18. 파이어베이스

1. 파이어베이스 이해하기
2. 파이어베이스 연동하기
3. 인증 기능 이용하기
4. 파이어스토어 데이터베이스
5. 파이어베이스 스토리지
6. 파이어베이스 클라우드 메시징

1. 파이어베이스 이해하기

- **파이어베이스란?**
  - 안드로이드 앱에서 파이어베이스를 이용하면 서버리스 컴퓨팅을 구현할 수 있다.

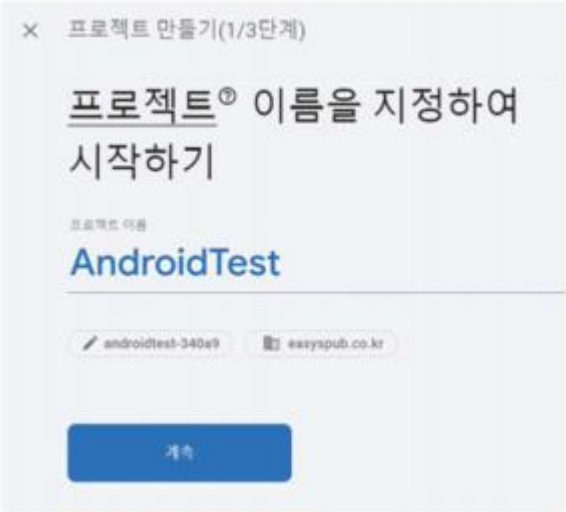
  - 파이어베이스는 모바일 앱과 웹 애플리케이션을 개발하는 데 필요한 여러 가지 기능을 제공
    - Authentication: 인증, 회원가입 및 로그인 처리
    - Cloud Message: 알림 전송
    - Firebase Database: 앱 데이터 저장 및 동기화
    - Realtime Database: 실시간 데이터 저장 및 동기화
    - Storage: 파일 저장소
    - Hosting: 웹 호스팅
    - Functions: 서버 관리 없이 모바일 백엔드 코드 실행
    - Machine Learning: 모바일 개발자용 머신러닝

1. 파이어베이스 이해하기

- **파이어베이스 요구 사항**
  - 안드로이드 앱에서 파이어베이스와 연동을 위한 환경
    - 구글 계정
    - 안드로이드 스튜디오
    - 그래들 4.1 이상
    - minSdk 16 이상

2. 파이어베이스 연동하기

- **파이어베이스 프로젝트 생성하기**
  - console.firebase.google.com에 접속
  - 프로젝트 만들기



2. 파이어베이스 연동하기

- **파이어베이스에 앱 등록하기**
  - 안드로이드 앱을 등록
  - <google-services.json 다운로드>를 클릭하여 파일을 내려받은 후 안드로이드 앱 모듈의 루트 디렉터리에 복사

2. 파이어베이스 연동하기

빌드 그래들에 파이어베이스 라이브러리 추가하기

- 프로젝트 수준의 build.gradle 파일

• 프로젝트 수준의 빌드 그래들

```
buildscript {
    (... 생략 ...)
    dependencies {
        classpath 'com.google.gms:google-services:4.3.10'
        (... 생략 ...)
    }
}
```

