Firebase Chat App

1. Realtime DB

- **실시간 DB:** DB에서 데이터를 실시간으로 주고받는것을 의미
- Firebase의 DB는 개발자가 직접 구축하는게 아니라 Firebase에서 기본적으로 제공하는 폼을 사용.
- DB는 안드로이드에서 API 사용을 통해 실시간으로 앱 데이터 저장 및 동기화.

2. Firebase 안드로이드 환경 구축

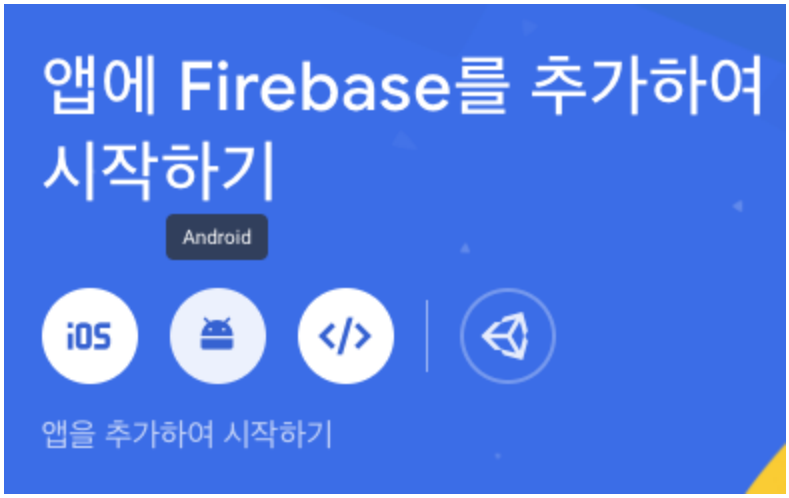
- 안드로이드 프로젝트에서 파이어베이스 API 사용을 위해서는 기본적으로 Firebase에 프로젝트를 생성해서, 내 안드로이드 프로젝트에서 Firebase를 사용할 수 있도록 해야함.
- Realtime DB를 사용하기 위해 규칙설정 필요.

3. 안드로이드 프로젝트에 파이어베이스 등록 과정

→ 파이어베이스 홈페이지 접속 및 프로젝트 생성
https://firebase.google.com/?hl=ko

→ **안드로이드 앱에 파이어베이스 추가**

- 현재 개발 중인 안드로이드의 패키지 명 작성 및 앱 등록



- json 파일 설치 후 Android App 루트 디렉터리에 추가.

- Firebase SDK 추가.

  → project/build_gradle

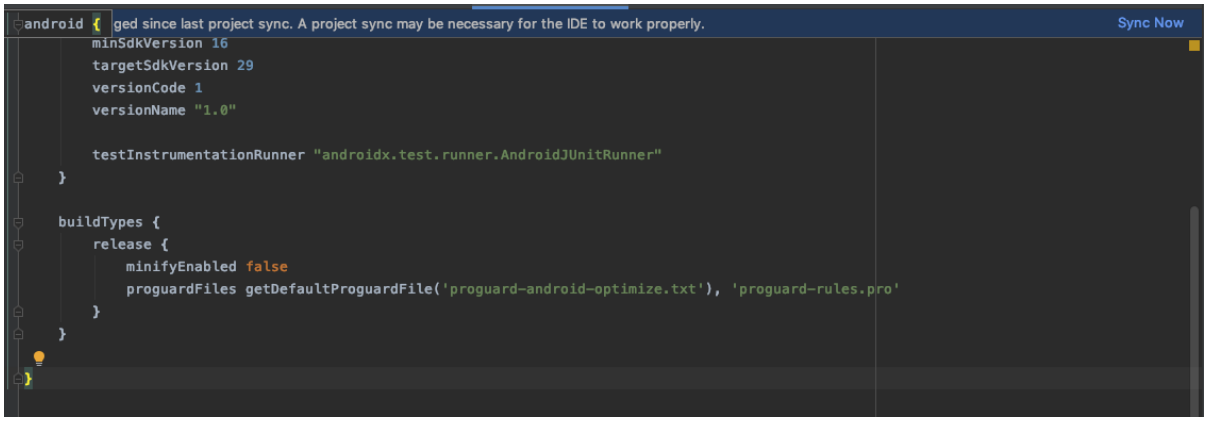
```
buildscript {

    repositories {
        google() //이 부분 추가
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.6.3'
        classpath 'com.google.gms:google-services:4.3.3' //이 부분 추가
    }
}

allprojects {
    repositories {
        google() //이 부분 추가
```

```
        jcenter()

    }
}
```

