캡스톤디자인 Ⅱ 중간보고서(표지)

프로젝트명 : 생체 인식을 활용한 스마트 도어락

캡스톤 디자인II, 중간보고서

Version 1.0

개발 팀원 명(팀리더): 구나영
고수연
송가은
대표 연락처: 010-4259-0466
e-mail: 20191399@edu.hanbat.ac.kr

캡스톤 디자인 Ⅱ 중간보고서 내용

1. 요구사항 정의서에 명세된 기능에 대하여 현재까지 진척된 결과 및 그 내용을 기술하시오.

- 소스코드 일부(코드 작성지침의 준수 확인용) 및 ver1.0 실행파일(환경설치 문서 포함)

애플리케이션

사용자 등록 코드(합치기 전)

EditUserActivity.kt

```
class EditUserActivity : AppCompatActivity() {

    //리스트는 리사이클러뷰에 표시될 이미지의 URI를 저장하는 역할
    var list = ArrayList<Uri>()
    //MultiImageAdapter 클래스의 인스턴스를 생성
    //list는 이미지 URI 리스트를, this는 현재 액티비티(또는 컨텍스트)를 의미
    //리사이클러뷰의 아이템을 관리하고 표시하는 역할
    val adapter = MultiImageAdapter(list, this)

    lateinit var binding: ActivityEditUserBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityEditUserBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // 앱에서 외부 저장소에 접근하기 위해 사용자로부터 권한을 요청하는 부분
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
            1
        )

        //액티비티 전환할 때 putExtra로 type 정보 함께 보냄
        val type = intent.getStringExtra("type")

        if (type.equals("ADD")) {
            binding.btnSave.text = "추가하기"
        } else {
            binding.btnSave.text = "수정하기"
        }


        //이미지 가져오기 버튼 클릭 리스너
        binding.getImage.setOnClickListener {
```

```
            //이미지 선택 위한 액션 설정한 객체 생성
            //Intent.ACTION_PICK -> 데이터 선택하는 액션
            var intent = Intent(Intent.ACTION_PICK)

            //이미지 데이터 가져올 위치를 설정
            //MediaStore.Images.Media.EXTERNAL_CONTENT_URI -> 외부 저장소에 있는 이
미지를 가리키는 uri
            intent.data = MediaStore.Images.Media.EXTERNAL_CONTENT_URI

            //다중 선택을 허용하기 위한 추가정보를 설정
            //true로 설정하여 다중 선택 활성화
            intent.putExtra(Intent.EXTRA_ALLOW_MULTIPLE, true)

            //액션을 컨텐츠 가져오기로 설정
            intent.action = Intent.ACTION_GET_CONTENT

            //액티비티를 실행하여 이미지를 선택하고 결과를 받기위해 호출, onActivityResult()에
서 결과 처리
            //intent -> 실행할 액티비티와 관련된 정보 포함
            //200 -> 요청코드
            startActivityForResult(intent, 200)
        }

        //수평방향으로 아이템 배치하는 레이아웃 매니저 생성, false-> 역순스크롤 비활성화
        val layoutManager = LinearLayoutManager(this, LinearLayoutManager.HORIZONTAL,
false)
        //리사이클러뷰의 레이아웃 매니저를 설정
        binding.recyclerView.layoutManager = layoutManager
        //리사이클러뷰에 사용할 어댑터 설정
        binding.recyclerView.adapter = adapter

        //save 버튼 클릭 리스너
        binding.btnSave.setOnClickListener {
            //입력된 제목 가져오기
            val title = binding.UserTitle.text.toString()
            //입력된 내용 가져오기
            val content = binding.UserContent.text.toString()
            // 현재 시간을 yyyy-MM-dd HH:mm 형식으로 포맷하여 가져오기
            val currentDate =
                SimpleDateFormat("yyyy-MM-dd HH:mm").format(System.currentTimeMillis())



            //입력된 제목, 내용, 날짜 등으로 User 객체 생성
            val user = User(0, title, content, currentDate, false)
```

```
            //type add인 경우
            if (type.equals("ADD")) {
                //제목과 내용이 모두 비어있지 않는경우 확인
                if (title.isNotEmpty() && content.isNotEmpty()) {
                    //user 객체 생성
                    val user = User(0, title, content, currentDate, false)
                    //결과로 반환할 intent 객체 생성 및 User객체와 flag 값 추가
                    val intent = Intent().apply {
                        putExtra("user", user)
                        putExtra("flag", 0)
                    }
                    //결과 코드와 함께 intent를 설정
                    setResult(RESULT_OK, intent)

                    // 이미지 파이어베이스 스토리지에 데이터 저장
                    uploadDataToFirebase(user, list)
                    //글로벌 스코프에서 코루틴을 실행
                    //화면전환이 딜레이시간만큼 안되는 현상을 막기위해
                    //애플리케이션 전역에서 실행되는 독립적인 백그라운드 작업으로 실행
                    GlobalScope.launch {
                        //스토리지에 이미지 파일을 전송한 후 파이어베이스에 업로드 되는데까지
                        //시간이 걸리는데 서버로 바로 신호를 주면 이미지 파일이 없는 오류가
발생
                        //그래서 delay를 주고 서버로 신호 전송
                        delay(15000)
                        // 서버로 신호를 보내는 부분
                        postUsername(user.title)
                    }
                    finish()

                }
            } else {
                // 수정
            }

        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        //돌아온 결과가 성공적이고 요청코드가 200인 경우 확인
        if (resultCode == RESULT_OK && requestCode == 200) {
            //이미지 URI 담고 있는 list 초기화
            list.clear()
```

```
            if (data?.clipData != null) { // 사진 여러개 선택한 경우
                //선택한 사진의 개수 가져옴
                val count = data.clipData!!.itemCount
                //20장 초과 시 메시지 표시하고 함수 종료
                if (count > 20) {
                    Toast.makeText(applicationContext, "사진은 20장까지 선택 가능합니다.",
Toast.LENGTH_LONG)
                    return
                }
                //선택한 각 사진에 대해 반복문 실행
                for (i in 0 until count) {
                    //i번째 이미지의 URI를 가져옴
                    val imageUri = data.clipData!!.getItemAt(i).uri
                    //리스트에 추가
                    list.add(imageUri)
                }
            } else { // 단일 선택
                data?.data?.let { uri ->
                    //선택 이미지 uri 가져오기
                    val imageUri: Uri? = data?.data
                    //사진의 uri가 null이 아닌 경우에만 처리
                    if (imageUri != null) {
                        //리스트 추가
                        list.add(imageUri)
                    }
                }
            }
            //어댑터에 변경사항 알려 리사이클러뷰 갱신
            adapter.notifyDataSetChanged()

        }

    }

    //스토리지에 사진 올리기
    private fun uploadDataToFirebase(user: User, imageList: ArrayList<Uri>) {

        //스토리지 초기화
        //파이어베이스 스토리지 인스턴스 가져오기
        val storage = FirebaseStorage.getInstance()
        //파이어베이스 스토리지의 루트 참조 가져오기
        val storageRef = storage.reference

//        val db = FirebaseFirestore.getInstance()
```

```
//          val collectionRef = db.collection("users")

        //업로드된 이미지의 다운로드 URL 저장하기 위한 빈문자열 배열 초기화
        val imageUrls = ArrayList<String>()
        //이미지 업로드 작업을 저장하기 위한 빈 작업 목록 초기화
        val uploadTasks = mutableListOf<Task<Uri>>()

        // 타이틀 업로드-> 서버에서 폴더 받아서 할때 이거때문에 오류나니까 빼주기
//          val titleRef = storageRef.child("${user.title}/title.txt")
//          val titleUploadTask = titleRef.putBytes(user.title.toByteArray())

        //사진 순서대로 보내는 코드
        for (i in 0 until imageList.size) {
            //현재 순회중인 이미지의 uri 가져옴
            val imageUri = imageList[i]
            //업로드할 이미지의 이름 생성
            val imageName = "${user.title}/image_${System.currentTimeMillis()}_$i.jpg"

            //이미지를 업로드할 파이어베이스 스토리지 내의 경로 지정
            val imagesRef = storageRef.child(imageName)
            //이미지를 업로드하는 작업 생성
            val uploadTask = imagesRef.putFile(imageUri)

            //이미지 업로드 작업 완료시 다운로드 url을 가져오는 작업 생성
            val completionTask = uploadTask.continueWithTask { task ->
                if (!task.isSuccessful) {
                    task.exception?.let {
                        throw it
                    }
                }
                imagesRef.downloadUrl
            }

            //이미지 업로드가 성공한 경우 처리할 작업을 정의
            completionTask.addOnSuccessListener { uri ->
                // 이미지 업로드 성공 시 다운로드 URL 저장
                val imageUrl = uri.toString()
                imageUrls.add(imageUrl)



            }

            completionTask.addOnFailureListener { exception ->
                // 이미지 업로드 실패 시 처리
                Log.e(TAG, "Image upload failed.", exception)
```

```
            }
            uploadTasks.add(completionTask)
        }

    }


//    //파이어스토리에 올리는 부분 신경ㄴㄴ
//    private fun saveDataToFirestore(
//        user: User,
//        imageUrls: ArrayList<String>,
//        collectionRef: CollectionReference
//    ) {
//        val data = hashMapOf(
//            "title" to user.title,
//            "images" to imageUrls
//        )
//
//        collectionRef
//            .add(data)
//            .addOnSuccessListener { documentReference ->
//                Log.d(TAG, "DocumentSnapshot added with ID: ${documentReference.id}")
//
//                // 업로드 완료 메시지 출력 또는 다른 작업 수행
//                                        Toast.makeText(this,   "데이터가업로드되었습니다.",
Toast.LENGTH_SHORT).show()
//                finish()
//            }
//            .addOnFailureListener { e ->
//                Log.e(TAG, "Error adding document", e)
//                                    Toast.makeText(this,   "데이터   업로드에   실패했습니다.",
Toast.LENGTH_SHORT).show()
//            }
//    }
//
//    private fun getCollectionReference(): CollectionReference {
//        val db = FirebaseFirestore.getInstance()
//        return db.collection("users")
//    }

    companion object {
        private const val TAG = "EditUserActivity"
    }

    //서버에 post로 request하기
```

```
    private fun postUsername(username: String) {
        //i/o작업을 수행하기 위한 코루틴 스코프 생성(백그라운드 실행)
        CoroutineScope(Dispatchers.IO).launch {
            try {
                //서버의 기본 url
                val baseUrl = "http://192.168.0.5:5000/"
                //서버의 엔드포인트(경로)설정
                val endpoint = "userjoin"
                //요청에 포함될 사용자 이름
                val paramName = "user_name"

                //서버로 보낼 요청의 url 생성, 사용자 이름 utf-8로 인코딩
                val                                    url                                    =
"${baseUrl}${endpoint}?${paramName}=${URLEncoder.encode(username, "UTF-8")}"
                //빈 JSONObject를 생성
                val jsonBody = JSONObject()
                //OkHttp 클라이언트 인스턴스를 생성
                val client = OkHttpClient()
                //요청의 미디어 타입을 설정
                val MEDIA_TYPE_JSON = "application/json; charset=utf-8".toMediaType()

                //JSONObject에 "user_name" 키와 사용자 이름 값을 설정
                jsonBody.put("user_name", username)

                //JSONObject를 문자열 형태로 변환
                val requestBody = jsonBody.toString()

                //요청 객체를 생성
                val request = Request.Builder()
                    .url(url)
                    .post(requestBody.toRequestBody(MEDIA_TYPE_JSON))
                    .build()

                //요청을 동기적으로 실행하고 응답을 받음
                //use 함수를 사용하여 응답 처리가 끝난 후에 자원을 해제
                client.newCall(request).execute().use { response ->
                    //응답이 성공적이지 않은 경우 예외를 발생
                    if    (!response.isSuccessful)    throw    IOException("Unexpected    code
$response")

                    val responseBody = response.body?.string()

                    //응답 바디의 내용을 출력
                    println(response.body!!.string())
```

```
                }
                //예외가 발생한 경우 해당 예외를 처리
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }
    }
}
```

