

# 모바일 프로그래밍

## (나의 앱 만들기)



2020.07.06

201636055 컴퓨터공학과 장주명  
201835634 컴퓨터공학과 김영현  
201835663 컴퓨터공학과 방은지

# 목차

1. 나의 앱 만들기.....	3
1.1 애플리케이션 개요.....	3
1.2 애플리케이션 구성.....	4
1.3 애플리케이션 실행화면.....	5
1.4 애플리케이션 구현.....	10
2. 나의 앱 만들기 후기.....	48

# 1. 나의 앱 만들기

## 1.1 애플리케이션 개요

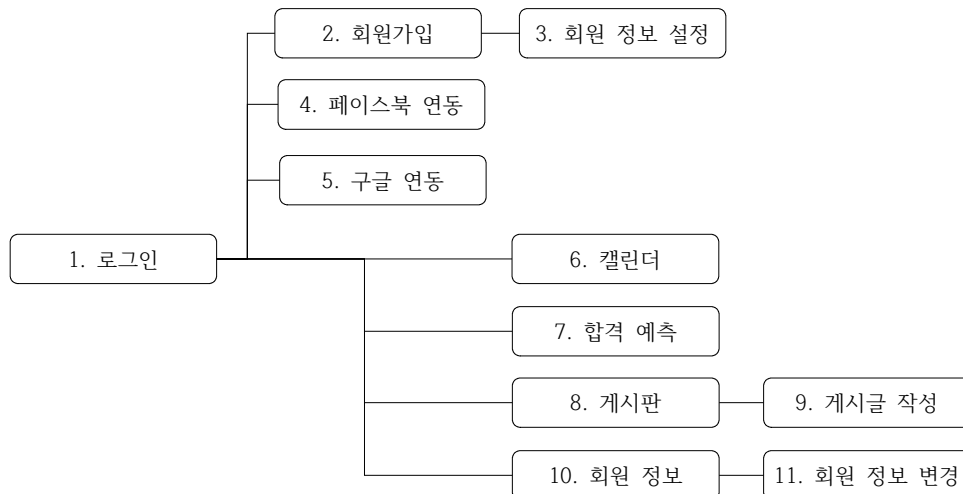
### 1) App 이름

『자격증의 민족』

### 2) App 개요

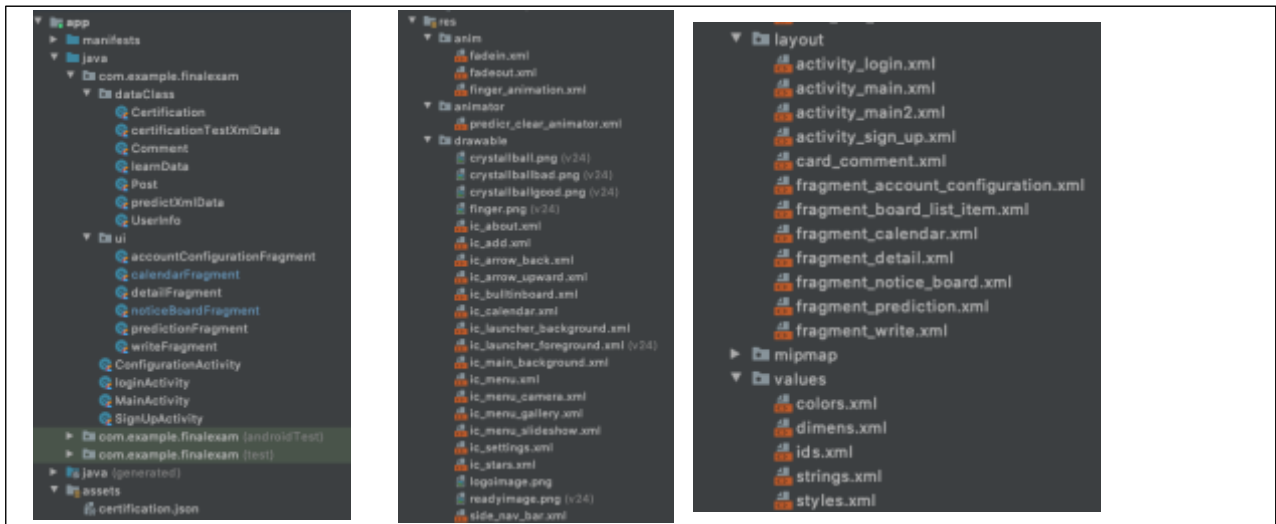
본 어플은 자격증을 공부하는 사람들을 겨냥한 어플리케이션으로 자신의 시험 일정을 확인함과 동시에 D-day를 지정, 오늘의 공부를 체크하면서 자신의 공부 진행도를 확인할 수 있으며 커뮤니티를 이용한 자격증에 대한 이야기를 자유롭게 할 수 있으며 즐길거리로 자신이 미래에 자격증 시험이 합격할 것인지 확인해볼 수 있는 어플입니다.

#### ◇주요 기능



## 1.2 애플리케이션 구성

### 1) 프로젝트 구성도



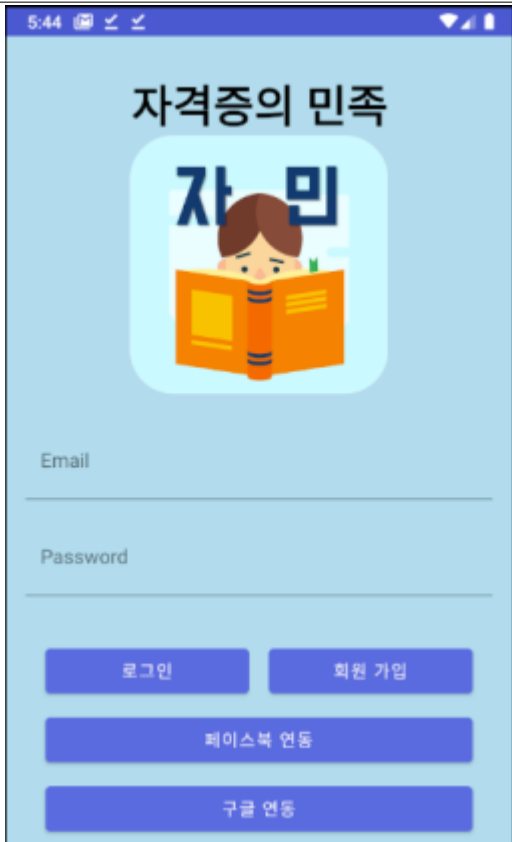
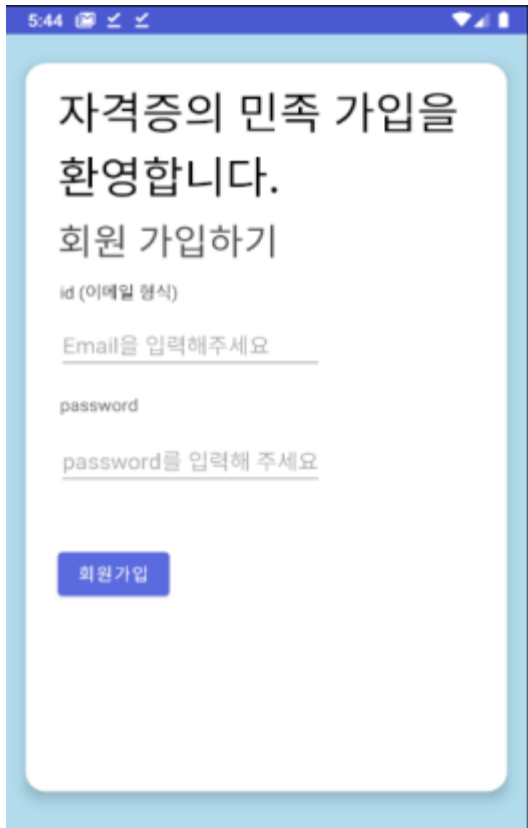
### 2) 메인 화면 구성도

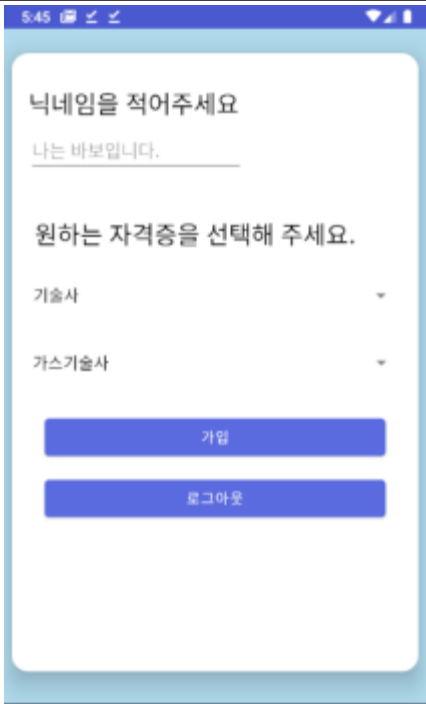




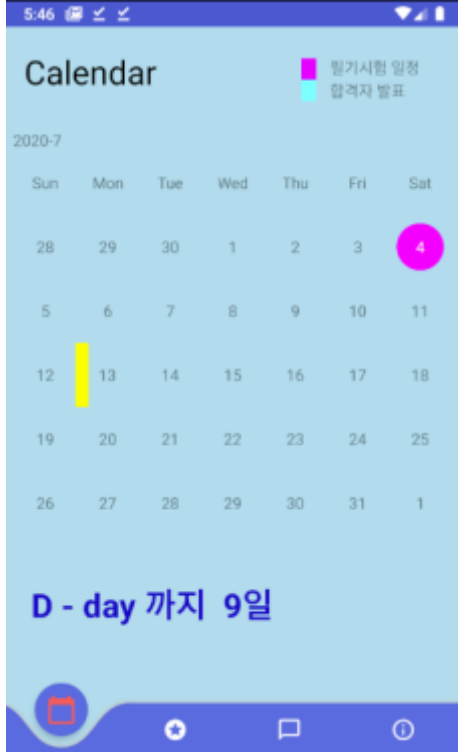
### 3) 실행 기능

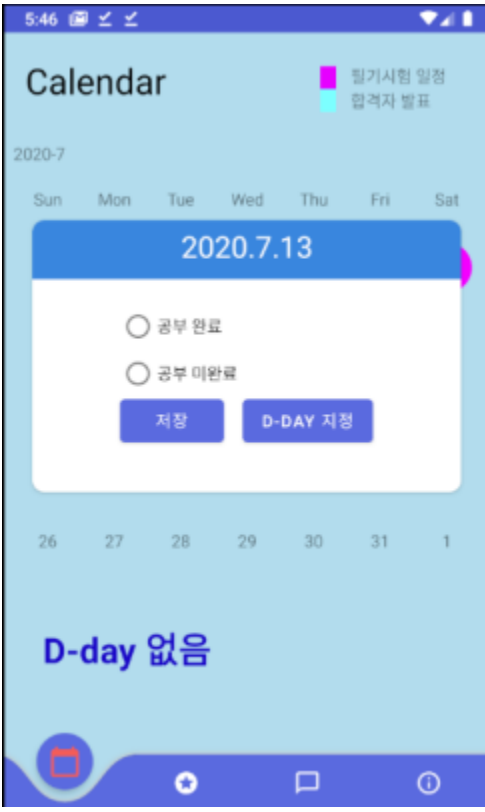
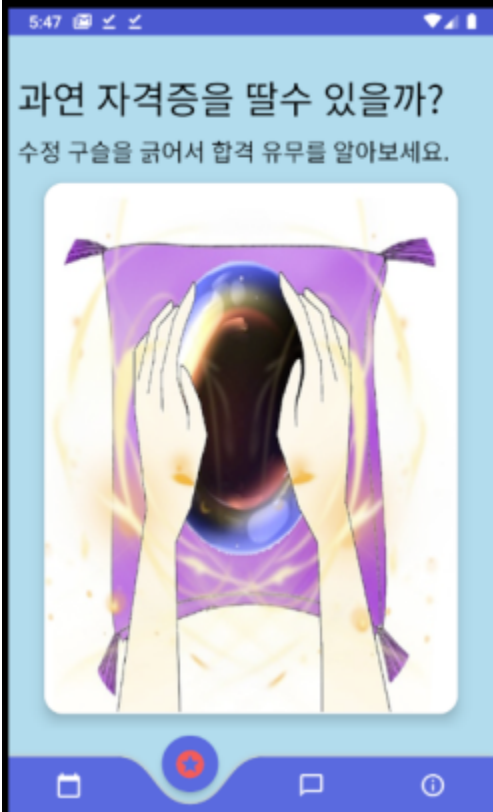