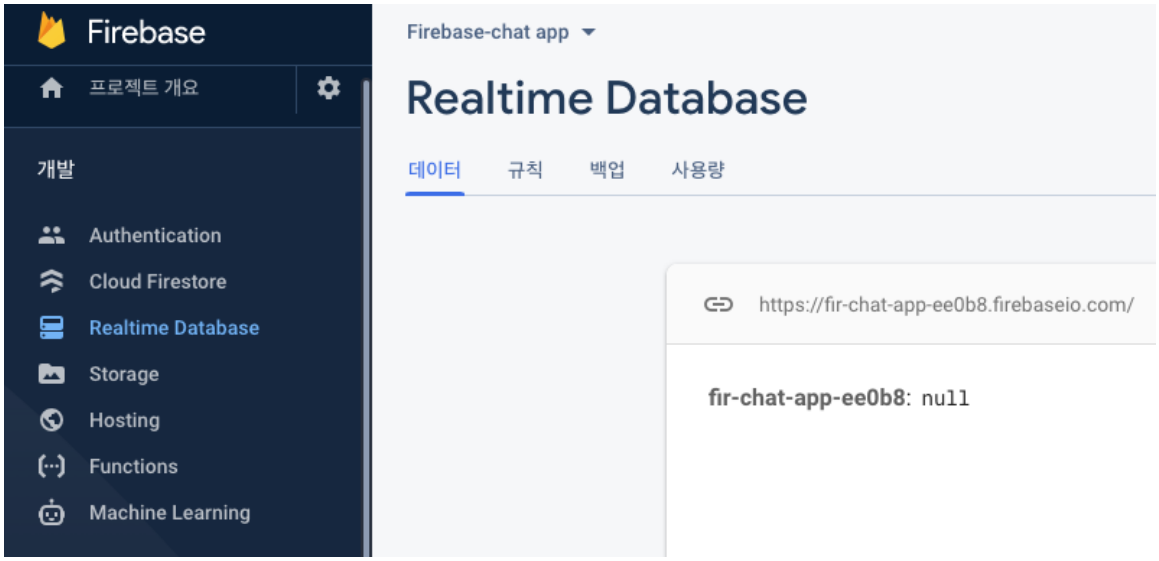
→ project/app/build_gradle

```
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services' // 이 부분 추가

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.google.firebase:firebase-analytics:17.2.2' // 이부분 추가
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

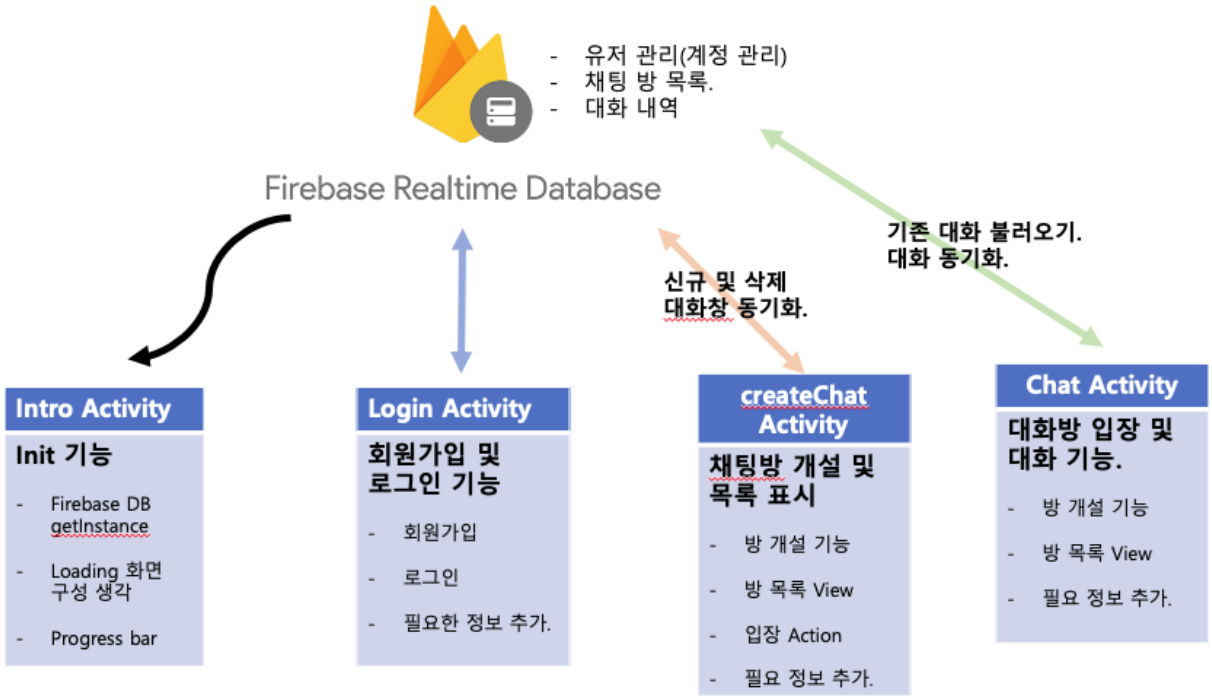
→ gradle 파일 수정 후 IDE위의 Synk now 눌러서 동기화.



- 파이어베이스의 DB 추가 및 테스트 모드로 시작.



4. 채팅 어플 proto 설계서

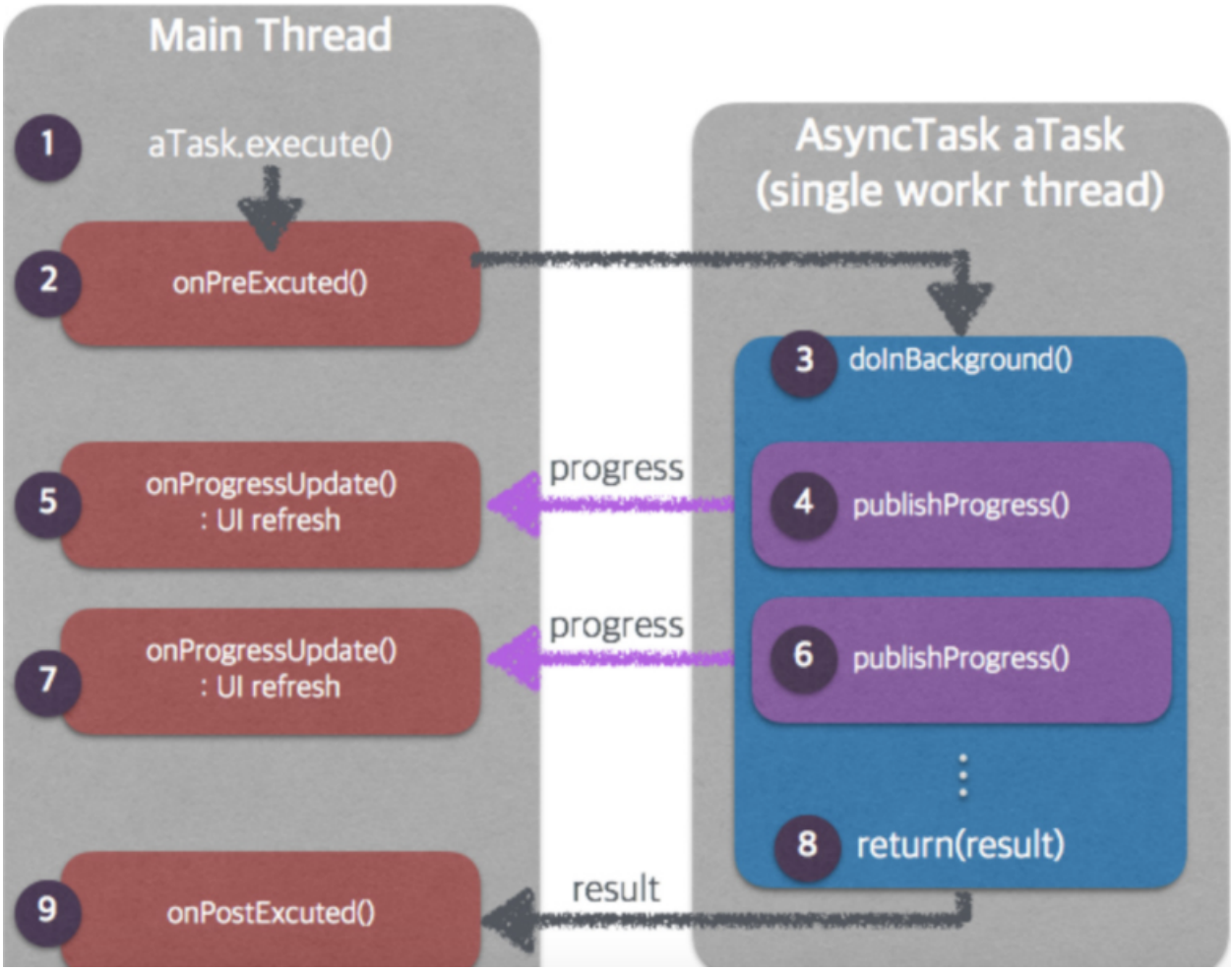


4-1. IntroActivity

부가 개념

AsyncTask

- 안드로이드에서 요구하는 메인 스레드와 작업 스레드의 분리 구조를 보다 쉽게 구현해주는 추상 클래스.



동작 순서

1. excute() 명령어를 통해 AsyncTask를 실행.
2. AsyncTask로 백그라운드 작업을 실행하기 전에 OnPreExcuted()실행됨, 이 부분에는 사용자 UI에 ProgressBar 표시등 본격적인 작업 스레드에 들어가기 전에 작업 진

행줄을 표시하는 구현이 들어감. 스레드 작업 이전에 수행할 동작을 구현.

3. 새로 만든 스레드에서 백그라운드 작업을 수행함. execute() 메소드를 호출할 때 사용된 파라미터를 전달 받음.