UseretcScreen.kt

```
class UseretcScreen : AppCompatActivity() {
    lateinit var binding: ActivityUseretcBinding
    lateinit var userViewModel: UserViewModel
    lateinit var userAdapter: UserAdapter




    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityUseretcBinding.inflate(layoutInflater)
        setContentView(binding.root)

        //수정 2차 시도
        UserRepository.initialize(applicationContext) // UserRepository 초기화 추가

        userViewModel = ViewModelProvider(this).get(UserViewModel::class.java)

        //수정 1차 시도
        //userViewModel = ViewModelProvider(this, ViewModelProvider.AndroidViewModelFactory.getInstance(application))[UserViewModel::class.java]

        //원래코드
        //userViewModel = ViewModelProvider(this)[UserViewModel::class.java]

        userViewModel.userList.observe(this) {
            userAdapter.update(it)
        }

        userAdapter = UserAdapter(this)
        binding.rvUserList.layoutManager = LinearLayoutManager(this)
        binding.rvUserList.adapter = userAdapter
```

```
        binding.fabAdd.setOnClickListener {
            val intent = Intent(this, EditUserActivity::class.java).apply {
                putExtra("type", "ADD")
            }
            requestActivity.launch(intent)
        }

//                                          userAdapter.setItemCheckBoxClickListener(object:
UserAdapter.ItemCheckBoxClickListener {
//              override fun onClick(view: View, position: Int, itemId: Long) {
//                  CoroutineScope(Dispatchers.IO).launch {
//                      val user = userViewModel.getOne(itemId)
//                      user.isChecked = !user.isChecked
//                      userViewModel.update(user)
//                  }
//              }
//          })
    }

    private                 val                 requestActivity                 =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
        if (it.resultCode == RESULT_OK) {
            val user = it.data?.getSerializableExtra("user") as User

            when(it.data?.getIntExtra("flag", -1)) {
                0 -> {
                    CoroutineScope(Dispatchers.IO).launch {
                        userViewModel.insert(user)
                    }
                    Toast.makeText(this, "추가되었습니다.", Toast.LENGTH_SHORT).show()
                }
            }
        }
    }

}
```

외부인 방문 확인 코드(합치기 전)

BindingAdapter.kt

```
object BindingAdapter {
    @JvmStatic
    @BindingAdapter("app:imageUrl")
    fun loadImage(imageView: ImageView, url: String){
```

```
        val storage: FirebaseStorage = FirebaseStorage.getInstance("gs://framerun-cloud.appspot.com/")
        val storageReference = storage.reference
        val pathReference = storageReference.child("Outsider_Img/$url")

        pathReference.downloadUrl.addOnSuccessListener { uri ->
            Glide.with(imageView.context)
                .load(uri)
                .diskCacheStrategy(DiskCacheStrategy.NONE)
                .centerCrop()
                .into(imageView)
        }
    }
}
```

InfoDTO.kt

```
data class InfoDTO(
    val thumbnail: String,
    val title : String
    )
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding



    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        MyFirebaseMessagingService().getFirebaseToken()


    }
}
```

MyFirebaseMessagingService.kt

```
class MyFirebaseMessagingService : FirebaseMessagingService() {


    private val TAG = "FirebaseService"

    private lateinit var binding: ActivityMainBinding
    private lateinit var manager: RecyclerView.LayoutManager



    override fun onNewToken(token: String) {
        Log.d(TAG, "new Token: $token")
```

```
        val pref = this.getSharedPreferences("token", Context.MODE_PRIVATE)
        val editor = pref.edit()
        editor.putString("token", token).apply()
        editor.commit()
        Log.i(TAG, "성공적으로 토큰을 저장함")
    }


    override fun onMessageReceived(remoteMessage: RemoteMessage) {
        Log.d(TAG, "From: " + remoteMessage!!.from)

        Log.d(TAG, "Message data : ${remoteMessage.data}")

        if(remoteMessage.data.isNotEmpty()){
            manager = GridLayoutManager(this, 3)

            var intruder =
InfoDTO(remoteMessage.data["image"].toString(),remoteMessage.data["title"].toString())
            var intruderList = listOf(intruder)



            binding.recyclerView.apply{
                adapter = RecyclerAdapter(intruderList)
                layoutManager = manager
            }

            sendNotification(remoteMessage)
        }else {
            Log.e(TAG, "data가 비어있습니다. 메시지를 수신하지 못했습니다.")
        }
    }


    private fun sendNotification(remoteMessage: RemoteMessage) {

        val uniId: Int = (System.currentTimeMillis() / 7).toInt()



        val intent = Intent(this, MainActivity::class.java)
        for(key in remoteMessage.data.keys){
            intent.putExtra(key, remoteMessage.data.getValue(key))
        }
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP) // Activity Stack 을 경로만 남김
(A-B-C-D-B => A-B)
```

```
        val pendingIntent = PendingIntent.getActivity(this, uniId, intent, PendingIntent.FLAG_ONE_SHOT or PendingIntent.FLAG_MUTABLE)

        val channelId = "my_channel"

        val soundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION)

        val notificationBuilder = NotificationCompat.Builder(this, channelId)
            .setSmallIcon(R.mipmap.ic_launcher)
            .setContentTitle(remoteMessage.data["title"].toString())
            .setContentText(remoteMessage.data["body"].toString())
            .setAutoCancel(true)
            .setSound(soundUri)
            .setContentIntent(pendingIntent)

        val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(channelId, "Notice", NotificationManager.IMPORTANCE_DEFAULT)
            notificationManager.createNotificationChannel(channel)
        }

        notificationManager.notify(uniId, notificationBuilder.build())
    }

    fun getFirebaseToken() {
        FirebaseMessaging.getInstance().token.addOnSuccessListener {
            Log.d(TAG, "token=${it}")
        }
    }
}
```

RecyclerAdapter.kt

```
class RecyclerAdapter(private val itemList: List<InfoDTO>): RecyclerView.Adapter<RecyclerAdapter.MyViewHolder>(){

    inner class MyViewHolder(val binding: ItemInfoBinding): RecyclerView.ViewHolder(binding.root) {
        fun bind(item: InfoDTO) {
```

```
                binding.info = item
            }
        }

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
            val inflater = LayoutInflater.from(parent.context)
            val listItemBinding = ItemInfoBinding.inflate(inflater, parent, false)
            return MyViewHolder(listItemBinding)
        }

        override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
            holder.bind(itemList[position])
        }

        override fun getItemCount(): Int {
            return itemList.size
        }
    }
```

웹서버 Flask

사용자 등록 (얼굴)

```
# '''model (사전 학습 가중치 + 모델 호출)'''
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
weight = 'kface.r34.arc.pt'
ckpt = torch.load(weight, map_location=device)  # load checkpoint

model = ckpt['backbone'].to(device)
model = DataParallel(model)

# 'main': 이름, __name__: 모듈명, url_prefix='/': URL 프리픽스
bp = Blueprint('userjoin', __name__, url_prefix='/')

# 1-1. 어플에서 파이어베이스로 사진 전송이 완료됐음을 인지하고, 파이어베이스에서 사진을 가져오는 코드
# 어플에서 1을 받으면 파이어베이스에서 데이터 가져오기 -> 완료되면 벡터코드 실행


@bp.route('/userjoin', methods=['GET', 'POST'])  # ★ POST 변경
def getUserData():

    # 어플에서 /userjoin?user_name = 사용자이름 URL 접속한 후
    user_name = request.args.get('user_name')
    print('User Join >>', user_name)

    # 새 폴더를 만들 경로 지정 (사용자 이름으로 폴더 생성할 것)
```

```
    new_folder_path = 'C:/Framerun/user/' + user_name

    # 폴더가 이미 존재하는지 확인
    if not os.path.exists(new_folder_path):
        # 폴더를 생성합니다.
        os.makedirs(new_folder_path)
        print("Folder created successfully.")
    else:
        print("The folder already exists.")

    # folder_path = user_name.encode(
    #     'iso-8859-1').decode('utf-8') + '/'  # 파이어베이스 폴더
    folder_path = user_name+'/'
    print("user_name:", user_name)

    bucket = storage.bucket()
    blobs = bucket.list_blobs(prefix=folder_path)

    for blob in blobs:
        filename = blob.name.split('/')[-1]
        if filename == "":
            continue

        # Download the file from Firebase Storage
        blob.download_to_filename(filename=new_folder_path + '/' + filename)
        print(filename)

    print(user_name + "'s images downloaded successfully.")

    # /userjoin/getvertor로 이동
    response = requests.post(getvectorURL + '?user_name=' + user_name)

    return 'successful'  # 클라이언트에게 응답 전송


# 1-2. 받은 사용자 사진으로 벡터값 저장
@bp.route('/userjoin/getvector', methods=['POST'])  # ★ POST 변경
def vector():
    print("<<Get Vector>>")
    t1 = datetime.now()
    user_name = request.args.get('user_name')
    folder_path = './user/' + user_name  # 폴더 경로
    pic = []

    for root, dirs, files in os.walk(folder_path):
```

```
        for file in files:
            file_path = posixpath.join(root, file)
            pic.append(file_path)

    '''얼굴 특징 벡터 추출'''
    def getFeature(model, image):
        # image = cv2.imread(img_path)
        image = cv2.resize(image, (112, 112))
        image = image.transpose((2, 0, 1))
        image = image[np.newaxis, :, :, :]
        image = image.astype(np.float32, copy=False)
        image -= 127.5
        image /= 127.5

        data = torch.from_numpy(image)
        data = data.to(torch.device('cuda'))
        feature = model(data)
        feature = feature.data.cpu().numpy()[0]

        return feature

    # 벡터값 계산
    v = {}
    index = 1  # 벡터값 인덱스
    cnt = 0  # 벡터값 저장 성공 카운트

    for p in pic:
        image = cv2.imread(p)
        obj = RetinaFace.detect_faces(p)

        try:
            for key in obj.keys():
                identity = obj[key]
                facial_area = identity["facial_area"]
                crop_image = image[facial_area[1]
                    :facial_area[3], facial_area[0]:facial_area[2]]

            feature = getFeature(model, crop_image)  # 얼굴의 특징 벡터 추출
            v[p] = feature

            # npy 파일로 저장하기
            np.save('./user/' + user_name + str(index) + '.npy', feature)

            print(index, "벡터값 저장 성공")
            index += 1
```

```
            cnt += 1
        except AttributeError:
            print(index, "벡터값 저장 실패")
            index += 1
            continue
    t2 = datetime.now()
    print('벡터값 계산 및 저장 소요 시간:', t2-t1)
    print('%d장 사진 중, %d개 벡터값 저장' % (len(pic), cnt))

    return 'done!'
```

● 해당 코드는 어플리케이션에서 등록한 사용자의 얼굴 사진을 파이어베이스를 통해 전달받아 서버에 등록하는 코드이다.

얼굴 인식 서버

```
# '''model (사전 학습 가중치 + 모델 호출)'''
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
weight = 'kface.r34.arc.unpg.wisk1.0.pt'
ckpt = torch.load(weight, map_location=device)  # load checkpoint

model = ckpt['backbone'].to(device)
model = DataParallel(model)

# 'main': 이름, __name__: 모듈명, url_prefix='/': URL 프리픽스
bp = Blueprint('run_face', __name__, url_prefix='/')

# 3. 카메라 모듈로 찍힌 사진 1장의 벡터값과 유저의 벡터값과 비교 (얼굴 인식)
# 사진 촬영→ 웹서버에서 벡터값 비교(얼굴인식)→ if 1 : return 열림 else:  어플 알림 보내는 웹서버 요청

# /face_one: 사진 한 장을 받아서 얼굴 인식

@bp.route('/face_one', methods=['POST'])
def face_one():
    # t1 = datetime.now()

    # 유저의 벡터값 가져오기
    s1 = datetime.now()
    npy_path = 'c:/FRAMERUN/user/'
    npy_list = [f for f in os.listdir(
        npy_path) if f.endswith('.npy')]  # ndarray 파일 이름 list

    vector_list = {}  # ndarray dictionary
    for file_name in npy_list:
        user_vector = np.load(npy_path + file_name)
        vector_list[file_name] = user_vector
```

```
    s2 = datetime.now()
    print('유저의 벡터값 가져오기 done')
    print("유저 벡터값 가져오는 시간 :", s2-s1)

    # 3-1. 벡터값 추출
    # (1) 사진 가져오기
    s1 = datetime.now()
    file = request.files['file']
    filename = file.filename
    filepath = 'c:/FRAMERUN/sensor_img/'
    file.save(filepath +  secure_filename(filename))
    s2 = datetime.now()
    print('사진 가져오기 done')

    print("사진 가져오는 시간 :", s2-s1)

    # (2) 벡터 추출
    s1 = datetime.now()
    image = cv2.imread(filepath +  filename)
    obj = RetinaFace.detect_faces(filepath +  filename)

    try:
        for key in obj.keys():

            identity = obj[key]
            facial_area = identity["facial_area"]
            crop_image = image[facial_area[1]:facial_area[3], facial_area[0]:facial_area[2]]

        feature = getFeature(model, crop_image)  # 얼굴의 특징 벡터 추출
        print('벡터 추출 done')

    except AttributeError:
        print("벡터 추출 fail")

    # t2 = datetime.now()
    # print(t2-t1)
    s2 = datetime.now()
    print("벡터 추출 시간:", s2-s1)

    # 3-2. 벡터값 비교
    s1 = datetime.now()

    def cosine_metric(x1, x2):
        return np.dot(x1, x2) / (np.linalg.norm(x1) * np.linalg.norm(x2))
    try:
```

```
        for v1 in vector_list:
            sim = cosine_metric(vector_list[v1], feature)
            print(v1 + '과 비교: ', sim)

            # case1. 얼굴 인식 성공
            if sim >= 0.85:
                t3 = datetime.now()
                print('얼굴 인식 성공', v1)
                # print(t3-t1)
                return ('approval')

    except UnboundLocalError:
        print("에러 발생")
        return ('error')

    s2 = datetime.now()
    print("벡터값 비교", s2-s1)
```

● 해당 코드는 라즈베리파이에서 촬영한 사진으로 얼굴 인식을 하는 코드이다.

