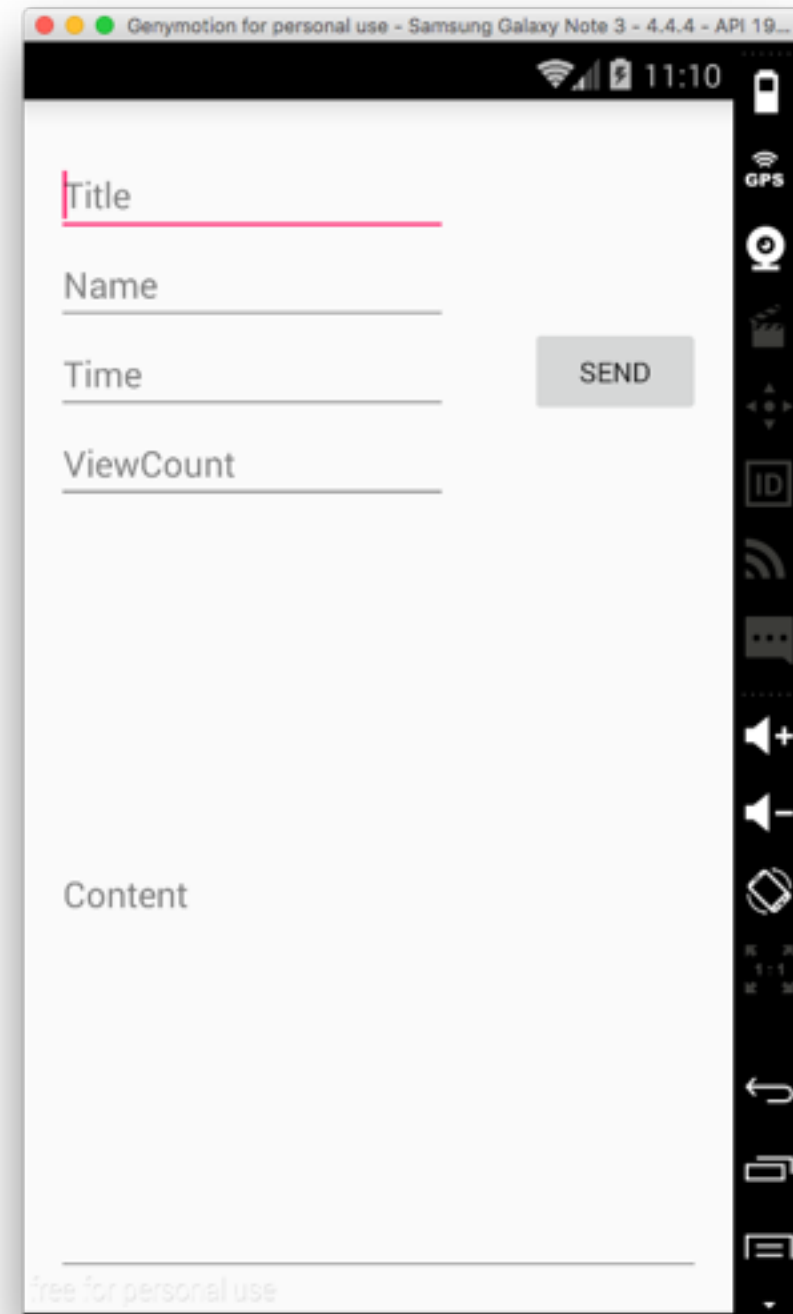
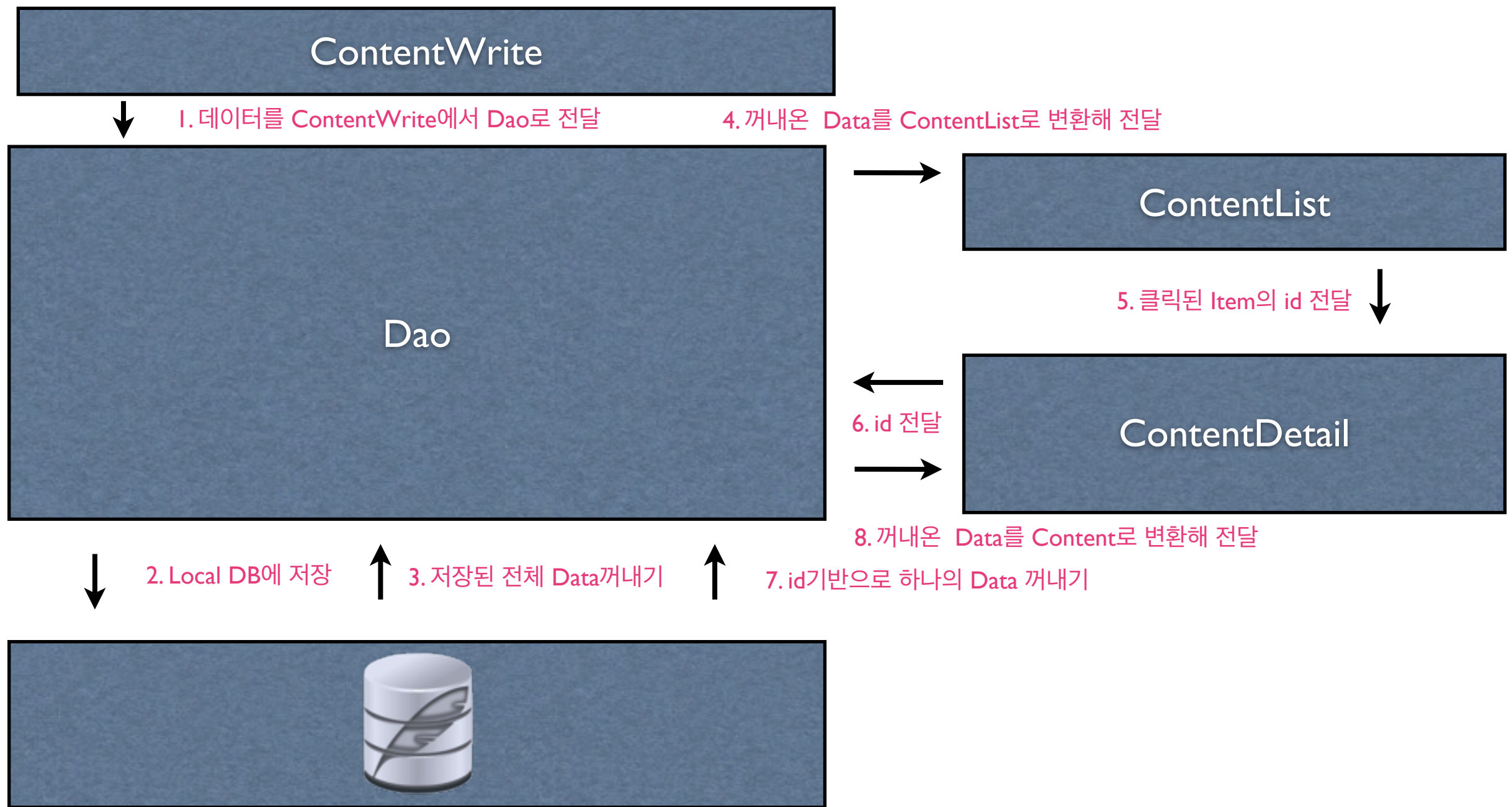