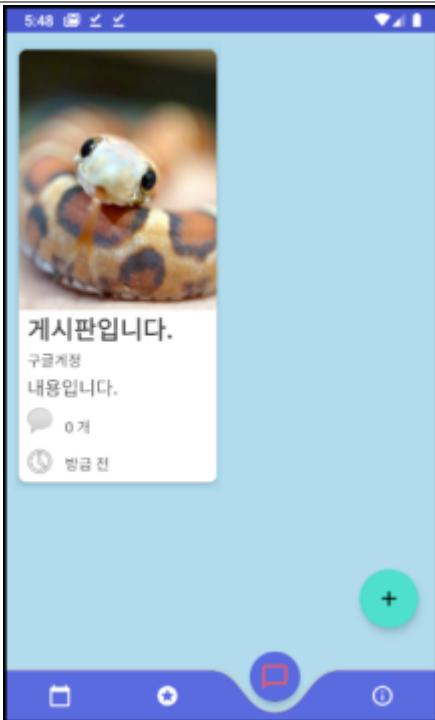
주요 기능	상세 기능
캘린더 기능	필기시험 일정 & 합격자 발표일 표시, & 공부 유무 표시 & 디데이 체크기능
합격 예측 기능	합격 또는 불합격 예측
게시판 기능	게시글 작성 및 공유
회원 관리 기능	회원 정보(닉네임, 자격증 정보) 변경

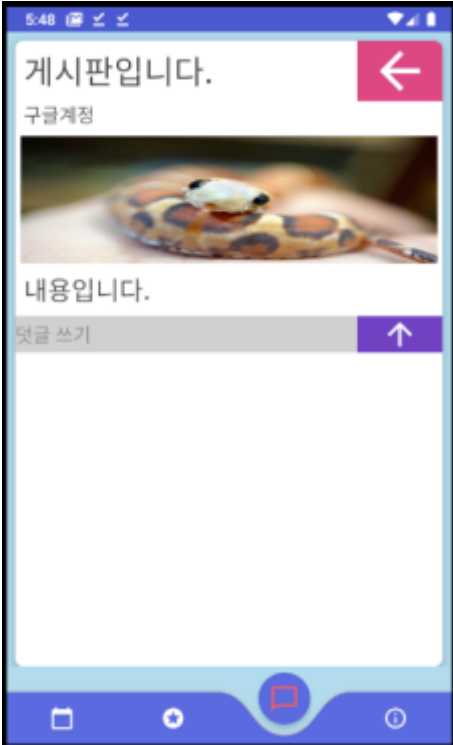
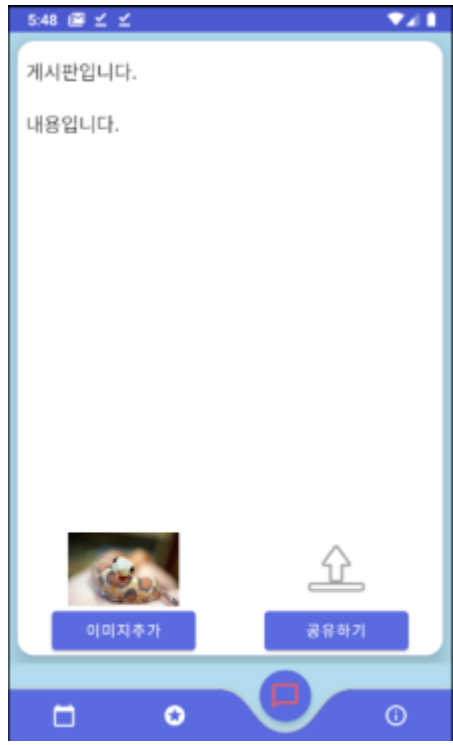

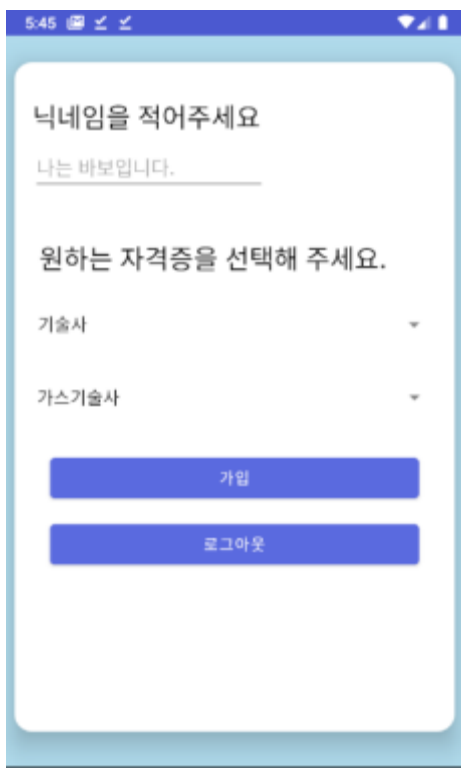
### 1.3 애플리케이션 실행화면

화면 1	화면 2
로그인 화면	회원가입 화면
 <p>The login screen features a light blue background. At the top, it says '자격증의 민족' (Nation of Certificate) and '자민' (Jamin) with an illustration of a person reading a book. Below this are input fields for 'Email' and 'Password'. At the bottom, there are four buttons: '로그인' (Login), '회원 가입' (Sign Up), '페이스북 연동' (Facebook Login), and '구글 연동' (Google Login).</p>	 <p>The sign-up screen has a light blue background. It displays the text '자격증의 민족 가입을 환영합니다.' (Welcome to the Nation of Certificate membership) and '회원 가입하기' (Sign up). Below this, it asks for 'id (이메일 형식)' (id in email format) and 'password', with corresponding input fields. A '회원가입' (Sign up) button is at the bottom.</p>

화면 3	화면 4
회원 정보 설정 화면	페이스북 연동
	

화면 5	화면 6
구글 연동	캘린더 화면
	

화면 7	화면 8
캘린더 공부 추가 화면	합격 예측 화면1
	
화면 9	화면 10
합격 예측 화면2	게시판 메인 화면
	

화면 11	화면 12
게시글 세부 내용 화면	게시글 작성 화면
	
화면 13	화면 14
회원 정보 화면	회원 정보 변경 화면
	



#### 1.4 데이터 베이스 구현

NoSql 형식의 파이어베이스 실시간 데이터 베이스를 이용했습니다.

유저정보 클래스 (프로젝트 파일의 dataClass - UserInfo.kt)

1rbEEq7bCLeMuRNJuONHazZ13cR2

```
cert1: "기술사"  
cert2: "가스기술사"  
d_day_day: 4  
d_day_month: 7  
d_day_year: 2020  
jmcdData: 752
```

게시판 내용 클래스 (프로젝트 파일의 dataClass - Post.kt)

-MBO-hRi2PxqC9wrMTKs

```
bgUri: "https://firebasestorage.googleapis.com/v0/b/fin..."  
commentCount: 1  
descWriteTime: -1593852482719  
message: "내용입니다."  
postId: "-MBO-hRi2PxqC9wrMTKs"  
title: "게시판입니다."  
writeTime: 1593852486609  
writerId: "구글계정"
```

댓글 클래스 (프로젝트 파일의 dataClass - Comment.kt)

-MBO-o40t3v3gaC7HJm9

```
commentId: "구글계정"  
message: "댓글입니다."  
postId: "-MBO-hRi2PxqC9wrMTKs"  
writeTime: 1593852510727
```

## 1.5 애플리케이션 구현

화면(UI)	설명
	<p>어플을 처음 실행하면 로그인을 하는 화면입니다. 이메일 형식으로 로그인을 하며</p> <p>각 버튼을 이용해서 특정 화면으로 넘어갑니다.</p> <p>회원가입 : 회원가입을 하는 창으로 넘어갑니다. 페이스북 연동 : 페이스북 연동하는 페이지로 넘어갑니다. 구글 연동 : 구글 연동을 하는 페이지로 넘어갑니다. 로그인 : 로그인 후에 회원 정보를 등록하는 페이지로 넘어갑니다.</p> <p>화면 넘감을 페이드인 아웃으로 구현하였습니다. ActionBar를 숨김처리했습니다.</p> <p>Constraint로 화면을 구성해서 다양한 크기의 기기에 맞게 화면이 조절됩니다.</p> <p>연동 로그인의 경우에는 해당 로그인을 하면 토큰을 반환 받아서 해당 토큰을 파이어베이스 로그인 토큰으로 1대1 교환을 받아서 로그인을 하는 방식으로 구현했습니다.</p> <p>또한 앱을 실행할 때 이전에 이미 로그인 한적이 있다면 로그인을 건너뛰게 만들었습니다.</p>
<p style="text-align: center;">loginActivity.kt</p> <pre>//가장 처음에 로그인을 하는 화면의 코틀린 파일입니다. class loginActivity : AppCompatActivity() {  //----- //전역변수 설정  //구글 로그인 변수 lateinit var mGoogleSignInClient: GoogleSignInClient lateinit var mGoogleSignInOptions: GoogleSignInOptions var RC_SIGN_IN = 1  //파이어베이스에서 이용하는 변수이다. private lateinit var firebaseAuth: FirebaseAuth  //페이스북에서 이용 lateinit var callbackManager: CallbackManager  //----- //이전에 로그인을 했으면 바로 넘어가는 기능을 제공합니다.</pre>	

```

//onCreate보다 먼저 실행되는 함수이다.
override fun onStart() {
    super.onStart()
    overridePendingTransition(R.anim.fadein, R.anim.fadeout)
    val user = FirebaseAuth.getInstance().currentUser
    if (user != null) {
        //이미 로그인 되어 있다면 패스한다.
        startActivity(MainActivity.getLaunchIntent(this))
        overridePendingTransition(R.anim.fadein, R.anim.fadeout)
        finish()
    }
}
} //end of onStart

```

```

//-----
//onCreate함수입니다.
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    //액션바 숨기
    supportActionBar?.hide()
    setContentView(R.layout.activity_login)

    //페이스북 개발자용 해시키 생성함수다.
    //다른 기기에서 로그인할 경우에는 페이스북 개발자 사이트에 들어가서 해당값을 바꿔주어야 한다.
    printHashKey(this)

    firebaseAuth = FirebaseAuth.getInstance()

    //처음 변수들의 초기화를 담당한다.
    configureSignIn()

    //구글 로그인 버튼을 누를 때
    googleLoginButton.setOnClickListener {
        signInGoogle()
    }

    //페이스북 로그인을 누를 때
    facebookLoginButton.setOnClickListener {
        signInFacebook()
    }

    //회원가입버튼을 누를 때
    Login_SignUpButton.setOnClickListener {
        startActivity(Intent(this, SignUpActivity::class.java))
        overridePendingTransition(R.anim.fadein, R.anim.fadeout)
    }

    //기본 이메일로 로그인을 할 때
    Login_loginButton.setOnClickListener {
        loginEmail()
    }
} //end of onCreate

```

```

//-----

```

```

// 각종 변수들의 초기화를 담당하는 함수이다.
private fun configureSignIn() {
    mGoogleSignInOptions
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
    .requestIdToken(getString(R.string.default_web_client_id))
    .requestEmail()
    .build()

    mGoogleSignInClient = GoogleSignIn.getClient(this, mGoogleSignInOptions)

    //페이스북 로그인 변수 할당
    callbackManager = CallbackManager.Factory.create()
}

//-----
//기본 이메일로 로그인을 하는 함수이다.
fun loginEmail(){
    val email = Login_email.text.toString()
    val password = Login_password.text.toString()

    if (email.isNullOrBlank() || password.isNullOrBlank()){
        Toast.makeText(applicationContext,"이메일 비밀번호를 입력해 주세
요.",Toast.LENGTH_LONG).show()
        return
    }

    FirebaseAuth.getInstance().signInWithEmailAndPassword(email,password).addOnCompleteListener {
        task ->
            if (task.isSuccessful){
                startActivity(ConfigurationActivity.getLaunchIntent(this))
                overridePendingTransition(R.anim.fadein, R.anim.fadeout)
            }else{
                Toast.makeText(applicationContext,"로그인 실패",Toast.LENGTH_LONG).show()
            }
        }
    }

//-----
//구글 로그인을 해주는 함수이다.
private fun signInGoogle() {
    val signInIntent: Intent = mGoogleSignInClient.signInIntent
    startActivityForResult(signInIntent, RC_SIGN_IN)
    overridePendingTransition(R.anim.fadein, R.anim.fadeout)
}

//-----
//페이스북에서 로그인을 하게 해주는 함수이다.
fun signInFacebook() {
    LoginManager.getInstance().loginBehavior = LoginBehavior.WEB_VIEW_ONLY
    LoginManager.getInstance().loginWithReadPermissions(
        this, Arrays.asList(
            "public_profile", "email"