잠금 모드 설정 서버

```
@bp.route('/mode_set', methods=['GET', 'POST'])
def mode_setting():

    req_mode = request.args.get('req_mode')  # /mode_set?req_mode = 값
    with open(r"C:₩Users₩User₩Desktop₩Framerun 실험₩잠금모드 서버 실험₩mode.txt", "w") as file:
        file.write(req_mode)
    print("Mode Set >> " + req_mode)

    return req_mode + " Mode Setting Done"

    '''
    임시비밀번호 모드
    {'모드': OO, '비밀번호': NONE or OOOO, '유효시간': NONE or YYYY-MMMM-DDDD OO:OO:OO} 형식으로 요청 받아서 db에 저장
    유효시간에 맞춰서 타이머(?) 적용해서 시간 지나면 모드 내용 자동대로 수정되야함
    안되면 잠금 모드 확인 서버 코드 ▼ 에서 임시비밀번호 모드 시간 유효한건지 체크 -> 시간 지났을 경우
    기본 비번모드(1)로 내용 수정하게끔 코드 만들기

    '''
```

● 해당 코드는 사용자가 설정한 잠금 모드를 전달받아 서버에 저장하는 코드이다.

잠금 모드 확인 서버

```
# 잠금 모드 확인 서버
# 기본 비밀번호:1 | 다중 잠금:2 | 임시 비밀번호:3

@bp.route('/mode_check', methods=['GET', 'POST'])
def mode_check():

    with open(r"C:₩Users₩User₩Desktop₩Framerun 실험₩잠금모드 서버 실험₩mode.txt", "r") as file:
        mode_type = file.read()
    print("Mode Check >> " + mode_type)

    return mode_type

    '''
    임시비밀번호 모드
    {'모드': OO, '비밀번호': NONE or OOOO, '유효시간': NONE or YYYY-MMMM-DDDD OO:OO:OO} 형식으로 요청 받아서 db에 저장
    유효시간에 맞춰서 타이머(?) 적용해서 시간 지나면 모드 내용 자동대로 수정되야함
    안되면 잠금 모드 확인 서버 코드 ▼ 에서 임시비밀번호 모드 시간 유효한건지 체크 -> 시간 지났을 경우
    기본 비번모드(1)로 내용 수정하게끔 코드 만들기

    '''
```

● 해당 코드는 잠금 모드를 라즈베리파이에 전달하는 코드이다.

Bcript 알고리즘 예시

```
import bcrypt

password = '1234'
hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

print(hashed_password)
save_password = hashed_password.decode('utf-8')

new_password = '1234'
bcrypt.checkpw(new_password.encode('utf-8'),save_password.encode('utf-8'))
# -> True
```

● 해당 코드는 비밀번호를 DB에 저장하기 위해 해시값으로 암호화하는 코드이다.

얼굴 인식 모델

얼굴 인식 모델 - Backbone (ResNet)

```
# 모델 아키텍처
import torch
import torch.nn as nn
import math
import torch.utils.model_zoo as model_zoo
import torch.nn.utils.weight_norm as weight_norm
import torch.nn.functional as F


# __all__ = ['ResNet', 'resnet18', 'resnet34', 'resnet50', 'resnet101',
#            'resnet152']


model_urls = {
    'resnet18': 'https://download.pytorch.org/models/resnet18-5c106cde.pth',
    'resnet34': 'https://download.pytorch.org/models/resnet34-333f7ec4.pth',
    'resnet50': 'https://download.pytorch.org/models/resnet50-19c8e357.pth',
    'resnet101': 'https://download.pytorch.org/models/resnet101-5d3b4d8f.pth',
    'resnet152': 'https://download.pytorch.org/models/resnet152-b121ed2d.pth',
}


def conv3x3(in_planes, out_planes, stride=1):
    """3x3 convolution with padding"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                     padding=1, bias=False)


class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        residual = x
```

```
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.relu(out)

        return out


class IRBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None, use_se=True):
        super(IRBlock, self).__init__()
        self.bn0 = nn.BatchNorm2d(inplanes)
        self.conv1 = conv3x3(inplanes, inplanes)
        self.bn1 = nn.BatchNorm2d(inplanes)
        self.prelu = nn.PReLU()
        self.conv2 = conv3x3(inplanes, planes, stride)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride
        self.use_se = use_se
        if self.use_se:
            self.se = SEBlock(planes)

    def forward(self, x):
        residual = x
        out = self.bn0(x)
        out = self.conv1(out)
        out = self.bn1(out)
        out = self.prelu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        if self.use_se:
            out = self.se(out)
```

```
        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.prelu(out)

        return out


class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride,
                               padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, planes * self.expansion, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        residual = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.relu(out)
```

```
        return out


class SEBlock(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SEBlock, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
                nn.Linear(channel, channel // reduction),
                nn.PReLU(),
                nn.Linear(channel // reduction, channel),
                nn.Sigmoid()
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y


class ResNetFace(nn.Module):
    def __init__(self, block, layers, use_se=True):
        self.inplanes = 64
        self.use_se = use_se
        super(ResNetFace, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.prelu = nn.PReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)
        self.bn4 = nn.BatchNorm2d(512)
        self.dropout = nn.Dropout()
        self.fc5 = nn.Linear(512 * 8 * 8, 512)
        self.bn5 = nn.BatchNorm1d(512)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.xavier_normal_(m.weight)
            elif isinstance(m, nn.BatchNorm2d) or isinstance(m, nn.BatchNorm1d):
                nn.init.constant_(m.weight, 1)
```

```
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.xavier_normal_(m.weight)
                nn.init.constant_(m.bias, 0)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes * block.expansion,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(planes * block.expansion),
            )
        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample, use_se=self.use_se))
        self.inplanes = planes
        for i in range(1, blocks):
            layers.append(block(self.inplanes, planes, use_se=self.use_se))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.prelu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.bn4(x)
        x = self.dropout(x)
        x = x.view(x.size(0), -1)
        x = self.fc5(x)
        x = self.bn5(x)

        return x


class ResNet(nn.Module):

    def __init__(self, block, layers):
        self.inplanes = 64
        super(ResNet, self).__init__()
```

```
        # self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,
        #                        bias=False)
        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1,
                               bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        # self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, layers[0], stride=2)
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)
        # self.avgpool = nn.AvgPool2d(8, stride=1)
        # self.fc = nn.Linear(512 * block.expansion, num_classes)
        self.fc5 = nn.Linear(512 * 8 * 8, 512)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes * block.expansion,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(planes * block.expansion),
            )

        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes * block.expansion
        for i in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        # x = self.maxpool(x)
```

```
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        # x = nn.AvgPool2d(kernel_size=x.size()[2:])(x)
        # x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc5(x)

        return x

def resnet18(pretrained=False, **kwargs):
    """Constructs a ResNet-18 model.
    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = ResNet(BasicBlock, [2, 2, 2, 2], **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['resnet18']))
    return model

def resnet34(pretrained=False, **kwargs):
    """Constructs a ResNet-34 model.
    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = ResNet(BasicBlock, [3, 4, 6, 3], **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['resnet34']))
    return model

def resnet50(pretrained=False, **kwargs):
    """Constructs a ResNet-50 model.
    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = ResNet(Bottleneck, [3, 4, 6, 3], **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['resnet50']))
    return model

def resnet101(pretrained=False, **kwargs):
    """Constructs a ResNet-101 model.
    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
```

```
    """
    model = ResNet(Bottleneck, [3, 4, 23, 3], **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['resnet101']))
    return model

def resnet152(pretrained=False, **kwargs):
    """Constructs a ResNet-152 model.
    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = ResNet(Bottleneck, [3, 8, 36, 3], **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['resnet152']))
    return model

def resnet_face18(use_se=True, **kwargs):
    model = ResNetFace(IRBlock, [2, 2, 2, 2], use_se=use_se, **kwargs)
    return model
```

- https://github.com/ronghuaiyang/arcface-pytorch
- https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

얼굴 인식 모델 - Metric

```
# 모델 metircs : 평가지표
from __future__ import print_function
from __future__ import division
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import Parameter
import math


class ArcMarginProduct(nn.Module):
    def __init__(self, in_features, out_features, s=30.0, m=0.50, easy_margin=False):
        super(ArcMarginProduct, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.s = s
        self.m = m
        self.weight = Parameter(torch.FloatTensor(out_features, in_features))
        nn.init.xavier_uniform_(self.weight)

        self.easy_margin = easy_margin
        self.cos_m = math.cos(m)
```

```
        self.sin_m = math.sin(m)
        self.th = math.cos(math.pi - m)
        self.mm = math.sin(math.pi - m) * m

    def forward(self, input, label):
        cosine = F.linear(F.normalize(input), F.normalize(self.weight))
        sine = torch.sqrt((1.0 - torch.pow(cosine, 2)).clamp(0, 1))
        phi = cosine * self.cos_m - sine * self.sin_m
        if self.easy_margin:
            phi = torch.where(cosine > 0, phi, cosine)
        else:
            phi = torch.where(cosine > self.th, phi, cosine - self.mm)

        one_hot = torch.zeros(cosine.size(), device='cuda')
        one_hot.scatter_(1, label.view(-1, 1).long(), 1)

        output = (one_hot * phi) + ((1.0 - one_hot) * cosine)
        output *= self.s

        return output


class AddMarginProduct(nn.Module):
    def __init__(self, in_features, out_features, s=30.0, m=0.40):
        super(AddMarginProduct, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.s = s
        self.m = m
        self.weight = Parameter(torch.FloatTensor(out_features, in_features))
        nn.init.xavier_uniform_(self.weight)

    def forward(self, input, label):
        cosine = F.linear(F.normalize(input), F.normalize(self.weight))
        phi = cosine - self.m

        one_hot = torch.zeros(cosine.size(), device='cuda')
        one_hot.scatter_(1, label.view(-1, 1).long(), 1)

        output = (one_hot * phi) + ((1.0 - one_hot) * cosine)
        output *= self.s

        return output

    def __repr__(self):
```

```
        return self.__class__.__name__ + '(' ₩
                + 'in_features=' + str(self.in_features) ₩
                + ', out_features=' + str(self.out_features) ₩
                + ', s=' + str(self.s) ₩
                + ', m=' + str(self.m) + ')'


class SphereProduct(nn.Module):
    def __init__(self, in_features, out_features, m=4):
        super(SphereProduct, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.m = m
        self.base = 1000.0
        self.gamma = 0.12
        self.power = 1
        self.LambdaMin = 5.0
        self.iter = 0
        self.weight = Parameter(torch.FloatTensor(out_features, in_features))
        nn.init.xavier_uniform(self.weight)

        self.mlambda = [
            lambda x: x ** 0,
            lambda x: x ** 1,
            lambda x: 2 * x ** 2 - 1,
            lambda x: 4 * x ** 3 - 3 * x,
            lambda x: 8 * x ** 4 - 8 * x ** 2 + 1,
            lambda x: 16 * x ** 5 - 20 * x ** 3 + 5 * x
        ]

    def forward(self, input, label):
        self.iter += 1
        self.lamb = max(self.LambdaMin, self.base * (1 + self.gamma * self.iter) ** (-1 *
self.power))

        cos_theta = F.linear(F.normalize(input), F.normalize(self.weight))
        cos_theta = cos_theta.clamp(-1, 1)
        cos_m_theta = self.mlambda[self.m](cos_theta)
        theta = cos_theta.data.acos()
        k = (self.m * theta / 3.14159265).floor()
        phi_theta = ((-1.0) ** k) * cos_m_theta - 2 * k
        NormOfFeature = torch.norm(input, 2, 1)

        one_hot = torch.zeros(cos_theta.size())
        one_hot = one_hot.cuda() if cos_theta.is_cuda else one_hot
```

```
        one_hot.scatter_(1, label.view(-1, 1), 1)

        output = (one_hot * (phi_theta - cos_theta) / (1 + self.lamb)) + cos_theta
        output *= NormOfFeature.view(-1, 1)

        return output

    def __repr__(self):
        return self.__class__.__name__ + '(' ₩
                + 'in_features=' + str(self.in_features) ₩
                + ', out_features=' + str(self.out_features) ₩
                + ', m=' + str(self.m) + ')'
```

- https://github.com/wujiyang/Face_Pytorch/blob/master/margin/ArcMarginProduct.py
- https://github.com/ronghuaiyang/arcface-pytorch

얼굴 인식 모델 - Loss

```
import torch
import torch.nn as nn

class FocalLoss(nn.Module):

    def __init__(self, gamma=0, eps=1e-7):
        super(FocalLoss, self).__init__()
        self.gamma = gamma
        self.eps = eps
        self.ce = torch.nn.CrossEntropyLoss()

    def forward(self, input, target):
        logp = self.ce(input, target)
        p = torch.exp(-logp)
        loss = (1 - p) ** self.gamma * logp
        return loss.mean()
```