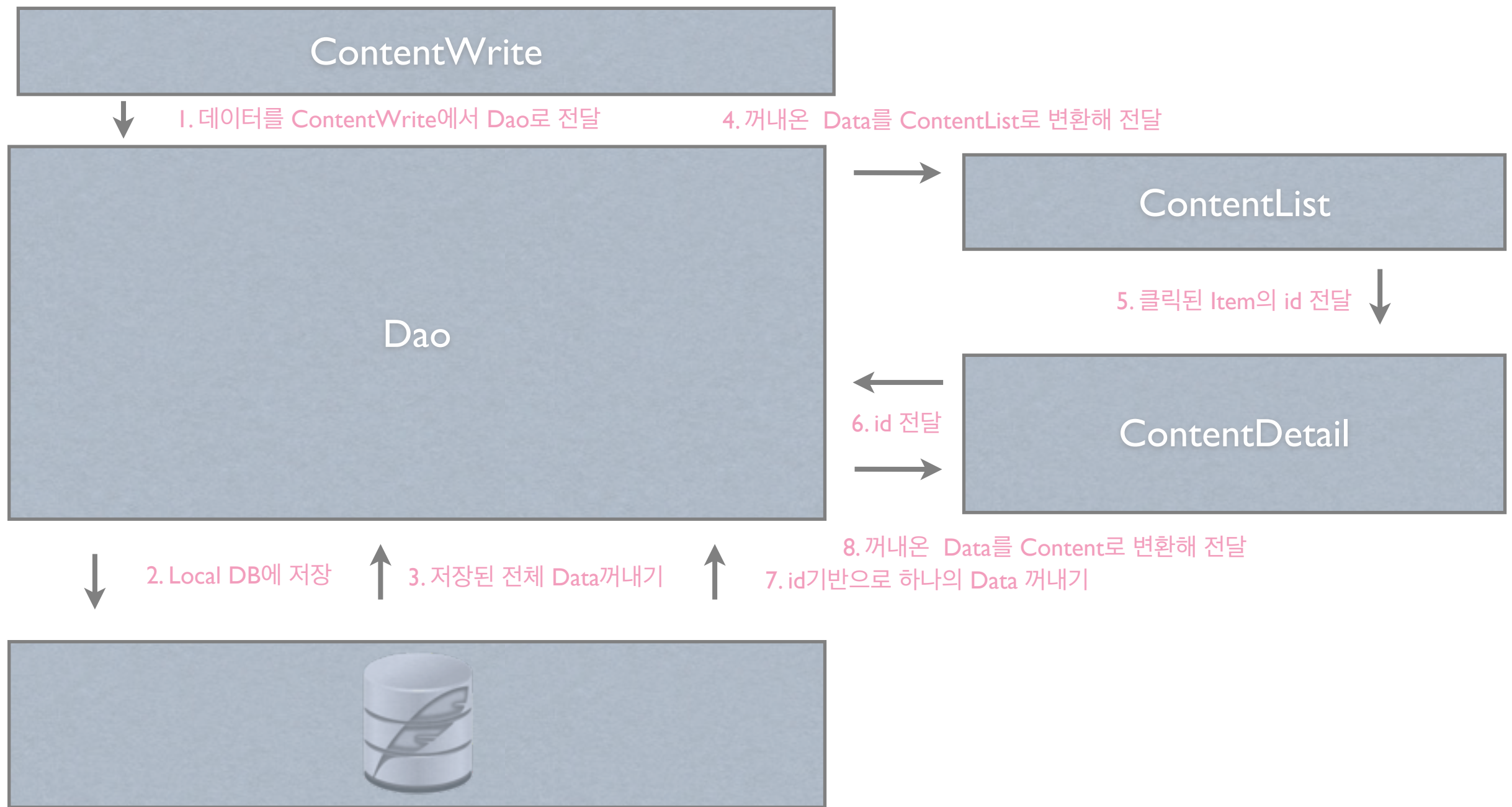
Android Network



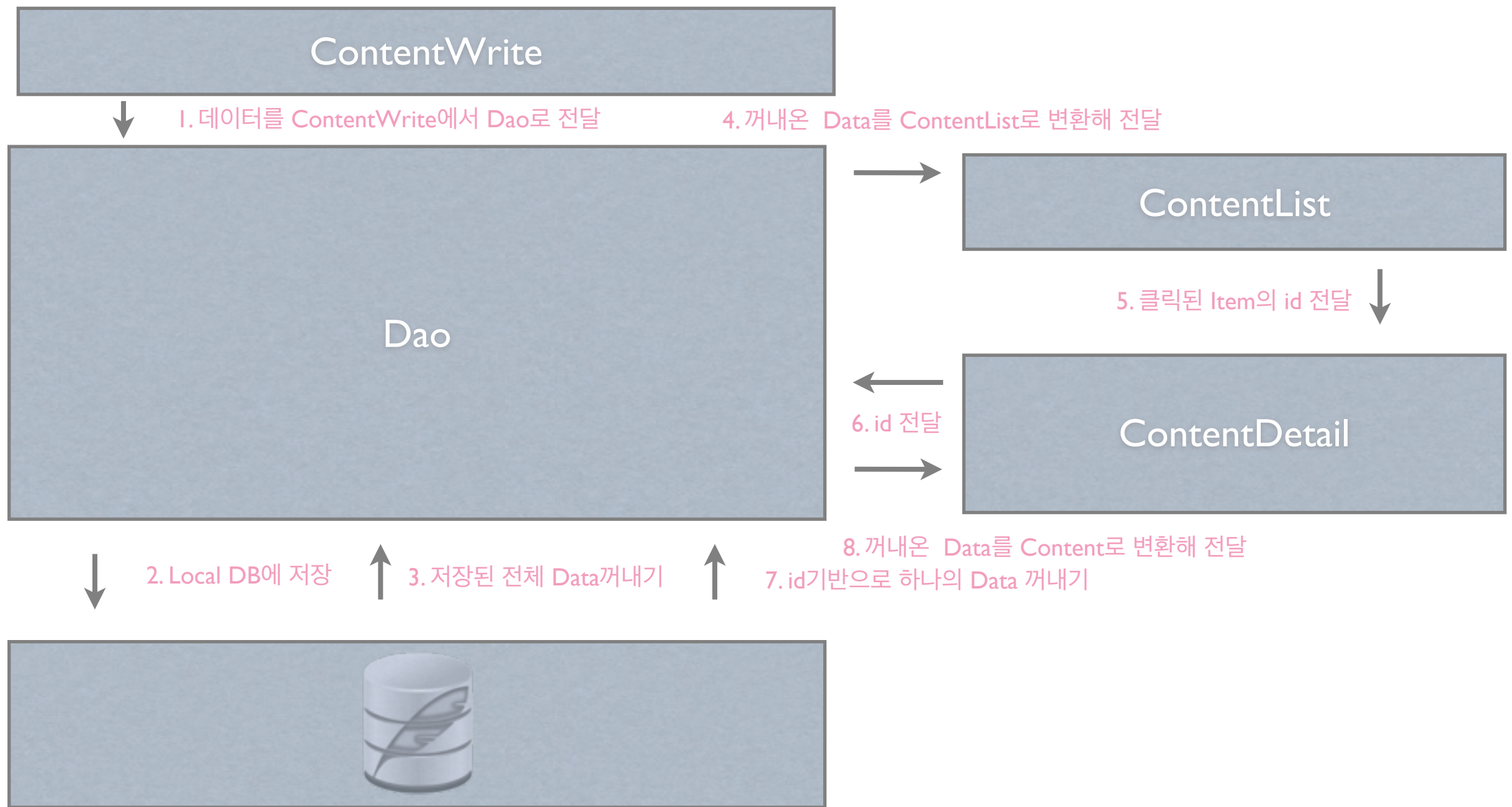
지금까지 우리는 화면 구성,
DataBase를 붙이는 방법에 대해서 배워봤습니다.



