『KU 동아리 정보 앱 개발』3 조 최종보고서

모바일프로그래밍(3194)

지도교수: 지정희

2024 년 6 월 20 일

202112439 최예름

201911174 박성준

202011293 박성근

202012602 김종우

목차

제 1 장 프로젝트 주제 1-3

제 2 장 상세 구현 방법 1-4

제 1 절 프로젝트 아키텍쳐 1-4

제 2 절 사용자 등록 및 인증 기능 2-4

제 3 절 동아리 소개 및 카테고리 기능 3-6

제 4 절 마이페이지 및 관심 동아리 등록 기능 4-8

제 5 절 알림 기능 5-9

제 6 절 애플리케이션 설정 기능 6-10

제 3 장 사용자 매뉴얼 6-12

제 1 절 회원 관리 1-12

제 2 절 화면 전환 bar 및 Progress bar 2-15

제 3 절 동아리 정보 조회 3-16

제 4 절 마이페이지 4-19

제 5 절 동아리 공지사항 알림 5-21

제 6 절 애플리케이션 설정 6-22

제 4 장 사용자 중심 애플리케이션 디자인 요소 6-25

제 1 절 UX 디자인 요소 1-25

제 2 절 UI 디자인 요소 2-30

제 1장 프로젝트 주제

동아리 활동을 통하여 자아를 탐색하고 진로를 개발하는 건국대학교 학생들을 대상으로, KU 동아리 정보 애플리케이션을 개발하였습니다.

기존에 동아리 정보 탐색에 활용되던 에브리타임, KUNG, 동아리 인스타그램, 학교 공식 홈페이지의 동아리 정보 등을 분석하고, 학생들이 불편함을 느낀 사항들을 선행적으로 분석하였습니다. 에브리타임의 복잡한 정보 탐색 경로, 동아리 상세 정보의 분산, 신청 기한 알림의 부재로 인한 지원 기간 놓침 등의 문제가 존재했습니다. 이러한 문제점을 보완하며, 다음과 같은 프로젝트 목표를 설정하였습니다.

A. KU 동아리 정보 애플리케이션은 건국대학교 내 모든 **중앙 동아리와 과 동아리의 모집 공고, 활동 내용, 일정** 등 관련 정보를 통합적으로 제공합니다.

B. 동아리 정보를 **카테고리를 통해 체계적으로 분류**함으로써, 학생들의 검색 및 탐색이 용이하게 합니다.

C. 애플리케이션을 통하여 동아리 활동에 대한 **학생들의 관심과 참여도**를 높입니다.

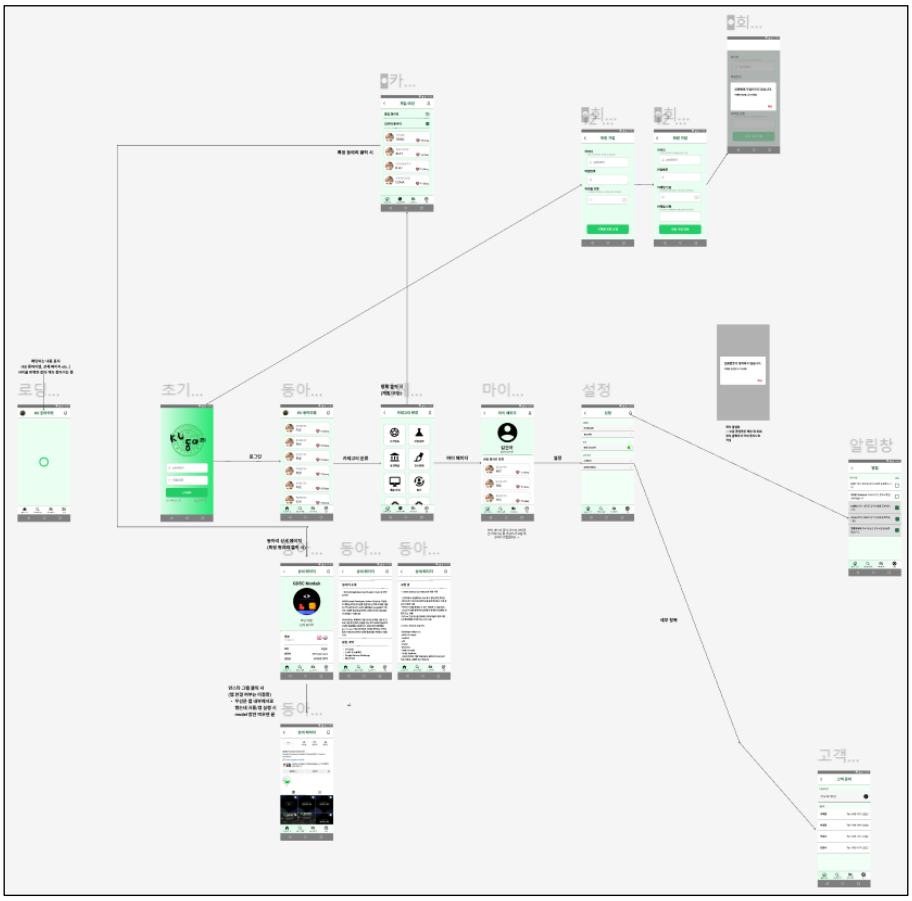


그림 1-1. KU 동아리 정보 앱 Wireframe

제 2장 상세 구현 방법

제 1절 프로젝트 아키텍쳐

DAO, Repository, ViewModel, UI 로 구성 요소를 모듈화함으로써, 유지 보수를 용이하게 하였습니다. 여러 개의 Screen 들은 NavHost 와 NavRoutes 에 등록하여 편리하게 navigation 합니다. NoSQL 기반의 Firebase Realtime Database 를 연동했으며, DAO 를 통해 데이터에 접근합니다. AWS EC2 서버는 API 를 통하여 사용자 인증을 위한 JWT 발급 및 검증을 수행합니다. 또한, 관리자용 기능을 따로 구현하지 않았기 때문에, Cron Job 을 통해 공지사항을 자동으로 Insert 하고 공지사항 등록 및 알림 기능을 테스트하였습니다.

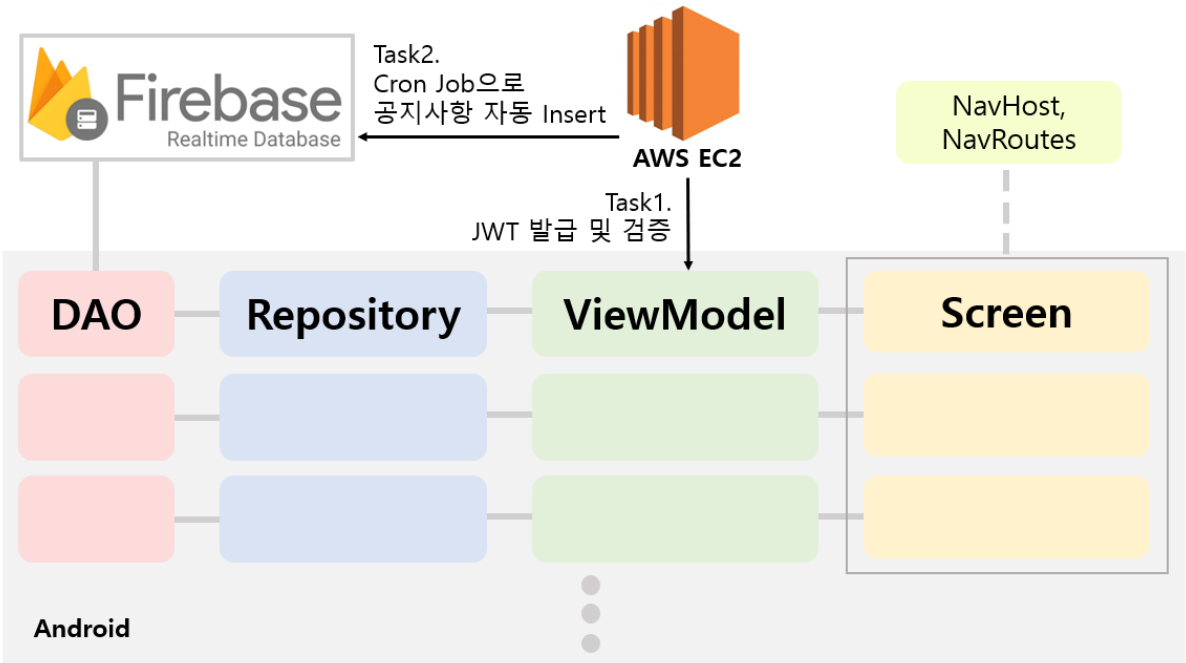


그림 2-1. 프로젝트 아키텍쳐

제 2절 사용자 등록 및 인증 기능

회원가입에서 학교 포탈에 등록된 이메일 주소를 아이디로 입력하도록 했습니다. 유저가 이메일 인증 요청을 하면, Gmail SMTP 를 통해 실제 학교 메일로 인증번호를 전송하기 때문에, 건국대학교 학생만 가입이 가능합니다. 정규식을 사용하여 비밀번호를 반드시 6 자리 이상 작성하도록 했고, 확인용으로 입력한 비밀번호와 일치해야 합니다. 학과는 DropDownMenu 를 통해 사용자가 편리하게 선택할 수 있도록 했습니다.