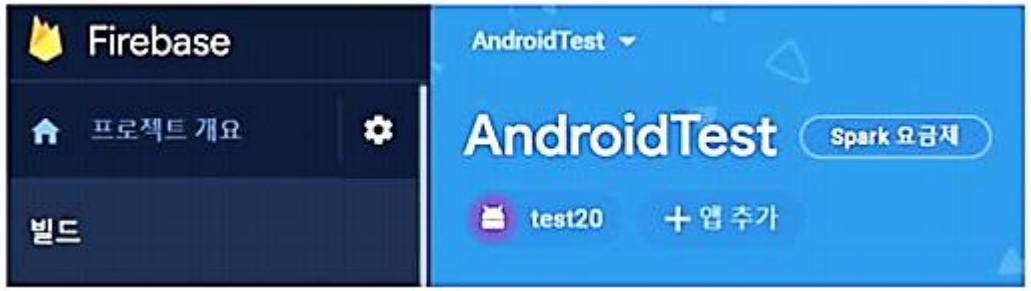
- 모듈 수준의 build.gradle 파일

• 모듈 수준의 빌드 그래들

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'com.google.gms.google-services'
}
(... 생략 ...)
dependencies {
    (... 생략 ...)
    implementation platform('com.google.firebase:firebase-bom:29.0.0')
}
```

2. 파이어베이스 연동하기

- 파이어베이스 콘솔에서 앱 등록 완료



3. 인증 기능 이용하기

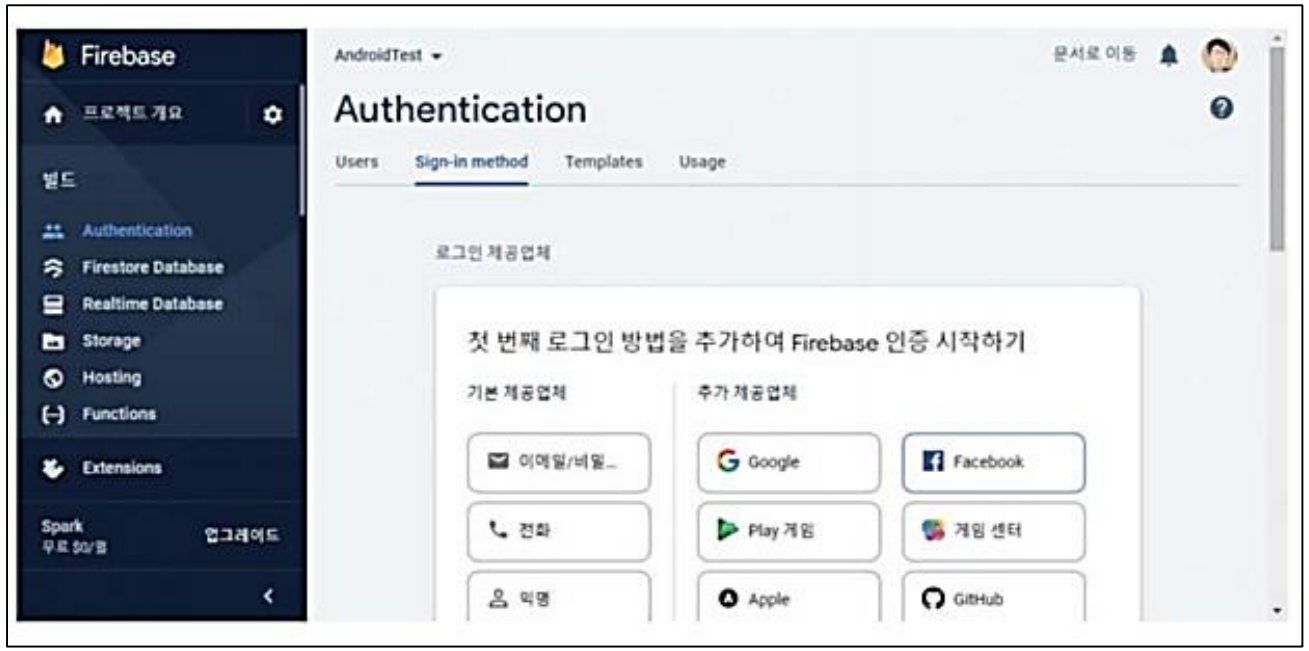
▪ 이메일/비밀번호 인증

- 이메일/비밀번호 인증은 사용자의 회원가입 정보인 이메일/비밀번호를 파이어베이스에 저장하고 이를 바탕으로 로 그인을 처리하는 방식

- 로그인할 때는 등록된 이메일 서버와 연동하지 않고 파이어베이스에서 처리

3. 인증 기능 이용하기

- 이메일/비밀번호 인증 사용 설정하기



• 파이어베이스 인증 라이브러리 등록

```
implementation 'com.google.firebase:firebase-auth-ktx:21.0.1'
```

3. 인증 기능 이용하기

- 파이어베이스 인증 객체 얻기
  - FirebaseAuth 객체는 Firebase.auth로 얻음.

• 파이어베이스 인증 객체 얻기

```
lateinit var auth: FirebaseAuth
(... 생략 ...)
auth = Firebase.auth
```

- 회원가입하기
  - FirebaseAuth 객체의 createUserWithEmailAndPassword() 함수로 파이어베이스에 이메일/비밀번호를 등록

• 회원가입하기

```
auth.createUserWithEmailAndPassword(email, password)
    .addOnCompleteListener(this) { task ->
    (... 생략 ...)
}
```

3. 인증 기능 이용하기

- 콜백 함수의 매개변수인 Task<AuthResult> 객체로 회원가입 성공·실패를 판단

• 회원가입 성공·실패 판단

```
auth.createUserWithEmailAndPassword(email, password)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            (... 생략 ...)
        } else {
            Log.w("kkang", "createUserWithEmail:failure", task.exception)
        }
    }
```

3. 인증 기능 이용하기

- 이메일/비밀번호 등록에 성공했을 때 FirebaseUser 클래스의 sendEmailVerification() 함수로 인증 메일 전송

• 인증 메일 발송하기(등록 성공했을 때)

```
auth.createUserWithEmailAndPassword(email, password)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            auth.currentUser?.sendEmailVerification()
                ?.addOnCompleteListener { sendTask ->
                    if (sendTask.isSuccessful) {
                        // ...
                    } else {
                        // ...
                    }
                }
        } else {
            Log.w("kkang", "createUserWithEmail:failure", task.exception)
        }
    }
```

3. 인증 기능 이용하기

- 로그인하기
  - 로그인을 처리할 때는 signInWithEmailAnd Password() 함수를 이용

• 로그인 처리

```
auth.signInWithEmailAndPassword(email, password)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            // 로그인 성공했을 때 처리
        } else {
            // 로그인 실패했을 때 처리
        }
    }
```

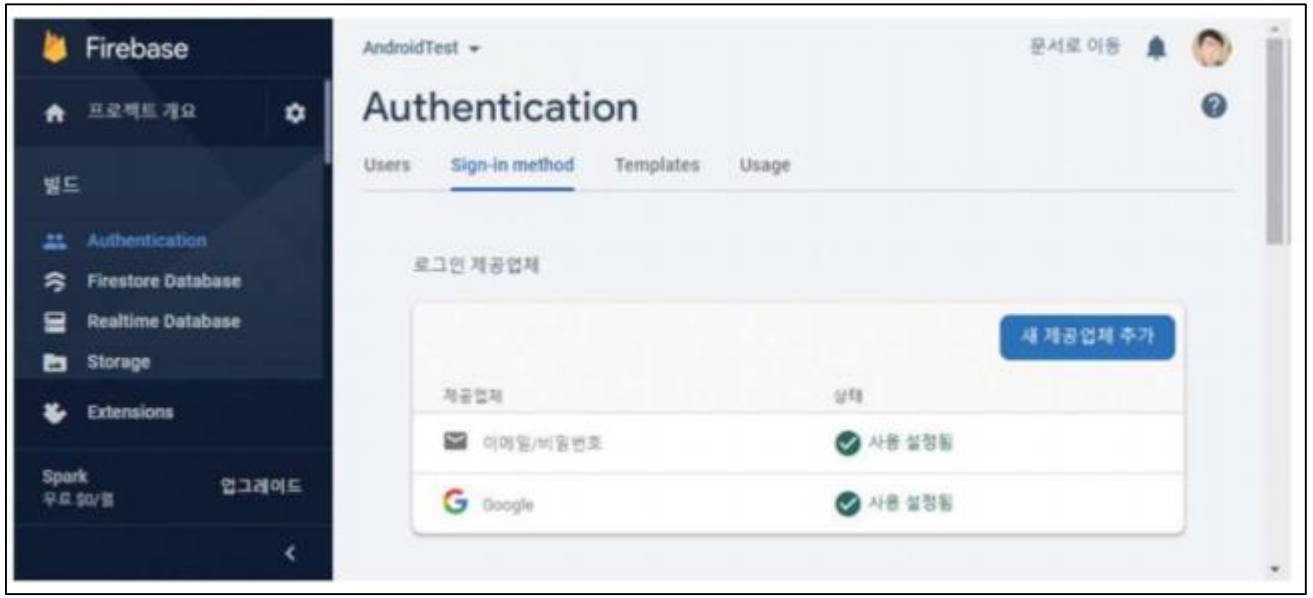
  - 사용자 정보 가져오

• 사용자 정보 가져오기