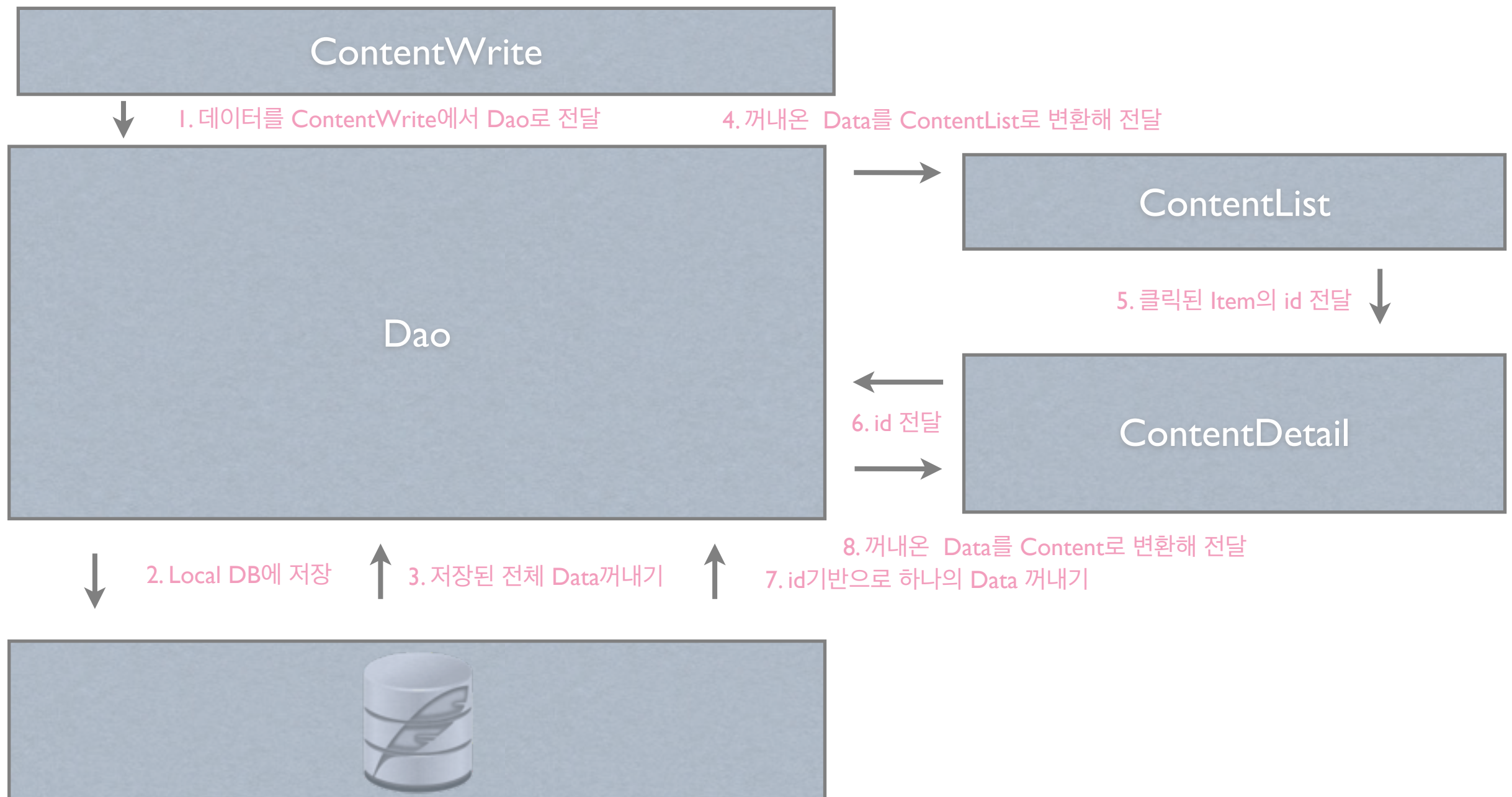
현재의 구조를 눈으로 볼 수 있게
정리해 보면 이와 같습니다.