```

```

    )
    )
    LoginManager.getInstance()
        .registerCallback(callbackManager, object : FacebookCallback<LoginResult> {
            override fun onSuccess(result: LoginResult?) {
                firebaseAuthWithFacebook(result)
            }

            override fun onCancel() {

            }

            override fun onError(error: FacebookException?) {

            }

        })
    })
}

//-----
//구글에서 로그인한 정보를 파이어베이스에 연동하는 코드이다.
fun firebaseAuthWhitGoogle(acct: GoogleSignInAccount?) {
    var credential = GoogleAuthProvider.getCredential(acct?.idToken, null)
    firebaseAuth.signInWithCredential(credential).addOnCompleteListener { task ->
        if (task.isSuccessful) {
            println("Signup success")
            //홈 액티비티로 넘어간다.
            Toast.makeText(applicationContext, "로그인에 성공했습니다.",
                Toast.LENGTH_LONG).show()
            startActivity(ConfigurationActivity.getLaunchIntent(this))
            overridePendingTransition(R.anim.fadein, R.anim.fadeout)
        } else {
            //로그인 실패인 거시다.
        }
    }
}

//-----
//페이스북에서 로그인한 정보를 파이어베이스에 연동하는 함수이다.
private fun firebaseAuthWithFacebook(result: LoginResult?) {
    var credential = FacebookAuthProvider.getCredential(result?.accessToken?.token!!)
    FirebaseAuth.getInstance().signInWithCredential(credential).addOnCompleteListener { task ->
        if (task.isSuccessful) {
            println("facebook login")
            Toast.makeText(applicationContext, "로그인에 성공했습니다.",
                Toast.LENGTH_LONG).show()
            startActivity(ConfigurationActivity.getLaunchIntent(this))
            overridePendingTransition(R.anim.fadein, R.anim.fadeout)
        }
    }
}

//-----

```

```

-----
//액티비티에서 결과값을 받았을 때 실행하는 함수이다.
//로그인을 하고 나서 토큰을 받아 올 것이다.
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    callbackManager.onActivityResult(requestCode, resultCode, data)

    if (requestCode == RC_SIGN_IN) {
        val task: Task<GoogleSignInAccount> =
GoogleSignIn.getSignedInAccountFromIntent(data)
        try {
            val account = task.getResult(ApiException::class.java)
            firebaseAuthWhitGoogle(account)
        } catch (e: ApiException) {
            Toast.makeText(this, "Google sign in failed:", Toast.LENGTH_LONG).show()
        }
    }
}

//-----
// 외부에서 로그아웃을 하고 화면전환을 할 수 있게 해주는 함수이다.
// 메인에서 호출하는 것을 볼 수 있다.
companion object {
    fun getLaunchIntent(from: MainActivity) = Intent(from, LoginActivity::class.java).apply {
        addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
    }
    fun getLaunchIntent(from: ConfigurationActivity) = Intent(from, LoginActivity::class.java).apply
{
    addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
}

    fun getLaunchIntent(from: Context) = Intent(from, LoginActivity::class.java).apply {
        addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
    }
}
}

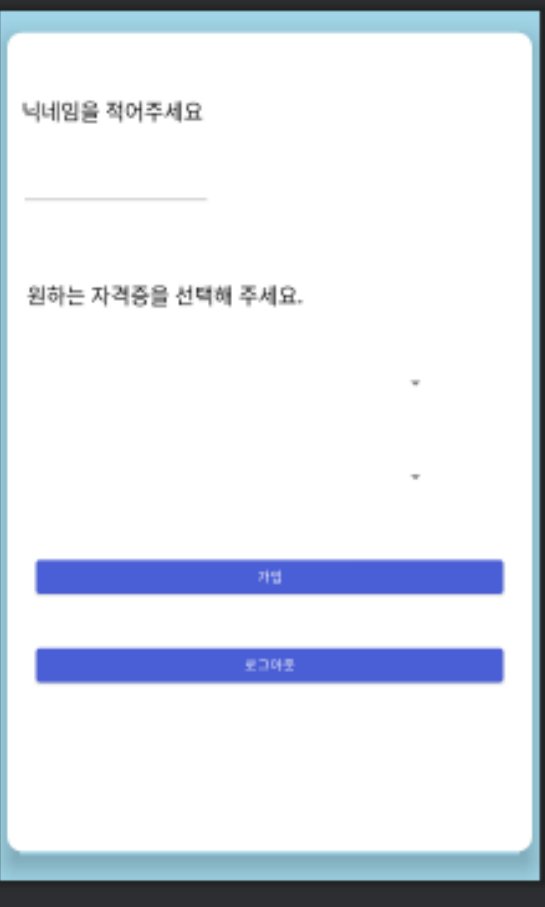
```

화면(UI)	설명
	<p>회원가입을 하는 창입니다.</p> <p>이메일과 비밀번호를 입력하면 파이어베이스의 Auth부분에 해당 값을 등록하여 로그인을 할 수 있습니다.</p> <p>또한 회원 가입이 실패하면 이메일의 형식이 갖추어졌는지 확인하라는 메시지를 출력합니다.</p> <p>마찬가지로 액션바를 제거하고 화면 전환 효과를 페이드인 아웃으로 구현하였습니다.</p>
SingUpActivity.kt	
<pre> //회원 가입을 하는 페이지의 코틀린파일입니다. class SignUpActivity : AppCompatActivity() { //----- //onCreate 함수입니다. override fun onCreate(savedInstanceState: Bundle?) {     super.onCreate(savedInstanceState)     //액션바 숨기     supportActionBar?.hide()     //화면 전환 효과를 페이드인,아웃으로 전환 합니다.     overridePendingTransition(R.anim.fadein, R.anim.fadeout)     setContentView(R.layout.activity_sign_up)      //회원 가입 버튼을 눌렀을때     SignUpCompleteButton.setOnClickListener {         createEmailId()     } }  }  //----- //회원 가입을 기능하는 함수입니다. fun createEmailId(){     var email = SignUpEmailText.text.toString()     var password = SignUpPasswordText.text.toString()      FirebaseAuth.getInstance().createUserWithEmailAndPassword(email,password).addOnCompleteListener { task -&gt;         if (task.isSuccessful){ </pre>	

```

        println("signup success")
        FirebaseAuth.getInstance().signOut()
        finish()
    }else{
        Toast.makeText(applicationContext,"이메일 형식으로 해주세요
",Toast.LENGTH_LONG).show()
    }
}
} //end of createEmailId
}

```

화면(UI)	설명
	<p>이번에는 로그인을 하면 나오는 유저 정보 설정 화면입니다.</p> <p>닉네임을 지정하고 원하는 자격증을 지정해서 선택합니다.</p> <p>자격증리스트는 공공 데이터 포털의 csv데이터를 json형식으로 바꾸어서 JSONObject을 이용하는 방식으로 파싱을 진행해서 각각의 스피너 어댑터에 넣어주는 방식으로 구현했습니다.</p> <p>공공데이터 - 국가기술종목 목록 정보  <a href="https://www.data.go.kr/data/15003024/fileData.do">https://www.data.go.kr/data/15003024/fileData.do</a></p> <p>그 후에 가입을 누르면 됩니다.</p> <p>또한 기본 회원 가입시 유저 아이콘을 기본으로 등록합니다.</p> <p>유저 이미지를 지정하는 부분은 따로 구현하지 못했습니다.</p>
SplashActivity.kt	
<pre> //로그인을 마치고 원하는 자격증과 닉네임을 정하는 부분입니다. class ConfigurationActivity : AppCompatActivity() {  //----- //전역변수 설정  //자격증 리스트를 보여주기 위한 리스트 입니다. var arr = arrayListOf&lt;String&gt;() var jmcList = arrayListOf&lt;Int&gt;() </pre>	



```

//userInfo Class에 저장하기 위한 변수입니다.
var certificatPosition1 : String = ""
var certificatPosition2 : String = ""
var jmcdData : Int = 0

// 인덱스 1,2,3,4 순서이다.
var certificationList = arrayOf("기술사","기능장","기사","기능사")

//JSON을 파싱한 값이다.
val certificationData by lazy { readJson() }
val jsonarr by lazy { JSONArray(certificationData) }

//-----
//onCreate 함수입니다.
override fun onCreate(savedInstanceState: Bundle?)
{
    super.onCreate(savedInstanceState)
    //액션바 숨기기
    supportActionBar?.hide()
    setContentView(R.layout.activity_main)

    //처음 변수들의 초기화를 담당합니다.
    initData()

    //버튼에 로그아웃 기능을 넣는다.
    signOutButton.setOnClickListener{
        signOut()
    }

    //닉네임을 지정하고 다음 화면으로 넘어가는 기능을 제공합니다.
    createNickButton.setOnClickListener {
        pushDatabase()
        pushNickName()
        goNextActivity()
    }

    //스피너에 어댑터와 리스너를 연결합니다.
    //세부사항 스피너를 개선하는 역할을 합니다.
    settingSpinner()

} //end of onCreate

//-----
//스피너를 설정하는 함수입니다.
private fun settingSpinner() {
    topSpinner.adapter =
    ArrayAdapter<String>(this, android.R.layout.simple_spinner_dropdown_item, certificationList)
    topSpinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener{
        override fun onNothingSelected(parent: AdapterView<*>?) {
            Toast.makeText(applicationContext, "자격증 종류를 선택해 주세요.", Toast.LENGTH_LONG).show()
        }

        override fun onItemSelected(
            parent: AdapterView<*>?,

```

```

        view: View?,
        position: Int,
        id: Long
    ) {

        //하위 자격증 항목 리스트인 arr을 새로 만들어 줍니다.
        settingCertficationList(position)

        //하위 스피너의 어댑터입니다.
        var adpt
        ArrayAdapter<String>(this@ConfigurationActivity, android.R.layout.simple_spinner_dropdown_item, arr)
        bottomSpinner.adapter = adpt

        // 하위 스피너의 리스너입니다.
        bottomSpinner.onItemSelectedListener = object :
        AdapterView.OnItemSelectedListener{
            override fun onNothingSelected(parent: AdapterView<*>?) {

            }

            override fun onItemSelected(
                parent: AdapterView<*>?,
                view: View?,
                position: Int,
                id: Long
            ) {
                certificatPosition2 = arr[position]
                jmcdData = jmcdList[position]
            }
        }
    } //end of on Item Selected -> 상위스피너
} //end of SpinnerListener -> 상위 스피너
} //end of setting Spinner

//-----
//상위 자격분류에 맞는 하위 자격분류를 만들어 줍니다.
fun settingCertficationList(position: Int) {
    certificatPosition1 = certificationList[position]

    //세부사항 스피너의 값을 수정합니다.
    Toast.makeText(applicationContext, "${certificationList[position]}를
        선택했습니다.", Toast.LENGTH_LONG).show()

    arr.clear()
    jmcdList.clear()
    for (i in 0 .. jsonarr.length() - 1)
    {
        var jsonobj = jsonarr.getJSONObject(i)
        if (jsonobj.getString("SERIESNM") == certificationList[position]){
            arr.add(jsonobj.getString("JMFLDNM"))
            jmcdList.add(jsonobj.getInt("JMCD"))
        }
    }
}

//-----

```

```
fun pushNickName () {
```

```
if (nickNameText.text.isNullOrBlank() && nickNameText.hint.isNullOrBlank()) {
    nickNameText.setText("나는 바보입니다.")
}
```

```

        .setPhotoUri(Uri.parse("https://firebasestorage.googleapis.com/v0/b/finalexam-77fdc.appspot.com/o/user.png?alt=media&token=8696e110-d8f1-4218-8cb9-e28088f60c1f"))
        .build()
    }
}