4. doInBackground()에서 중간 중간 진행 상태를 UI에 업데이트 하도록 하려면 publishProgress() 메소드 호출.
5. onProgressUpdate() 메소드는 publishProgress()가 호출 될 때 마다 자동으로 호출되는 함수로 작업 스레드를 실행하는 도중에 UI처리를 담당. 일반적으로 작업 진행 정도를 표시하는 용도로 사용.
6. doInBackground() 메소드에서 작업이 끝나면 onPostExcuted로 결과 파라미터를 리턴하면서 그 리턴값을 통해 스레드 작업이 끝났을 때의 동작을 구현.

여기서 핵심은 onPreExecute(), onProgressUpdate(), onPostExecute() 메소드는 메인 스레드에서 실행되므로 UI 객체에 자유롭게 접근 가능.

기본형인 **AsynkTask <Params, Progree, Result>**

- Params: doInBackground 파라미터 타입이 되며, execute 메소드 인자 값이 됨.
- Progress: doInBackground 작업 시 진행 단위의 타입으로, onProgressUpdate 파라미터 타입.
- Result: doInBackground의 리턴값으로 onPostExecute 파라미터 타입.

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    // 버튼을 클릭하면 파일 다운로드 경로를 파라미터로 AsyncTask 실행
    public void OnClick(View view) {
        switch (view.getId()) {
            case R.id.button:
                try {
                    new DownloadFilesTask().execute(new URL("파일 다운로드 경로1"));
```

```
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                }
                break;
        }
    }

    private class DownloadFilesTask extends AsyncTask {
        @Override
        protected void onPreExecute() {
            super.onPreExecute();
        }

        @Override
        protected Long doInBackground(URL... urls) {
                // 전달된 URL 사용 작업

            return total;
        }

        @Override
        protected void onProgressUpdate(Integer... progress) {
            // 파일 다운로드 퍼센티지 표시 작업
        }

        @Override
        protected void onPostExecute(Long result) {
            // doInBackground 에서 받아온 total 값 사용 장소
        }
    }
}
```

제약조건 및 단점

- Task는 오직 한번만 실행 가능.
- 하나의 객체이므로 재사용이 불가능함.
- 구현한 Activity 종료와 같이 종료되지 않음.
- Activity 종료 후 재시작 시 AsyncTask의 Reference가 되지 않음.
-