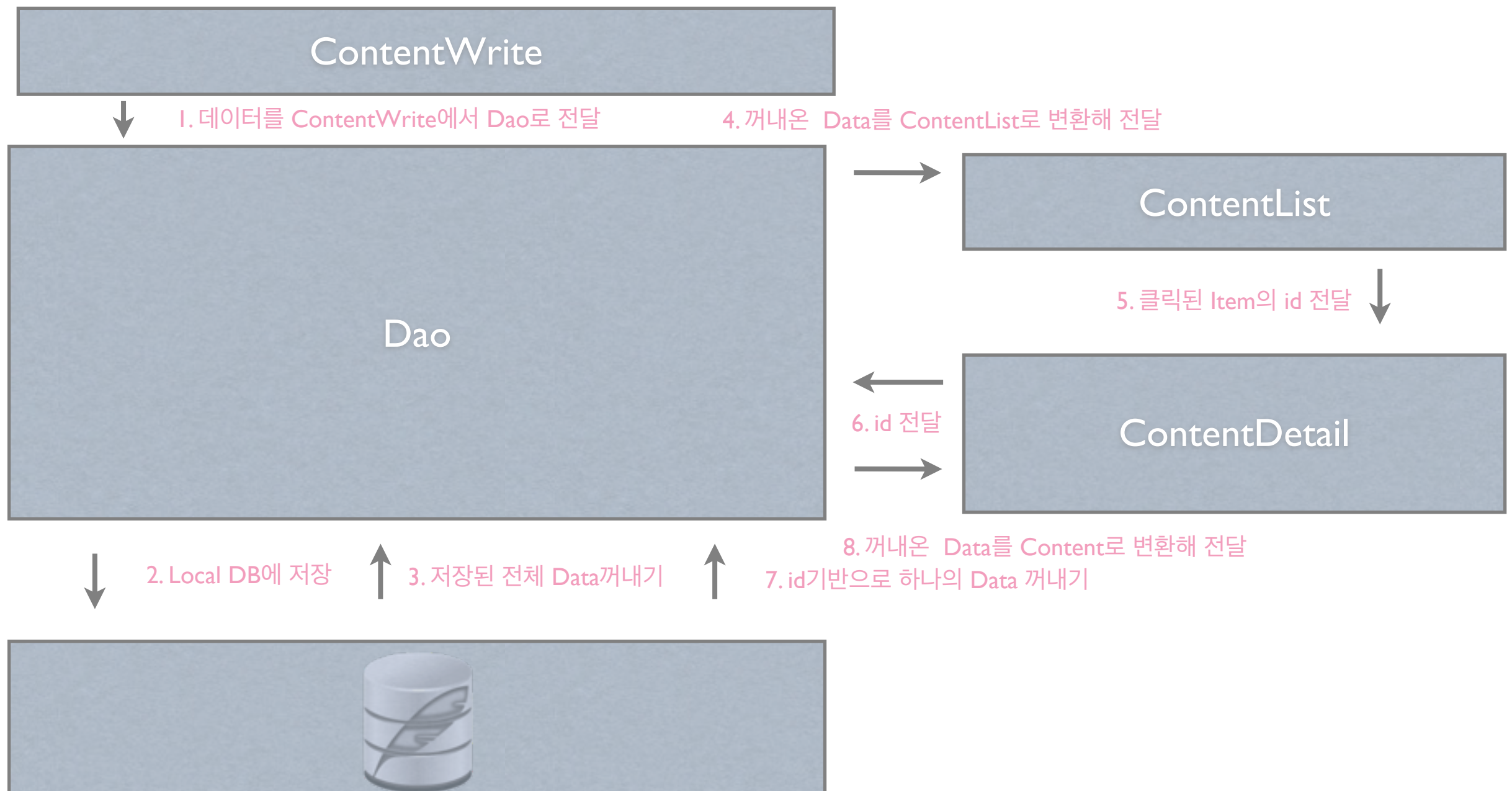
Local DB를 기반으로 동작하는 게시판 어플입니다.



즉 나혼자 쓰는 메모 어플이라 보아도 무방합니다.



이제 진짜 게시판처럼 동작하기 위해
네트워크와 연동을 해보도록 하겠습니다.



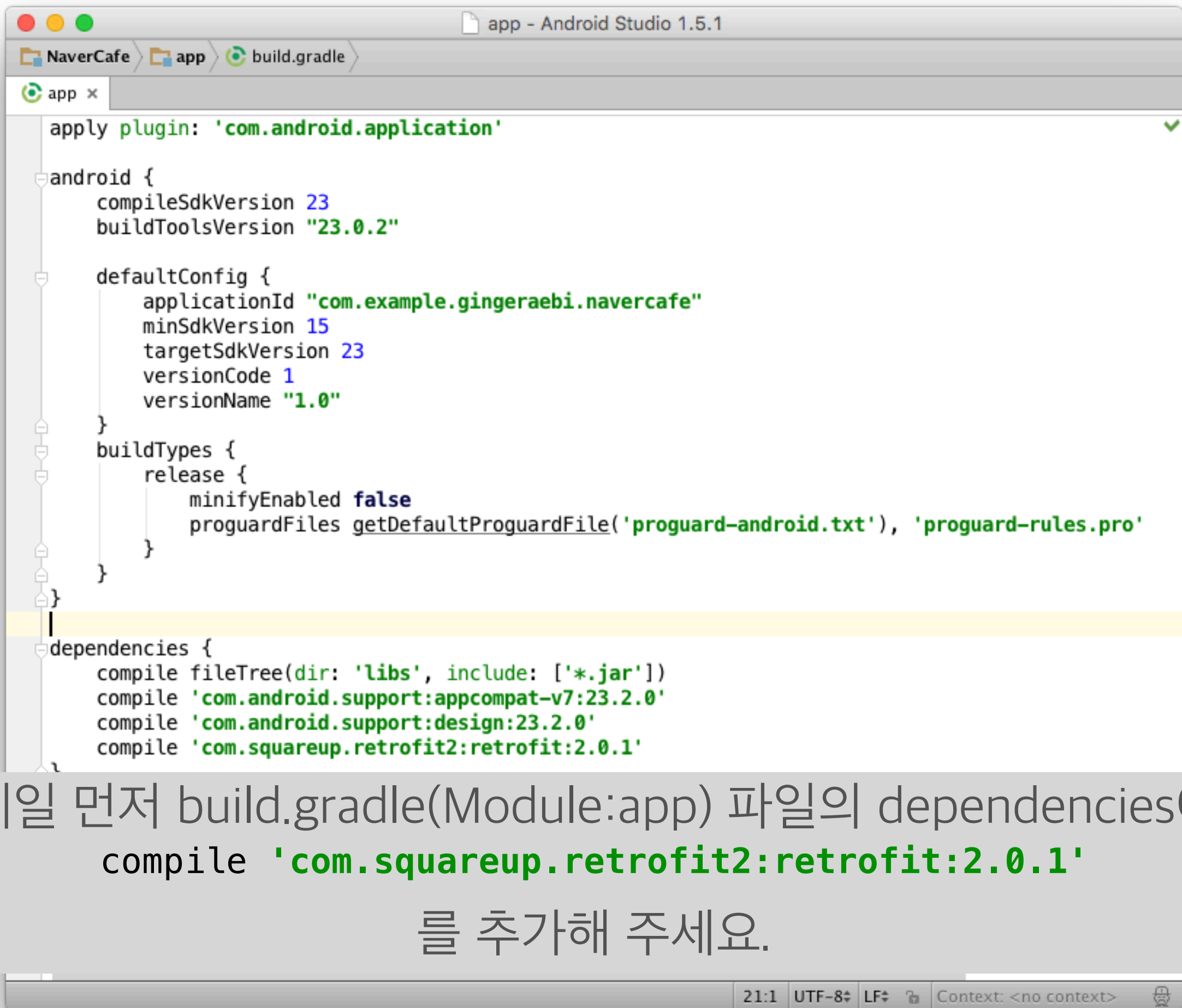
즉, 서버를 붙여 다른사람들이 올린게시글 또한
내 어플리케이션에서 확인 할 수 있도록 해보겠습니다.



HTTP 클라이언트가 애플리케이션 개발의 중심이라고도 할 정도로 Android 애플리케이션에서는 HTTP 통신을 다루는 부분의 비중이 큽니다.
(돈버는 앱들은 다 네트워크가 있습니다!!!)

우리는 retrofit이라는
오픈소스를 사용하여 네트워크 통신을 할 것입니다.