일반적인 로그인 로직을 활용하였고, 자동 로그인 기능을 구현하는 과정에서 Json Web Token(JWT)과 DataStore, Retrofit 을 사용했습니다. 일반적으로 JWT 는 인증에 필요한 정보들을

암호화시킨 JSON 토큰으로서, JWT 토큰을 HTTP 헤더에 실어 서버가 클라이언트를 식별하는 방식으로 사용됩니다. Header, Payload, Signature 로 구성되어 인증 신뢰성을 가지며, 유효한 토큰만 있으면 로그인이 가능하다는 장점이 있어 사용하게 되었습니다. 자동 로그인 시 유저의 정보를 직접적으로 저장하지 않고, Datastore 에 토큰을 저장하도록 했습니다.

AWS EC2 에서는 node.js 기반 서버들을 관리하는 pm2 모듈로 Express 서버를 무중단 실행했습니다. Express 서버는 /login, /verifyToken, /sendIdToken 엔드포인트를 가지며, 각각 커스텀 토큰 생성, 토큰 유효성 검증, 클라이언트로부터 토큰 수신의 역할을 합니다. ApiService 인터페이스를 통해 애플리케이션이 해당 서버에 POST 요청을 보낼 수 있습니다. 네트워크 통신을 위해 Retrofit 과 OkHttpClient 등을 함께 사용했습니다.

```
interface ApiService {
    @POST("/login")
    fun login(@Body request: LoginRequest): Call<TokenResponse>

    @POST("/verifyToken")
    fun verifyToken(@Body request: VerifyTokenRequest): Call<VerifyTokenResponse>

    @POST("/sendIdToken")
    fun sendIdToken(@Body request: IdTokenRequest): Call<ServerResponse>
}
```

그림 2-2. Android 앱을 JWT 생성 및 검증 서버와 연결하는 API

```
// Create JWT
app.post('/login', async (req, res) => {
    const { userId } = req.body;
    console.log('Received login request:', userId);

    try {
        const customToken = await admin.auth().createCustomToken(userId);
        console.log('Sent login response to client');
        res.json({ token: customToken });
    } catch (error) {
        console.log('Error creating custom token:', error);
        res.status(500).send('Internal server error');
    }
});
```

그림 2-3. AWS EC2 에서 실행 중인 Express 서버의 JWT 생성 로직

```
// Verify JWT
app.post('/verifyToken', async (req, res) => {
    const { token } = req.body;
    console.log(token);
    if (!token) {
        return res.status(401).json({ message: 'Please give the JWT' });
    }

    try {
        const decodedToken = await admin.auth().verifyIdToken(token);
        const uid = decodedToken.uid;
        console.log(uid);
        req.user = uid;
        res.json({ message: 'Valid JWT' });
        console.log('Sent verifyToken response to client');
    } catch (error) {
        console.log(error);
        return res.status(401).json({ message: 'Invalid JWT' });
    }
});

// Client send ID token to server.
app.post('/sendIdToken', async (req, res) => {
    const { idToken } = req.body;
    console.log('Received ID token from client:', idToken);
    res.json({ success: true, message: 'Received ID token successfully' });
});
```

그림 2-4. AWS EC2 에서 실행 중인 Express 서버의 JWT 수신 및 검증 로직

제 3절 동아리 소개 및 카테고리 기능

로그인을 한 후 사용자가 동아리 목록을 탐색하고 각 동아리의 세부 정보를 확인할 수 있도록 구현했습니다. 이를 위해 Firebase Realtime Database 를 사용하여 동아리 정보 데이터를 가져오며 ClubViewModel 로 데이터를 화면 간 주고 받아 동아리 목록과 개별 동아리의 상세 정보를 표시하였습니다. 또한 동아리 상세 화면에서 인스타그램 아이콘을 클릭하면 웹뷰를 통해 해당 동아리의 인스타그램 페이지로 이동할 수 있습니다.

Navigation Component 로 동아리 목록에서 상세 페이지로의 이동과 웹뷰로의 이동을 관리하였고 각 동아리에 대한 사용자의 좋아요 상태를 Firebase DB 에 저장하고 이를 통해 사용자가 좋아요를 누르거나 취소할 수 있습니다. 좋아요의 상태는 Firebase DB 로부터 UserLikedClub(어떤 사용자가 어디 동아리에 좋아요를 눌렀는지 정보가 들어있는 데이터)를 가져와 clubId 로 식별하여 해당 동아리의 총 좋아요 수 가 몇 개인지를 구했고 또한, 현재 로그인한 사용자가 동아리 좋아요를 전에 눌렀는지 확인하기 위해 userId 로 확인하는 작업을 하였습니다.

```
park99999
fun countLikes(clubId: Int, likes: List<UserLikedClub>): Int {
    return likes.count { it.clubId == clubId }
}
park99999
fun isUserLikedClub(clubId: Int, userId: String, likes: List<UserLikedClub>): Boolean {
    return likes.any { it.clubId == clubId && it.userId == userId }
}
```

그림 2-5. 동아리 좋아요 수 확인 로직

```
park99999
fun getAllLikedByClub(): Flow<List<UserLikedClub>> = callbackFlow { this: ProducerScope<List<UserLikedClub>>
    val table = firebaseDB.getReference( path: "KuclubDB/UserLikedClub")
    val listener = object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            val itemList = mutableListOf<UserLikedClub>()
            for (itemSnapshot in snapshot.children) {
                val item = itemSnapshot.getValue(UserLikedClub::class.java)
                item?.let { itemList.add(it) }
                Log.d( tag: "testtest1",item.toString())
            }
            trySend(itemList)
        }
        override fun onCancelled(error: DatabaseError) {
            close(error.toException())
        }
    }
    table.addListenerForSingleValueEvent(listener)
    awaitClose {
        table.removeEventListener(listener)
    }
}
```

그림 2-6. UseLikedClubDao 에서 Firebase DB 로 데이터 가져오는 로직

```
update = { webView ->
    navClubViewModel.selectedClub?.clubInsta?.let { webView.loadUrl(it) }
}
```

그림 2-7. webView 에서 선택한 동아리의 인스타 URL 을 연결하는 로직

제 4절 마이페이지 및 관심 동아리 등록 기능

관심 동아리 등록을 위해 Clubs 테이블과 User 테이블을 associative 테이블인 UserLikedClub 을 만들었습니다. 해당 테이블은 userId 와 clubId 를 합친 기본키를 사용하였습니다. 그리고 각 tuple 은 userId 와 clubId 를 가지고 있습니다. User 가 좋아요 아이콘을 클릭할 시 InsertLiked 기능을 통해 UserLikedClub 테이블에 값을 추가해주었습니다. 또한 좋아요 아이콘을 한 번 더 클릭할 시 deleteLiked 를 통해 UserLikedClub 테이블에서 해당하는 userId 와 clubId 를 가진 tuple 을 제거해 주었습니다. 이를 통해 userId 를 기준으로 UserLikedClub 을 찾으면, User 의 관심 동아리를 얻을 수 있고, clubId 를 기준으로 UserLikedClub 을 찾으면 좋아요 수를 구할 수 있게 만들었습니다.

이러한 데이터베이스 구조를 바탕으로 마이페이지를 구성하였습니다. 마이페이지에는 NavUserViewModel 에서 User 의 정보를 받아와 User 의 프로필을 보여주었으며, 그 아래에 관심 동아리를 LazyColumn 형태로 보여주었습니다. 관심 동아리가 하나도 없을 때와 관심 동아리가 있을 때를 구분하기 위해 userId 를 이용하여 관심 동아리가 존재하는지 확인하였습니다. 관심 동아리가 없을 때의 Column 과 관심 동아리가 있을 때의 Column 을 함수로 나누어 두가지 상황에 적절하게 화면이 나타나도록 만들었습니다. 관심 동아리가 있을 땐, 관심 동아리 리스트의 동아리 item 을 클릭하면 동아리 상세페이지로 이동할 수 있게 만들었습니다.

```
class UserLikedClubDao(private val firebaseDB:FirebaseDatabase) {
    Jongu Kim +1
    suspend fun insertLiked(userLikedClub: UserLikedClub){
        var table = firebaseDB.getReference( path: "KuclubDB/UserLikedClub")
        table.child( pathString: userLikedClub.clubId.toString()+userLikedClub.userId).setValue(userLikedClub)
    }
    Jongu Kim +1
    suspend fun deleteLiked(userLikedClub: UserLikedClub){
        val table = firebaseDB.getReference( path: "KuclubDB/UserLikedClub")
        table.child( pathString: userLikedClub.clubId.toString()+userLikedClub.userId).removeValue()
    }
```