```
currentUser?.let {
    val isEmailVerified = currentUser.isEmailVerified
    val email = currentUser.email
    val uid = currentUser.uid
}
```

3. 인증 기능 이용하기

- **구글 인증**
  - 구글 인증 사용 설정하기



• 플레이 서비스 인증 라이브러리 등록

```
implementation 'com.google.android.gms:play-services-auth:19.2.0'
```

3. 인증 기능 이용하기

- 구글 인증 처리하기
  - 안드로이드 시스템에 설치된 구글의 인증 앱과 연동하여 처리

• 구글 인증 앱 실행

```
val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
    .requestIdToken(getString(R.string.default_web_client_id))
    .requestEmail()
    .build()
val signInIntent = GoogleSignIn.getClient(this, gso).signInIntent
requestLauncher.launch(signInIntent)
```

인텐트 객체 생성

인텐트 시작

3. 인증 기능 이용하기

- 구글 인증 처리

```
val requestLauncher = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult())
{
    val task = GoogleSignIn.getSignedInAccountFromIntent(it.data)
    try {
        val account = task.getResult(ApiException::class.java)!!
        val credential = GoogleAuthProvider.getCredential(account.idToken, null)
        MyApplication.auth.signInWithCredential(credential)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    // 구글 로그인 성공
                } else
                    // 구글 로그인 실패
                }
            }
    } catch (e: ApiException) {
        // 예외 처리
    }
}
```

실습1 : 회원가입과 로그인 기능 만들기

- **1단계. 이메일/비밀번호 인증 설정하기**
  - 파이어베이스 콘솔에서 이메일/비밀번호인증을 사용하겠다고 설정
- **2단계. 구글 인증 설정하기**
  - 구글 인증을 활성화
- **3단계. 빌드 그래들 설정하기**
  - 파이어베이스 이메일/비밀번호와 구글 인증을 사용하는 라이브러리를 등록
- **4단계. MyApplication 클래스 작성과 등록**
  - MyApplication 클래스를 작성하고 매니페스트에 등록
- **5단계. 인증 처리 액티비티 추가하기**
  - AuthActivity라는 이름으로 새로운 액티비티를 만듬.

실습1 : 회원가입과 로그인 기능 만들기

- **6단계. 실습 파일 복사하기**
  - layout과 menu 두 디렉터리를 현재 모듈의 res 디렉터리 아래에 복사
  - 코틀린 소스가 있는 디렉터리의 AuthActivity.kt와 MainActivity.kt 파일을 앱의 소스 영역에 복사
- **7단계. 인증 액티비티 작성하기**
  - AuthActivity.kt 파일 작성

4. 파이어스토어 데이터베이스

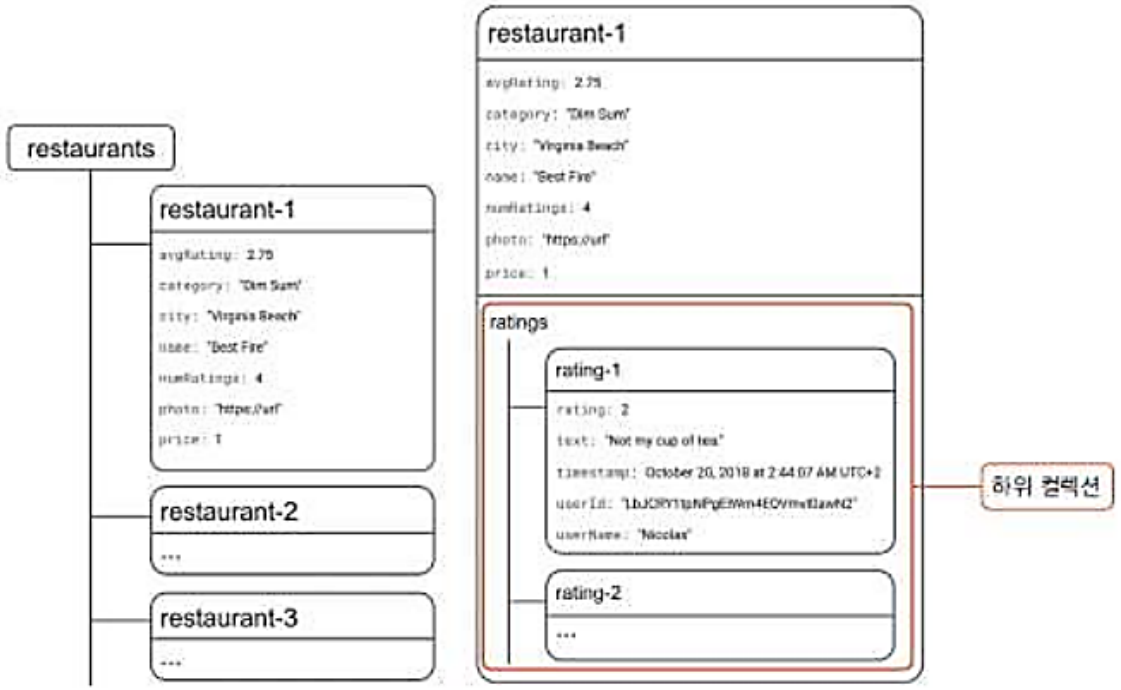
- 파이어스토어 사용 설정

• 파이어스토어 사용 등록

