ylabel('the number of persons');  max\_points = 10  max1\_points = 10  max2\_points = 10  max3\_points = 10  line, = ax.plot([], [], lw=1, c='blue', marker='d', ms=2)  line1, = ax1.plot([], [], lw=1, c='red', marker='d', ms=2)  line2, = ax2.plot([], [], lw=1, c='green', marker='d', ms=2)  line3, = ax3.plot([], [], lw=1, c='yellow', marker='d', ms=2)  line, = ax.plot(np.arange(max\_points),  np.ones(max\_points, dtype=np.float) \* np.nan, lw=1, c='blue', ms=1)  line1, = ax1.plot(np.arange(max1\_points),  np.ones(max1\_points, dtype=np.float) \* np.nan, lw=1, c='red', ms=1)  line2, = ax2.plot(np.arange(max2\_points),  np.ones(max2\_points, dtype=np.float) \* np.nan, lw=1, c='green', ms=1)  line3, = ax3.plot(np.arange(max3\_points),  np.ones(max3\_points, dtype=np.float) \* np.nan, lw=1, c='yellow', ms=1)  def init():  return line  def init1():  return line1  def init2():  return line2  def init3():  return line3  def animate(i):  y = firebase.get('Distribution\_DB/Kpu/people\_number', None)  old\_y = line.get\_ydata()  new\_y = np.r\_[old\_y[1:], y]  line.set\_ydata(new\_y)  return line  def animate1(i):  y1 = firebase.get('Distribution\_DB/Olive/people\_number', None)  # y1 = random.randint(0,10)  old\_y1 = line1.get\_ydata()  new\_y1 = np.r\_[old\_y1[1:], y1]  line1.set\_ydata(new\_y1)  return line1  def animate2(i):  y2 = firebase.get('Distribution\_DB/Sanyung/people\_number', None)  # y2 = random.randint(0,10)  old\_y2 = line2.get\_ydata()  new\_y2 = np.r\_[old\_y2[1:], y2]  line2.set\_ydata(new\_y2)  return line2  def animate3(i):  y3 = firebase.get('Distribution\_DB/Edong/people\_number', None)  # y3 = random.randint(0,10)  old\_y3 = line3.get\_ydata()  new\_y3 = np.r\_[old\_y3[1:], y3]  line3.set\_ydata(new\_y3)  return line3  anim = animation.FuncAnimation(fig, animate, init\_func=init, frames=50, interval=200, blit=False)  anim1 = animation.FuncAnimation(fig, animate1, init\_func=init2, frames=50, interval=200, blit=False)  anim2 = animation.FuncAnimation(fig, animate2, init\_func=init2, frames=50, interval=200, blit=False)  anim3 = animation.FuncAnimation(fig, animate3, init\_func=init3, frames=50, interval=200, blit=False)  plt.tight\_layout()  plt.show() |
| 설명 | Firebase의 Distribution\_DB에 저장되어 있는 식당 별 현재 인원, Waiting\_time\_DB의 대기시간 등을 그래프, 표를 통해 디스플레이에 출력한다. |

|  |  |
| --- | --- |
| Bayes\_algorithm() | |
| 분류 | 라즈베리파이 모듈/함수 |
| 기능 | 대기시간 예측 |
| 리턴 값 | void |
| 소스코드 | #from firebase import firebase  from thinkbayes2 import Suite  #firebase = firebase.FirebaseApplication("https://graduate-work-462b3.firebaseio.com/", None)  #kpu\_people\_result = firebase.get('/Distribution\_DB/Kpu/people\_number', None)  #Edong\_people\_result = firebase.get('/Distribution\_DB/Edong/people\_number', None)  class calculate(Suite):  cal\_a = '1분'  cal\_b = '2분'  cal\_c = '3분'  cal\_d = '4분'  mixes = {  cal\_a: dict(p1=0.4, p2=0.3, p3=0.2, p4=0.1),  cal\_b: dict(p1=0.3, p2=0.4, p3=0.2, p4=0.1),  cal\_c: dict(p1=0.2, p2=0.3, p3=0.4, p4=0.1),  cal\_d: dict(p1=0.1, p2=0.2, p3=0.3, p4=0.4),  }  def Likelihood(self, data, hypo):  mix = self.mixes[hypo]  like = mix[data]  return like  kpu\_people\_number = 10  Edong\_people\_number = 7  max\_num = 0  end\_hypo='0분'  DBdata=['1분', '2분', '3분', '4분']  hypos = DBdata  pmf = calculate(hypos)  pmf.Update('p1')  for hypo, prob in pmf.Items():  if prob >= max\_num:  max\_num = prob  end\_hypo = hypo  print(end\_hypo, max\_num)  #print(kpu\_people\_result)  #print(Edong\_people\_result) |
| 설명 | Firebase의 DB 내의 식당의 현재 분포 인원을 가져와 예상 조리 대기시간을 측정한다. |

1. ProtoType 구현





1) 라즈베리파이(디스플레이)

라즈베리파이 디스플레이는 PI카메라로부터 딥러닝을 톻해 분석된 사람의 수를 서버를 참조하여 matplot 라이브러리를 사용하여 실시간 꺾은선 그래프로 출력한다.

2) 라즈베리파이(카메라)

멀티 스레드를 사용하여 스레드1에서는 PI카메라를 통해 영상을 촬영하고 저장한다. 스레드2에서는 저장된 영상을 불러와 딥러닝을 통해 프레임 단위로 사람을 분석하고, 사람의 수까지 카운트 한다. 이후 서버를 통해 firebase DB에 데이터를 저장한다.

3) 모바일

해당 어플리케이션을 실행하였을 때, 계정이 없을 경우 사용자가 원하는 이메일과 비밀번호를 통해 회원가입을 할 수 있다. 비밀번호를 잊어버렸을 경우, 해당 이메일로 비밀번호 재설정 메일이 발송된다. 회원일 경우 등록한 이메일과 비밀번호를 입력하여 로그인 하고, Dialogflow를 이용한 챗 봇 기능을 사용할 수 있다.

챗 봇에서 이용자가 원하는 기능을 키워드로 입력하면, 챗 봇은 학습된 데이터를 기반으로 이용자의 인텐트(화자의 의도, 키워드)를 파악한 후 그에 적절한 응답을 한다.

이용자가 분포도를 입력하면, 챗 봇은 해당 Activity를 자동으로 띄워주고, 각 식당 별 인원을 실시간 그래프로 변환하여 보여준다.

이용자가 대기시간을 입력하면, 챗 봇은 해당 Activity를 자동으로 띄워주고, 각 식당의 예상 대기시간을 테이블을 통해 보여준다.

4) 관리자 웹 앱

NodeJS로 관리자 서버를 구축한 후 해당 Firebase 서버에서 배포하는 특정한 Key를 가지고 있는 운영체제에서만 접속 가능하며, 특정한 명령어를 통해 실행시킬 경우 로컬 호스트로 해당 웹 앱에 접속이 가능하다.

웹 앱 에서는 DB로의 데이터 삽입, 데이터 삭제 및 특정 키워드의 데이터 검색, 특정한 데이터의 수정이 가능하다.

5) 테스트 스텁

NodeJS로 테스트 스텁을 만들어 실제 주문 시스템을 가상 하위 모듈로 대체한다. 기본적인 데이터(주문할 메뉴, 조리 완성까지의 시간) 삽입이 가능하다.