그림 2-8. UserLikedClub 에 값 추가, 삭제 기능

```
suspend fun getAllLiked(userId:String, onResult: (List<String>) -> Unit) {
    val clubIds = mutableListOf<String>()
    val table = firebaseDB.getReference( path: "KuclubDB/UserLikedClub")
    val query = table.orderByChild( path: "userId").equalTo(userId)
    query.addListenerForSingleValueEvent(object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            for (userLikedClubSnapshot in dataSnapshot.children) {
                val userLikedClubId = userLikedClubSnapshot.getValue(UserLikedClub::class.java)
                userLikedClubId?.let { clubIds.add(it.clubId.toString()) }
                Log.d( tag: "Insert Club", msg: "리스트가 업데이트 되었습니다"+userLikedClubId.toString())
            }
            onResult(clubIds)
        }

        override fun onCancelled(databaseError: DatabaseError) {
            onResult(emptyList())
        }
    })
}
```

그림 2-9. dao 단계의 관심 동아리 찾는 기능

```
val clubLikedList = navClubViewModel.clubLiked.value
navClubViewModel.getAllLiked(userId)
    if(clubLikedList == null) {
        noLikedClub(userId, navClubViewModel)
    } else if(clubLikedList.size == 0) {
        noLikedClub(userId, navClubViewModel)
    }
    else {
        likedClubUI(userId, clubLikedList, navController, navClubViewModel)
    }
```

그림 2-10. 관심 동아리 있을 때, 없을 때에 따른 함수 설정

제 5절 알림 기능

동아리별 공지사항 등록시 사용자가 알림을 수신할 수 있도록, Notice_Notification 채널과 notificationManger 를 생성하였습니다. 6 절에서 자세히 설명할 Cron job 코드가 실행되어 주기적으로 새로운 공지사항이 추가되면, 이 이벤트를 감지하여 알림을 보냅니다. 'KuclubDB/Notice' 테이블에 ValueEventListener 를 연결하여, 데이터가 변경되면 dataSnapshot 을 얻도록 했습니다. lastProcessedNoticeId 보다 큰 noticeNum 을 가진 경우에만 새로운 공지사항으로 판단하였습니다. 새로운 공지사항의 clubName 정보를 sendNoticeNotification 메서드에 전달하여, '[동아리명] 새로운 공지사항이 있습니다.' 형태로 알림을 보내게 됩니다.

```
fun sendNoticeNotification(context: Context, clubName: String, msg: String) {
    val channelId = "Notice_Notification"
    val notificationId = 1

    val intent = Intent(context, MainActivity::class.java).apply {
        flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    }
    val pendingIntent = PendingIntent.getActivity(context, 0, intent,
                        PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE)

    val notification = NotificationCompat.Builder(context, channelId)
        .setContentTitle("공지사항 알림")
        .setContentText("[$clubName] $msg")
        .setPriority(NotificationCompat.PRIORITY_HIGH)
        .setContentIntent(pendingIntent)
        .setAutoCancel(true)
        .setSmallIcon(R.drawable.konkuk_logo)
        .build()

    val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE)
                              as NotificationManager
    notificationManager.notify(notificationId, notification)
}
```

그림 2-11. Pending intent 와 공지사항 채널을 통해 알림을 보내는 코드

```
fun monitorNewNotices(context: Context) {
    val table = firebaseDB.getReference("KuclubDB/Notice")

    table.addValueEventListener(object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            dataSnapshot.children.forEach { noticeSnapshot ->
                val notice = noticeSnapshot.getValue(Notice::class.java)
                notice?.let {
                    if (!initialDataLoaded) {
                        lastProcessedNoticeId = notice.noticeNum
                    } else {
                        if (notice.noticeNum > lastProcessedNoticeId) {

                            sendNoticeNotification(context, notice.clubName, "새로운 공지사항이 있습니다.")
                            lastProcessedNoticeId = notice.noticeNum
                        }
                    }
                }
            }
            initialDataLoaded = true
        }

        override fun onCancelled(databaseError: DatabaseError) {
            Log.e(TAG, "Failed to read data:", databaseError.toException())
        }
    })
}
```

그림 2-12. 공지사항 추가 이벤트를 모니터링하고 새로운 공지사항 정보로 알림을 발송하는 코드

제 6절 애플리케이션 설정 기능

앱을 설치한 후 처음 실행했을 때, 알림 권한 허용을 묻는 다이얼로그를 띄웠습니다. AndroidManifest.xml 파일에 POST_NOTIFICATION 에 대한 설정을 추가한 후 진행했으며, 이후 설정 페이지에서 원하는 대로 허용 여부를 변경할 수 있도록 했습니다. 또한 rememberPermissionState 를 통해 현재 허용 여부에 따라 ON 또는 OFF 텍스트가 표시됩니다.

```
val requestPermissionLauncher
    = rememberLauncherForActivityResult(contract = ActivityResultContracts.RequestPermission()) {
    isGranted ->
      if (isGranted)
          Toast.makeText(context, "알림 권한이 허용되었습니다.", Toast.LENGTH_SHORT).show()
      else
          if (!permissionState.status.shouldShowRationale)
              showDialog = true
}

LaunchedEffect(key1 = permissionState) {
    if(!permissionState.status.isGranted && !permissionState.status.shouldShowRationale)
        requestPermissionLauncher.launch(android.Manifest.permission.POST_NOTIFICATIONS)
}
```

그림 2-13. 알림 퍼미션 다이얼로그 구현 코드

공지사항 목록에서 상세 페이지로 navigate 할 때, 선택한 notice 객체의 noticeNum 을 함께 전달하여 선택한 공지사항의 내용이 나오도록 했습니다. 현재 기획한 프로젝트에는 관리자용

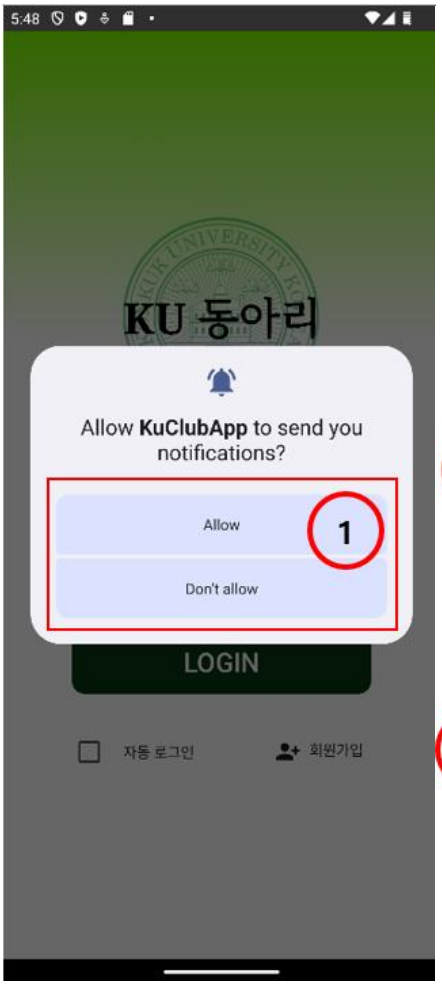
기능을 따로 구현하지 않았기 때문에, Firebase DB 에 직접 Notice 데이터를 몇 개 추가해서 사용했었습니다. 공지사항 알림 기능 테스트를 고려한 결과, 서버측에서 일정한 주기마다 자동으로 공지사항을 추가하는 기능을 구현하면 편리할 것으로 보였습니다. AWS EC2 의 Crontab 에서 일정한 주기로, Firebase DB 에 Notice 데이터를 추가하는 node.js 코드를 실행하게 했습니다.