얼굴 인식 모델 - Train (AI HUB Dataset)

```
from __future__ import print_function
import os
import torch
from torch.utils import data
import torch.nn.functional as F
import torchvision
import torch
import numpy as np
import random
import time
from torch.nn import DataParallel
from torch.optim.lr_scheduler import StepLR
from test import *

def save_model(model, save_path, name, iter_cnt):
    save_name = os.path.join(save_path, name + '_' + str(iter_cnt) + '.pth')
    torch.save(model.state_dict(), save_name)
    return save_name

    opt = Config() # 파라미터값 정의한 클래스 불러오기

    print('{} train iters per epoch:'.format(len(train_loader)))

    # loss 종류 2개
    if opt.loss == 'focal_loss':
        criterion = FocalLoss(gamma=2)
    else:
        criterion = torch.nn.CrossEntropyLoss()

    # backbone 종류 3개
    if opt.backbone == 'resnet18':
        model = resnet_face18(use_se=opt.use_se)
    elif opt.backbone == 'resnet34':
        model = resnet34()
    elif opt.backbone == 'resnet50':
        model = resnet50()

    # metric 종류 4개
    if opt.metric == 'add_margin':
        metric_fc = AddMarginProduct(512, opt.num_classes, s=30, m=0.35)
    elif opt.metric == 'arc_margin':
        metric_fc = ArcMarginProduct(512, opt.num_classes, s=30, m=0.5,
easy_margin=opt.easy_margin)
    elif opt.metric == 'sphere':
```

```
        metric_fc = SphereProduct(512, opt.num_classes, m=4)
    else:
        metric_fc = nn.Linear(512, opt.num_classes)

    #print(model) # 모델 출력
    model.to(device)
    model = DataParallel(model)
    metric_fc.to(device)
    metric_fc = DataParallel(metric_fc)

    # optimizer 종류 2개
    if opt.optimizer == 'sgd':
        optimizer = torch.optim.SGD([{'params': model.parameters()}, {'params': metric_fc.parameters()}],
                                    lr=opt.lr, weight_decay=opt.weight_decay)
    else:
        optimizer = torch.optim.Adam([{'params': model.parameters()}, {'params': metric_fc.parameters()}],
                                     lr=opt.lr, weight_decay=opt.weight_decay)
    scheduler = StepLR(optimizer, step_size=opt.lr_step, gamma=0.1)

    ## 위까지는 파라미터 설정값 실제로 불러오는 작업

    start = time.time()
    for i in range(opt.max_epoch):
        scheduler.step()

        model.train()
        for ii, (data, label) in enumerate(train_loader): # index, train_loader(image, label)
            data, label = data.to(device), torch.tensor(list(label)).to(device) # label.to(device)
# label = label.to(device).long()

            feature = model(data) # ResNetFace에서 임베딩 벡터 추출
            output = metric_fc(feature, label) # Arcface에서 얼굴 특징 벡터와 대응하는 레이블 간의 유사도를 나타내는 값
            loss = criterion(output, label)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            iters = i * len(train_loader) + ii

            if iters % opt.print_freq == 0:
                output = output.data.cpu().numpy()
                output = np.argmax(output, axis=1)
```

```
                label = label.data.cpu().numpy()
                # print(output)
                # print(label)
                acc = np.mean((output == label).astype(int))
                speed = opt.print_freq / (time.time() - start)
                time_str = time.asctime(time.localtime(time.time()))
                print('{} train epoch {} iter {} {} iters/s loss {} acc {}'.format(time_str, i, ii,
speed, loss.item(), acc))

                start = time.time()

        if i % opt.save_interval == 0 or i == opt.max_epoch:
            save_model(model, opt.checkpoints_path, opt.backbone, i)

        model.eval()

        acc = kface_test(model, img_paths, identity_list, opt.valid_list, opt.test_batch_size)
```

```
64 train iters per epoch:
Mon Apr 24 07:32:35 2023 train epoch 0 iter 0 354.19510057178854 iters/s loss 21.25853157043457 acc 0.0
Mon Apr 24 07:32:56 2023 train epoch 1 iter 36 4.775952388139594 iters/s loss 18.24370002746582 acc 0.0
Mon Apr 24 07:33:17 2023 train epoch 3 iter 8 4.930049211936328 iters/s loss 13.43145751953125 acc 0.0
Mon Apr 24 07:33:37 2023 train epoch 4 iter 44 4.8499074577764 iters/s loss 10.012166023254395 acc 0.02054794520547945
Mon Apr 24 07:33:58 2023 train epoch 6 iter 16 4.848468314683477 iters/s loss 3.736023426055908 acc 0.3561643835616438
Mon Apr 24 07:34:18 2023 train epoch 7 iter 52 4.920543152458291 iters/s loss 1.0647798776626587 acc 0.7123287671232876
Mon Apr 24 07:34:39 2023 train epoch 9 iter 24 4.91975388562596 iters/s loss 0.12075347453355789 acc 0.910958904109589
Mon Apr 24 07:34:59 2023 train epoch 10 iter 60 4.917016876149931 5 iters/s loss 0.11994481831789017 acc 0.9041095890410958
Mon Apr 24 07:35:20 2023 train epoch 12 iter 32 4.75400094594815 iters/s loss 0.07194720208644867 acc 0.9246575342465754
Mon Apr 24 07:35:40 2023 train epoch 14 iter 4 4.906543668664931 iters/s loss 0.0208061579614877 7 acc 0.9452054794520548
Mon Apr 24 07:36:01 2023 train epoch 15 iter 40 4.836103264190755 iters/s loss 0.09412864595651627 acc 0.9315068493150684
Mon Apr 24 07:36:21 2023 train epoch 17 iter 12 4.907705090814514 iters/s loss 0.06568977981805801 acc 0.9383561643835616
Mon Apr 24 07:36:42 2023 train epoch 18 iter 48 4.903024583355822 iters/s loss 0.07742913067340851 acc 0.9452054794520548
Mon Apr 24 07:37:02 2023 train epoch 20 iter 20 4.907231901369027 iters/s loss 0.0860198512673378 acc 0.9315068493150684
Mon Apr 24 07:37:23 2023 train epoch 21 iter 56 4.758554231367555 iters/s loss 0.03612002357840538 acc 0.958904109589041
Mon Apr 24 07:37:44 2023 train epoch 23 iter 28 4.900381208939938 iters/s loss 0.03355381265282631 acc 0.952054794520548
Mon Apr 24 07:38:04 2023 train epoch 25 iter 0 4.8271004205222985 iters/s loss 0.25968384742736816 acc 0.910958904109589
Mon Apr 24 07:38:25 2023 train epoch 26 iter 36 4.8989537870939355 iters/s loss 0.011474824510514736 acc 0.9657534246575342
Mon Apr 24 07:38:45 2023 train epoch 28 iter 8 4.900563109134022 iters/s loss 0.04932386800646782 acc 0.9452054794520548
Mon Apr 24 07:39:05 2023 train epoch 29 iter 44 4.898437031844995 5 iters/s loss 0.06299326568841934 acc 0.9246575342465754
Mon Apr 24 07:39:26 2023 train epoch 31 iter 16 4.816665658769553 iters/s loss 0.10155296325683594 acc 0.910958904109589
Mon Apr 24 07:39:47 2023 train epoch 32 iter 52 4.8301288277263295 iters/s loss 0.1694885641336441 acc 0.9315068493150684
Mon Apr 24 07:40:08 2023 train epoch 34 iter 24 4.823258056364437 iters/s loss 0.07399053126573563 acc 0.9383561643835616
Mon Apr 24 07:40:28 2023 train epoch 35 iter 60 4.890249773731578 iters/s loss 0.05139030888676643 4 acc 0.9452054794520548
Mon Apr 24 07:40:49 2023 train epoch 37 iter 32 4.89268149077482 iters/s loss 0.09810648113489151 acc 0.9246575342465754
Mon Apr 24 07:41:09 2023 train epoch 39 iter 4 4.901529519291099 iters/s loss 0.0365196093916893 acc 0.9178082191780822
Mon Apr 24 07:41:29 2023 train epoch 40 iter 40 4.906470832577695 iters/s loss 0.1776903122663498 acc 0.9315068493150684
Mon Apr 24 07:41:50 2023 train epoch 42 iter 12 4.746745045734584 iters/s loss 0.03792773187160492 acc 0.952054794520548
Mon Apr 24 07:42:11 2023 train epoch 43 iter 48 4.828626128641209 5 iters/s loss 0.18628177046775818 acc 0.9246575342465754
Mon Apr 24 07:42:32 2023 train epoch 45 iter 20 4.896048056143072 iters/s loss 0.14679822325706482 acc 0.9178082191780822
Mon Apr 24 07:42:52 2023 train epoch 46 iter 56 4.903471165131584 5 iters/s loss 0.03656154125928879 acc 0.952054794520548
Mon Apr 24 07:43:12 2023 train epoch 48 iter 28 4.898041300341083 iters/s loss 0.04933818802237511 acc 0.958904109589041
```

사용자 얼굴 임베딩 벡터 저장

```
# 사용자 등록 시 test_image_path 경로에 이미지를 저장
# 사용자 사진 n개 서버(path)에 다 올리고 나서 OK 버튼 누르면 아래 코드 실행 (한번 실행)
# 사용자 등록 시 사용자의 임베딩 벡터를 저장하는 코드

test_image_path = sorted(glob.glob('test/*.jpg'))

for path in test_image_path:
        image = cv2.imread(path, cv2.IMREAD_COLOR)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY).astype(np.float32)

        obj = RetinaFace.detect_faces(path)

        for key in obj.keys():
    identity = obj[key]

    facial_area = identity["facial_area"]

    #cv2.rectangle(image,  (facial_area[2],  facial_area[3]),  (facial_area[0],facial_area[1]),
(255,255,255),1)

                # 이미지 자르기
                crop = image[90:252,97:216] # 1 : 3 , 0 : 2

          # resnetface 모델에 넘기기
                feature = model(crop)

                # txt 파일에 임베딩 벡터 저장
                f = open("vector.txt","w+")
                f.write(feaure, "₩n")

f.close()
```

사용자 일치 여부 판별

```
def cos_similarity(f1, f2): # f : feature
    return dot(f1, f2)/(norm(f1)*norm(f2))

def isUser(img_path):
    visitor = cv2.imread(path, cv2.IMREAD_COLOR)
    visitor = cv2.cvtColor(visitor,cv2.COLOR_BGR2GRAY).astype(np.float32)

        obj = RetinaFace.detect_faces(img_path)

        for key in obj.keys():
        identity = obj[key]
        facial_area = identity["facial_area"]

        # 이미지 자르기
        crop = visitor[facial_area[1]: facial_area[3],facial_area[0]:facial_area[2]] # 1 : 3 , 0 :
2

        # resnetface 모델에 넘기기
        feature = model(crop)

        # txt 파일 임베딩 벡터 읽기
        f = open("vector.txt")
    features = f.readlines()

    count = 0
```

```
    for v in features:
        v = v.strip()
        if cos_similarity(v, feature)) > 0.7: # threshold
            count += 1

    if count >= 4:
        return 1 # 동일 인물이라 판정
    else:
        return 0 # 타인이라 판정

# 추가 고려
# 동일 인물일 경우 vector에 append 이번 feature (강화학습)
# 타인일 경우, others에 append 이번 feature
f.close()
```

화자 인식 모델

화자 인식 모델 – 데이터 로드

```
import librosa
import wave
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
import os

CHANNELS = 1
RATE = 44100 # bit rate
CHUNK = int(RATE / 10) # 버퍼 사이즈 1초당 44100 bit rate -> 100ms 단위
RECORD_SECONDS = 5 # 녹음할 시간 설정
WAVE_OUTPUT_FILENAME = "output.wav"
DATA_PATH = "./data/"
X_train = [] #train_data 저장할 공간
X_test = []
Y_train = []
Y_test = []
tf_classes = 0
def load_wave_generator(path):

    batch_waves = []
    labels = []
    X_data = []
    Y_label = []
    global X_train, X_test, Y_train, Y_test, tf_classes

    folders = list(set(os.listdir(path)) - set(['.ipynb_checkpoints']))
    #print(folders)
    for folder in folders:
```

```
        if not os.path.isdir(path):continue
        # if folder == '.ipynb_checkpoints' : continue
        files = os.listdir(path+"/"+folder)
        print("Foldername :",folder,"-",len(files),"파일")
        for wav in files:
            if not wav.endswith(".wav"):continue
            else:
                print("Filename :",wav) #.wav 파일이 아니면 continue
                y, sr = librosa.load(path+"/"+folder+"/"+wav)
                mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=128,
hop_length=int(sr*0.01),n_fft=int(sr*0.02)).T
                # print(mfcc)
                # print("SHAPE", mfcc.shape)
                X_data.extend(mfcc)
               # print(len(mfcc))

                label = [0 for i in range(len(folders))]
                label[tf_classes] = 1

                for i in range(len(mfcc)):
                    Y_label.append(label)
                #print(Y_label)
        tf_classes = tf_classes+1
        print(tf_classes)
    #end loop
    print("X_data :",np.shape(X_data))
    print("Y_label :",np.shape(Y_label))
    X_train, X_test, Y_train, Y_test = train_test_split(np.array(X_data), np.array(Y_label))
    xy = (X_train, X_test, Y_train, Y_test)
    np.save("./data.npy",xy)
load_wave_generator(DATA_PATH)

print(tf_classes," classes")
print("X_train :",np.shape(X_train))
print("Y_train :",np.shape(Y_train))
print("X_test :",np.shape(X_test))
print("Y_test :",np.shape(Y_test)) # one-hot encoding, 해당하는 클래스 인덱스에 1, 그 외 0
```