이 방법은 안드로이드가 네트워크를 사용하는
정석적인 방법은 아님을 인지하시길 바랍니다.



```
app - Android Studio 1.5.1
NaverCafe > app > build.gradle
app x
apply plugin: 'com.android.application' ✓

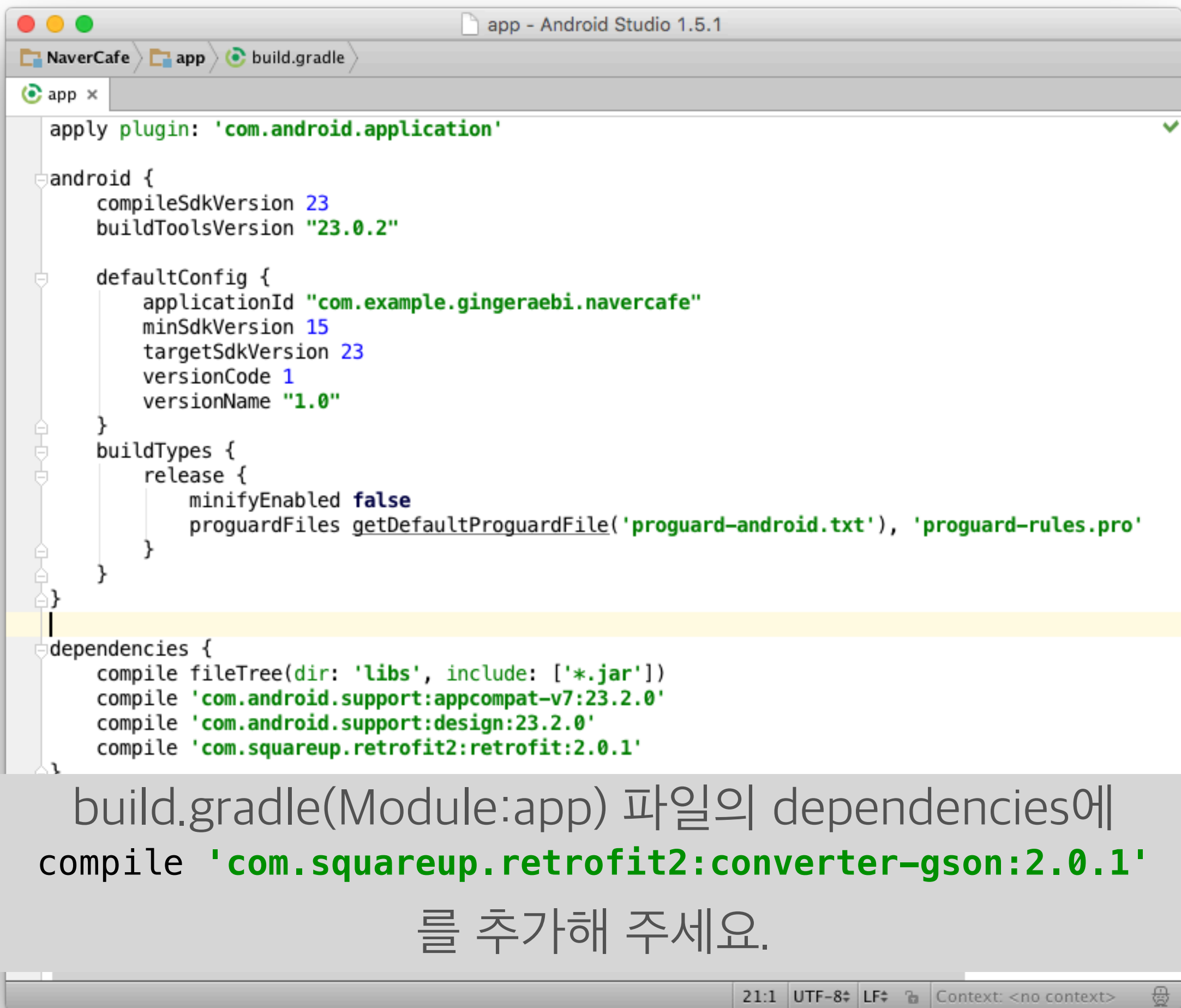
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

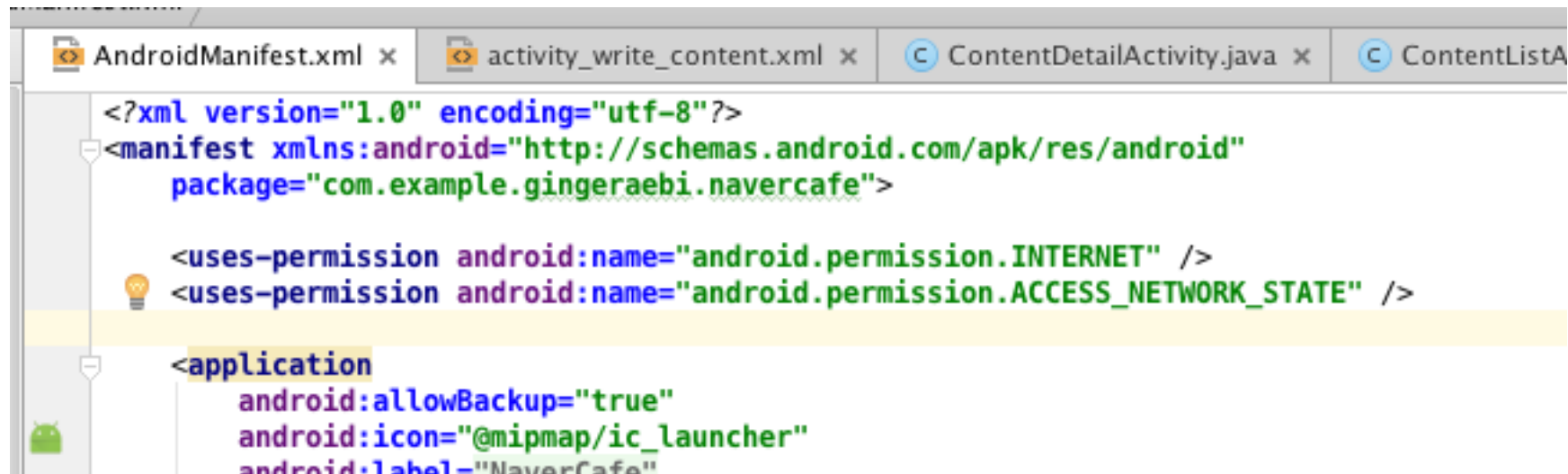
    defaultConfig {
        applicationId "com.example.gingeraebi.navercafe"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.2.0'
    compile 'com.android.support:design:23.2.0'
    compile 'com.squareup.retrofit2:retrofit:2.0.1'
```

21:1 UTF-8 LF Context: <no context>

제일 먼저 build.gradle(Module:app) 파일의 dependencies에
compile 'com.squareup.retrofit2:retrofit:2.0.1'
를 추가해 주세요.






```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.gingeraebi.navercafe">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="NaverCafe"
```

AndroidManifest파일에 Network에 관련된 권한을 추가해 줍니다.