```
dependencies {
    (... 생략 ...)
    implementation 'com.google.firebase:firebase-firestore-ktx:21.2.1'
}
```

파이어스토어 데이터 모델

- 파이어스토어는 NoSQL 데이터베이스
- 컬렉션으로 정리되는 문서에 데이터가 저장
- 문서에는 키-값 쌍의 데이터가 저장되며 모든 문서는 컬렉션에 저장



4. 파이어스토어 데이터베이스

파이어스토어 보안 규칙

- 보안 규칙은 콘솔의 [규칙] 탭에서 설정
- match와 allow 구문을 조합해서 작성
- match 구문으로 데이터베이스 문서를 식별하고 allow 구문으로 접근 권한을 작성

• 모든 문서의 읽기/쓰기 거부

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {          // 모든 문서
      allow read, write: if false;  // 읽기/쓰기 거부
    }
  }
}
```

• 모든 문서의 읽기/쓰기 허용

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if true;   // 읽기/쓰기 허용
    }
  }
}
```

4. 파이어스토어 데이터베이스

• 인증된 사용자에게만 모든 문서의 읽기/쓰기 허용

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth.uid != null;
    }
  }
}
```

인증된 사용자에게만 읽기/쓰기 허용

• 자신의 데이터만 읽기/쓰기 허용

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, update, delete: if request.auth.uid == userId;
      allow create:  if request.auth.uid != null;
    }
  }
}
```

자신의 데이터만 읽기, 수정, 삭제 허용

인증된 사용자에게만 문서 생성 허용

• 문서에 저장된 데이터 활용

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /cities/{city} {
      allow read: if resource.data.visibility == 'public';
    }
  }
}
```

문서의 visibility값이 public일 때만 읽기 허용

• 전달받은 데이터 활용

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {{
      allow update: if request.resource.data.population > 0
                    && request.resource.data.name == resource.data.name;
    }
  }
}
```

전달받은 데이터가 0 이상일 때만 population 데이터 수정 허용.
단, name 데이터는 수정할 수 없음

4. 파이어스토어 데이터베이스

데이터 저장하기

- FirebaseFirestore 객체로 컬렉션을 선택하고 문서를 추가하거나 가져오는 작업을 함

• 파이어스토어 객체 얻기

```
var db: FirebaseFirestore = FirebaseFirestore.getInstance()
```

- add() 함수로 데이터 저장하기
  - 데이터를 저장하려면 먼저 컬렉션을 선택하고 문서 작업을 하는 CollectionReference 객체를 얻어야 함
  - CollectionRe ference 객체의 add(), set(), get() 등의 함수로 문서 작업

4. 파이어스토어 데이터베이스

- add() 함수로 데이터 저장

```
val user = mapOf(
    "name" to "kkang",
    "email" to "a@a.com",
    "avg" to 10
)
val colRef: CollectionReference = db.collection("users")
val docRef: Task<DocumentReference> = colRef.add(user)
docRef.addOnSuccessListener { documentReference ->
    Log.d("kkang", "DocumentSnapshot added with ID: ${documentReference.id}")
}
docRef.addOnFailureListener { e ->
    Log.w("kkang", "Error adding document", e)
}
```

성공 콜백

실패 콜백

4. 파이어스토어 데이터베이스

- 객체 저장하기
  - Users 컬렉션에 User 클래스의 객체를 저장

• 객체 저장하기

```
class User(val name: String, val email: String, val avg: Int,
    @JvmField val isAdmin: Boolean, val isTop: Boolean)
val user = User("kim", "kim@a.com", 20, true, true)
db.collection("users")
    .add(user)
```

- set() 함수로 데이터 저장하기

• set() 함수로 데이터 저장

```
val user = User("lee", "lee@a.com", 30)
db.collection("users")
    .document("ID01")
    .set(user)
```

▶ 실행 결과

4. 파이어스토어 데이터베이스

▪ 데이터 업데이트와 삭제

- update() 함수로 데이터 업데이트하기

• 특정 필드값만 업데이트

```
db.collection("users")
    .document("ID01")
    .update("email", "lee@b.com")
```

- delete() 함수로 데이터 삭제하기

• 여러 필드값 업데이트

```
db.collection("users")
    .document("ID01")
    .update(mapOf(
        "name" to "lee01",
        "email" to "lee@c.com"
    ))
```

• 문서 전체 삭제

```
db.collection("users")
    .document("ID01")
    .delete()
```

4. 파이어스토어 데이터베이스

데이터 불러오기

- get() 함수로 컬렉션의 전체 문서 가져오기

• 전체 문서 가져오기

```
db.collection("users")
    .get()
    .addOnSuccessListener { result ->
        for (document in result) {
            Log.d("kkang", "${document.id} => ${document.data}")
        }
    }
    .addOnFailureListener { exception ->
        Log.d("kkang", "Error getting documents: ", exception)
    }
```

4. 파이어스토어 데이터베이스

- get() 함수로 단일 문서 가져오기

• 단일 문서 가져오기

```
val docRef = db.collection("users").document("ID01")
docRef.get()
    .addOnSuccessListener { document ->
        if (document != null) {
            Log.d("kkang", "DocumentSnapshot data: ${document.data}")
        } else {
            Log.d("kkang", "No such document")
        }
    }
    .addOnFailureListener { exception ->
        Log.d("kkang", "get failed with ", exception)
    }
```

• 문서를 객체에 담기