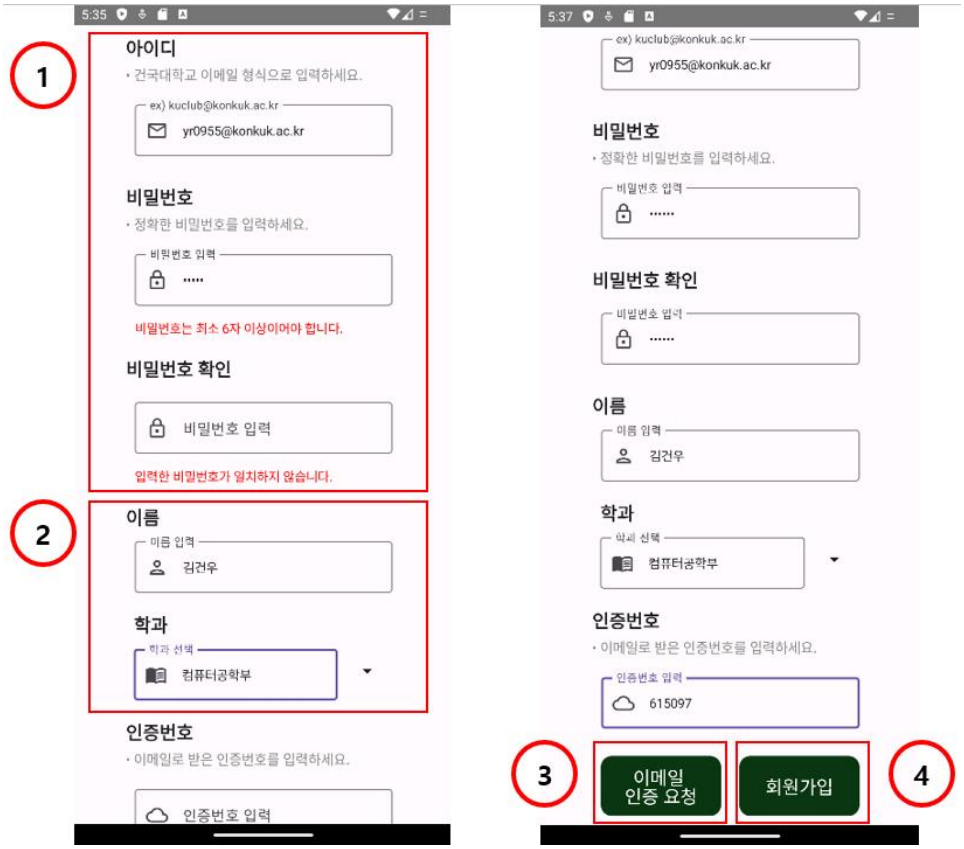
```
const db = admin.database();
const noticeRef = db.ref('/KuclubDB/Notice');

async function addNotice() {
  const snapshot = await noticeRef.once('value');
  const notices = snapshot.val();
  (중략)
  const newNotice = {
    noticeContent: "This is a scheduled notice content.",
    noticeDt: dateString,
    noticeNum: noticeNum,
    noticeTitle: "Scheduled Notice Title"
  };

  await noticeRef.child(noticeNum.toString()).set(newNotice);
}

addNotice().catch(console.error);
```

그림 2-14. Firebase DB 에 자동으로 공지사항 추가하는 서버측 Cron Job 코드

제 3장 사용자 매뉴얼

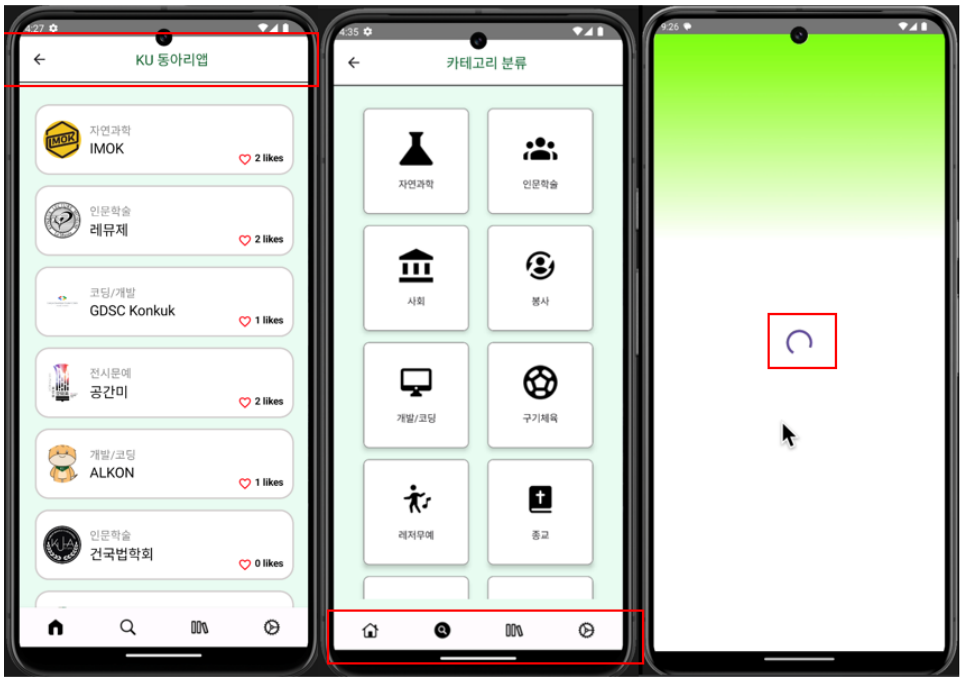
제 1절 회원 관리

대분류	회원 관리	중분류	로그인
화면			
설명	1. 앱 처음 설치시 알림 설정 - 사용자의 선호에 따라 허용 / 거부 선택 2. 계정이 없는 경우, 회원 가입 버튼 클릭 3. 이미 계정이 있는 경우, ID / PW 입력 4. 자동 로그인 체크 가능 5. 로그인 버튼 클릭		

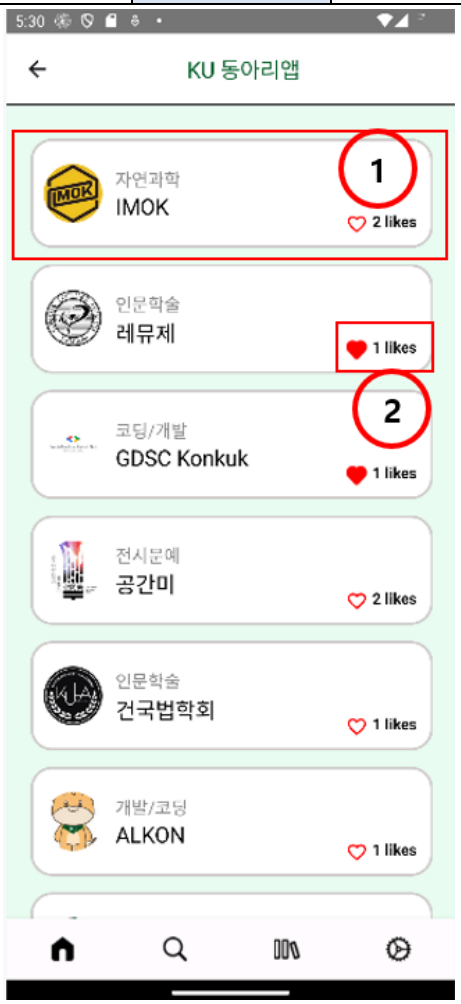
대분류	회원 관리	중분류	회원 가입
화면			
설명	1. 가입할 ID / PW 작성 - 실제로 건국대학교 포털에 가입된 이메일 주소를 작성해야 함. - 비밀번호 6 자리 이상, 두 비밀번호의 일치 조건을 만족해야 함. 2. 이름 및 학과 정보 작성 - 이름은 직접 작성하고, 학과는 드롭다운 메뉴에서 선택. 3. 이메일 인증 요청 버튼 클릭 후 인증번호 작성 - 건국대학교 이메일 계정에서 인증번호 확인 가능. 4. 회원가입 버튼 클릭		

대분류	회원 관리	중분류	회원 가입 완료
화면			
설명	1. 로그인 화면으로 이동 버튼 클릭 - 회원가입이 완료되었으니, 로그인 화면으로 이동.		