화자 인식 모델 – Neural Network (1)

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
X_train, X_test, Y_train, Y_test = np.load("./data.npy",allow_pickle=True)
X_train = X_train.astype("float")
X_test = X_test.astype("float")
tf.reset_default_graph()
tf.random.set_random_seed(777) #set_seed
```

```
learning_rate = 0.001
training_epochs = 150
keep_prob = tf.placeholder(tf.float32)
sd = 1 / np.sqrt(128) # default 20

X = tf.placeholder(tf.float32, [None, 128])
Y = tf.placeholder(tf.float32, [None, tf_classes])
# W = tf.Variable(tf.random_normal([216, 200]))
# b = tf.Variable(tf.random_normal([200]))
# hidden layer 1
W1 = tf.get_variable("w1",
    #tf.random_normal([216, 180], mean=0, stddev=sd),
        shape=[128, 256],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256], mean=0, stddev=sd), name="b1")
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 = tf.nn.dropout(L1, keep_prob = keep_prob)
# hidden layer 2
W2 = tf.get_variable("w2",
    #tf.random_normal([180, 150], mean=0, stddev=sd),
         shape=[256, 256],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([256], mean=0, stddev=sd), name="b2")
L2 = tf.nn.tanh(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob = keep_prob)
# hidden layer 3
W3 = tf.get_variable("w3",
    #tf.random_normal([150, 100], mean=0, stddev=sd),
            shape=[256, 256],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([256], mean=0, stddev=sd), name="b3")
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
L3 = tf.nn.dropout(L3, keep_prob = keep_prob)
# hidden layer 4
W4 = tf.get_variable("w4",
    #tf.random_normal([100, 50], mean=0, stddev=sd),
             shape=[256, 128],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([128], mean=0, stddev=sd), name="b4")
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4) # 4차 히든레이어는 'Relu' 함수를 쓴다.
L4 = tf.nn.dropout(L4, keep_prob = keep_prob)
```

```
# hidden layer 5
W5 = tf.get_variable("w5",
    #tf.random_normal([100, 50], mean=0, stddev=sd),
            shape=[128, 128],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([128], mean=0, stddev=sd), name="b5")
L5 = tf.nn.relu(tf.matmul(L4, W5) + b5) # 5차 히든레이어는 'Relu' 함수를 쓴다.
L5 = tf.nn.dropout(L5, keep_prob = keep_prob)
# hidden layer 6
W6 = tf.get_variable("w6",
    #tf.random_normal([100, 50], mean=0, stddev=sd),
            shape=[128, 128],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b6 = tf.Variable(tf.random_normal([128], mean=0, stddev=sd), name="b6")
L6 = tf.nn.relu(tf.matmul(L5, W6) + b6) # 6차 히든레이어는 'Relu' 함수를 쓴다.
L6 = tf.nn.dropout(L6, keep_prob = keep_prob)
# hidden layer 7
W7 = tf.get_variable("w7",
    #tf.random_normal([100, 50], mean=0, stddev=sd),
            shape=[128, 128],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b7 = tf.Variable(tf.random_normal([128], mean=0, stddev=sd), name="b7")
L7 = tf.nn.relu(tf.matmul(L6, W7) + b7) # 7차 히든레이어는 'Relu' 함수를 쓴다.
L7 = tf.nn.dropout(L7, keep_prob = keep_prob)
# output layer
W8 = tf.get_variable("w8",
    #tf.random_normal([50, tf_classes], mean=0, stddev=sd),
           shape=[128, tf_classes],
                    initializer = tf.truncated_normal_initializer(stddev=0.1))
                    # initializer=tf.contrib.layers.xavier_initializer())
b8 = tf.Variable(tf.random_normal([tf_classes], mean=0, stddev=sd), name="b8")
hypothesis = tf.matmul(L7, W8) + b8
#cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
#optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
batch_size=1
x_len = len(X_train)

if(x_len%2==0):
```

```
    batch_size = 2
elif(x_len%3==0):
    batch_size = 3
elif(x_len%4==0):
    batch_size = 4
else:
    batch_size = 1
split_X = np.split(X_train,batch_size)
split_Y = np.split(Y_train,batch_size)
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for epoch in range(training_epochs):
    avg_cost = 0
    for i in range(batch_size):
        batch_xs = split_X[i]
        batch_ys = split_Y[i]
        feed_dict = {X:batch_xs, Y:batch_ys, keep_prob: 0.7}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / batch_size

    print('Epoch:', '%04d' % (epoch), 'cost =', '{:.9f}'.format(avg_cost))
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy: ", sess.run(accuracy, feed_dict={X: X_test, Y:Y_test, keep_prob:1}))
print('Learning Finished!')
```

●

화자 인식 모델 – Neural Network (2)

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout

def create_speaker_recognition_model(input_shape, num_classes):
    model = Sequential([
        Flatten(input_shape=input_shape),  # Flatten the input for dense layers
        Dense(256, activation='relu'),
        Dropout(0.5),  # Dropout layer to reduce overfitting
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Output layer with softmax activation for
classification
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
# Input shape for the model (should match the shape of your MFCC features)
input_shape = X_train[0].shape
# Number of classes (speakers) in the dataset
num_classes = tf_classes
# Create the model
speaker_model = create_speaker_recognition_model(input_shape, num_classes)
# Train the model
batch_size = 8
epochs = 200  # Adjust the number of epochs as needed
X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)
Y_test = np.array(Y_test)
speaker_model.fit(X_train, Y_train, validation_data=(X_test, Y_test), batch_size=batch_size, epochs=epochs)
# Evaluate the model
loss, accuracy = speaker_model.evaluate(X_test, Y_test)
print('Test accuracy:', accuracy)
```

멀티 모달 모델

음성&이미지 멀티 모달 모델 - 데이터셋 구축 (1) voice

```
import glob
import json
import pandas as pd
path = glob.glob("/workspace/CAPS2/speech/sample/label/*/*/*/*.json")
female_list = []
male_list = []
for j in path:
    with open(j, 'r') as f:
        js = json.load(f)

    name = js["Speaker"]["SpeakerName"]

    if js["Speaker"]["Gender"] == "Female":
        female_list.append(name)
    else:
        male_list.append(name)
female_list = list(set(female_list))
male_list = list(set(male_list))
f_df = pd.DataFrame(female_list, columns=['voice_id'])
f_df['gender'] = "Female"
m_df = pd.DataFrame(male_list, columns=['voice_id'])
m_df['gender'] = "Male"
df = pd.concat([f_df, m_df], axis=0)
print(df)
```

```
print("=======TOTAL========")
print(f"여자 음성: {len(female_list)}개")
print(f"남자 음성: {len(male_list)}개")
df.to_csv("../CSV/voice_list.csv", index=False)
```

음성&이미지 멀티 모달 모델 - 데이터셋 구축 (2) face

```
import glob
import json
import pandas as pd
path = glob.glob("/workspace/CAPS2/face/face_parsing/label/*/*.json")
female_list = []
male_list = []
for j in path:
    with open(j, 'r') as f:
        js = json.load(f)

    name = js["id"]
    # print(js["gender"])
    if js["gender"] == "female":
        female_list.append(name)
    else:
        male_list.append(name)
female_list = list(set(female_list))
male_list = list(set(male_list))
f_df = pd.DataFrame(female_list, columns=['face_id'])
f_df['gender'] = "Female"
m_df = pd.DataFrame(male_list, columns=['face_id'])
m_df['gender'] = "Male"
df = pd.concat([f_df, m_df], axis=0)
print(df)
print("=======TOTAL========")
print(f"여자 얼굴: {len(female_list)}개")
print(f"남자 얼굴: {len(male_list)}개")
df.to_csv("../CSV/face_list.csv", index=False)
```

음성&이미지 멀티 모달 모델 - 데이터셋 구축 (3) voice&face

```
voice_list = pd.read_csv("voice_list.csv")
face_list = pd.read_csv("face_list.csv")

female_data = min(len(voice_list[voice_list["gender"] == "Female"]), len(face_list[face_list["gender"] == "Female"]))
male_data = min(len(voice_list[voice_list["gender"] == "Male"]), len(face_list[face_list["gender"] == "Male"]))

female_df = pd.DataFrame([], columns = ["face_id", "voice_id", "gender"])
for i in range(female_data):
    face_id, f_gender = face_list[face_list["gender"] == "Female"].iloc[i]["face_id"], face_list[face_list["gender"] == "Female"].iloc[i]["gender"]
    voice_id, v_gender = voice_list[voice_list["gender"] == "Female"].iloc[i]["voice_id"], voice_list[voice_list["gender"] == "Female"].iloc[i]["voice_id"]
    # print(face_id, voice_id)

    df = pd.DataFrame([[face_id, voice_id, "Female"]], columns = ["face_id", "voice_id", "gender"])
    # print(df)
    female_df = pd.concat([female_df, df], ignore_index = True)
#print(female_df)

male_df = pd.DataFrame([], columns = ["face_id", "voice_id", "gender"])
for i in range(male_data):
    face_id, f_gender = face_list[face_list["gender"] == "Male"].iloc[i]["face_id"], face_list[face_list["gender"] == "Male"].iloc[i]["gender"]
    voice_id, v_gender = voice_list[voice_list["gender"] == "Male"].iloc[i]["voice_id"], voice_list[voice_list["gender"] == "Male"].iloc[i]["voice_id"]
    # print(face_id, voice_id)
    df = pd.DataFrame([[face_id, voice_id, "Male"]], columns = ["face_id", "voice_id", "gender"])
    # print(df)
    male_df = pd.concat([male_df, df], ignore_index = True)
#print(male_df)

# 여자, 남자 데이터 합치기
new_dataset = pd.concat([male_df, female_df], ignore_index = True)
new_dataset.to_csv("../CSV/face&voice_match_list.csv", index=False)
```

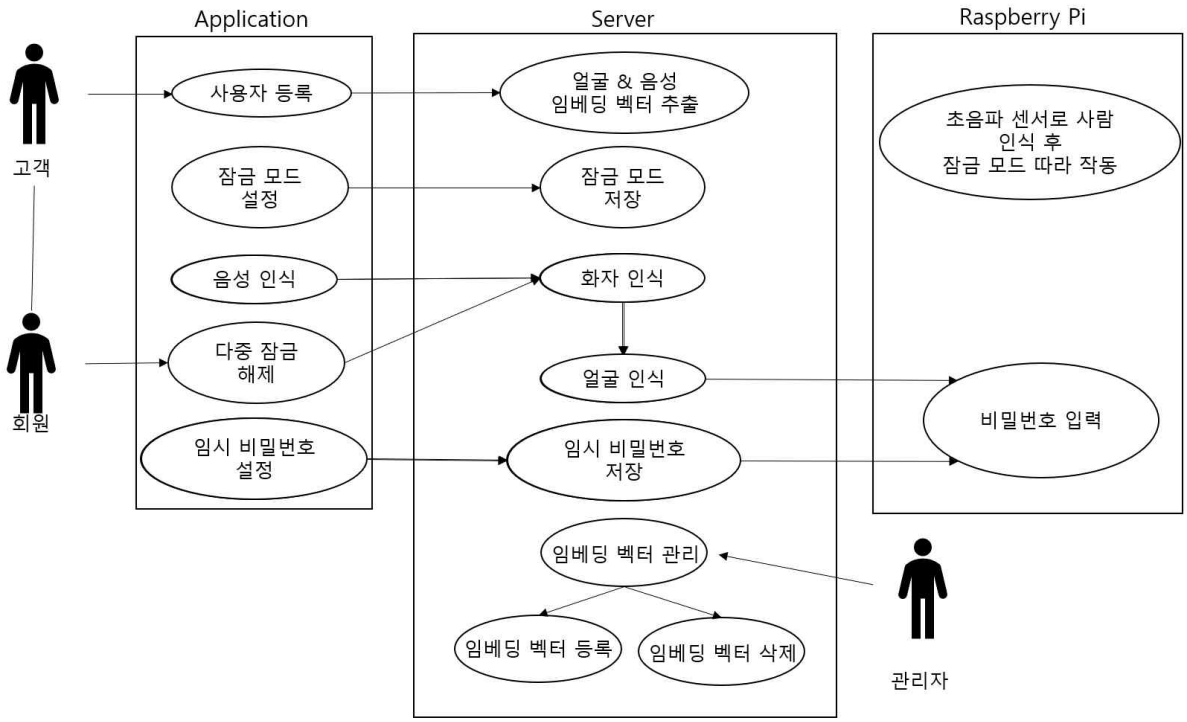
- **시스템(하드웨어, 시스템)구성도, 또는 소프트웨어 아키텍처**

1. 서버
   - 모델 저장 : 한국인 안면 데이터로 훈련된 얼굴 인식 모델, 화자 인식 데이터로 훈련된 화자 인식 모델 저장
   - 코드 저장 : 저장된 사용자의 음성&얼굴 임베딩 벡터를 이용하여 본인 일치 여부를 확인하고 동일인물인 경우 이중 잠금 상태였던 도어락의 잠금 상태를 일차적으로 해제
   - 잠금 모드 저장 및 확인: 사용자가 설정한 잠금 모드를 '잠금 모드 설정 서버'를 통해 저장하고, 사용자가 도어락 잠금 해제 시도 시 '잠금 모드 확인 서버'를 통해 활성화된 잠금 모드를 전달한다.
   - 임시 비밀번호 관리: 사용자가 임시 비밀번호 모드를 활성화하면 시간 정보와 비밀번호 해시값을 전달받는다. 이후, 방문자가 임시 비밀번호로 도어락 잠금 해제할 경우에 방문자가 입력한 비밀번호 해시값과 비교하여 일치 여부를 판별한다.