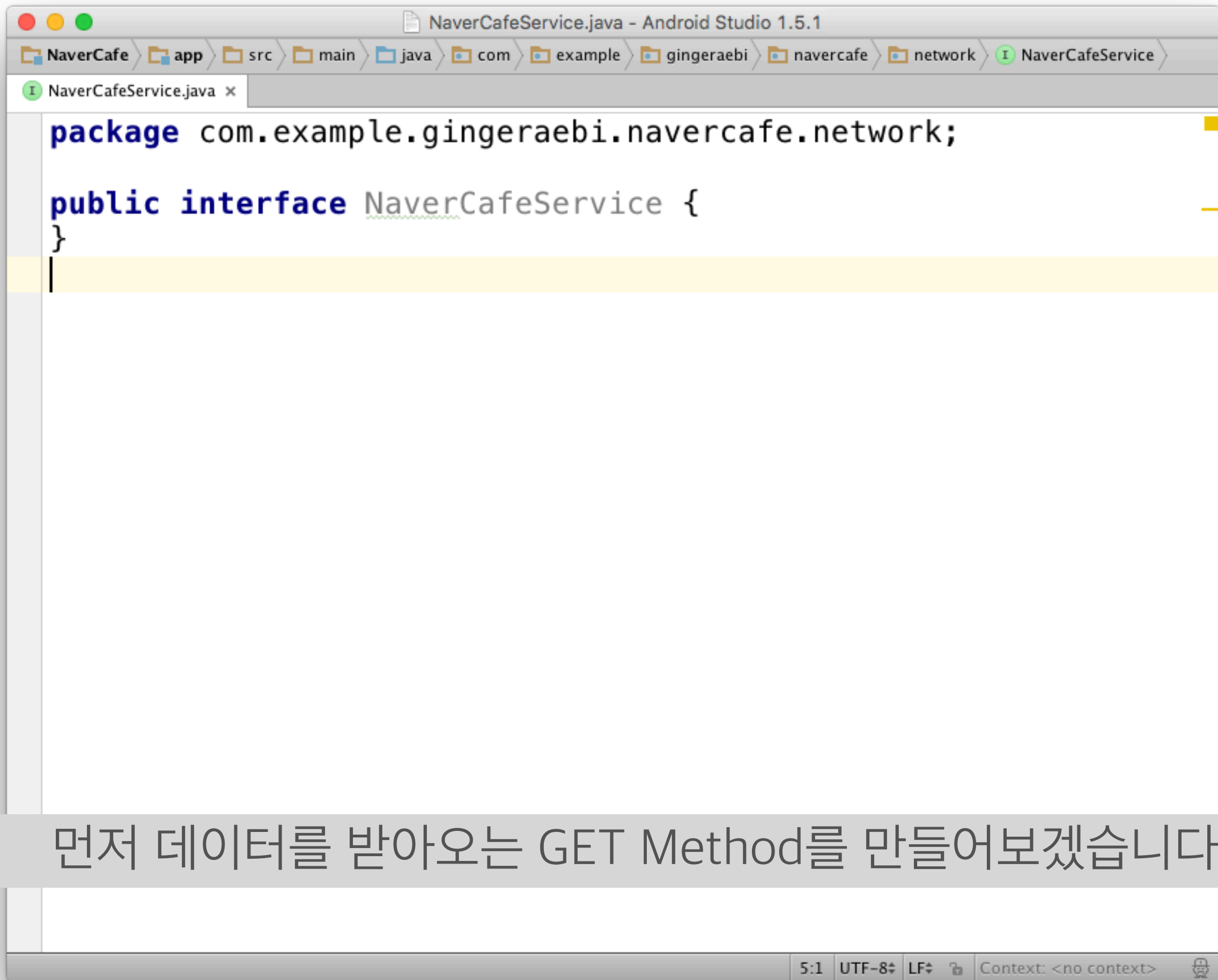


```
package com.example.gingeraebi.navercafe.network;

public interface NaverCafeService {
}
```

5:1 UTF-8 LF Context: <no context>

그 후 ~~Service의 이름을 가진 interface를 하나 선언해주세요.
이 인터페이스는 request URL과 메소드를 매핑할 때 사용될 것입니다.



먼저 데이터를 받아오는 GET Method를 만들어보겠습니다.


```
NaverCafeService.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > I NaverCafeService
I NaverCafeService.java x
package com.example.gingeraebi.navercafe.network;

import com.example.gingeraebi.navercafe.vo.Content;

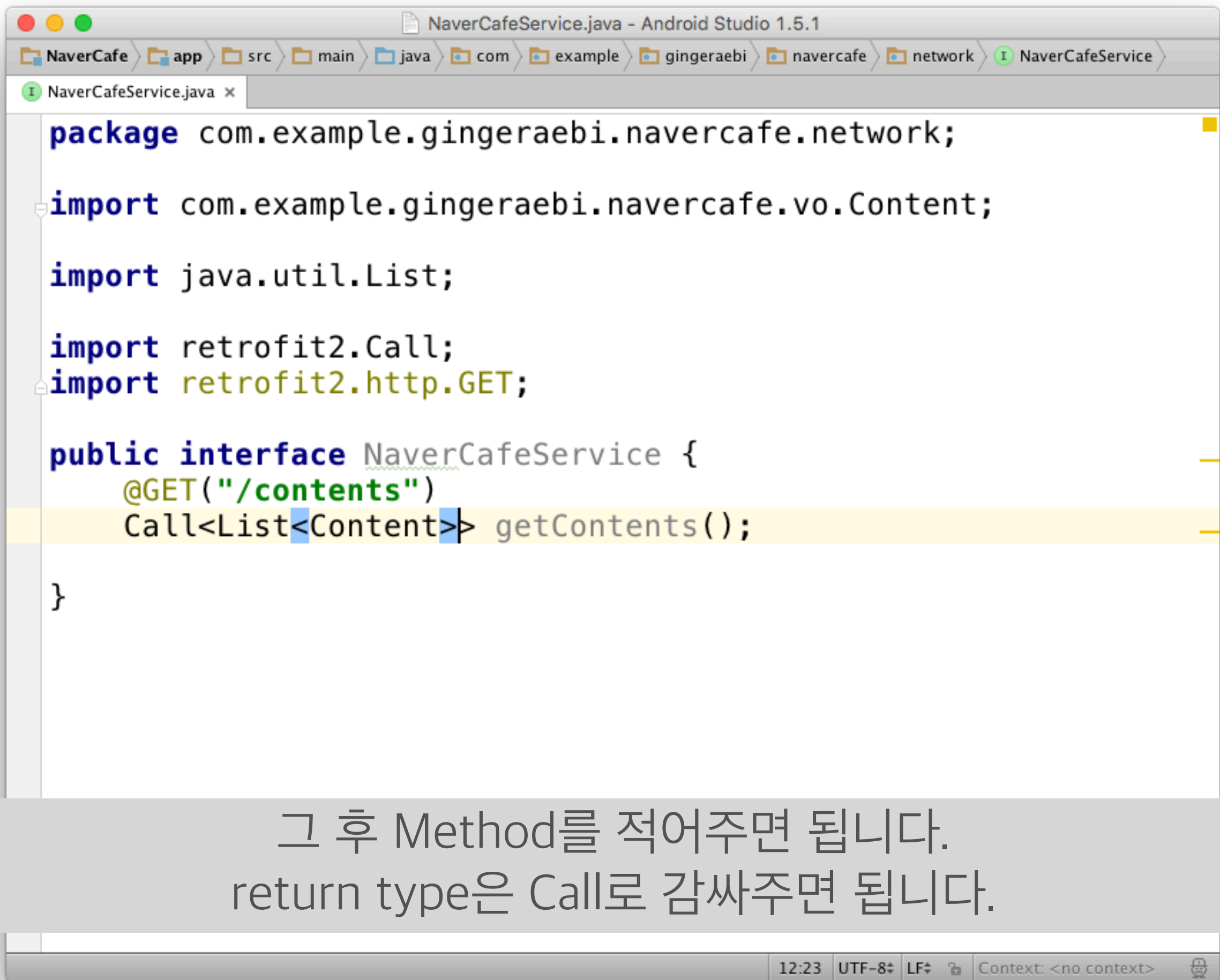
import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;

public interface NaverCafeService {
    @GET("/contents")
    Call<List<Content>> getContents();
}

12:23 UTF-8 LF Context: <no context>
```