```
class User{
    var name: String? = null
    var email: String? = null
    var avg: Int = 0
}
val docRef = db.collection("users").document("ID01")
docRef.get().addOnSuccessListener { documentSnapshot ->
    val selectUser = documentSnapshot.toObject(User::class.java)
    Log.d("kkang", "name: ${selectUser?.name}")
}
```

4. 파이어스토어 데이터베이스

- **whereXXX() 함수로 조건 설정**

• 조건에 맞는 문서 가져오기

```
db.collection("users")
    .whereEqualTo("name", "lee")
    .get()
    .addOnSuccessListener { documents ->
        for (document in documents) {
            Log.d("kkang", "${document.id} => ${document.data}")
        }
    }
    .addOnFailureListener { exception ->
        Log.w("kkang", "Error getting documents: ", exception)
    }
```

5. 파이어베이스 스토리지

- 스토리지 사용 설정

• 스토리지 라이브러리 등록

```
dependencies {
    implementation 'com.google.firebase:firebase-storage-ktx'
}
```

5. 파이어베이스 스토리지

파일 올리기

- 스토리지를 이용할 때는 먼저 FirebaseStorage 객체를 얻는다.
- 파일을 가리키는 StorageReference를 만든다.
- 스토리지에 파일을 올릴 때는 StorageReference의 putBytes(), putFile(), putStream() 함수를 이용

• 스토리지 객체 얻기

```
val storage: FirebaseStorage=Firebase.storage
```

• 스토리지 참조 만들기

```
val storageRef: StorageReference = storage.reference
val imgRef: StorageReference = storageRef.child("images/a.jpg")
```

5. 파이어베이스 스토리지

- putBytes() 함수로 바이트 값 저장하기
  - Bitmap 객체를 만들고 여기에 Canvas 객체로 뷰의 내용을 그린 후 반환

• 화면을 비트맵 객체에 그리기

```
fun getBitmapFromView(view: View): Bitmap? {
    var bitmap = Bitmap.createBitmap(view.width, view.height, Bitmap.Config.ARGB_8888)
    var canvas = Canvas(bitmap)
    view.draw(canvas)
    return bitmap
}
```

- 이미지 내용이 바이트값으로 저장

• 이미지를 바이트값으로 읽기

```
val bitmap = getBitmapFromView(binding.addPicImageView)
val baos = ByteArrayOutputStream()
bitmap?.compress(Bitmap.CompressFormat.JPEG, 100, baos)
val data = baos.toByteArray()
```

5. 파이어베이스 스토리지

- putBytes() 함수로 스토리지에 저장

• 바이트값을 스토리지에 저장하기

```
var uploadTask = imgRef.putBytes(data)
uploadTask.addOnFailureListener {
    Log.d("kang", "upload fail.....")
}.addOnCompleteListener { taskSnapshot ->
    Log.d("kkang", "upload success ...")
}
```

- UploadTask 객체로 파일의 URL을 가져올 수도 있음.

• 파일의 URL 얻기

```
val urlTask = uploadTask.continueWithTask { task ->
    if (!task.isSuccessful) {
        task.exception?.let {
            throw it
        }
    }
    imgRef.downloadUrl
}.addOnCompleteListener { task ->
    if (task.isSuccessful) {
        val downloadUri = task.result   // 스토리지에 저장된 파일의 URL
        Log.d("kkang", "upload url ...$downloadUri")
    } else {
        (... 생략 ...)
    }
}
```

5. 파이어베이스 스토리지

- putStream() 함수로 저장하기
  - 파일에서 데이터를 읽는 FileInputStream 객체를 putStream() 함수의 매개변수로 지정하여 업로드

• 파일 스트림으로 업로드

```
val stream = FileInputStream(File(filePath))
val uploadTask = imgRef.putStream(stream)
```

- putFile() 함수로 저장하기

• 파일 경로로 업로드

```
var file = Uri.fromFile(File(filePath))
val uploadTask = imgRef.putFile(file)
```

- 업로드 파일 삭제

• 파일 삭제

```
val imgRef: StorageReference = storageRef.child("images/a.jpg")
imgRef.delete()
    .addOnFailureListener {
        Log.d("kkang," failure.............")
    }
    .addOnCompleteListener {
        Log.d("kkang," success.............")
    }
```

5. 파이어베이스 스토리지

- **파일 내려받기**
  - getBytes() 함수로 바이트값 가져오기

• 내려받은 파일의 바이트값 가져오기

```
val  storageRef: StorageReference = storage.reference
val imgRef: StorageReference = storageRef.child("images/a.jpg")
val ONE_MEGABYTE: Long = 1024 * 1024 ← 내려받는 최대 바이트 수 지정
imgRef.getBytes(ONE_MEGABYTE).addOnSuccessListener {
    val bitmap = BitmapFactory.decodeByteArray(it, 0, it.size)
    binding.downloadImageView.setImageBitmap(bitmap)
}.addOnFailureListener {
    Log.d("kkang"," failure.............")
}
```

5. 파이어베이스 스토리지

- getFile() 함수로 가져오기

• 로컬 저장소에 파일 내려받기

```
val imgRef: StorageReference = storageRef.child("images/a.jpg")
val localFile = File.createTempFile("images", "jpg")
imgRef.getFile(localFile).addOnSuccessListener {
    val bitmap = BitmapFactory.decodeFile(localFile.absolutePath)
    binding.downloadImageView.setImageBitmap(bitmap)
}.addOnFailureListener {
    // 오류 처리
}
```

- downloadUrl() 함수로 URL 얻기

• 스토리지 파일의 URL 얻기