제 2절 화면 전환 bar 및 Progress bar

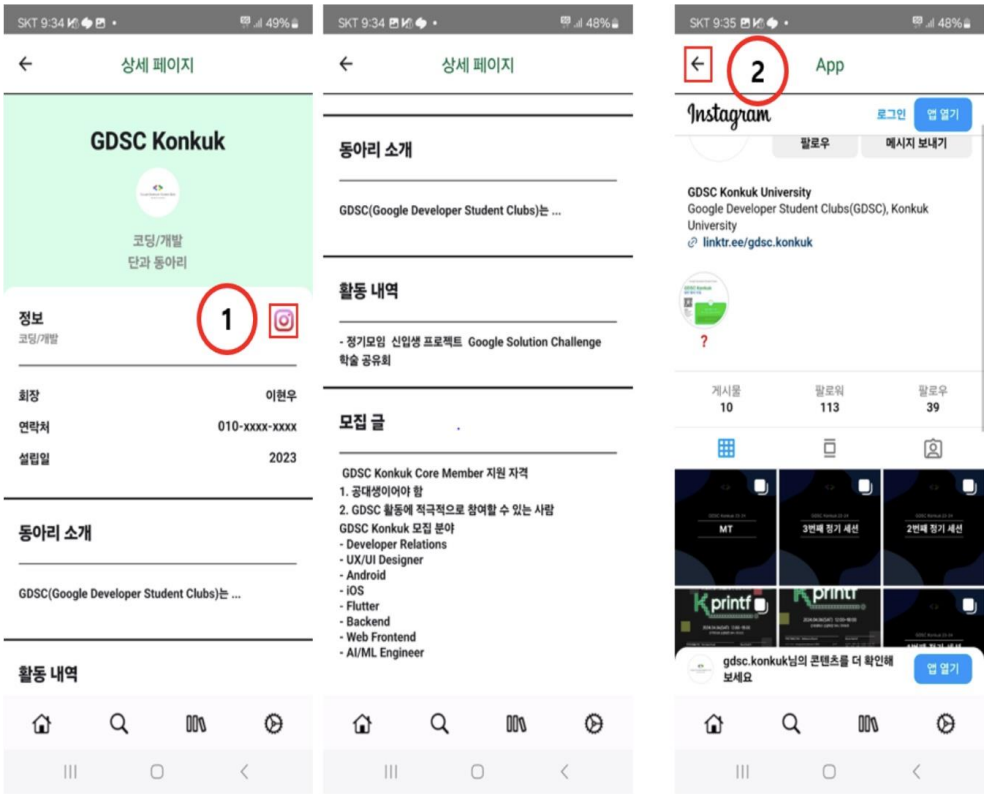
대분류	화면 전환 bar 및 Progress bar	중분류	Navigation, Top, Progress bar
화면			
설명	1. BottomNavigationBar - 화면 아래에 표시된 하단 네비게이션바의 경우 총 4 개의 페이지로 이동할 수 있는 네비게이션바 - 클릭한 페이지에 대해서는 아이콘의 모양이 두꺼워지도록 표현 2. TopBar - 현재 표시되는 페이지를 나타낼 수 있도록 중앙부에 페이지명을 표시 - 왼쪽에는 이전 페이지로 되돌아갈 수 있는 버튼이 존재 - 오른쪽에는 '알림'창으로 가는 버튼이 존재. 3. ProgressBar - 로그인 요청이 서버로 전송된 후, 서버로부터 사용자 정보를 비동기적으로 받아오는 동안, 사용자 인터페이스(UI)에서는 스피너(spinner) 애니메이션을 표시하여 사용자에게 로그인 처리가 진행 중임을 알림.		

제 3절 동아리 정보 조회

대분류	동아리 정보 조회	중분류	동아리 목록
화면			
설명	건국대학교 내에 있는 모든 동아리를 보여줍니다. 동아리 리스트에는 각각 동아리의 동아리 대분류(중앙 동아리, 단과 동아리), 동아리 이미지, 동아리 이름, 좋아요 개수 를 보여줍니다. 1. 선택한 동아리의 동아리 상세 정보 화면으로 넘어갑니다. 2. 사용자가 기존에 좋아요를 눌렀었다면 빈 하트 모양으로 표시되고 사용자가 하트를 누르면 꽉 찬 하트로 바뀌고 좋아요수가 1 올라갑니다. 반대로도 작동합니다.		

대분류	동아리 정보 조회	중분류	동아리 상세 정보

화면

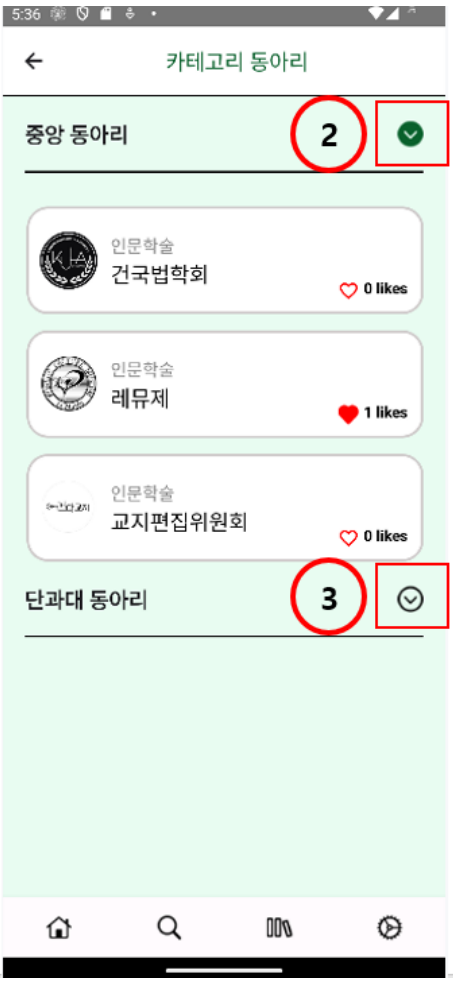


설명

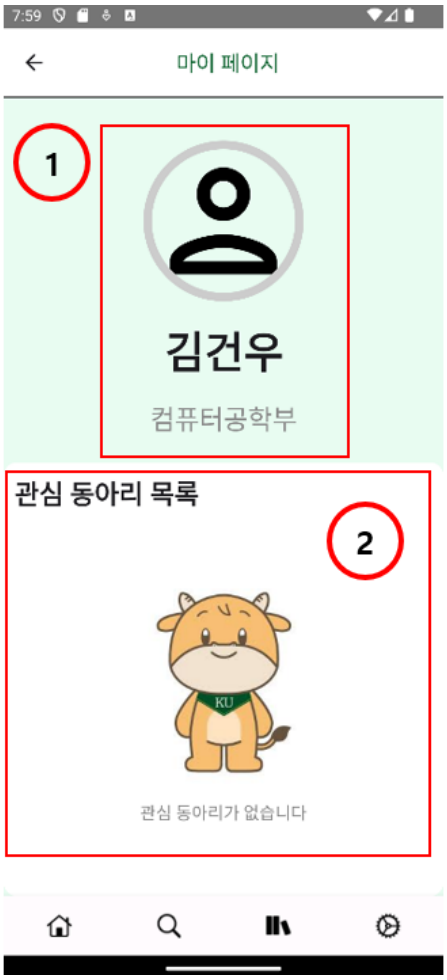
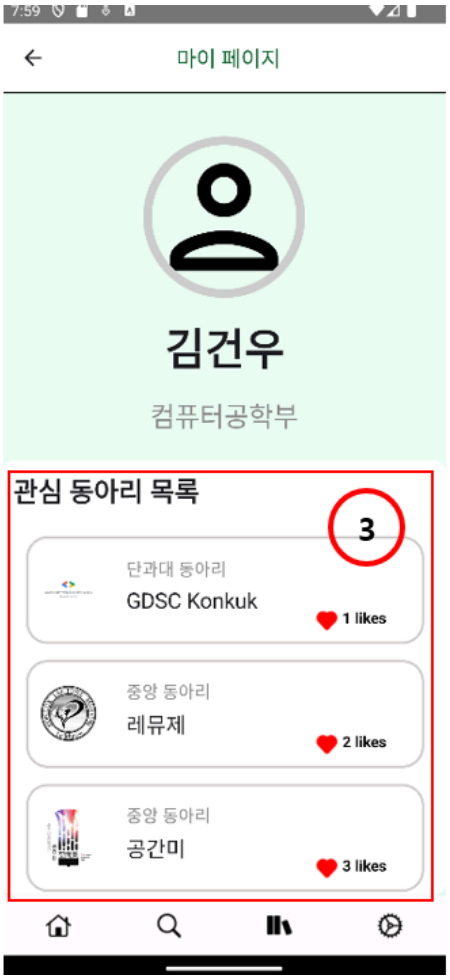
동아리 상세보기 페이지에서는 동아리 소개, 활동 내역, 모집글 등등 동아리 관련 정보를 열람할 수 있습니다.

1. 인스타 아이콘을 클릭하면 webview 로 해당 동아리의 인스타 주소로 이동합니다.

2. 상단의 뒤로가기 버튼을 누르면 이전 화면으로 이동합니다.

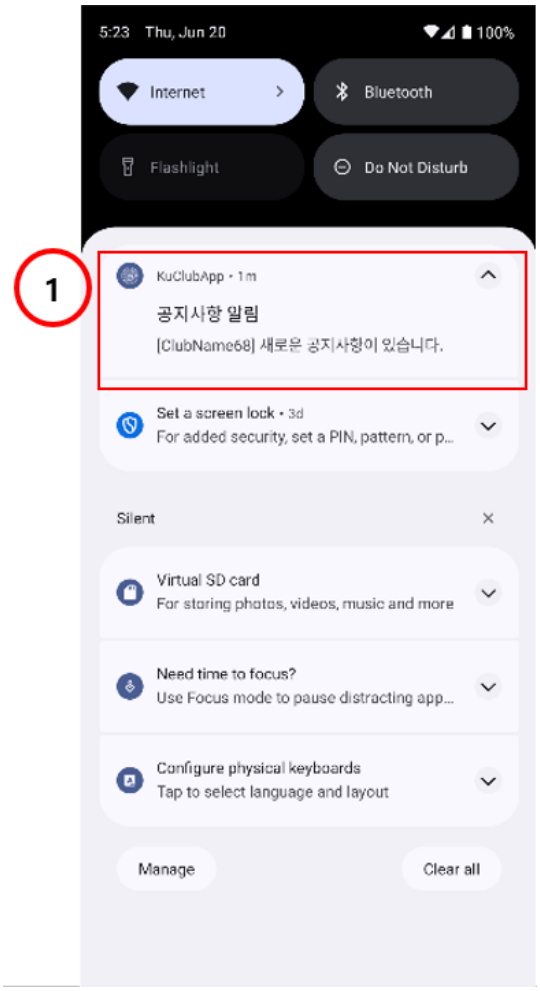
대분류	동아리 정보 조회	중분류	동아리 카테고리
화면			
설명	1. 선택한 특정 카테고리 동아리 리스트 화면으로 넘어갑니다. 2. 카테고리 동아리 리스트 화면은 중앙 동아리, 단과대 동아리 별로 나뉘어 있으며, 해당 버튼을 클릭 시 중앙 동아리로 분류된 동아리 리스트들을 보여주고 버튼의 색상을 바꿉니다.. 3. 해당 버튼을 클릭 시 단과대 동아리로 분류된 리스트들을 보여주고 버튼의 색상을 바꿉니다. 이 화면에서도 마찬가지로 리스트를 클릭하면 동아리 상세 정보 화면으로 넘어가고 좋아요를 누를 수 있습니다.		