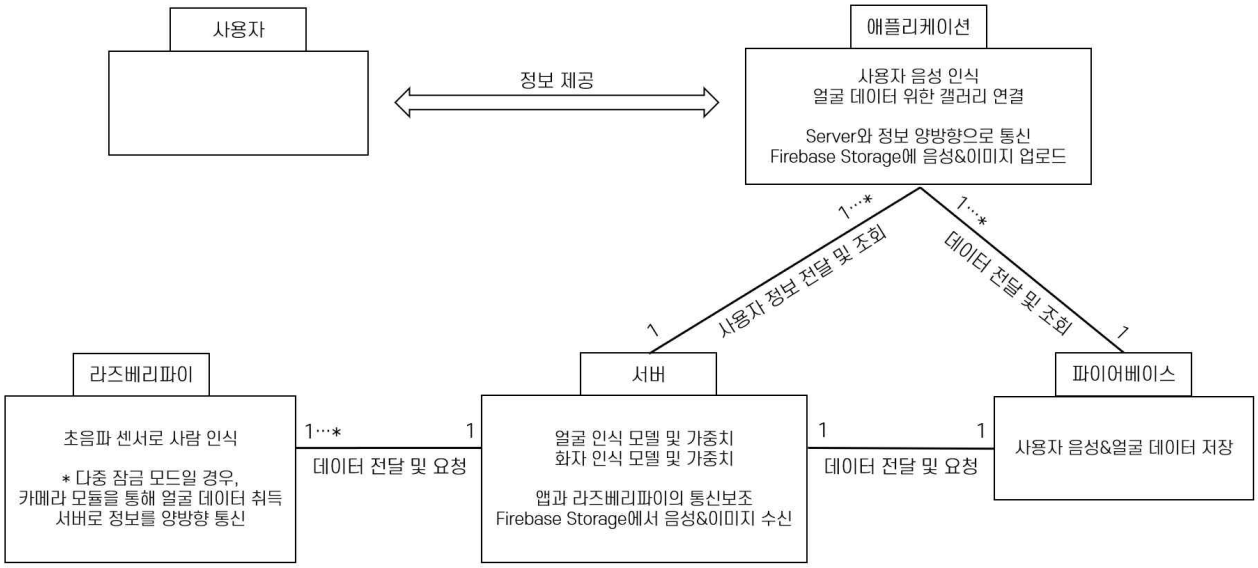
2. 데이터 비교 알고리즘 :

   입력된 음성 및 사진에서 임베딩 벡터를 추출해 서버에 저장되어있는 사용자의 임베딩 벡터와의 유사도를 계산하여 임계치보다 높을 때 동일 인물로 판단

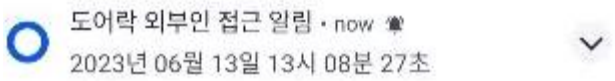
- **기능별 상세 요구사항(또는 유스케이스)**

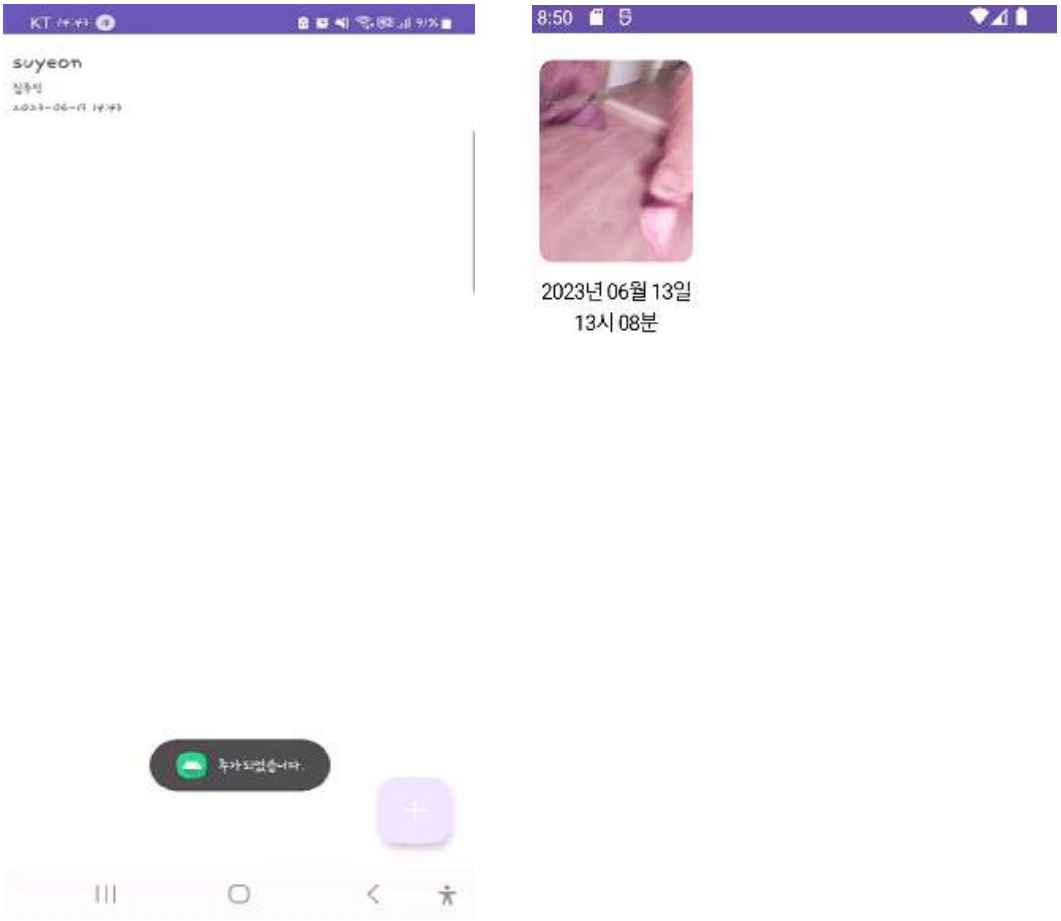


- **설계 모델(클래스 다이어그램, 클래스 및 모듈 명세서)**

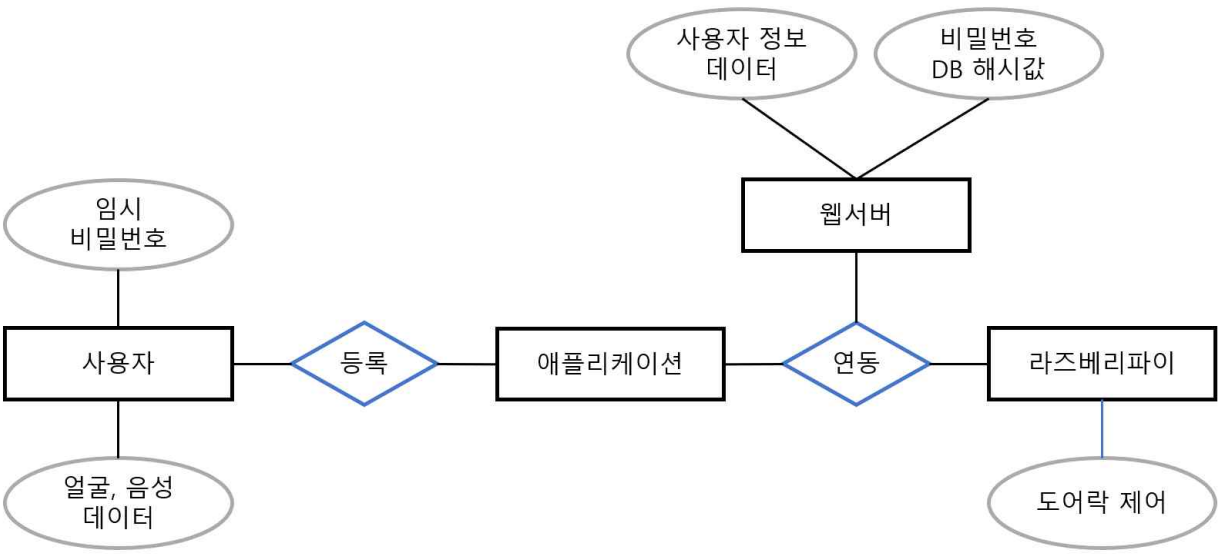


- UI 분석/설계 모델/프로토타입





- E-R 다이어그램/DB 설계 모델(테이블 구조)



- **테스트 계획서, 테스트 케이스 기술서 등등**

프로젝트 최종 기능 구현을 위해 팀원 간 애플리케이션, 라즈베리파이, 웹서버, 얼굴 인식 모델 부분으로 나누어 역할 분담을 하였다. 테스트 내용은 다음과 같다.

1) 어플리케이션과 웹서버 간의 사용자 등록 과정

: 사용자의 얼굴 사진과 음성을 어플리케이션을 통해 파이어베이스에 업로드가 완료되면, 웹서버에 사용자 등록 요청을 보낸다. 요청을 받은 웹서버에서는 얼굴 및 음상 임베딩 추출 과정을 거쳐 사용자 정보를 저장한다.

2) 도어락의 초음파 센서 인식 과정
: 초음파 센서를 통해 사람을 인식하면, 네트워크 연결 여부를 확인한다.
네트워크 연결이 없다면 (오프라인) 기본 비밀번호 모드로 도어락을 제어하며, 네트워크 연결이 있다면 웹서버에 잠금 모드 확인 요청을 한다. 이 과정에서 웹서버가 네트워크 연결이 없을 상황을 대비하여 정해진 시간 안에 모드 응답을 받지 못하면 오프라인으로 인식하여 기본 비밀번호 모드를 사용한다.
서버로부터 모드 응답을 받으면 3가지의 잠금 모드 중 해당하는 모드로 잠금 해제 과정을 거친다.

3) 잠금 해제 과정
: 잠금 모드 확인 서버를 통해 전달받은 모드로 잠금 해제 과정이 이루어진다.
잠금 모드는 3가지가 있다. (1) 기본 비밀번호 모드 (2) 다중 잠금 모드 (3) 임시 비밀번호 모드
(1) 기본 비밀번호 모드의 경우는 오프라인에서와 같이 기본적으로 작동하는 비밀번호 모드이다. 도어락 내의 DB에 있는 비밀번호 해시값과 사용자가 키패드에 입력한 비밀번호 해시값을 비교하여 개폐 여부를 결정한다.
(2) 다중 잠금 모드는 얼굴 인식과 화자 인식이 결합된 모드이다.
도어락의 카메라 센서로 사진을 캡쳐하여 파이어베이스에 전송하고, 웹서버에 요청을 보낸다.
요청을 받은 웹서버에서 얼굴 인식 모델로 사용자인지 판별한다.
어플리케이션에서 음성 인식을 거친 음성 파일을 파이어베이스에 전송하고, 웹서버에 요청을 보낸다.
요청을 받은 웹서버에서 화자 인식 모델로 사용자인지 판별한다.
얼굴 인식과 화자 인식 모두 통과해야만 도어락 잠금이 해제된다.
(3) 임시 비밀번호 모드는 방문자에게 일정 시간동안 기존의 비밀번호가 아닌, 랜덤한 임시 비밀번호를 부여하여 잠금 해제를 허락하는 모드이다.
방문자가 도어락의 키패드로 비밀번호를 입력하면 서버 DB에 저장되어 있는 임시 비밀번호 해시값과 비교하여 비밀번호 일치 여부를 확인한 후 도어락 잠금이 해제된다.

4) 방문자(침입자) 어플리케이션 알림
방문자가 도어락 잠금 해제를 시도하여 실패하였을 경우, 카메라 센서를 통해 방문자 사진을 캡쳐하여 파이어베이스에 업로드한다. 그리고 FCM을 통해 사용자의 어플리케이션에 알림을 보내어 방문자의 사진을 확인할 수 있다.

2. 프로젝트 수행을 위해 적용된 추진전략, 수행 방법의 결과를 작성하고, 만일 적용과정에서 문제점이 도출되었다면 그 문제를 분석하고 해결방안을 기술하시오.