```
val imgRef: StorageReference = storageRef.child("images/a.jpg")
imgRef.downloadUrl.addOnSuccessListener {
    Log.d("kkang", "download uri : $it")
}.addOnFailureListener {
    // 오류 처리
}
```

5. 파이어베이스 스토리지

- firebase-ui-storage 라이브러리 이용

• firebase-ui-storage 라이브러리 등록

```
plugins {
    (... 생략 ...)
    id 'kotlin-kapt'
}
(... 생략 ...)
dependencies {
    (... 생략 ...)
    implementation 'com.github.bumptech.glide:glide:4.11.0'
    implementation 'com.google.firebase:firebase-storage-ktx'
    implementation 'com.firebaseui:firebase-ui-storage:7.1.0'
    kapt 'com.github.bumptech.glide:compiler:4.11.0'
}
```

5. 파이어베이스 스토리지

- firebase-ui-storage 라이브러리 이용
  - AppGlideModule을 상속받는 클래스를 작성
  - 클래스에 @GlideModule이라는 애너테이션을 적용

• 글라이드 모듈 클래스 작성

```
@GlideModule
class MyAppGlideModule : AppGlideModule() {
    override fun registerComponents(context: Context, glide: Glide, registry: Registry)
    {
        registry.append(
            StorageReference::class.java, InputStream::class.java,
            FirebaseImageLoader.Factory()
        )
    }
}
```

- firebase-ui-storage 라이브러리를 이용하면 Glide의 load() 함수에 Storage Reference를 직접 전달해서 이미지를 쉽게 내려받을 수 있다.

• 이미지 내려받기