```

}

```
if (nickNameText.text.toString().isNullOrEmpty()){
    Toast.makeText(applicationContext, "닉네임을
        그대로 합니다.", Toast.LENGTH_LONG).show()
}
```

```
    }else{
        val profileUpdate = UserProfileChangeRequest.Builder()
            .setDisplayName("${nickNameText.text}")
```

```

.setPhotoUri(Uri.parse("https://firebasestorage.googleapis.com/v0/b/finalexam-77fdc.appspot.com/o/user.png?alt=media&token=8696e110-d8f1-4218-8cb9-e28088f60c1f"))
    .build()

```

}

```
//end of pushNickName
```

//선택한 자격증을 파이어베이스 데이터베이스에 저장하는 역할을 합니다.

```
fun pushDatabase() {
```

//기능사인지 기사인지 구분하는 기능을 합니다.

```
//certificatPosition1
```

```
//세부 자격증을 표시하는 역할을 합니다.
```

```
//certificatPosition2
```

```
//자격증 종목 코드입니다.
```

```

//jmcdData

val userinfo = UserInfo(
    certificatPosition1,
    certificatPosition2,
    jmcdData
)

//파이어베이스 데이터베이스에 저장합니다.
FirebaseDatabase.getInstance().reference
    .child("users")
    .child("${FirebaseAuth.getInstance().currentUser?.uid}")
    .setValue(userinfo)
} //end of pushDatabase

//-----
//에셋 폴더에서 JSON을 읽어 옵니다.
fun readJson(): String? {
    var json : String? = null
    try {
        val inputStream : InputStream = assets.open("certification.json")
        json = inputStream.bufferedReader().use { it.readText() }

    } catch (e : IOException){

    }

    return json
}

//-----
//백스택을 쌓지 않고 화면을 넘기는 역할을 합니다.
companion object {
    fun getLaunchIntent(from: Context) = Intent(from, ConfigurationActivity::class.java).apply {
        addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
    }
}
}

```

화면(UI)	설명
	<p>로그인을 완료했을 경우의 메인 화면입니다.</p> <p>meowBottomNavigation 라이브러리를 이용해서 하단 액션바를 만들었습니다.</p> <p>참조 : <a href="https://github.com/oneHamidreza/MeowBottomNavigation">https://github.com/oneHamidreza/MeowBottomNavigation</a></p> <p>해당 바는 동적으로 버튼을 넣을 수 있습니다.</p> <p>버튼을 누르면 위는 Fragment 부분으로 해당 Fragment가 전환되는 방식으로 화면전환을 합니다.</p> <p>로그아웃을 하는 함수를 지니고 있어서 Fragment에서 로그아웃을 하는 경우 MainActivity에서 SignOut()함수를 이용하면 됩니다.</p> <p>백스택을 남기지 않고 화면전환을 하는 companion object의 함수가 있습니다.</p>

#### MainActivity.kt

```
//로그인이 완료된 후의 메인 클래스입니다.
class MainActivity : AppCompatActivity() {

    //로그인한 유저의 정보입니다.
    val user = FirebaseAuth.getInstance().currentUser
    //-----

    //onCreate 함수입니다.
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main2)
        overridePendingTransition(R.anim.fadein, R.anim.fadeout)

        //처음에 하단에 app bar를 만들어 줍니다.
        meowBottomNavigation.add(MeowBottomNavigation.Model(1,R.drawable.ic_calendar))
        meowBottomNavigation.add(MeowBottomNavigation.Model(2,R.drawable.ic_stars))
        meowBottomNavigation.add(MeowBottomNavigation.Model(3,R.drawable.ic_bulletinboard))
        meowBottomNavigation.add(MeowBottomNavigation.Model(4,R.drawable.ic_about))

        //하단 네비게이션에 리스너를 붙여서 프래그먼트도 바뀌게 합니다.
        meowBottomNavigation.setOnClickMenuListener {
            when(it.id){
                1 -> {setFragment(calendarFragment.newInstance())}
                2 -> {setFragment(predictionFragment.newInstance())}
                3 -> {setFragment(noticeBoardFragment.newInstance())}
                4 -> {setFragment(accountConfigurationFragment.newInstance())}
            }
        }
    }
}
```

```

        //처음 Fragment를 캘린더로 지정한다.
        setFragment(calendarFragment.newInstance())
        meowBottomNavigation.show(1)
    }//end of onCreate

//-----
//메뉴 전환시 Fragment를 전환하는 함수 입니다.
fun setFragment(fragment : Fragment){
    supportFragmentManager
        .beginTransaction()
        .replace(R.id.mainFrame,fragment,"mainActivity")
        .commit()
} //end of setFragment

//-----
// 세션 로그아웃 함수
fun signOut() {
    lateinit var googleSignInClient: GoogleSignInClient

    //세션 로그아웃 구현
    var gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken(getString(R.string.default_web_client_id))
        .requestEmail()
        .build()

    //구글 클라이언트를 연결시킵니다.
    googleSignInClient = GoogleSignIn.getClient(this,gso)
    FirebaseAuth.getInstance().signOut();

    // 구글 세션 로그 아웃
    googleSignInClient?.signOut()

    //페이스북 세션 로그아웃
    LoginManager.getInstance().logout()

    //초기 화면으로 돌아간다.
    finish()
    startActivity(loginActivity.getLaunchIntent(this))
    overridePendingTransition(R.anim.fadein, R.anim.fadeout)
} //end of signOut

//-----
//액티비티 스택을 쌓지 않고 화면을 넘어가게 해주는 싱글톤 함수입니다.
companion object {
    fun getLaunchIntent(from: Context) = Intent(from, MainActivity::class.java).apply {
        addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK)
    }
}
}

```

화면(UI)	설명
	<p>메인 화면에 들어가는 Fragment입니다.</p> <p>먼저 각 계정에서 등록한 국가기술 종목의 대분류(기사, 기능사)를 불러옵니다.</p> <p>나온 대분류에 맞는 공공데이터의 국가 기술종목 자격시험 Api를 호출해서 시험일과 합격자 발표일을 Xml로 호출, 이를 다시 파싱해서 캘린더에 보여주는 방식으로 작동시켰습니다.</p> <p>또한 날짜를 입력하면 그날의 공부를 했는지 안했는지 체크하는 기능을 넣었습니다.</p> <p>또한 계정마다 D-day를 체크하는 기능을 만들어서 시험이나 특정일로부터 얼마나 남았는지 아니면 얼마나 지났는지 확인하는 기능을 넣었습니다.</p> <p>해당 D-day기능은 프래그먼트의 생명주기에 맞춰 onPause()를 실행할 때 파이어베이스에 D-day를 저장하도록 하였습니다.</p> <p>또한 현재 코로나 19 사태로 인해 시험일정 Api에 XXXXXXXX로 일정이 반환되기도 하는데 해당 부분은 건너뛰고 체크를 하게 만들었습니다.</p>
calendarFragment.kt	
<pre> class calendarFragment : Fragment() {      //리스너 해제를 담당합니다.     var listener: ChildEventListener? = null      //api의 값     var Xmldata: ArrayList&lt;certificationTestXmlData&gt;? = null      //원하는 자격증의 대분류     var result: String = ""      //선택한 날짜     var year = 0     var month = 0     var day = 0      //D-day로 지정한 날짜     var dDay_Year = 0 </pre>	

```

var dDay_Month = 0
var dDay_Day = 0

val now = LocalDate.now()

//오늘의 날자
val calendar = Calendar.getInstance()

//선택한 날짜
val dCalendar = Calendar.getInstance()

val dayOfMilliSecond = (24 * 60 * 60 * 1000)

override fun onCreateView(view: View, savedInstanceState: Bundle?) {
    super.onCreateView(view, savedInstanceState)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        //원하는 자격증이 뭔지 확인하는 함수입니다.
        whatWantCertificat()

        //원하는자격증에 따라 url을 호출합니다.
        callUrlAndXmlParse(result)
    }

    override fun onPause() {
        super.onPause()
        discheckTestDate()

        //파이어베이스 데이터베이스에 저장합니다.
        FirebaseDatabase.getInstance().reference
            .child("users")
            .child("${FirebaseAuth.getInstance().currentUser?.uid}")
            .child("d_day_year")
            .setValue(dDay_Year)

        //파이어베이스 데이터베이스에 저장합니다.
        FirebaseDatabase.getInstance().reference
            .child("users")
            .child("${FirebaseAuth.getInstance().currentUser?.uid}")
            .child("d_day_month")
            .setValue(dDay_Month)

        //파이어베이스 데이터베이스에 저장합니다.
        FirebaseDatabase.getInstance().reference
            .child("users")
            .child("${FirebaseAuth.getInstance().currentUser?.uid}")
            .child("d_day_day")
            .setValue(dDay_Day)

        Log.d("리스너 제거", "리스너 제거")
        calendarView.unMarkDate(DateData(dDay_Year, dDay_Month, dDay_Day))
    }

    fun checkTestDate() {

```



```

var date: LocalDate? = null

try {
    for (data in this!!.Xmldata!!) {
        //필기시험 일자
        try {
            if (data.docexamdt != "XXXXXXXX" || data.docexamdt != "") {
                date = LocalDate.parse(
                    data.docexamdt,
                    DateTimeFormatter.ofPattern("yyyyMMdd")
                )
                calendarView.markDate(
                    DateData(
                        date.year,
                        date.monthValue,
                        date.dayOfMonth
                    ).setMarkStyle(MarkStyle.BACKGROUND, Color.MAGENTA)
                )
                Toast.makeText(requireContext(), "마킹됨", Toast.LENGTH_LONG).show()
                Log.d("date", "${date.year} , ${date.monthValue} , ${date.dayOfMonth}")
            }
            if (data.docpassdt != "XXXXXXXX") {
                //필기시험 합격자 발표일
                date = LocalDate.parse(
                    data.docpassdt,
                    DateTimeFormatter.ofPattern("yyyyMMdd")
                )
                calendarView.markDate(
                    DateData(
                        date.year,
                        date.monthValue,
                        date.dayOfMonth
                    ).setMarkStyle(MarkStyle.BACKGROUND, Color.CYAN)
                )
            }
        } catch (e: DateTimeParseException){
            e.printStackTrace()
        }
    }
} catch (e : NullPointerException){
    e.printStackTrace()
}
}

```

//-----

```

fun discheckTestDate() {

    var date : LocalDate? = null
    try {
        for (data in this!!.Xmldata!!) {

            try {

                //필기시험 일자
                if (data.docexamdt != "XXXXXXXX") {
                    date = LocalDate.parse(

```

```

        data.docexamdt,
        DateTimeFormatter.ofPattern("yyyyMMdd")
    )
    calendarView.unMarkDate(
        DateData(
            date.year,
            date.monthValue,
            date.dayOfMonth
        ).setMarkStyle(MarkStyle.BACKGROUND, Color.MAGENTA)
    )
    Toast.makeText(requireContext(), "마킹됨", Toast.LENGTH_LONG).show()
    Log.d("date", "${date.year} , ${date.monthValue} , ${date.dayOfMonth}")
}
if (data.docpassdt != "XXXXXXXX") {
    //필기시험 합격자 발표일
    date = LocalDate.parse(
        data.docpassdt,
        DateTimeFormatter.ofPattern("yyyyMMdd")
    )
    calendarView.unMarkDate(
        DateData(
            date.year,
            date.monthValue,
            date.dayOfMonth
        ).setMarkStyle(MarkStyle.BACKGROUND, Color.CYAN)
    )
}
} catch (e: DateTimeParseException){
    e.printStackTrace()
}
}
} catch (e : NullPointerException){
    e.printStackTrace()
}
}
}

```

//-----

```

fun whatWantCertificat(){

    val user = FirebaseAuth.getInstance().currentUser
    //원하는 자격증을 바인딩합니다.
    //d-day도 가져옵니다.
    if (user != null) {
        FirebaseDatabase.getInstance().reference
            .child("users")
            .child(user.uid)
            .addListenerForSingleValueEvent(object : ValueEventListener {
                override fun onCancelled(p0: DatabaseError) {
                }

                override fun onDataChange(p0: DataSnapshot) {
                    try {

                        var data = p0?.value as Map<String, Any>

                        result = data["cert1"].toString()
                    }
                }
            })
    }
}