- 문제해결을 위해 적용한 방법(또는 기법) 결과, (문제점, 해결방안)

[문제점] - 사용하려는 얼굴 인식 모델이 서양인 안면 데이터로 학습되어 동양인에 대한 성능이 떨어짐
[해결방안] - AI HUB에서 한국인 안면 데이터를 요청하여 사용

[문제점] - 리액트 네이티브 관련 에뮬레이터 및 버전 관련하여 여러 가지 오류가 발생하여 많이 시도를 해보았지만 고쳐지지 않음
[해결방안] - 개발 환경을 리액트 네이티브에서 안드로이드 스튜디오로 변경 - 사용 언어로 JAVA를 선택하고자 했으나 안드로이드 스튜디오의 버전이 업그레이드 되면서 지원되는 JAVA 라이브러리가 제한되어 Kotlin언어를 선택

- 팀원의 책임 및 역할 수행에 대한 결과, (문제점, 해결방안)

[문제점] - 팀원 전원이 프로젝트를 진행해본 경험이 없음 - 팀원 간의 개인 역량이 다름
[해결방안] - 주기적으로 진행 상황을 보고하며 목표를 유연하게 수정 - 회의 때 서로 필요한 부분 피드백

- 프로젝트 일정계획에 맞추지 못한 경우의 문제점, 해결 방안

[문제점] - 계획된 일정보다 기능 구현이 늦어지는 문제점 발생
[해결방안] - 주기적으로 프로젝트 일정계획 리마인드 - 팀원 개인 역할분담에 있어 계획이 변경되는 상황에는 일정계획을 수정하여 공유

- 요구사항 변경관리의 결과, (문제점, 해결 방안) 등등.

[문제점] - 프로젝트와 관련된 경험 및 지식이 부족해 어떠한 방향으로 수정해야 할지 잘 알지 못함
[해결방안] - 교수님과의 미팅을 통해 요구사항 수집 - 관련 서적 및 웹서핑을 참고하여 공부하고 적용하기 - 매주 한 주 동안의 결과 공유 및 상호 피드백

[별첨1]

프로젝트명 : 생체 인식을 활용한 스마트 도어락

소프트웨어 요구사항 정의서

Version 1.0

개발 팀원 명(팀리더): 구나영
고수연
송가은
대표 연락처: 010-4259-0466
e-mail: 20191399@edu.hanbat.ac.kr

목차

요구사항 정의서에 사용되는 양식 설명

요구사항 고유번호(ID): 제안요청서에 정의된 요구사항에 대해 계약, 사업수행, 사업완료 및 검수까지 변경, 삭제, 수정 여부에 대한 추적관리를 위해 고유의 번호를 부여하도록 한다.

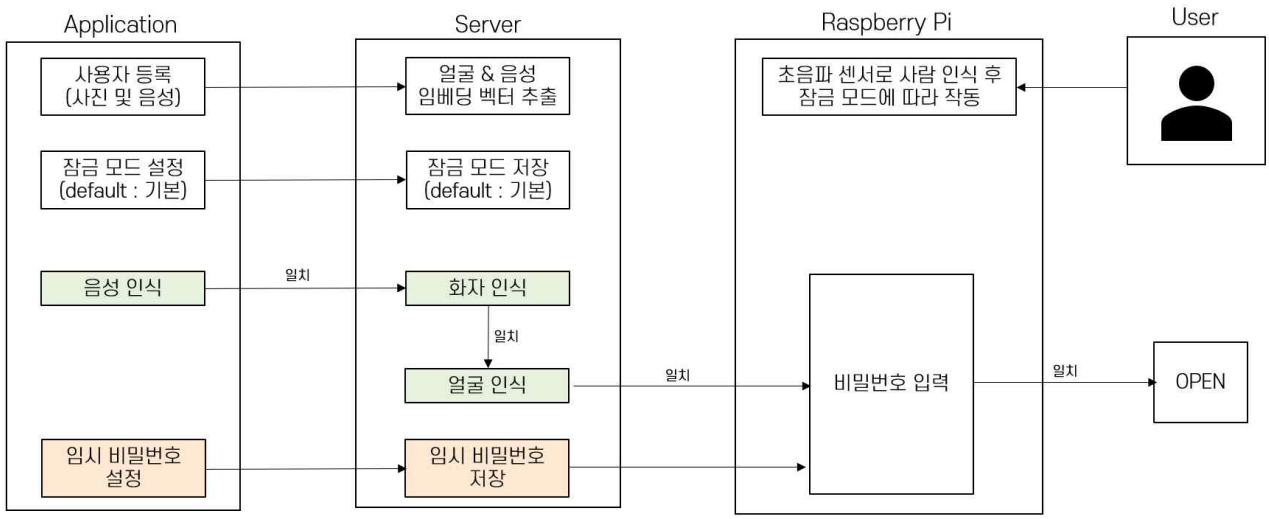
요구사항 구분 및 ID부여 규칙

요구사항 구분		ID 부여 규칙
시스템 장비 구성 요구사항	Equipment Composition Requirement	ECR-000
기능 요구사항	System Function Requirement	SFR-000
성능 요구사항	Performance Requirement	PER-000
인터페이스 요구사항	System Interface Requirement	SIR-000
데이터 요구사항	Data Requirement	DAR-000
테스트 요구사항	Test Requirement	TER-000
보안 요구사항	Security Requirement	SER-000
품질요구사항	Quality Requirement	QUR-000
제약사항	Constraint Requirement	COR-000
프로젝트 관리 요구사항	Project Mgmt. Requirement	PMR-000
프로젝트 지원 요구사항	Project Support Requirement	PSR-000

요구사항 세부내용 작성표 양식 및 항목설명

요구사항 고유번호		(설명) 요구사항 추적관리를 위해 독립적인 고유번호(ID) 부여
요구사항 명칭		(설명) 요구사항 명칭을 작성함
요구사항 분류		(설명) 요구사항 분류기준에 따른 분류를 기입
요구사항 상세 설명	**정의**	(설명) 요구사항 정의
	세부 내용	(설명) 요구사항 구체적인 세부 내용을 설명
산출정보		(설명) 해당기능을 통해 산출되는 결과물 혹은 정보를 표기
관련 요구사항		(설명) 정의된 요구사항과 관련된 요구사항에 대해 기술
요구사항 출처		(설명) 기능 도출내용에 대한 출처(source) 표기

1. 시스템 개요



2. 시스템 장비 구성요구사항

요구사항 고유번호		ECR-001		
요구사항 명칭		라즈베리파이		
요구사항 분류		시스템 장비 구성 요구사항	응락수준	필수
요구사항 상세설명	정의	센서 인식 및 도어락 핵심 기능 구현		
	세부 내용	- 장비 품목 : 라즈베리파이4 B+ - 장비 수량 : 1개 - 장비 기능 : 초음파 센서로 사람을 인식하여 도어락 잠금 모드에 맞게 잠금 해제를 시도한다. 어플리케이션과 서버에서 전달된 도어락 핵심 기능을 구현하기 위한 장치이다. - 장비 성능 및 특징: 1.5GHz 64비트 쿼드 코어 ARM Cortex-A72 CPU, 2.4/5.0 GHz IEEE 802.11ac 무선, 블루투스 5.0, Gigabit Ethernet, USB 3.0		

요구사항 고유번호		ECR-002		
요구사항 명칭		웹서버		
요구사항 분류		시스템 장비 구성 요구사항	응락수준	필수
요구사항 상세설명	정의	- 어플리케이션(사용자)와 도어락(라즈베리파이) 사이의 통신		
	세부 내용	- 장비 품목 : Flask - 장비 수량 : 1개 - 장비 기능 : 어플리케이션으로 전달받은 사용자 정보를 저장하고 인식하여 도어락 잠금 해제 여부를 결정한다. 임시 비밀번호 정보를 DB에 저장하여 사용자가 입력한 비밀번호와 해시값 비교를 한다. - 장비 성능 및 특징 : Python 언어로 구현되는 Flask 서버, 코드를 관리하기 쉽도록 Blueprint로 나누었다.		

요구사항 고유번호		ECR-003		
요구사항 명칭		애플리케이션		
요구사항 분류		시스템 장비 구성 요구사항	응락수준	필수
요구사항 상세설명	정의	- 애플리케이션 UI, 서버 연동		
	세부 내용	- 장비 품목 : Android studio(language : kotlin) - 장비 수량 : 1개 - 장비 기능 : 사용자 정보 전달, 라즈베리파이와의 통신 - 장비 성능 및 특징 : 사용자 인증을 위한 사진, 음성정보와 사용자 정보를 서버에게 전달 및 라즈베리파이와 소통을 위해 서버에게 정보 전달, 도어락 모드 변경 전달을 라즈베리파이와 소통을 위한 서버에게 정보 전달		

요구사항 고유번호		ECR-004		
요구사항 명칭		얼굴 인식		
요구사항 분류		시스템 장비 구성 요구사항	응락수준	필수
요구사항 상세설명	정의	- 얼굴 임베딩 벡터 추출 및 비교 알고리즘		
	세부 내용	- 장비 품목 : Python 코드 - 장비 수량 : 1개 - 장비 기능 : 인물 판별 - 장비 성능 및 특징 : 라즈베리파이에서 촬영된 사진에서 얼굴 영역 검출 및 얼굴 임베딩 벡터 추출을 수행하고, 추출된 벡터를 인자값으로 입력하여 등록된 사용자와의 얼굴 일치 여부를 확인하는 코드		

요구사항 고유번호		ECR-004		
요구사항 명칭		화자 인식		
요구사항 분류		시스템 장비 구성 요구사항	응락수준	필수
요구사항 상세설명	정의	- 화자 임베딩 벡터 추출 및 비교 알고리즘		
	세부 내용	- 장비 품목 : Python 코드 - 장비 수량 : 1개 - 장비 기능 : 인물 판별 - 장비 성능 및 특징 : 애플리케이션에서 음성 인식 API를 사용하여 랜덤 문장과 일치하는 경우, 음성 파일을 서버로 전송하여 음성 임베딩 벡터를 추출한 뒤, 추출된 벡터를 인자값으로 입력하여 등록된 사용자와의 음성 일치 여부를 확인하는 코드		

3. 기능 요구사항

요구사항 고유번호		SFR-001		
요구사항 명칭		방문자 사진 촬영		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	방문자 촬영하여 서버에 전달		
	세부 내용	- 다중 잠금 모드일 때 활용된다. - 도어락과 연결된 라즈베리파이에서 초음파 센서를 통해 사람을 인식하고, 카메라 촬영 후 서버에 전달 - 서버에 전달된 사진은 방문자인지 외부인인지 판단하여 외부인일 경우, 화자 인식이 되더라도 잠금 해제되지 않음 (다중 잠금) - 외부인 사진을 사용자가 확인할 수 있도록 애플리케이션에 전달 - 사용언어 : Python 3.7.3		

요구사항 고유번호		SFR-002		
요구사항 명칭		사용자 정보 전달		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	사용자 정보를 서버에 전달		
	세부 내용	- 애플리케이션에서 사용자의 이름과 사진 및 음성 파일을 서버에 전달. - 서버에 위치한 얼굴 인식 모델이 전달받은 사용자 사진 정보를 활용하여 학습. - 이후 라즈베리 파이의 카메라를 통해 서버에 전달받은 사용자 얼굴 캡쳐 파일이 사용자와 유사한지 판단. - 서버에 위치한 화자 인식 모델이 전달받은 사용자 사진 정보를 활용하여 학습. - 이후 애플리케이션 내에서 음성인식기술로 오늘의 단어 확인 후 음성파일을 파이어베이스 스토리지에 올려 화자 인식 기술을 활용해 화자 구분. - 사용언어 : Kotlin		

요구사항 고유번호		SFR-003		
요구사항 명칭		웹서버와 소통		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	서버를 통해 라즈베리파이와 소통		
	세부 내용	- 다중 잠금, 도어락 원격 제어, 임시 비밀번호 부여를 애플리케이션을 통해 실행 - 애플리케이션에서 버튼을 누를 시 binary를 서버에 전달하고 서버는 라즈베리파이에게 전달하여 조건에 맞을 시 각 기능을 수행. - 사용언어 : Kotlin		

요구사항 고유번호		SFR-004		
요구사항 명칭		얼굴 인식		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	얼굴 임베딩 벡터 추출 및 일치 여부 판단 알고리즘		
	세부 내용	- 서버에서 받은 데이터를 입력하여 코드 실행 - 전달된 이미지에 대한 크기 조절 및 얼굴 영역 검출을 통한 얼굴 임베딩 벡터 추출 - 추출된 벡터와 등록된 사용자의 벡터 사이의 cos 유사도를 계산하여 일치 여부 판단 및 결과(0 또는 1) 반환 - 사용언어 : Python 3.10 (OpenCV)		

요구사항 고유번호		SFR-005		
요구사항 명칭		화자 인식		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	음성 임베딩 벡터 추출 및 일치 여부 판단 알고리즘		
	세부 내용	- 서버에서 받은 데이터를 입력하여 코드 실행 - 추출된 벡터와 등록된 사용자의 벡터 사이의 cos 유사도를 계산하여 일치 여부 판단 및 결과(0 또는 1) 반환 - 사용언어 : Python 3.10		