어떤 Http Method로 실행될 것인지 @를 통해서
정해주고 (GET 혹은 POST) 괄호안에 URL을 입력해줍니다.



```
package com.example.gingeraebi.navercafe.network;

import com.example.gingeraebi.navercafe.vo.Content;

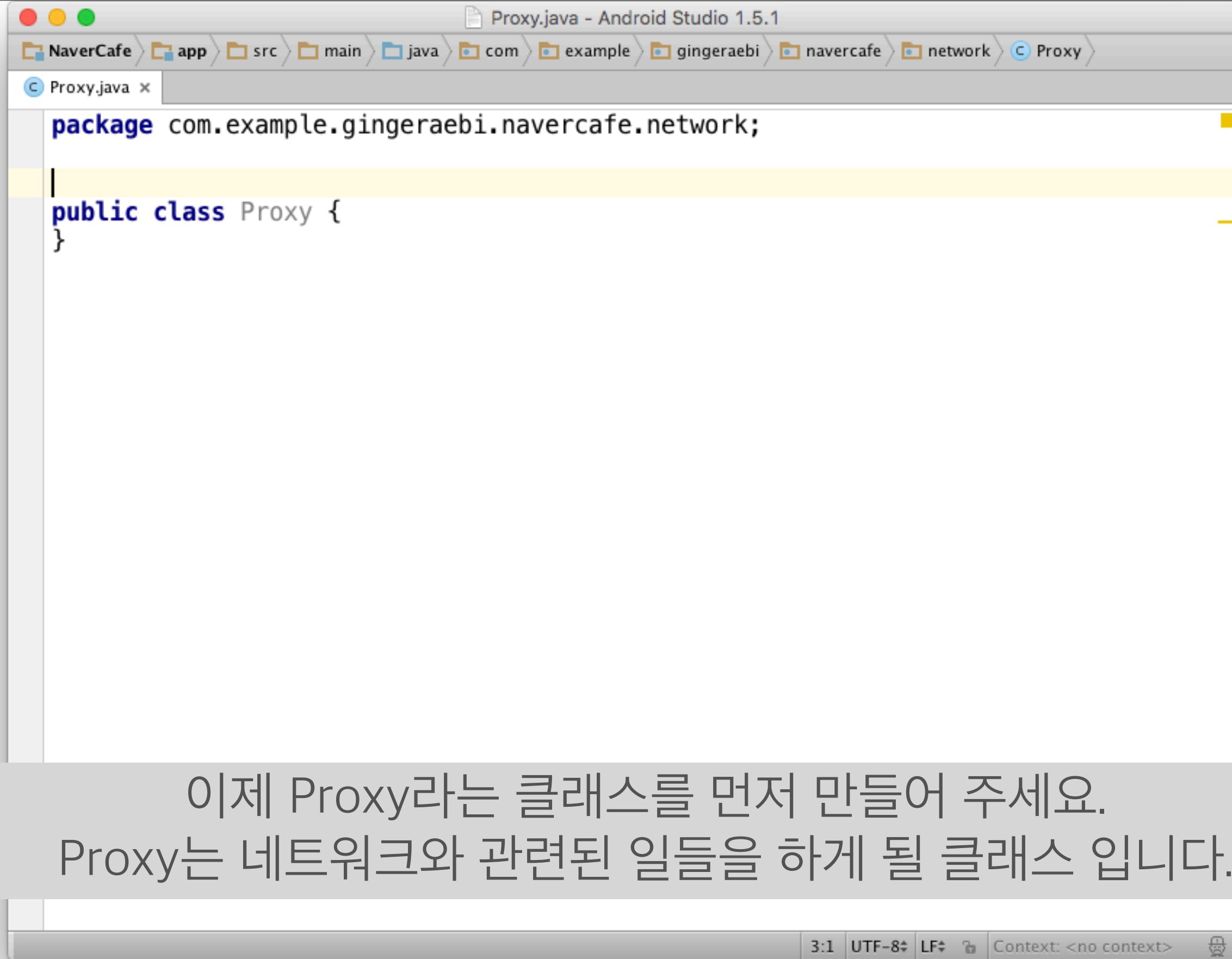
import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;

public interface NaverCafeService {
    @GET("/contents")
    Call<List<Content>> getContents();
}
```

12:23 UTF-8 LF Context: <no context>

그 후 Method를 적어주면 됩니다.
return type은 Call로 감싸주면 됩니다.



```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x
package com.example.gingeraebi.navercafe.network;

public class Proxy {
}
```

3:1 UTF-8 LF Context: <no context>

이제 Proxy라는 클래스를 먼저 만들어 주세요.
Proxy는 네트워크와 관련된 일들을 하게 될 클래스 입니다.


```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

package com.example.gingeraebi.navercafe.network;

import android.content.Context;
import retrofit2.Retrofit;

public class Proxy {

    private Context context;

    public Proxy(Context context) {
        this.context = context;
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://localhost:1000/")
            .build();

        NaverCafeService service = retrofit.create(NaverCafeService.class);
    }
}
```

12:36 UTF-8 LF Context: <no context>

생성자에 Retrofit 객체를 선언해줍니다.
baseUrl에는 본인의 서버 URL을 적어주면 됩니다.