```

```

        dDay_Year = (data["d_day_year"] as Long).toInt()
        dDay_Month = (data["d_day_month"] as Long).toInt()
        dDay_Day = (data["d_day_day"] as Long).toInt()

        Log.d("onDataChange_d-day", "${dDay_Year} ${dDay_Month}
        ${dDay_Day}")

        if (dDay_Year == 0) {
            //디데이 지정을 하지 않은 경우
            //디데이를 설정할 때까지 감추기
            tv_content.visibility = View.INVISIBLE

            //오늘을 디데일로 하기
            dDay_Year = now.format(DateTimeFormatter.ofPattern("yyyy")).toInt()
            dDay_Month = now.format(DateTimeFormatter.ofPattern("MM")).toInt()
            dDay_Day = now.format(DateTimeFormatter.ofPattern("dd")).toInt()
        }

        year = now.format(DateTimeFormatter.ofPattern("yyyy")).toInt()
        month = now.format(DateTimeFormatter.ofPattern("MM")).toInt()
        day = now.format(DateTimeFormatter.ofPattern("dd")).toInt()

        checkD_Day()

    } catch (e: TypeCastException) {
        e.printStackTrace()
    }
    //원하는자격증에 따라 url을 호출합니다.
    callUrlAndXmlParse(result)
}
}
}

fun checkD_Day() {

    dCalendar.set(dDay_Year, dDay_Month - 1, dDay_Day)

    var t = calendar.timeInMillis
    var d = dCalendar.timeInMillis
    //디데이 날짜에서 오늘 날짜를 뺀 값을 일 단위로 바꾼다.
    var r = (d - t) / (dayOfMilliSecond)

    calendarView.markDate(
        DateData(dDay_Year, dDay_Month, dDay_Day)
            .setMarkStyle(MarkStyle.LEFTSIDEBAR, Color.YELLOW)
    )
    try {
        Log.d("today", "${year} ${month} ${day}")
        if (r > 0) {
            tv_content.text = "D-day : -" + r.toString()
        } else if (r == 0L) {
            tv_content.text = "D-day : D-Day!!"
        } else {
            tv_content.text = "D-day : +" + (r * -1).toString()
        }
    }
}

```

```

    }
    Log.d("d-day", "${r.toString()}")
} catch (e: IllegalStateException) {
    e.printStackTrace()
} catch (e: java.lang.NullPointerException) {
    e.printStackTrace()
}
}

fun callUrlAndXmlParse(result: String) {

    var url = ""

    when (result) {
        "기사" -> {
            url =

"http://openapi.q-net.or.kr/api/service/rest/InquiryTestInformationNTQ SVC/getEList?serviceKey=QR6Eti6
A0zHnybQIRlidIklUWdlf9bl5bt3coyBro2ldfN%2FsxvulwJ8O4HNQA hBV%2Fia8yktKc1xmVa29qQaDMA%3D
%3D&"

        }
        "기능사" -> {
            url =

"http://openapi.q-net.or.kr/api/service/rest/InquiryTestInformationNTQ SVC/getCList?serviceKey=QR6Eti6
A0zHnybQIRlidIklUWdlf9bl5bt3coyBro2ldfN%2FsxvulwJ8O4HNQA hBV%2Fia8yktKc1xmVa29qQaDMA%3D
%3D&"

        }
        "기능장" -> {
            url =

"http://openapi.q-net.or.kr/api/service/rest/InquiryTestInformationNTQ SVC/getMCList?serviceKey=QR6Et
i6A0zHnybQIRlidIklUWdlf9bl5bt3coyBro2ldfN%2FsxvulwJ8O4HNQA hBV%2Fia8yktKc1xmVa29qQaDMA%3
D%3D&"

        }
        "기술사" -> {
            url =

"http://openapi.q-net.or.kr/api/service/rest/InquiryTestInformationNTQ SVC/getPEList?serviceKey=QR6Et
i6A0zHnybQIRlidIklUWdlf9bl5bt3coyBro2ldfN%2FsxvulwJ8O4HNQA hBV%2Fia8yktKc1xmVa29qQaDMA%3
D%3D&"

        }
    }

    requestVolley(url)
}

//-----
//-----
//volley로 url 호출의 값을 받아오는 역할을 합니다.
fun requestVolley(url: String) {
    //volley로 함격을 api값 가져오기
    var result : String? = null
    // Volley의 request queue를 만듭니다.
    val queue = Volley.newRequestQueue(context)

    val stringRequest = StringRequest(

```

```

Request.Method.GET, url,
Response.Listener<String> { response ->
    // Display the first 500 characters of the response string.
    result = String(response.toByteArray())

    //xml 파싱을 위한 준비를 하고 파싱을 합니다.
    xmlparseReady(result!!)

},
Response.ErrorListener { Log.d("Volley","Volley에러") }
)

// url request를 실행합니다.
queue.add(stringRequest)
}

```

//-----

```

//xml 파싱을 위한 준비를 하고 xml 파싱을 합니다.
fun xmlparseReady(result: String) {
    val pullParserFactory : XmlPullParserFactory
    try {
        pullParserFactory = XmlPullParserFactory.newInstance()
        val parser = pullParserFactory.newPullParser()
        val inputStream = ByteArrayInputStream(result?.toByteArray(charset("euc-kr")))
        parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES,false)
        parser.setInput(inputStream,null)

        //가져온 값을 파싱해줍니다.
        Xmlldata = parseXml(parser)

        //파싱이 완료되고 실행하는 기능입니다.
        //시험 일정을 체크하는 함수입니다.
        checkTestDate()
        Log.d("checkComplete","checkComplete")

    }catch (e : XmlPullParserException){
        e.printStackTrace()
    }catch (e : IOException){
        e.printStackTrace()
    }
}

```

//-----

```

//Xml 파싱하는 것을 담당하는 함수입니다.
@Throws (XmlPullParserException::class, IOException::class)
fun parseXml(parser: XmlPullParser?): ArrayList<certificationTestXmlData>? {
    var dataArray : ArrayList<certificationTestXmlData>? = null
    var eventType = parser?.eventType
    var data : certificationTestXmlData? = null

    while (eventType != XmlPullParser.END_DOCUMENT){
        val tagName : String
        when(eventType){

```

```

XmlPullParser.START_DOCUMENT->dataArray = ArrayList()
XmlPullParser.START_TAG -> {
    tagName = parser!!.name
    if (tagName == "item"){
        data = certificationTestXmlData()
    }else if (data != null){
        if (tagName == "description"){
            data.description = parser.nextText()
        }
        if (tagName == "docexamdt"){
            data.docexamdt = parser.nextText()
        }
        if (tagName == "docpassdt"){
            data.docpassdt = parser.nextText()
        }
        if (tagName == "docregenddt"){
            data.docregenddt = parser.nextText()
        }
        if (tagName == "docregstartdt"){
            data.docregstartdt = parser.nextText()
        }
        if (tagName == "pracexamenddt"){
            data.pracexamenddt = parser.nextText()
        }
        if (tagName == "pracexamstartdt"){
            data.pracexamstartdt = parser.nextText()
        }
        if (tagName == "pracpassdt"){
            data.pracpassdt = parser.nextText()
        }
        if (tagName == "pracregenddt"){
            data.pracregenddt = parser.nextText()
        }
        if (tagName == "pracregstartdt"){
            data.pracregstartdt = parser.nextText()
        }
    }
}
}
XmlPullParser.END_TAG -> {
    tagName = parser!!.name
    if (tagName.equals("item", ignoreCase = true) && data != null){
        dataArray!!.add(data)
    }
}
}
eventType = parser!!.next()
}
return dataArray
} //end of ParseXml
}

```

화면(UI)	설명
	<p>메인화면의 프래그먼트중 하나인 합격예측 프래그먼트입니다.</p> <p>자신이 예정된 시험에서 합격할지 못하는지 재미삼아 예측을 하는 기능입니다.</p> <p>처음에 프래그먼트를 실행하면 자신의 자격증의 대분류(기사, 기능사)에 맞는 필기시험 합격률 Api를 호출 반환된 xml파일을 파싱하여 각 합격률에 맞는 시험 통과를 예측할 수 있습니다.</p> <p>예를 들어 기능장의 합격률이 33%이면 33%의 확률도 합격이 나옵니다. (Toast메세지의 내용 : 자신의 합격률, Api의 필기시험 합격률)</p> <p>또한 수정구슬을 scratchView로 만들어서 즉석복권 긁듯이 만들 수 있게 하였습니다.</p> <p>일정 퍼센테이지 이상 긁으면 scratchView가 사라지면서 합격 불합격이 완전하게 나타나게 됩니다.</p> <p>또한 Api로 호출을 하고 합격률을 예측할 때까지 합격 유무를 확인할 수 없게 하였으며 내부적으로 합격유무가 준비가 되면 손가락이 나와서 움직이며 준비가 되었음을 알려줍니다.</p>
predictionFragment.kt	
<pre>// 합격예측 프래그먼트의 코틀린파일 입니다.  class predictionFragment : Fragment() {      //api의 값     var predictXml : ArrayList&lt;predictXmlData&gt;? = null     val percentOfOpenPredict = 0.085f      //-----     //onStart 함수입니다.     @RequiresApi(Build.VERSION_CODES.O)     override fun onStart() {         super.onStart()     } }</pre>	

```

//volley 호출을 합니다.
//내부적으로 xml과 이미지 세팅도 합니다.
//volley에서 값을 받은 다음에만 가능하기 때문에 함수를 따로 분리하지 못했습니다.
val url =
"http://openapi.q-net.or.kr/api/service/rest/InquiryStatSVC/getGradPiPassList?serviceKey=5uS4tOJQnOf
w3%2FjgZo8p9AY9kOz3VywROltpVQrcjFIHYo6OWPw5yJKuqvm4A%2BJ6mPHQjFhJxuZd%2BqkLBw4T0w
%3D%3D&baseYY=2019&"
requestVolley(url)

// 스크래치 기능의 리스너를 넣었습니다.
settingScratchView()

} //end of onStart

//-----
//volley로 url 호출의 값을 받아오는 역할을 합니다.
fun requestVolley(url: String) {
    //volley로 합격률 api값 가져오기
    var result : String? = null
    // Volley의 request queue를 만듭니다.
    val queue = Volley.newRequestQueue(context)

    val stringRequest = StringRequest(
        Request.Method.GET, url,
        Response.Listener<String> { response ->
            // Display the first 500 characters of the response string.
            result = String(response.toByteArray(charset("euc-kr")), charset("euc-kr"))

            //xml 파싱을 위한 준비를 하고 파싱을 합니다.
            xmlparseReady(result!!)

            //volley로 값을 가져온 상태에서만 가능하기 때문에 여기에 위치되어있습니다.
            //랜덤 함수를 해서 합격 예측을 합니다.
            settingPredictAndSettingImage()
        },
        Response.ErrorListener { Log.d("Volley","Volley에러") }
    )

    // url request를 실행합니다.
    queue.add(stringRequest)
}

//-----
//랜덤한 값을 받아서 xml에서 받은 합격률보다 높으면 불합
//낮으면 합격 처리로 예언하는 함수입니다.
fun settingPredictAndSettingImage() {
    var predictResult = Random(LocalDateTime.now().hashCode()).nextInt(100)+1
    Log.d("time","${LocalDateTime.now()}")
    var cert1 = ""

    FirebaseAuth.getInstance().currentUser?.uid?.let {
        FirebaseDatabase.getInstance().reference
            .child("users")
            .child(it)
    }
}