요구사항 고유번호		SFR-006		
요구사항 명칭		방문자 화자 인식		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	방문자 음성을 서버에 전달		
	세부 내용	- 다중 잠금 모드일 때 활용된다. - 애플리케이션에서 음성 인식이 통과가 되면, 웹서버에 음성 파일을 전달한 후 화자 인식을 한다. - 서버에 전달된 사진은 방문자인지 외부인인지 판단하여 외부인일 경우, 화자 인식이 되더라도 잠금 해제되지 않음 (다중 잠금) - 외부인 사진을 사용자가 확인할 수 있도록 애플리케이션에 전달 - 사용언어 : Python 3.7.3		

요구사항 고유번호		SFR-007		
요구사항 명칭		방문자 사진 촬영		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	방문자 촬영하여 서버에 전달		
	세부 내용	- 다중 잠금 모드일 때 활용된다. - 도어락과 연결된 라즈베리파이에서 초음파 센서를 통해 사람을 인식하고, 카메라 촬영 후 서버에 전달 - 서버에 전달된 사진은 방문자인지 외부인인지 판단하여 외부인일 경우, 화자 인식이 되더라도 잠금 해제되지 않음 (다중 잠금) - 외부인 사진을 사용자가 확인할 수 있도록 애플리케이션에 전달 - 사용언어 : Python 3.7.3		

요구사항 고유번호		SFR-008		
요구사항 명칭		임시 비밀번호 설정 및 관리		
요구사항 분류		기능 요구사항	응락수준	필수
요구사항 상세설명	정의	사용자가 임시 비밀번호를 설정하고, 웹서버가 관리한다.		
	세부 내용	- 임시 비밀번호 모드일 때 활용된다. - 사용자가 어플리케이션을 통해 임시 비밀번호 모드를 활성화하면 해시화된 비밀번호가 웹서버에 전달된다. - 임시 비밀번호는 모드 활성화된 시점으로부터 10분간 유효하다. - 외부인이 임시 비밀번호를 입력하면 웹서버에서 해시값을 비교한 후 일치 여부를 판별한다. - 사용언어 : Kotlin, Python 3.7.3		

4. 성능 요구사항

요구사항 고유번호		PER-001		
요구사항 명칭		사진 촬영 전송		
요구사항 분류		성능 요구사항	응락수준	필수
요구사항 상세설명	정의	센서 인식 및 사진 촬영, 전송 시간		
	세부 내용	- 센서 인식 및 사진 촬영 시간은 0.5초 이내로 한다. - 촬영된 사진을 서버에 전송되는 시간 1초 이내로 한다.		

요구사항 고유번호		PER-002		
요구사항 명칭		음성 파일 전송		
요구사항 분류		성능 요구사항	응락수준	필수
요구사항 상세설명	정의	음성 인식 및 음성 파일 전송 시간		
	세부 내용	- 오늘의 단어를 특정하는 음성 인식 기술은 1초 이내로 한다. - 음성 파일을 파이어베이스 스토리지에 업로드 되는 시간을 1초 이내로 한다.		

요구사항 고유번호		PER-003		
요구사항 명칭		사진 데이터 처리		
요구사항 분류		성능 요구사항	응락수준	필수
요구사항 상세설명	정의	얼굴 임베딩 벡터 추출 및 일치 여부 시간		
	세부 내용	- 해당 코드의 실행 시간은 2초 이내로 수행 - 처리시간이 10초가 지나면 해당 작업을 무효화 (인식 실패) - 추출된 데이터와 서버 내부의 데이터와의 일치 여부를 판단하여 결과(0 또는 1)를 라즈베리파이에 반환하며, 결과가 0일 경우에 서버에 이미지를 서버에 저장하고, 애플리케이션에 해당 이미지를 전송		

요구사항 고유번호		PER-004		
요구사항 명칭		음성 데이터 처리		
요구사항 분류		성능 요구사항	응락수준	필수
요구사항 상세설명	정의	음성 임베딩 벡터 추출 및 일치 여부 시간		
	세부 내용	- 해당 코드의 실행 시간은 1초 이내로 수행 - 처리시간이 10초가 지나면 해당 작업을 무효화 (인식 실패) - 추출된 데이터와 서버 내부의 데이터와의 일치 여부를 판단하여 결과(0 또는 1)를 라즈베리파이에 반환		

요구사항 고유번호		PER-005		
요구사항 명칭		잠금 모드 설정 및 확인		
요구사항 분류		성능 요구사항	응락수준	필수
요구사항 상세설명	정의	잠금 모드 설정 서버 및 잠금 모드 확인 서버		
	세부 내용	- 어플리케이션을 통해 잠금 모드가 변경되면 웹서버(잠금 모드 설정 서버)에 전달한다. - 도어락 초음파 센서 인식이 되면 잠금 모드 확인 서버에 요청을 통해 모드를 확인한다. - 응답을 통해 받은 모드에 맞게 잠금 해제 과정을 거친다.		

5. 인터페이스 요구사항

요구사항 고유번호		SIR-001		
요구사항 명칭		방문자 얼굴 데이터 관리		
요구사항 분류		사용자 인터페이스	응락수준	필수
요구사항 상세설명	정의	방문자 얼굴 이미지 및 본인 일치 여부 실시간 확인		
	세부 내용	- 등록된 사용자는 실시간 얼굴 인식 알고리즘을 통한 결과 확인 - 사용자는 반환 기록(log)을 확인 가능		

요구사항 고유번호		SIR-002		
요구사항 명칭		사용자 정보 등록		
요구사항 분류		사용자 인터페이스	응락수준	필수
요구사항 상세설명	정의	사용자 정보 등록		
	세부 내용	- 사용자정보 입력을 위한 입력칸 및 사진 등록, 음성 녹음 버튼 구현 - 결과값(개인정보) 반환 인터페이스 구현		

요구사항 고유번호		SIR-003		
요구사항 명칭		임시 비밀번호		
요구사항 분류		사용자 인터페이스	응락수준	필수
요구사항 상세설명	정의	임시 비밀번호 발급		
	세부 내용	- 임시 비밀번호를 발급하기 위한 버튼 및 결과값 반환 인터페이스 구현		

요구사항 고유번호		SIR-004		
요구사항 명칭		방문자 확인		
요구사항 분류		사용자 인터페이스	응락수준	필수
요구사항 상세설명	정의	침입자 이미지 확인		
	세부 내용	- 갤러리 형식의 침입자 확인 인터페이스 구현 - 방문자 발생 시 pop 알림 구현		

요구사항 고유번호		SIR-005		
요구사항 명칭		잠금 모드 변경		
요구사항 분류		사용자 인터페이스	응락수준	필수
요구사항 상세설명	정의	잠금 모드 변경		
	세부 내용	- 잠금 모드(다중 잠금 모드, 기본 모드, 임시비밀 모드) 변경 인터페이스 구현		

요구사항 고유번호		SIR-006		
요구사항 명칭		음성 인식		
요구사항 분류		사용자 인터페이스	응락수준	필수
요구사항 상세설명	정의	음성 인식 일치 확인 및 파일 전송		
	세부 내용	- 오늘의 단어 일치 확인을 위한 음성 인식 일치 확인 인터페이스 구현 - 일치 시 음성 파일이 파이어베이스 스토리지로 전송		

6. 데이터 요구사항

요구사항 고유번호	DAR-001		
요구사항 명칭	얼굴 데이터 수집		
요구사항 분류	데이터 요구사항	응락수준	필수
요구사항 상세설명	- 애플리케이션에 사용자를 등록하기 위해서 얼굴 데이터는 최소 20개를 서버로 전송하여 자료를 구축한다. - 방문자 사진 데이터는 라즈베리파이의 카메라로 촬영하여 수집한 후, 서버에 전송하여 사용자 일치 여부를 확인한다. - 얼굴 인식 모델을 학습시키기 위하여 AI-HUB의 임의의 얼굴 데이터셋을 활용한다.		

요구사항 고유번호	DAR-002		
요구사항 명칭	음성 데이터 수집		
요구사항 분류	데이터 요구사항	응락수준	필수
요구사항 상세설명	- 애플리케이션에 사용자를 등록하기 위해서 음성 데이터는 대략 5초 정도의 음성 파일 6개를 서버로 전송하여 자료를 구축한다. - 방문자 음성 데이터는 애플리케이션으로 수집한 후, firebase storage에 전송하여 사용자 일치 여부를 확인한다. - 화자 인식 모델을 학습시키기 위하여 AI-HUB의 임의의 화자 인식 데이터셋을 활용한다.		

요구사항 고유번호	DAR-003		
요구사항 명칭	임시 비밀번호 데이터 관리		
요구사항 분류	데이터 요구사항	응락수준	필수
요구사항 상세설명	- 임시 비밀번호는 웹서버에 해시 알고리즘을 통해 저장한다. - 임시 비밀번호의 경우 일정 시간이 지나면 폐기한다.		

7. 테스트 요구사항

요구사항 고유번호	TER-001		
요구사항 명칭	테스트 방안		
요구사항 분류	테스트	응락수준	필수
요구사항 상세설명	- 시스템은 핵심 기능인 사용자 등록을 베이스 라인으로 하며 제공하려는 모든 부가 기능을 구현하는 것을 목표로 한다. - 애플리케이션과 라즈베리파이(도어락) 소통은 중간에 서버를 두고 각 기기 간의 통신은 서버를 이용하도록 한다. - 얼굴 인식 모델 테스트를 위하여 여러 사용자의 얼굴 데이터는 AI-HUB에서 제공하는 임의의 데이터를 활용하여 진행한다. - 화자 인식 모델 테스트를 위하여 여러 사용자의 음성 데이터는 AI-HUB에서 제공하는 임의의 데이터를 활용하여 진행한다. - 최종 테스트는 최소 기능의 테스트가 완료된 후 부가 기능을 추가하여 구동하여 도출값을 기준으로 성공 여부를 정한다.		

8. 보안 요구사항

요구사항 고유번호	SER-001		
요구사항 명칭	도어락 비밀번호 DB 보안		
요구사항 분류	보안	응락수준	필수
요구사항 상세설명	- 도어락 비밀번호 정보는 우리가 구현하려는 핵심 기능을 위해서 서버에 저장해야 하므로 보안성이 요구된다. - 해시 알고리즘을 통해 해시값을 생성한 후 서버에 전달한다. - 임시 비밀번호 일치 여부를 확인할 때에도 알고리즘을 사용한다. - 임시 비밀번호의 경우 일정 시간이 지나면 폐기한다.		

9. 품질 요구사항

요구사항 고유번호		QUR-001		
요구사항 명칭		데이터 비교		
요구사항 분류		품질	응락수준	필수
요구사항 상세설명	정의	얼굴 데이터 알고리즘의 임계치		
	세부 내용	- 안면 데이터 시 일치 여부의 기준이 되는 최적의 임계치 값(threshold) 구하기 (일반적으로 0.5)		

요구사항 고유번호		QUR-002		
요구사항 명칭		데이터 비교		
요구사항 분류		품질	응락수준	필수
요구사항 상세설명	정의	음성 데이터 알고리즘의 임계치		
	세부 내용	- 음성 데이터 시 일치 여부의 기준이 되는 최적의 임계치 값(threshold) 구하기 (일반적으로 0.5)		

요구사항 고유번호		QUR-003		
요구사항 명칭		잠금 모드		
요구사항 분류		품질	응락수준	필수
요구사항 상세설명	정의	잠금 모드에 맞는 잠금 해제 과정		
	세부 내용	- 사용자가 설정한 잠금 모드를 토대로 도어락 해제 과정이 이루어져야 한다.		

10. 제약 사항

요구사항 고유번호	COR-001		
요구사항 명칭	얼굴 인식 모델 훈련		
요구사항 분류	제약 사항	응락수준	필수
요구사항 상세설명	- 기존 얼굴 인식 모델이 서양인을 기준으로 학습된 점을 고려했을 때, 큰 용량의 한국인 안면 데이터를 필요로 함 - 개인정보 등의 문제로 많은 양의 한국인 안면 데이터를 구하는 데 어려움이 있음		

11. 프로젝트 관리 요구사항

요구사항 고유번호	PMR-001		
요구사항 명칭	품질관리		
요구사항 분류	프로젝트 관리	응락수준	필수
요구사항 상세설명	- 최소기능인 얼굴 인식 기능 구현과 라즈베리파이와 애플리케이션 및 서버 간의 연동을 최우선 목표로 한다. - 최소기능이 작동된 후 부가 기능을 추가하고 성능 향상을 목표로 한다. - 팀원은 각 애플리케이션, 라즈베리파이, 인공지능 모델 중 맡은 역할을 수행한다. - 모든 팀원은 품질의 성능 향상과 관리를 위해 매주 부득이한 경우를 제외하고 대면으로 회의를 진행한다. - 프로젝트를 신속 정확하게 진행할 수 있도록 해당 전문 교수님께 자문을 구한다.		