```
val imgRef: StorageReference = storageRef.child("images/a.jpg")
Glide.with(this ... 생략 ...)
    .load(imgRef)
    .into(binding.downloadImageView)
```

실습2: 이미지 공유 앱 만들기

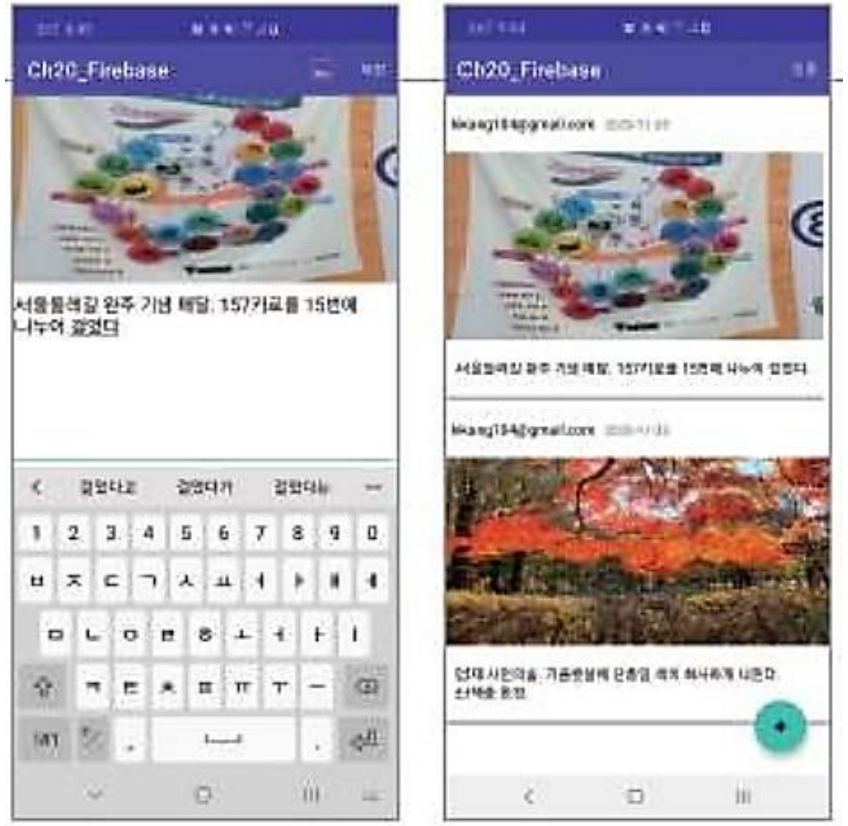
- **1단계. 파이어스토어 설정하기**
  - 파이어베이스콘솔에서새로운데이터베이스만들기
- **2단계. 스토리지 설정하기**
  - 스토리지를 설정
- **3단계. 빌드 그래들 설정하기**
  - 모듈 수준의 build.gradle 파일을 열어 파이어스토어와 스토리지 라이브러리를 추가
- **4단계. 데이터 입력 액티비티 생성하기**
  - 사용자가 데이터를 입력하는 화면으로 AddActivity라는 이름의 새로운 액티비티를 만듭니다.
- **5단계. 매티페스트 설정하기**
  - AndroidManifest.xml 파일을 열고 내용을 추가

실습2: 이미지 공유 앱 만들기

- **6단계. 실습 파일 복사하기**
  - layout, menu 디렉터리를현재모듈의같은위치에덮어쓰기
  - 코틀린 소스 파일이 있는 디렉터리에서 util, model, recycler 디렉터리와 AddActivity.kt 파일을 소스 영역에 복사
- **7단계. MyApplication.kt 작성하기**
  - MyApplication.kt 파일을 열고 앱 전역에서 FirebaseFirestore와 Firebase Storage 객체를 사용하도록 코드를 추가
- **8단계. AddActivity.kt 작성하기**
  - AddActivity.kt 파일을 열고 코드를 추가
- **6단계. MyAdapter.kt 작성하기**
  - MyAdapter.kt 파일을 열고 코드를 추가

실습2: 이미지 공유 앱 만들기

- **7단계. 메인 액티비티 작성하기**
  - MainActivity.kt 파일을 열고 코드를 추가
- **8단계. 앱 실행하기**

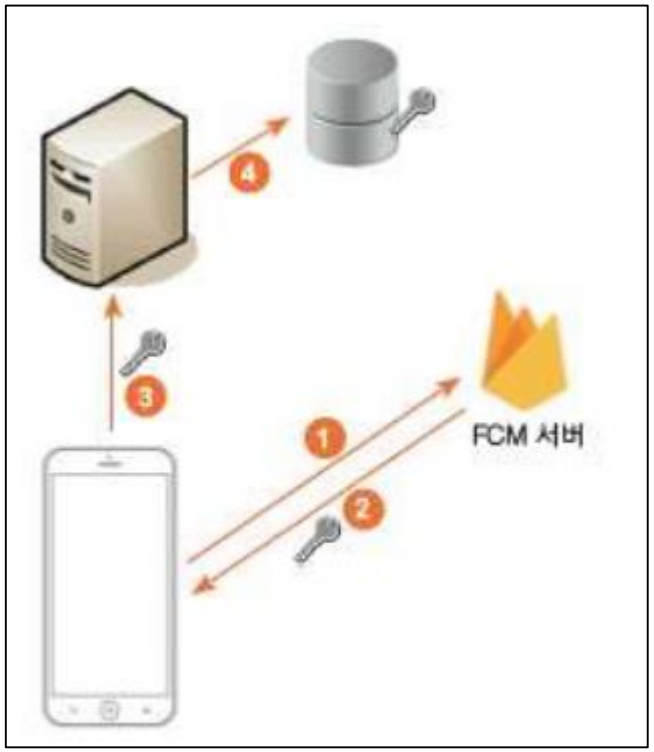


6. 파이어베이스 클라우드 메시징

1단계: 토큰 발급

- 클라우드 메시징을 이용하는 앱을 구분하는 식별 값이 필요
- 안드로이드 시스템이 FCM 서버에 자동으로 의뢰해서 발급

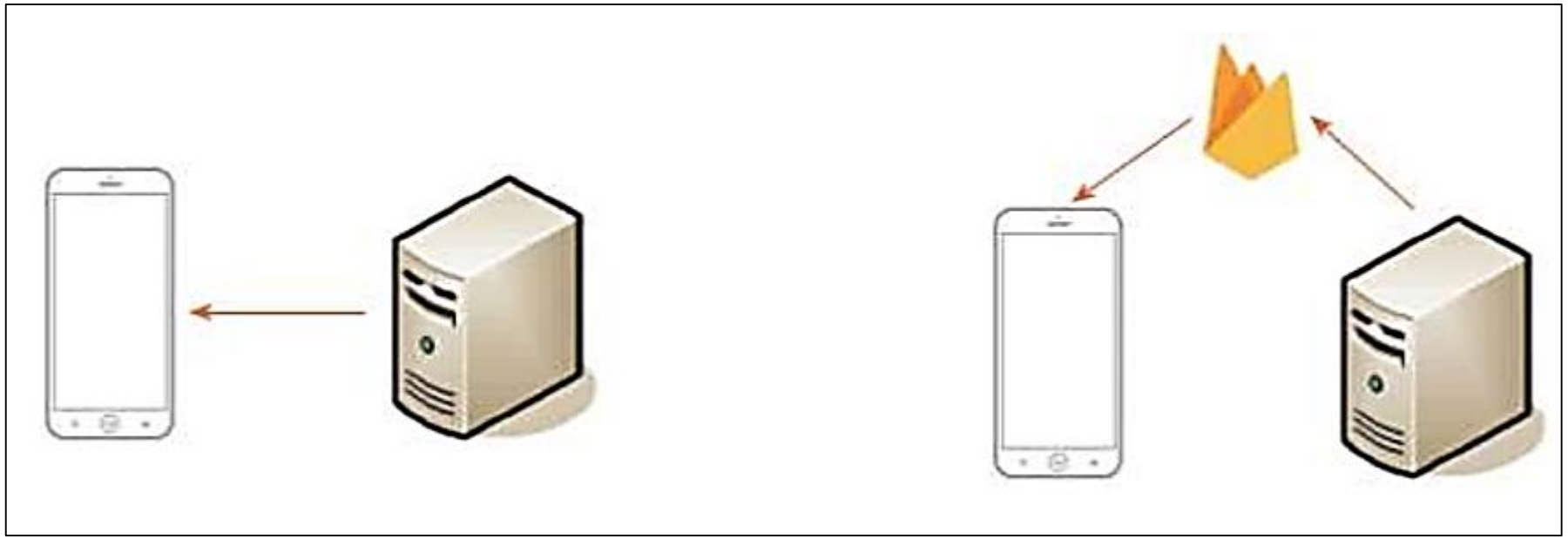
1. 클라우드 메시징을 이용하는 앱이 폰에 설치되면 안드로이드 시스템이 자동으로 FCM 서버에 토큰 발급을 요청합니다.
2. FCM 서버에서 앱을 식별하는 토큰을 발급해 폰에 전달합니다.
3. 앱에 전달된 토큰은 FCM 서버에서 메시지가 발생할 때 사용되므로 서버에 전달합니다.
4. 서버에서 전달받은 토큰을 데이터베이스에 저장합니다.



6. 파이어베이스 클라우드 메시징

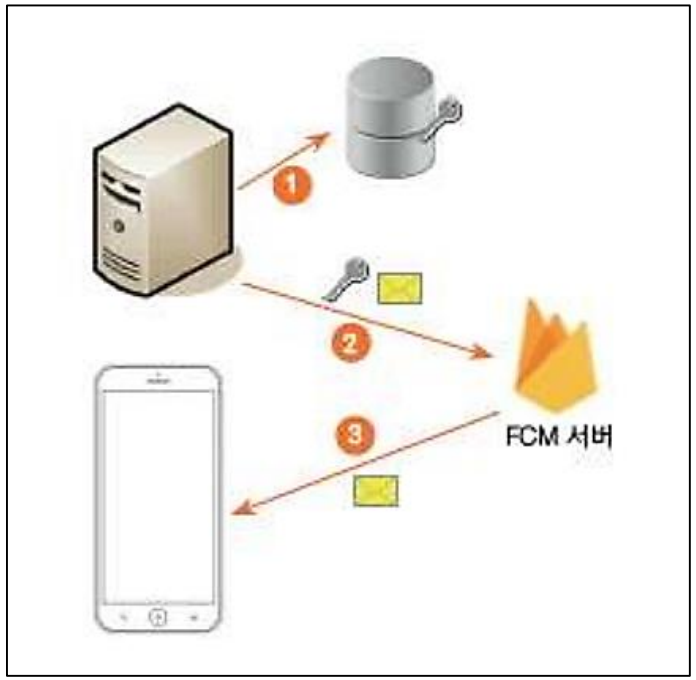
클라우드 메시징의 원리

- 파이어베이스 클라우드 메시징(Firebase cloud messaging), FCM은 서버에서 특정 상황이나 데이터가 발생할 때 앱에 알림을 전달하는 기능
- 서버에서 어떤 상황이 발생할 때 클라이언트(앱)에 데이터를 전달하는 것을 서버 푸시라고 한다.



6. 파이어베이스 클라우드 메시징

- **2단계: 서버에서 앱으로 데이터 전송**
  1. 서버에서 특정 상황이 발생하면 데이터베이스에 저장해 뒀던 토큰을 추출하여 앱을 식별합니다.
  2. 서버에서 사용자에게 전달할 메시지와 앱을 식별할 토큰을 FCM 서버에 전달합니다.
  3. FCM 서버에서 토큰을 분석해 해당 사용자의 폰에 메시지를 전달합니다.



6. 파이어베이스 클라우드 메시징

클라우드 메시징 설정하기

그래들 설정

• 구글 서비스 등록(프로젝트 수준 그래들)