```



```

        .addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onCancelled(p0: DatabaseError) {
            }

            override fun onDataChange(p0: DataSnapshot) {
                var data = p0.value as Map<String,Any>
                cert1 = data["cert1"].toString()

                // 각 합격률에 맞추어서 합격 예측을 합니다.
                when(cert1){
                    "기능사" -> { setPredict(predictResult , predictXml?.get(5)?.starisyy1) }
                    "기사" -> {setPredict(predictResult , predictXml?.get(2)?.starisyy1)}
                    "기능장" -> {setPredict(predictResult , predictXml?.get(1)?.starisyy1)}
                    "기술사" -> {setPredict(predictResult , predictXml?.get(0)?.starisyy1)}
                    else -> {Toast.makeText(context,"cert1의 값이 이상합니다.
                    ${cert1}",Toast.LENGTH_LONG).show()}
                }
            }
        }).toString()
    }
}

//-----
//xml 파싱을 위한 준비를 하고 xml 파싱을 합니다.
fun xmlparseReady(result: String) {
    val pullParserFactory : XmlPullParserFactory
    try {
        pullParserFactory = XmlPullParserFactory.newInstance()
        val parser = pullParserFactory.newPullParser()
        val inputStream = ByteArrayInputStream(result?.toByteArray(charset("euc-kr")))
        parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES,false)
        parser.setInput(inputStream,null)

        //가져온 값을 파싱해줍니다.
        predictXml = parseXml(parser)

    }catch (e : XmlPullParserException){
        e.printStackTrace()
    }catch (e : IOException){
        e.printStackTrace()
    }
}

//-----
//스크래치 뷰의 리스너를 지정해 줍니다.
//일정 확률 이상스크래치를 하면 스크래치 뷰가 사라지면서 전체 화면이 나타나게 됩니다.
//또한 약간이라도 굵으면 손가락 애니메이션이 보이지 않게 만들었습니다.
fun settingScratchView() {

    scratchView.setRevealListener(object : ScratchView.IRevealListener{
        override fun onRevealed(scratchView: ScratchView?) {
            AnimatorInflater.loadAnimator(context,R.animator.predicr_clear_animator).apply {
                setTarget(scratchView)
                start()
            }
        }
    })
}

```

```

    }
}

override fun onRevealPercentChangedListener(scratchView: ScratchView?, percent: Float)

{
    Log.d("percent","${percent}")

    //핑거를 사라지게 합니다.
    fingerImage.clearAnimation()
    fingerImage.visibility = View.INVISIBLE

    if (percent > percentOfOpenPredict){
        //0.1퍼센트 이상 큼으면 onRevealed 함수를 진행하고 리스너를 해제합니다.
        this.onRevealed(scratchView)

        //리스너를 해제합니다.
        scratchView?.setRevealListener(object :ScratchView.IRevealListener{
            override fun onRevealed(scratchView: ScratchView?)
            {

            }

            override fun onRevealPercentChangedListener(
                scratchView: ScratchView?,
                percent: Float
            ) {

            }

        })
    }
}

})//end of setRevealListener
}

//-----
//합격이미지를 설정하고 손가락 애니메이션의 작동을 담당합니다.
fun setPredict(predictResult: Int, starisyy1: String?) {
    //합격 확률 안쪽으로 들어온 경우
    if (predictResult < starisyy1!!.toInt()){
        predictionResultImageView?.setBackgroundColor(
            requireContext().getColor(R.color.goodnews) )
        predictionResultImageView?.setImageResource(R.drawable.crystalballgood)
    }else{
        predictionResultImageView?.setBackgroundColor(
            requireContext().getColor(R.color.badnews) )
        predictionResultImageView?.setImageResource(R.drawable.crystalballbad)
    }

    //이미지가 지정이 되면 예언을 하는 이미지가 사라지면서 손가락으로 스와이프를하는 애니메이
    션이 생성됩니다.
    //빠르게 화면전환을 했을때 예러가 나는것을 try로 잡았습니다.
    try {
        Toast.makeText(requireContext(),"${predictResult}
        ${starisyy1}",Toast.LENGTH_LONG).show()
        AnimatorInflater.loadAnimator(requireContext(), R.animator.predicr_clear_animator)
        ?.apply {
            addListener(object : AnimatorListenerAdapter() {

```

```

        //애니메이션이 끝나면 이미지를 안보이게 해서 수정이 클릭되게 합니다.
        override fun onAnimationEnd(animation: Animator?) {
            imageView2?.visibility = View.INVISIBLE
        }
    })
    setTarget(imageView2)
    start()
}

fingerImage?.visibility = View.VISIBLE
val animation = AnimationUtils.loadAnimation(requireActivity(), R.anim.finger_animation)
fingerImage?.startAnimation(animation)
} catch (e : IllegalStateException){
    e.printStackTrace()
}
}
}

```


//-----

```

//Xml 파싱하는 것을 담당하는 함수입니다.
@Throws (XmlPullParserException::class, IOException::class)
fun parseXml(parser: XmlPullParser?): ArrayList<predictXmlData>? {
    var dataArray : ArrayList<predictXmlData>? = null
    var eventType = parser?.eventType
    var data : predictXmlData? = null

    while (eventType != XmlPullParser.END_DOCUMENT){
        val tagName : String
        when(eventType){
            XmlPullParser.START_DOCUMENT->dataArray = ArrayList()
            XmlPullParser.START_TAG -> {
                tagName = parser!!.name
                if (tagName == "item"){
                    data = predictXmlData()
                } else if (data != null){
                    if (tagName == "statisyy1"){
                        data.starisyy1 = parser.nextText()
                    }
                }
            }
            XmlPullParser.END_TAG -> {
                tagName = parser!!.name
                if (tagName.equals("item", ignoreCase = true) && data != null){
                    dataArray!!.add(data)
                }
            }
        }
        eventType = parser!!.next()
    }
    return dataArray
} //end of ParseXml

```

화면(UI)	설명
	<p>메인화면의 커뮤니티 기능을 담당하는 게시판의 리스트를 보여주는 Fragment입니다.</p> <p>안드로이드 with Kotlin 책의 익명 SNS앱을 참조하여 만들었습니다.</p> <p>차별점으로 게시글을 StaggeredGridLayoutManager를 이용해서 뒤틀린 격자형태로 게시글이 보이도록 구현했습니다.</p> <p>또한 익명이 아닌 로그인한 사용자의 닉네임으로 글을 적게 만들었습니다.</p> <p>Floating Action Button을 누르면 글쓰는 화면으로 넘어갑니다.</p> <p>게시글을 누르면 글 상세보기 화면으로 넘어갑니다.</p> <p>파이어베이스의 실시간 데이터베이스에서 글을 읽어옵니다.</p> <p>프래그먼트에서 다른 프래그먼트로 전환시 파이어베이스의 리스너가 살아있는데 할당된 뷰가 사라져서 나오는 에러를 onPause()함수에서 리스너를 해제하는 방법으로 해결하였습니다.</p>
noticeBoardFragment.kt	
<pre> class noticeBoardFragment : Fragment() {      val posts:MutableList&lt;Post&gt; = mutableListOf()     var listener : ChildEventListener? = null      //프래그먼트 전환시 리스너를 해제합니다.     override fun onPause() {         super.onPause()         listener?.let { it1 -&gt;             FirebaseDatabase.getInstance().getReference("/Posts").removeEventListener(                 it1             )             Log.d("리스너 제거","리스너 제거")         }     }      //프래그먼트 전환시 리스너를 해제합니다.     override fun onResume() {         super.onResume() </pre>	

```

floatingActionButton.setOnClickListener {
    listener?.let { it1 ->
        FirebaseDatabase.getInstance().getReference("/Posts").removeEventListener(
            it1
        )
        Log.d("리스너 제거", "리스너 제거")
    }
    (activity as MainActivity).setFragment(writeFragment.newInstance())
}

//
var layoutManager = StaggeredGridLayoutManager(2, LinearLayoutManager.VERTICAL)
layoutManager.reverseLayout = true
notice_rv.layoutManager = layoutManager
notice_rv.adapter = MyAdapter()

//파이어베이스에서 게시글읽어오기
listener = FirebaseDatabase.getInstance().getReference("/Posts")
    .orderByChild("descWriteTime").addChildEventListener(object : ChildEventListener {
        //취소된 경우
        override fun onCancelled(snapshot: DatabaseError) {
            snapshot?.toException()?.printStackTrace()
        }

        //글의 순서가 이동된 경우
        override fun onChildMoved(snapshot: DataSnapshot, prevChildKey: String?) {
            snapshot?.let {
                val post = snapshot.getValue(Post::class.java)

                post?.let { post ->
                    //기존의 인덱스를 구한다.
                    val existIndex = posts.map { it.postId }.indexOf(post.postId)
                    //기존의 데이터를 지운다.
                    posts.removeAt(existIndex)
                    notice_rv.adapter?.notifyItemRemoved(existIndex)

                    //prevChildKey가 없는 경우 맨 마지막으로 이동 된것
                    if (prevChildKey == null) {
                        posts.add(post)
                        notice_rv.adapter?.notifyItemChanged(posts.size - 1)
                    } else {
                        //prev키 다음글로 추가한다.
                        val prevIndex = posts.map { it.postId }.indexOf(prevChildKey)
                        posts.add(prevIndex + 1, post)
                        notice_rv.adapter?.notifyItemChanged(prevIndex + 1)
                    }
                }
            }
        }

        //글이 변경되는 경우
        override fun onChildChanged(snapshot: DataSnapshot, prevChildKey: String?) {
            snapshot?.let { snapshot ->
                //snapshot의 데이터를 뽑아 Post객체로 가져옴
                val post = snapshot.getValue(Post::class.java)
                post?.let { post ->
                    //글이 변경된 경우 글의 앞의 데이터 인덱스에 데이터를 변경한다.

```

```

        val prevIndex = posts.map { it.postId }.indexOf(prevChildKey)
        posts[prevIndex + 1] = post
        notice_rv.adapter?.notifyItemChanged(prevIndex + 1)
    }
}

//글이 추가된 경우
override fun onChildAdded(snapshot: DataSnapshot, prevChildKey: String?) {
    snapshot?.let { snapshot ->
        //snap의 데이터를 Post객체로 가져옴
        val post = snapshot.getValue(Post::class.java)
        post?.let {
            Log.d("실행", "*****")
            //새글이 마지막 부분에 추가된 경우
            if (prevChildKey == null) {
                //글 목록을 저장하는 변수에 post객체 추가
                posts.add(it)
                //리사이클러 뷰에 글이 추가된것을 알린다.
                notice_rv.adapter?.notifyItemInserted(posts.size - 1)
            } else {
                //글이 중간에 삽입된 경우 prevChildKey로 한단계 앞의 데이터의 위치
                val prevIndex = posts.map { it.postId }.indexOf(prevChildKey)
                posts.add(prevIndex + 1, post)
                //리사이클러뷰 업데이트
                notice_rv.adapter?.notifyItemInserted(prevIndex + 1)
            }
        }
    }
}

//글이 삭제된 경우
override fun onChildRemoved(snapshot: DataSnapshot) {
    snapshot?.let {
        val post = snapshot.getValue(Post::class.java)

        post?.let { post ->
            val existIndex = posts.map { it.postId }.indexOf(post.postId)
            posts.removeAt(existIndex)
            notice_rv.adapter?.notifyItemRemoved(existIndex)
        }
    }
}

//게시글의 어댑터 입니다.
inner class MyViewHolder(itemView : View) : RecyclerView.ViewHolder(itemView){
    val writerId = itemView.witterName_text
    val title = itemView.title
    val content = itemView.subtitle
    val timeTextView = itemView.timeTextView
    val commentCountText = itemView.commentCountText
    val image = itemView.imageView3
}

inner class MyAdapter : RecyclerView.Adapter<MyViewHolder>() {