제 4절 마이페이지

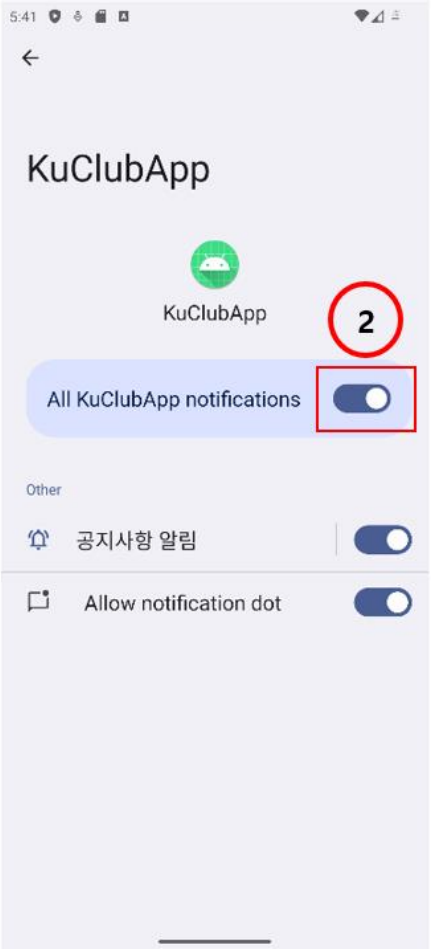
대분류	마이페이지	중분류	마이페이지
화면	 		
설명	1. 로그인한 현재 사용자의 프로필을 출력해 줍니다. 2. 관심 동아리가 없을 땐, 해당 화면을 출력합니다 . 3. 관심 동아리가 있을 땐, 관심 동아리 목록에 좋아요 누른 동아리를 LazyColumn 형태로 출력해 줍니다. 이 화면에서도 마찬가지로 리스트를 클릭하면 동아리 상세 정보 화면으로 넘어가고 좋아요를 누를 수 있습니다.		

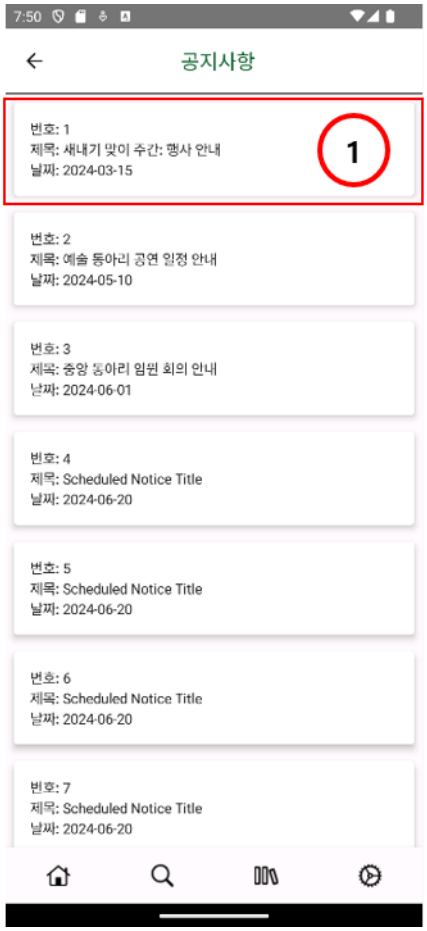
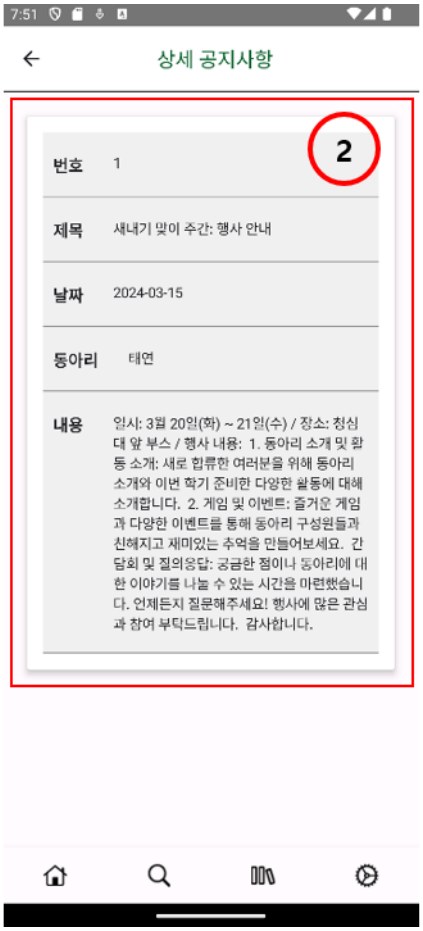
대분류	마이페이지	중분류	관심 동아리 상세 정보
화면			
설명	동아리 정보 조회 부분의 동아리 상세보기 페이지와 동일하게 동아리 소개, 활동 내역, 모집글 등등 동아리 관련 정보를 열람할 수 있습니다. 1. 인스타 아이콘을 클릭하면 webview 로 해당 동아리의 인스타 주소로 이동합니다. 2. 상단의 뒤로가기 버튼을 누르면 이전 화면으로 이동합니다.		

제 5절 동아리 공지사항 알림

대분류	동아리 공지사항 알림	중분류	동아리 공지사항 알림
화면			
설명	1. 동아리 공지사항 알림 수신 - 사용자가 공지사항 알림 수신을 허용한 경우 수신. - 대괄호 안에서 공지사항을 등록한 동아리명을 확인 가능.		

제 6절 애플리케이션 설정

대분류	애플리케이션 설정	중분류	설정 목록 및 알림 수신 동의
화면			
설명	1. 알림 수신 동의 여부 확인 및 설정 변경 - 현재 알림 수신 동의 여부를 ON / OFF 로 확인 가능 - 클릭하면 앱 설정 페이지로 이동 2. 알림 수신 동의 설정을 변경 3. 공지사항 클릭 후, 공지사항 페이지로 이동 4. 고객 문의 클릭 후, 고객 문의 페이지로 이동 5. 로그아웃 클릭 - 로그인 상태가 해제되며, 로그인 화면으로 이동		

대분류	애플리케이션 설정	중분류	공지사항
화면	 		
설명	1. 공지사항 목록 중 1 개 클릭 2. 선택한 공지사항의 상세 내용 확인 - 공지사항 번호, 제목, 등록 날짜, 내용을 확인 가능.		

대분류	애플리케이션 설정	중분류	고객 문의
화면			
설명	1. 고객 문의 가능한 공식 이메일 주소 안내 2. 개발자 연락처 안내		

제 4장 사용자 중심 애플리케이션 디자인 요소

제 1절 UX 디자인 요소

4.1.1 사용자 연구와 페르소나

4.1.1.1 사용자 연구

- 회의 기반 필요성 및 기대 : 모바일 프로그래밍 3 조의 회의를 통해 어플의 필요성과 기대를 파악하였습니다. 회의 결과, "동아리 공지사항을 알림으로 쉽게 확인하고, 다양한 동아리 정보를 한 번에 볼 수 있는 기능"이 필요하다는 결론에 도달했습니다.

4.1.1.2 User 페르소나

유저 페르소나는 아래와 같이 설정하였습니다.

- 이름: 김건국
- 나이: 20 대 초반 ~ 20 대 후반
- 직업: 건국대학교 재학생
- 목표: 동아리 활동을 효율적으로 관리하고, 모든 공지사항을 제때 확인하고 싶음.
- 행동 패턴: 하루에 2-3 시간을 스마트폰을 사용.
- 주요 니즈: 공지사항 알림, 건국대학교 교내 동아리 정보 확인.

4.1.2 사용자 여정 맵

사용자 여정 맵 단계는 다음과 같습니다.

1. 발견(Discovery):

   - 사용자가 애플리케이션에 대해 처음 알게 되는 단계.