참고 : genymotion에서 localhost로 접근시 10.0.3.2로 설정

```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

package com.example.gingeraebi.navercafe.network;

import android.content.Context;
import retrofit2.Retrofit;

public class Proxy {

    private Context context;

    public Proxy(Context context) {
        this.context = context;
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://localhost:1000/")
            .build();

        NaverCafeService service = retrofit.create(NaverCafeService.class);
    }
}
```

그 후 Service객체를 retrofit.create메소드로 초기화해주면 Service객체를 통해 등록해 놓은 메소드에 접근 가능해지게 됩니다.

```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

public class Proxy {

    private Context context;
    private NaverCafeService service;

    public Proxy() {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.3.2:9000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        service = retrofit.create(NaverCafeService.class);
    }

    public List<Content> getContents() {
        Call<List<Content>> call = service.getContents();

        try {
            List<Content> contents = call.execute().body();
            return contents;
        } catch (IOException e) {
            Log.e("Proxy", e.getMessage());
        }
    }
}
```

retrofit 객체에 .addConvertFactroy를 해 주었습니다.
결과값을 받을 때 GsonConverFatory를 설정해주면
결과 값으로 받은 JSON을 자동으로 VO객체로 변환해 줍니다.


```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

public class Proxy {

    private Context context;
    private NaverCafeService service;

    public Proxy() {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.3.2:9000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        service = retrofit.create(NaverCafeService.class);
    }

    public List<Content> getContents() {
        Call<List<Content>> call = service.getContents();

        try {
            List<Content> contents = call.execute().body();
            return contents;
        } catch (IOException e) {
            Log.e("Proxy", e.getMessage());
        }

        return null;
    }
}
```

이제 실제로 실행될 메소드를 하나 만들어보도록 하겠습니다.
(context는 일단 제거하였습니다.)

```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

public class Proxy {

    private Context context;
    private NaverCafeService service;

    public Proxy() {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.3.2:9000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        service = retrofit.create(NaverCafeService.class);
    }

    public List<Content> getContents() {
        Call<List<Content>> call = service.getContents();

        try {
            List<Content> contents = call.execute().body();
            return contents;
        } catch (IOException e) {
            Log.e("Proxy", e.getMessage());
        }

        return null;
    }
}
```

service.~~로 아까 인터페이스에 선언해 놓은 함수를 실행시키면 Call이라는 객체에 네트워크 연결 후 실행시킨 결과값이 담겨져서 날아오게 됩니다.

```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

public class Proxy {

    private Context context;
    private NaverCafeService service;

    public Proxy() {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.3.2:9000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        service = retrofit.create(NaverCafeService.class);
    }

    public List<Content> getContents() {
        Call<List<Content>> call = service.getContents();

        try {
            List<Content> contents = call.execute().body();
            return contents;
        } catch (IOException e) {
            Log.e("Proxy", e.getMessage());
        }

        return null;
    }
}
```

응답을 비동기적으로 받는 것이 아닌 동기적으로 받을 때는
call.excute()메서드로 받게 됩니다.
excute()메서드가 실행되는 순간 네트워크에 접근하게 됩니다.

```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

public class Proxy {

    private Context context;
    private NaverCafeService service;

    public Proxy() {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.3.2:9000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        service = retrofit.create(NaverCafeService.class);
    }

    public List<Content> getContents() {
        Call<List<Content>> call = service.getContents();

        try {
            List<Content> contents = call.execute().body();
            return contents;
        } catch (IOException e) {
            Log.e("Proxy", e.getMessage());
        }

        return null;
    }
}
```

또한, 우리는 response의 Body에서 값을 꺼내올 것이기 때문에 body()메서드로 값을 가져오게 됩니다.

```
Proxy.java - Android Studio 1.5.1
NaverCafe > app > src > main > java > com > example > gingeraebi > navercafe > network > Proxy
Proxy.java x NaverCafeService.java x

public class Proxy {

    private Context context;
    private NaverCafeService service;

    public Proxy() {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.3.2:9000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        service = retrofit.create(NaverCafeService.class);
    }

    public List<Content> getContents() {
        Call<List<Content>> call = service.getContents();

        try {
            List<Content> contents = call.execute().body();
            return contents;
        } catch (IOException e) {
            Log.e("Proxy", e.getMessage());
        }

        return null;
    }
}
```

이 때 아까 설정해준 convertFactory의 효과로 body를 자동적으로 VO객체로 바꿔서 받게 됩니다.


```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), WriteContentActivity.class);
        startActivity(intent);
    }
});

FloatingActionButton fab2 = (FloatingActionButton) findViewById(R.id.fab2);
fab2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        new Thread() {
            public void run() {
                Proxy proxy = new Proxy();
                List<Content> contents = proxy.getContents();
                for(Content content : contents) {
                    Log.i("TEST", content.toString());
                }
            }
        }.start();
    }
});
```

이제 테스트를 해보겠습니다.

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), WriteContentActivity.class);
        startActivity(intent);
    }
});

FloatingActionButton fab2 = (FloatingActionButton) findViewById(R.id.fab2);
fab2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        new Thread() {
            public void run() {
                Proxy proxy = new Proxy();
                List<Content> contents = proxy.getContents();
                for(Content content : contents) {
                    Log.i("TEST", content.toString());
                }
            }
        }.start();
    }
});
```

이제 테스트를 해보겠습니다.

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), WriteContentActivity.class);
        startActivity(intent);
    }
});

FloatingActionButton fab2 = (FloatingActionButton) findViewById(R.id.fab2);
fab2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        new Thread() {
            public void run() {
                Proxy proxy = new Proxy();
                List<Content> contents = proxy.getContents();
                for(Content content : contents) {
                    Log.i("TEST", content.toString());
                }
            }
        }.start();
    }
});
```

테스트 하기 위해 ListActivity에 fab를 하나
추가해 주었고 (과정 생략) onClickListner를 달아주었습니다.

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), WriteContentActivity.class);
        startActivity(intent);
    }
});

FloatingActionButton fab2 = (FloatingActionButton) findViewById(R.id.fab2);
fab2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        new Thread() {
            public void run() {
                Proxy proxy = new Proxy();
                List<Content> contents = proxy.getContents();
                for(Content content : contents) {
                    Log.i("TEST", content.toString());
                }
            }
        }.start();
    }
});
```

여기서 주의해야 할 점은 안드로이드에서 Network 접근은 MainThread에서 할 수 없게 만들어 놓았기 때문에 새로운 Thread를 만들어 실행시켜야 한다는 점입니다.

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), WriteContentActivity.class);
        startActivity(intent);
    }
});

FloatingActionButton fab2 = (FloatingActionButton) findViewById(R.id.fab2);
fab2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        new Thread() {
            public void run() {
                Proxy proxy = new Proxy();
                List<Content> contents = proxy.getContents();
                for(Content content : contents) {
                    Log.i("TEST", content.toString());
                }
            }
        }.start();
    }
});
```

새로 Thread를 만들어 run메소드 안에 Porxy 객체를 가져와 값을 꺼내오는 로직을 만들어 보았습니다.


```

public class GetContentsTest extends AsyncTask<Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... params) {
        Proxy proxy = new Proxy();
        List<Content> contents = proxy.getContents();
        for (Content content : contents) {
            Log.i("TEST", content.toString());
        }
        return null;
    }
}

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

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(getApplicationContext(), WriteContentActivity.class);
            startActivity(intent);
        }
    });

    FloatingActionButton fab2 = (FloatingActionButton) findViewById(R.id.fab2);
    fab2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            new GetContentsTest().execute();
        }
    });
}

```

가끔 thread가 문제가 생길수도 있으니 문제가 생긴다면
AsyncTask로 만들어봅시다. (안생기면 안해도 됨)

ContentListActivity

잘 찍히는군요.