```

```

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
            return
MyViewHolder(LayoutInflater.from(requireContext()).inflate(R.layout.fragment_board_list_item,parent,false
))
        }

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

        override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
            val post = posts[position]

            //데이터 바인딩
            if (!post.bgUri.isNullOrBlank()){
                Picasso.get().load(post.bgUri).fit().centerCrop().into(holder.image)
            }else{
                holder.image.setImageResource(0)
            }
            holder.title.text = post.title
            holder.content.text = post.message
            holder.timeTextView.text = getDiffTimeText(post.writeTime as Long)
            holder.commentCountText.text = post.commentCount.toString()+" 개"
            holder.writterId.text = post.writterId

            /*카드가 클릭이 되는 경우 DetailActivity를 실행한다.*/
            holder.itemView.cardView.setOnClickListener {
                listener?.let { it1 ->
                    FirebaseDatabase.getInstance().getReference("/Posts").removeEventListener(
                        it1
                    )
                    Log.d("리스너 제거","리스너 제거")
                }
                (activity as MainActivity).setFragment(detailFragment.newInstance(post))
            }
            /*카드가 클릭이 되는 경우 DetailActivity를 실행한다.*/
            holder.itemView.subtitle.setOnClickListener {
                listener?.let { it1 ->
                    FirebaseDatabase.getInstance().getReference("/Posts").removeEventListener(
                        it1
                    )
                    Log.d("리스너 제거","리스너 제거")
                }
                (activity as MainActivity).setFragment(detailFragment.newInstance(post))
            }
        }
    }
}

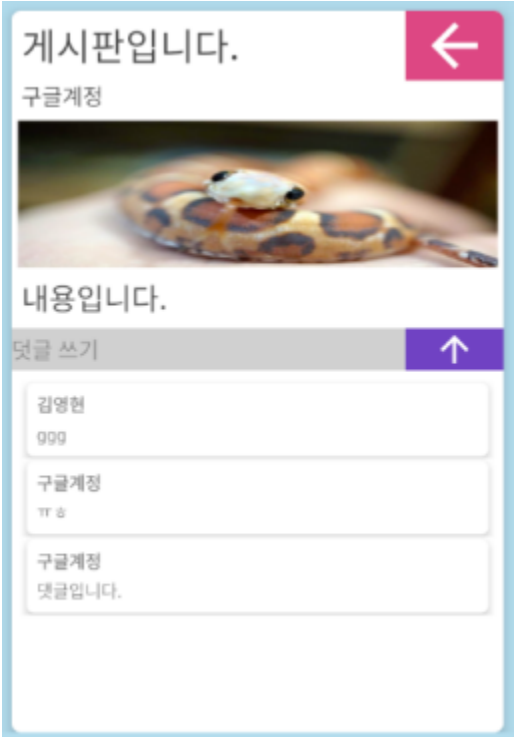
//게시글이 언제 나왔는지 알려주는 함수
fun getDiffTimeText(targetTime: Long): String {
    val curDateTime = DateTime()
    val targetDateTime = DateTime().withMillis(targetTime)
    val diffDay = Days.daysBetween(curDateTime, targetDateTime).days
    val diffHours = Hours.hoursBetween(targetDateTime, curDateTime).hours
    val diffMinutes = Minutes.minutesBetween(targetDateTime, curDateTime).minutes
    if (diffDay == 0) {
        if (diffHours == 0 && diffMinutes == 0) {
            return "방금 전"
        }
    }
    return if (diffHours > 0) {

```

```

        "" + diffHours + "시간 전"
    } else "" + diffMinutes + "분 전"
    } else {
        val format = SimpleDateFormat("yyyy년 MM월 dd일 HH:mm")
        return format.format(Date(targetTime))
    }
    }
}
}
}

```

화면(UI)	설명
	<p>게시판의 상세보기 화면입니다.</p> <p>게시글을 상세적으로 보여줍니다.</p> <p>또한 댓글 기능을 넣어서 댓글을 적고 버튼을 누르면 바로 댓글이 등록되도록 구현하였습니다.</p> <p>피카소 라이브러리를 이용해서 이미지를 로딩하는 작업을 하였습니다.</p> <p>댓글역시 파이어베이스의 실시간 데이터베이스를 이용해서 구현했습니다.</p> <p>댓글의 수를 받아와서 댓글을 쓸때마다 댓글수의 카운트를 늘리고 마지막으로 화면을 벗어날 때 게시글의 수를 파이어베이스에 저장하는 방식으로 구현했습니다.</p>

detailFragment.kt

```

class detailFragment(data: Post?) : Fragment() {

    val commentList = mutableListOf<Comment>()
    var commentlistener : ChildEventListener? = null
    var commentCount = 0

    var post = data

    override fun onCreateView(view: View, savedInstanceState: Bundle?) {
        super.onCreateView(view, savedInstanceState)

        backButton.setOnClickListener {
            (activity as MainActivity).setFragment(noticeBoardFragment.newInstance())
        }

        //이미 있던 댓글의 수
        commentCount = post?.commentCount!!

        detail_send_comment.setOnClickListener {
            var comment = Comment()
            var newRef =
                FirebaseDatabase.getInstance().getReference("Comments/${post?.postId}").push()

```



```

        comment.message = commentEditText.text.toString()
        commentEditText.setText("")
        comment.writeTime = ServerValue.TIMESTAMP
//        comment.writerId = FirebaseAuth.getInstance().currentUser?.displayName.toString()
        comment.postId = post?.postId.toString()
        comment.commentId = FirebaseAuth.getInstance().currentUser?.displayName.toString()

        newRef.setValue(comment)
        Toast.makeText(requireContext(), "저장 성공했습니다.", Toast.LENGTH_LONG).show()

        //댓글수 업데이트
        var commentListCount = FirebaseDatabase.getInstance().getReference("/Posts")
            .child("${post?.postId}")
        commentListCount.child("commentCount").setValue(commentCount + 1)
        commentCount += 1
    }

    detail_title_text.text = post?.title
    detail_content_text.text = post?.message
    detail_witter_text.text = post?.writerId
    try {
        Picasso.get()
            .load(post?.bgUri)
            .fit()
            .into(detail_ImageView)
    } catch (e: IllegalArgumentException) {
        e.printStackTrace()
    }

    val layoutManager = LinearLayoutManager(requireContext())
    comment_rv.layoutManager = layoutManager
    comment_rv.adapter = MyAdapter()

    //게시글의 ID로 댓글 목록에 ChildEventListener을 등록한다.
    commentlistener =
    FirebaseDatabase.getInstance().getReference("/Comments/${post?.postId}")
        .addChildEventListener(object : ChildEventListener{
            override fun onCancelled(p0: DatabaseError) {
                p0.toException()?.printStackTrace()
            }

            override fun onChildMoved(snapshot: DataSnapshot, prevChildKey: String?) {
                if (snapshot != null){
                    val comment = snapshot.getValue(Comment::class.java)
                    comment?.let {
                        val existIndex = commentList.map { it.commentId
}.indexOf(it.commentId)
                        //기존의 데이터를 지운다.
                        commentList.removeAt(existIndex)

                        //prevKey다음에 글에 추가를 한다.
                        val prevIndex = commentList.map { it.commentId
}.indexOf(prevChildKey)

                        commentList.add(prevIndex + 1, it)
                        comment_rv.adapter?.notifyItemInserted(prevIndex + 1)
                    }
                }
            }
        })
    }

```

```

    }

    override fun onChildChanged(snapshot: DataSnapshot, prevChildKey: String?) {
        snapshot?.let { snapshot ->
            val comment = snapshot.getValue(Comment::class.java)
            comment?.let {
                //prevKey다음에 글에 추가를 한다.
                val prevIndex = commentList.map { it.commentId
}.indexOf(prevChildKey)

                commentList[prevIndex+1] = comment
                comment_rv.adapter?.notifyItemChanged(prevIndex + 1)
            }
        }
    }

    override fun onChildAdded(snapshot: DataSnapshot, prevChildKey: String?) {
        snapshot?.let { snapshot ->
            val comment = snapshot.getValue(Comment::class.java)
            comment?.let {
                Log.d("덧글 등록", "덧글 등록")
                //새글의 마지막 부분에 추가한다.
                val prevIndex = commentList.map { it.commentId
}.indexOf(prevChildKey)

                commentList.add(prevIndex+1, comment)
                comment_rv.adapter?.notifyItemInserted(prevIndex + 1)
                comment_rv.scrollToPosition(prevIndex + 1)
            }
        }
    }

    override fun onChildRemoved(snapshot: DataSnapshot) {
        snapshot?.let { snapshot ->
            val comment = snapshot.getValue(Comment::class.java)
            comment?.let {
                //새글의 마지막 부분에 추가한다.
                val existIndex = commentList.map { it.commentId
}.indexOf(comment.commentId)

                commentList.removeAt(existIndex)
                comment_rv.adapter?.notifyItemRemoved(existIndex)
            }
        }
    }

    })

}

//화면을 벗어날 때 리스너를 제거합니다.
override fun onPause() {
    super.onPause()
    commentlistener?.let { it1 ->

        FirebaseDatabase.getInstance().getReference("/Comments/${post?.postId}").removeEventListener(
            it1
        )
        Log.d("리스너 제거", "리스너 제거")
    }
}

```

```

    }

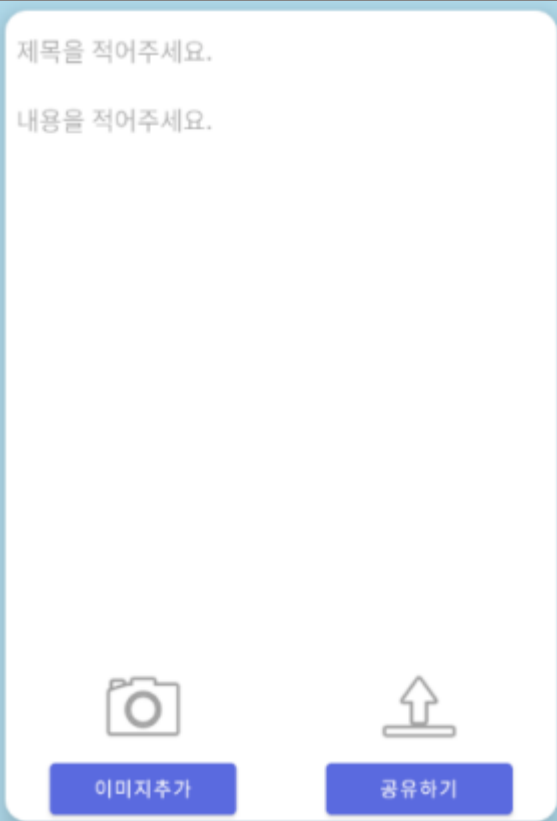
//댓글의 어댑터패턴 입니다.
    inner class MyViewHolder(itemView : View):RecyclerView.ViewHolder(itemView){
        val witter = itemView.comment_NickName_text
        val content = itemView.comment_content_text
    }

    inner class MyAdapter : RecyclerView.Adapter<MyViewHolder>() {
        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
            return
MyViewHolder(LayoutInflater.from(requireContext()).inflate(R.layout.card_comment,parent,false))
        }

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

        override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
            val comment = commentList[position]
            comment?.let {
                holder.witter.text = comment.commentId
                holder.content.text = comment.message
            }
        }
    }
}
}

```

화면(UI)	설명
	<p>글쓰기 화면 Fragment입니다.</p> <p>기존의 안드로이드 with Kotlin 책의 익명 SNS의 글쓰기 화면을 참조하여 만들었습니다.</p> <p>이미지 추가를 하면 암시적 인텐트를 이용해서 안드로이드 OS에서 사진을 uri로 받아옵니다.</p> <p>그후 공유하기를 누르면 먼저 uri의 사진을 파이어베이스의 Storage로 올리고 서버로 올라간 이미지의 다운로드 uri를 받아옵니다.</p> <p>그 후 받아온 uri를 다시 구글 파이어베이스의 실시간 데이터베이스에 등록해서 해당 uri를 피카소 라이브러리로 불러와서 표시할 수 있게 합니다.</p> <p>또한 내용 체크를 해서 내용이 없을 경우 글을 올리지 못하게 합니다.</p> <p>또한 글을 올릴 때 시간값 * -1을 해서 나중에 시간값으로 정렬을 하면 최신글이 위로 올라오게 만들었습니다.</p>

```

class writeFragment : Fragment() {

    val GET_GALLERY_IMAGE = 200;

    //선택된 이미지의 Uri
    var selectImageUri : Uri? = null

    override fun onCreateView(view: View, savedInstanceState: Bundle?) {
        super.onCreateView(view, savedInstanceState)
        /*이미지 업로드 버튼 구현*/
        galleryButton.setOnClickListener {
            var intent = Intent(Intent.ACTION_PICK)
            intent.setDataAndType(
                android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
                "image/*"
            )
            startActivityForResult(intent, GET_GALLERY_IMAGE)
        }

        /*게시글 올리기 버튼*/
        sendButton.setOnClickListener {
            if (contentEditText.text.isEmpty() || titleEditText.text.isEmpty()) {
                Toast.makeText(requireContext(), "제목과 내용을 입력해 주세요.",
                    Toast.LENGTH_LONG).show()
                return@setOnClickListener
            }
            //Post객체 생성
            val post = Post()
            //참조값을 할당합니다.
            val newRef = FirebaseDatabase.getInstance().getReference("Posts").push()
            //데이터를 할당합니다.
            post.writeTime = ServerValue.TIMESTAMP
            post.title = titleEditText.text.toString()
            post.message = contentEditText.text.toString()
            post.postId = newRef.key.toString()
            post.writerId = FirebaseAuth.getInstance().currentUser?.displayName!!
            post.descWriteTime = -1 * Date().time

            val storage = FirebaseStorage.getInstance().getReference("/image/${post.postId}")
            //나중에 이미지 업로드 부분 시간되면 구현하기
            val upload = selectImageUri?.let { it1 -> storage.putFile(it1) }

            upload?.addOnFailureListener{
                Log.d("image", "이미지 업로드 실패")
            }.addOnSuccessListener {
                Log.d("image", "이미지 업로드 성공")
            }

            val urlTask = upload?.continueWithTask{ task ->
                if (!task.isSuccessful){
                    task.exception?.let {
                        throw it
                    }
                }
            }
            storage.downloadUrl
        }
    }
}