   - 사용자의 행동: 소셜 미디어 광고를 통해 애플리케이션을 알게 됨.

2. 다운로드(Download):

- 애플리케이션을 다운로드하고 설치하는 단계.

- 사용자의 행동: 구글 플레이스토어에서 애플리케이션을 검색하고 다운로드.

3. 회원가입(Sign Up):

- 애플리케이션에 회원가입을 하는 단계.

- 사용자의 행동: 건국대학교 이메일 계정을 통해 회원가입.

4. 탐색(Exploration):

- 애플리케이션의 기능을 탐색하는 단계.

- 사용자의 행동: 메인 화면을 탐색하고, 동아리 리스트와 공지사항을 확인.

5. 사용(Usage):

- 애플리케이션의 주요 기능을 사용하는 단계.

- 사용자의 행동: 동아리 일정 확인, 공지사항 확인, 좋아요 버튼 클릭.

6. 유지(Retention):

- 애플리케이션을 지속적으로 사용하는 단계.

- 사용자의 행동: 매일 공지사항을 확인하고, 동아리 활동에 참여.

4.1.3 정보 구조

어플리케이션의 정보 구조는 상단 네비게이션 바와 각 화면의 구성 요소들로 이루어져 있습니다. 각 화면은 사용자가 특정한 기능이나 정보를 쉽게 찾고 접근할 수 있도록 설계되어 있으며, 이러한 구조는 사용자의 탐색 경험을 최적화하는 데 중점을 두고 있습니다. 사용자는 홈 화면, 검색, 카테고리, 설정 등 주요 기능에 빠르게 접근할 수 있으며, 각 화면 내에서 필요한 정보를 직관적으로 확인할 수 있습니다. 동아리 리스트 화면에서 특정 항목을 눌렀을 때의 동아리 상세 페이지는 사용자가 동아리에 대한 구체적인 정보를 확인할 수 있도록 돕습니다. 이 페이지는 동아리의 기본 정보, 활동 내역, 인스타그램 링크 등을 포함하여 사용자가 필요한 동아리 정보를 한눈에 볼 수 있도록 설계되었습니다.

4.1.4 화면별 상세 정보 구조

1. 동아리 리스트 화면

   - 동아리 정보 카드: 각 동아리의 이름과 카테고리, 그리고 '좋아요' 수를 표시.

2. 알림 화면

   - 상단 제목: '알림'

   - 알림 리스트: 각 동아리에서 등록한 새로운 공지사항을 목록으로 표시.

   - 읽음 상태 체크박스: 각 공지사항의 읽음 여부를 표시.

3. 마이 페이지 화면

   - 사용자 정보: 사용자 이름과 소속 정보 (박성근, 컴공).

   - 관심 동아리 목록: 사용자가 관심 있는 동아리 리스트.

4. 카테고리 분류 화면

   - 카테고리 타이틀: '카테고리 분류'

   - 카테고리 아이콘: 구기체육, 자연과학, 공연예술, 전시문예, 개발/코딩, 봉사 등의 아이콘과 함께 각 카테고리를 나타냄.

5. 설정 화면

   - 알림 설정: 알림 수신 동의 상태 표시 (현재 'OFF').

   - 고객 지원: 공지사항과 고객 문의 메뉴.

   - 사용자 설정: 로그아웃 옵션.

6. 동아리 상세 페이지 화면

   - 상단 제목: '상세 페이지'

   - 동아리 이름: 동아리의 이름을 표시.

   - 동아리 로고: 동아리의 로고 이미지.

   - 동아리 정보: 동아리 카테고리, 회장, 연락처, 설립일 등의 기본 정보.

   - 활동 내역: 동아리의 활동 내역을 목록으로 표시.

- 모집 글: 동아리 모집 글을 목록으로 표시.

- SNS 아이콘: 동아리의 소셜 미디어 링크 (인스타그램, 에브리타임)

4.1.5 와이어프레임 및 프로토타입

'KU 동아리 앱'의 와이어프레임과 프로토타입은 아래와 같습니다.