```
buildscript {
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        (... 생략 ...)
        classpath 'com.google.gms:google-services:4.3.10'
    }
}
```

• FCM 관련 라이브러리 등록(모듈 수준 그래들)

```
plugins {
    (... 생략 ...)
    id 'com.google.gms.google-services'
}
(... 생략 ...)
dependencies {
    (... 생략 ...)
    implementation platform('com.google.firebase:firebase-bom:29.0.0')
    implementation 'com.google.firebase:firebase-messaging-ktx:23.0.0'
    implementation 'com.google.firebase:firebase-analytics-ktx:20.0.0'
}
```

6. 파이어베이스 클라우드 메시징

- 매니페스트 설정

• 알림 자동 발생을 위한 메타 데이터 설정

```
<meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/ic_stat_ic_notification" />
<meta-data
    android:name="com.google.firebase.messaging.default_notification_color"
    android:resource="@color/colorAccent" />
<meta-data
    android:name="com.google.firebase.messaging.default_notification_channel_id"
    android:value="fcm_default_channel" />
```

6. 파이어베이스 클라우드 메시징

▪ 서비스 컴포넌트 작성하기

- 앱에서 FCM 토큰과 메시지를 받는 서비스를 작성
- FCM 서비스는 intent-filter의 action 문자열을 com.google.firebase.MESSAGING_EVENT로 선언

• 서비스 등록

```
<service
    android:name=".fcm.MyFirebaseMessageService"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

실습 3: 파이어베이스 클라우드 메시징

- 서비스 컴포넌트 클래스는 FirebaseMessagingService를 상속받아서 작성

• 서비스 코드

```
class MyFirebaseMessageService : FirebaseMessagingService() {
    override fun onNewToken(p0: String) {
        super.onNewToken(p0)
        Log.d("kkang", "fcm token.....$p0")
    }
    override fun onMessageReceived(p0: RemoteMessage) {
        super.onMessageReceived(p0)
        Log.d("kkang", "fcm message.....${p0.data}")
    }
}
```

실습 3: 서버에서 보내는 알림 받기

- **1단계. 빌드 그래들 설정하기**
  - 파이어베이스 클라우드 메시징을 사용해야하므로 모듈수준의 빌드 그래들 파일에 라이브러리를 등록
- **2단계. 실습 파일 복사하기**
  - drawable 디렉터리의 ic_stat_ic_notification.png 파일을 같은 위치에 복사
- **3단계. 색상 리소스 추가하기**
  - res/values 디렉터리의 colors.xml 파일을 열고 color 리소스를 추가
- **4단계. 서비스 컴포넌트 만들기**
  - [New → Service → Service]를 선택하여 MyFirebaseMessageService라는 이름으로 새로운 서비스 컴포넌트를 추가

실습 3: 서버에서 보내는 알림 받기

- **5단계. 매티페스트 설정하기**
  - 앱에서 FCM 메시지를 받기 위해 AndroidManifest.xml 파일을 열고 작성
- **6단계. 서비스 컴포넌트 작성하기**
  - MyFirebaseMessageService.kt 파일을 열고 토큰과 메시지를 로그에 출력하는 코드를 작성
- **7단계. 앱 실행하고 토큰 확인하기**
  - 안드로이드 스튜디오에서 로그캣 창에서 FCM 서버가 발급한 토큰을 확인
- **8단계. 파이어베이스 콘솔에서 비공개 키 내려받기**
  - FCM에 데이터를 전송할 때 사용할 비공개 키를 파이어베이스 콘솔에서 내려받습니다.

실습 3: 서버에서 보내는 알림 받기

- **9단계. 서버 구현하기**
  - FCM에 메시지를 전송하는 서버를 간단하게 Node.js로 구현
- **10단계. 서버 파일 작성하기**
  - fcm_node_server 디렉터리에서 node_fcm_server.js 파일을 만들고 메모장 등으로 열어서 작성
- **11단계. 서버에서 메시지 발송하기**
  - 서버를 실행하여 FCM 서버에 메시지를 전달