```

```

}?.addOnCompleteListener { task ->
    if (task.isSuccessful){
        post.bgUri = task.result.toString()

        newRef.setValue(post)
        Toast.makeText(requireContext(),"아마자 추가해서 저장
성공!!!",Toast.LENGTH_LONG).show()

        (activity as MainActivity).setFragment(noticeBoardFragment.newInstance())
    }
}

if(uriTask == null) {
    Log.d("게시글 업로드", "게시글 업로드")
    newRef.setValue(post)
    Toast.makeText(requireContext(), "이미지 없이 저장 성공!!!",
Toast.LENGTH_LONG).show()

    (activity as MainActivity).setFragment(noticeBoardFragment.newInstance())
}

}

}

}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == GET_GALLERY_IMAGE && resultCode == RESULT_OK && data != null
&& data.data != null){
        selectImageUri = data.data!!
        imageView6.setImageURI(selectImageUri)
    }
}
}

```

화면(UI)	설명
	<p>메인화면의 계정 설정 화면 Fragment입니다.</p> <p>자신이 설정한 이미지와 닉네임, 이메일, 원하는 자격증을 파이어베이스 데이터베이스에서 가져와서 나오게 만들었습니다.</p> <p>또한 정보를 변경하는 화면 로그아웃버튼 회원 탈퇴 기능을 만들었습니다.</p> <p>이미지를 동그랗게 만드는 Circle Image View를 이용해서 유저 이미지를 동그랗게 만드는 작업을 하였습니다.</p> <p>회원 탈퇴버튼을 누르면 파이어베이스의 Auth부분에서 계정이 삭제 됩니다.</p> <p>로그아웃 버튼을 누르면 세션로그아웃을 하고 로그인 화면으로 넘어갑니다.</p> <p>정보변경 버튼을 누르면 로그인에 성공하고 닉네임을 설정했던 부분으로 넘어갑니다.</p>

```
//사용자 계정 프래그먼트의 코틀린파일입니다.
```

```
class accountConfigurationFragment : Fragment() {
```

```
    val user = FirebaseAuth.getInstance().currentUser
```

```
//-----  
--
```

```
    //onCreateView함수입니다.
```

```
    //MainActivity에서의 로그아웃을 리스너로 등록합니다.
```

```
    override fun onCreateView(
```

```
        inflater: LayoutInflater, container: ViewGroup?,
```

```
        savedInstanceState: Bundle?
```

```
): View? {
```

```
    // Inflate the layout for this fragment
```

```
    val inflater = inflater.inflate(R.layout.fragment_account_configuration, container, false)
```

```
    inflater.logoutButton.setOnClickListener {
```

```
        (activity as MainActivity).signOut()
```

```
    }
```

```
    return inflater
```

```
}//end of onCreateView
```

```
//-----  
--
```

```
    //onStart 함수입니다.
```

```
    override fun onStart() {
```

```
        super.onStart()
```

```
    //화면에 유저 정보를 불러옵니다.
```

```
    binding.userInfo()
```

```
    //정보 변경을 위한 버튼 리스너 입니다.
```

```
    userInfoChangeButton.setOnClickListener {
```

```
        goUserInfoChange()
```

```
    }
```

```
    //회원 탈퇴를 하는 기능입니다.
```

```
    userWithdrawalButton.setOnClickListener {
```

```
        deleteUserInfo()
```

```
    }
```

```
}//end of onStart
```

```
//-----  
-----
```

```
    //유저의 회원 탈퇴를 말합니다.
```

```
    //계정의 삭제와 지정한 자격증 데이터 베이스를 삭제하는 기능을 합니다.
```

```
    fun deleteUserInfo() {
```

```
        //파이어베이스 데이터베이스에서 유저의 데이터를 삭제합니다.
```

```
        //파이어베이스 데이터베이스에 저장합니다.
```

```
        FirebaseDatabase.getInstance().reference
```

```
            .child("users")
```

```
            .child("${FirebaseAuth.getInstance().currentUser?.uid}")
```

```
            .removeValue()
```

```

// 파이어베이스 Auth의 유저를 삭제하는 기능입니다.
user?.delete()
    ?.addOnCompleteListener { task ->
        if (task.isSuccessful){
            Toast.makeText(context,"회원 탈퇴를 했습니다.",Toast.LENGTH_LONG).show()
            startActivity(loginActivity.getLaunchIntent(requireContext()))
            activity?.overridePendingTransition(R.anim.fadein, R.anim.fadeout)
        }
    }
}

//-----
//유저 정보를 바꾸기 위한 화면으로 넘어갑니다.
private fun goUserInfoChange() {
    //여기서는 백스택을 남기면서 이동합니다.
    startActivity(Intent(requireContext(),ConfigurationActivity::class.java))
    activity?.overridePendingTransition(R.anim.fadein, R.anim.fadeout)
    // 정보를 바로 고치기 위해서 onDestroy를 호출해 줍니다.
    onDestroy()
}

//-----
// 화면에 유저정보를 바인딩해주는 함수입니다.
fun bindingUserInfo() {
    // 닉네임을 설정해서 보여주는 역할을 합니다.
    userName.text = user?.displayName.toString()

    // 이메일을 보여줍니다.
    if (userEmailText.text.isNullOrEmpty()){
        userEmailText.text = ""
    }else{
        userEmailText.text = user?.email.toString()
    }

    // 유저 이미지를 가져와서 보여줍니다.
    Picasso.get()
        .load(user?.photoUrl)
        .into(userImage)


    //원하는 자격증을 바인딩합니다.
    if (user != null) {
        FirebaseDatabase.getInstance().reference
            .child("users")
            .child(user.uid)
            .addListenerForSingleValueEvent(object : ValueEventListener{
                override fun onCancelled(p0: DatabaseError) {
                }

                override fun onDataChange(p0: DataSnapshot) {
                    var data = p0?.value as Map<String,Any>
                    userCertificationText.text = data["cert2"].toString()
                }
            }).toString()
    }
}
} //end of bindUserInfo
}

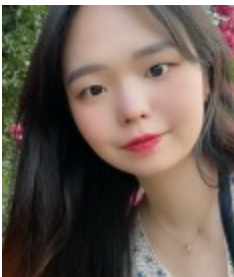
```

## 2. 나의 앱 만들기 후기

이름: 장주명

	<p><b>나의 소감:</b></p> <p>아무리 능력이 있다고 해도 개인에는 분명히 한계가 있습니다. 백지장도 맞들면 낫다는 속담처럼 개발자라는 직업을 가지기 위해서는 동료와의 협업, 커뮤니케이션이 필수입니다. 그래서 1인 프로젝트로 분명 혼자할 수 있었음에도 불구하고 팀 프로젝트를 진행하였습니다. 서로 팀원끼리 못하는 부분을 케어해 주면서 프로젝트를 진행했기에 여기까지 올 수 있었던 것 같습니다. 저 혼자였다면 분명 여기까지 오지 못했을 것입니다.</p> <p>여러명과의 협업을 하다보니 git을 이용한 버전관리를 이용해서 3명에서 같은 프로젝트를 하는데 git을 배우는 시간과 주제선정에서 시간을 많이 잡았지만 물고기가 아닌 낚시를 하는 방법을 배우는 것처럼 결과적으로는 저에게도 팀원들에게도 많은 배움의 기회가 되었을 것이라고 생각합니다.</p> <p>또한 막바지에 D-day기능을 넣자고 제안했는데 이미 만든 프로젝트에서 특정한 기능을 하나 추가한다는 것이 얼마나 어려운 일인지 알아가는 계기가 되었습니다.</p> <p>결과적으로 팀 프로젝트를 하면서 개발자로서의 소양인 소통과 역할분담에 대해 알아보는 계기가 되었다고 생각합니다.</p>
	<p><b>모바일 프로그래밍 강의를 통해 얻은 것:</b> 교수님의 모바일 프로그래밍 강의를 들으면서 참 많은 것을 배우는 계기가 되었습니다. 이전에 미리 안드로이드를 살짝 배워보았지만 그 당시에는 어댑터하나를 제대로 이해하지 못해서 힘겨워 했었습니다. 그래서 이번에 모바일 프로그래밍 수업을 들으면서 원론적인 이해부터 해서 오히려 이해가 잘되는 시간이었다고 생각합니다. 또한 저는 스스로 배울때는 기능적인 부분만을 생각해서 UI/UX쪽으로는 생각도 안하고 있었는데 교수님의 예제는 그러한 UI/UX부분도 깔끔하게 정리된 것을 보면서 보기 좋은 것이 먹기도 좋다는 옛 속담처럼 디자인 부분도 신경을 써야겠다고 느꼈습니다. 그래서 기말 프로젝트를 할 때 디자인 부분에서 “꼭 액션바를 안보이게 하자”라고 생각했었습니다. “뭉든지 확실한 이론을 먼저 배워야 실력이 빨리 늘어난다.”라는 옛말의 의미를 다시 곰 느낀 수업이었습니다.</p>

이름: 김영현

	<p><b>나의 소감:</b></p> <p>처음엔 수업도 겨우 따라가는 내가 어플 하나를 제대로 만들 수 있을까하는 걱정이 컸지만, 같이 하는 조원들의 도움 덕분에 좋은 결과물을 만들어 낼 수 있었던 것 같다.</p> <p>이번 코로나19의 영향으로 대면 회의 없이 웹엑스 화상회의로 모든 걸 진행해야 했는데, 그 과정에서 여러 어려운 점들이 있었지만, git을 사용한 협업과 서로의 배려로 큰 불편함 없이 진행되었다.이 프로젝트를 하고나서 어플의 한 기능을 구현을 한다고 해도 다양하고 많은 디테일을 요구한다는 것을 깨달았고, 타 어플을 사용할 때도 개발자의 관점으로 기능 구현에 대해 자세히 생각해 보게 될 것 같다.</p>
	<p><b>모바일 프로그래밍 강의를 통해 얻은 것:</b>온라인으로 실시간 강의를 들으면서 실습도 같이 해야 한다는 게 쉽지 않았지만, 우리가 어플을 사용하면서 흔히 자주 볼 수 있는 기능을 직접 구현해보는 과정에서 흥미로웠고, 졸업 프로젝트에서 안드로이드 어플을 제작하는데 큰 도움이 될 것 같다.</p>



이름: 방은지



**나의 소감:**

앱 만들기가 기말 과제라는 것을 듣고 처음에는 의욕적이었다. 하지만 주차가 더해 갈수록 점점 더 어려워지는 것을 보며 나의 능력의 한계를 느낄 수 있었다. 많은 부분을 해내기 위해 최선을 다했지만 노력을 한다고 결과로 이어지는 것이 아니라는 것을 깨닫게 되었다. 다행히 팀원들 덕분에 앱 만들기를 무사히 마쳤지만 다시금 공부의 필요성을 느끼는 계기가 되었다. 또한, 앱을 만들기 위해 수업 시간에 배운 것뿐만 아니라 인터넷을 여러 군데 찾아보면서 안드로이드에 대해 더 많이 배울 수 있었던 것 같다. 마지막으로 이번에 앱 만들기를 개인이 아닌 팀 프로젝트로 수행하면서 같이 무언가를 제작해 나가는 것이 어렵지만 끝내고 나니 보람차다는 것을 느꼈다.

**모바일 프로그래밍 강의를 통해 얻은 것:** 이번 학기에 모바일 프로그래밍 수업을 들으면서 사실을 따라가기가 조금 벅찼다. 코로나로 인해 사이버로 수업을 듣게 되면서 프로그램을 돌리고 수업 화면도 보는 것이 낯설었기 때문이다. 그래도 이 수업을 배우면서 힘들었던 것보다는 얻어가는 것이 더 많다고 생각한다.