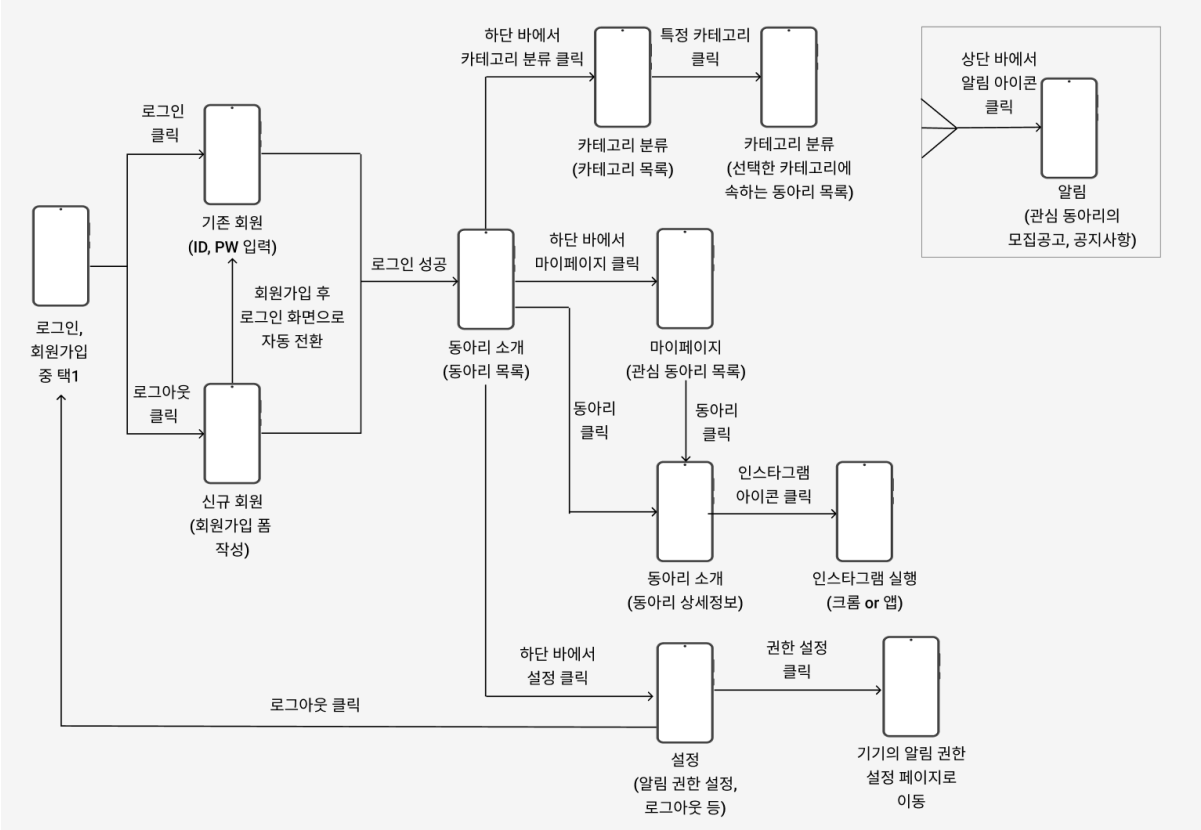


그림 4-1. KU 동아리 정보 앱 와이어프레임

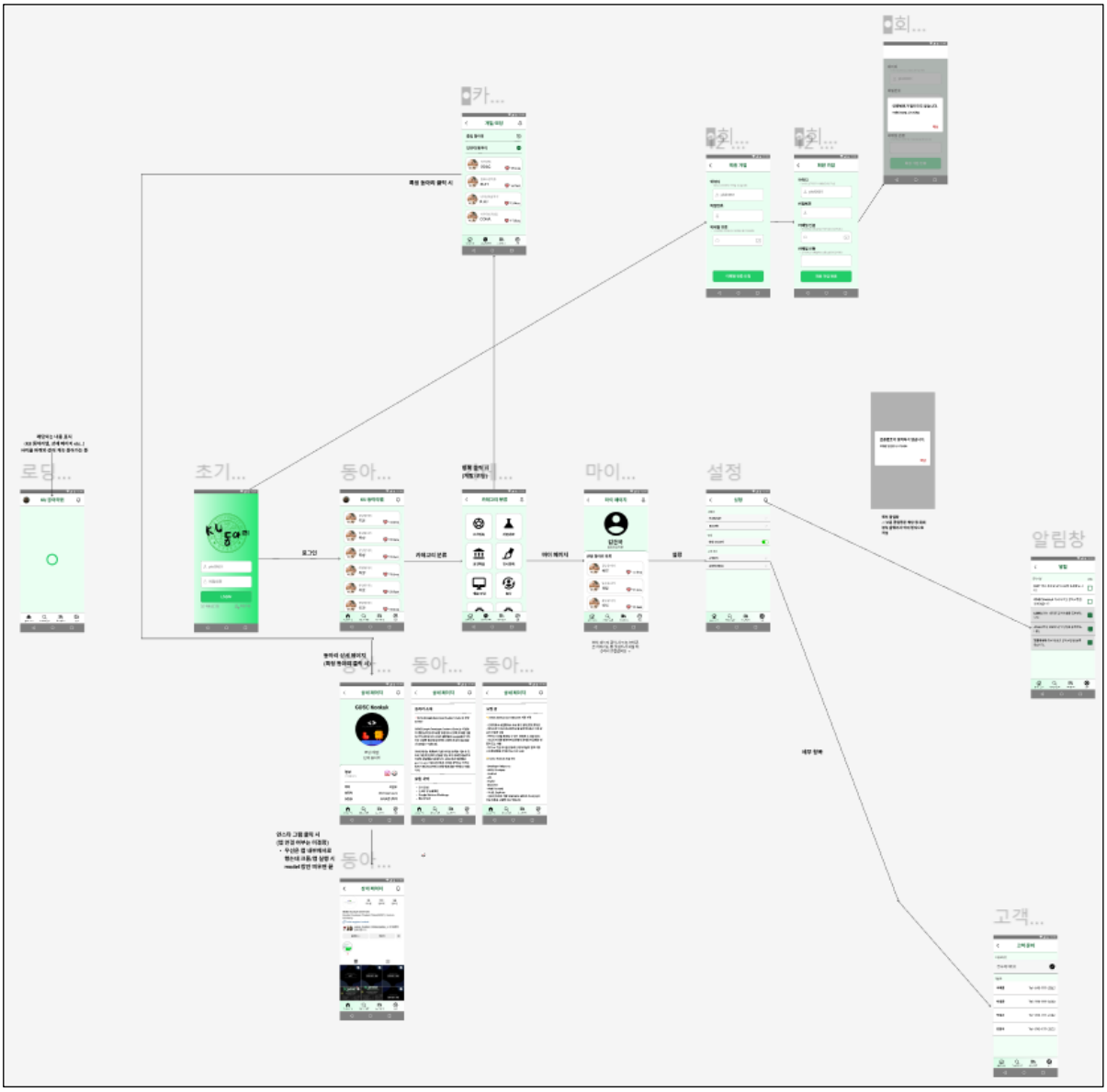


그림 4-2. KU 동아리 정보 앱 프로토타입

제 2절 UI 디자인 요소

4.2.1 비주얼 디자인

- 색상 팔레트: 건국대학교의 대표 색인 초록색 계열 색과 하얀색을 사용하였습니다.

  - background color: D9FDE8(opacity: 60%)

  - title color: 09612C

  - font-color: 000000, Color.gray

- 타이포그래피

  - Android 기본 글씨체 사용

- 아이콘 및 그래픽 요소

  - SF Symbols 에 내장된 아이콘 사용

4.2.2 레이아웃

- 레이아웃 그리드

  - 에브리타임 디자인을 참고하여 UI 요소들을 배치하였습니다.

  - TopApp 바를 상단에 배치하여 사용자에게 페이지에 대한 정보를 제공

  - 화면 중앙에 각 페이지 주요 기능을 보이는 UI 요소들 배치

  - BottomNavigation 바를 통해 페이지들 간의 이동을 할 수 있도록 함.

4.2.3 주요 화면 디자인

1. 동아리 리스트 화면

   목적: 사용자가 동아리 목록을 탐색하고 관심 있는 동아리를 선택할 수 있는 화면입니다.

   구성 요소:

   - 동아리 정보 카드: 각 동아리의 이름, 카테고리, 좋아요 수를 포함한 카드 형태의 목록.

- 상단 제목: 'KU 동아리앱'이라는 제목과 뒤로 가기 버튼이 포함된 상단 바.

- 좋아요 버튼: 각 동아리에 대한 좋아요 수를 표시하고 클릭할 수 있는 버튼.

2. 알림 화면

목적: 사용자에게 새로운 공지사항이나 업데이트를 알리는 화면입니다.

구성 요소:

- 공지사항 목록: 각 동아리에서 등록한 새로운 공지사항을 목록 형태로 표시.

- 읽음 상태 체크박스: 공지사항의 읽음 여부를 체크할 수 있는 상자.

- 상단 바: '알림'이라는 제목과 뒤로 가기 버튼이 포함된 상단 바.

3. 마이 페이지 화면

목적: 사용자의 프로필 정보와 관심 동아리 목록을 보여주는 화면입니다.

구성 요소:

- 프로필 섹션: 사용자의 이름과 소속 정보 (박성근, 컴공).

- 관심 동아리 목록: 사용자가 관심 있는 동아리 리스트를 표시. 관심 동아리가 없다면 '관심 동아리가 없습니다'로 표시.

- 상단 바: '마이 페이지'라는 제목과 뒤로 가기 버튼이 포함된 상단 바.

4. 카테고리 분류 화면

목적: 동아리와 활동을 카테고리별로 분류하여 사용자가 쉽게 탐색할 수 있도록 돕는 화면입니다.

구성 요소:

- 카테고리 타이틀: '카테고리 분류'라는 제목과 뒤로 가기 버튼이 포함된 상단 바.

- 카테고리 아이콘: 구기체육, 자연과학, 공연예술, 전시문예, 개발/코딩, 봉사 등의 아이콘과 함께 각 카테고리를 나타내는 버튼.

5. 설정 화면

목적: 어플리케이션의 다양한 설정을 조정할 수 있는 화면입니다.

구성 요소:

- 알림 설정: 알림 수신 동의 상태를 표시하고 조정할 수 있는 옵션.
- 고객 지원: 공지사항과 고객 문의 메뉴.
- 사용자 설정: 로그아웃 옵션.
- 상단 바: '설정'이라는 제목과 뒤로 가기 버튼이 포함된 상단 바.

6. 동아리 상세 페이지 화면

목적: 특정 동아리에 대한 상세 정보를 제공하는 화면입니다.

구성 요소:

- 동아리 이름: 동아리의 이름을 표시.
- 동아리 로고: 동아리의 로고 이미지.
- 동아리 정보: 동아리 카테고리, 회장, 연락처, 설립일 등의 기본 정보.
- 활동 내역: 동아리의 활동 내역을 목록으로 표시.
- 모집 글: 동아리 모집 글을 목록으로 표시.
- SNS 아이콘: 동아리의 소셜 미디어 링크 (인스타그램, 기타).
- 상단 바: '상세 페이지'라는 제목과 뒤로 가기 버튼이 포함된 상단 바.

4.2.4 인터랙션 디자인

어플리케이션 내에서 사용되는 버튼과 컨트롤 요소들에 대해 구체적으로 설명하겠습니다.

1. Bottom Navigation Bar

버튼 Hover 시

- 효과: 사용자가 Bottom Navigation Bar 의 버튼 위에 마우스를 올리면 버튼의 배경이 회색으로 변경됩니다.

- 목적: 시각적 피드백을 통해 사용자가 현재 어떤 버튼에 포커스를 두고 있는지 명확히 알 수 있게 합니다.

버튼 클릭 시:

- 효과: 사용자가 버튼을 클릭하면 해당 버튼이 더 진한 색상으로 표시됩니다.

- 목적: 클릭된 버튼이 현재 활성 상태임을 사용자가 쉽게 인식할 수 있도록 돕습니다.

2. Top App Bar

화살표 버튼 클릭 시:

- 효과: 사용자가 화살표 버튼을 클릭하면 이전에 위치하던 페이지로 이동합니다.

- 목적: 사용자가 쉽게 이전 페이지로 돌아갈 수 있게 하여 탐색을 용이하게 합니다.

종 버튼 클릭 시:

- 효과: 사용자가 종 버튼을 클릭하면 알림 페이지로 이동합니다.

- 목적: 사용자가 빠르게 알림 페이지에 접근할 수 있도록 하여 중요한 공지사항이나 업데이트를 확인할 수 있게 합니다.

3. 동아리 리스트 페이지

특정 카테고리 클릭 시:

- 효과: 사용자가 동아리 리스트 페이지에서 특정 카테고리를 클릭하면 해당 카테고리에 속한 동아리 목록이 표시됩니다.

- 목적: 사용자가 관심 있는 카테고리의 동아리를 쉽게 탐색할 수 있도록 하여, 원하는 정보를 빠르게 찾을 수 있게 합니다.

동아리 클릭 시:

- 효과: 사용자가 특정 동아리를 클릭하면 동아리 상세 페이지로 이동합니다.

- 목적: 사용자가 선택한 동아리에 대한 상세 정보를 확인할 수 있게 하여, 동아리의 활동 내용, 모집 글 등을 쉽게 접근할 수 있도록 합니다.

4..2.5 공통 인터랙션 디자인 요소

1. 입력필드

입력 필드

- 포커스 효과: 입력 필드를 클릭하면 테두리의 색상이 변경됩니다.

- 목적: 사용자가 현재 입력 중인 필드를 명확히 인지할 수 있도록 도와줍니다.