

모바일 앱

프로그래밍

보고서

- Book Search App 개선 -

4조

김민규

방세현

이정훈

최민성

**Ⅰ. 팀원소개**

|  |  |  |  |
| --- | --- | --- | --- |
| 조 번호 | 4조 | | |
| 담당 교수님 | 김순태 교수님 | | |
| 조원 | 이름 | 학과 | 학번 |
| 조원1 | 김민규 | 소프트웨어공학과 | 201513820 |
| 조원2 | 방세현 | 소프트웨어공학과 | 201510416 |
| 조원3 | 이정훈 | 소프트웨어공학과 | 201746148 |
| 조원4 | 최민성 | 소프트웨어공학과 | 201515237 |

**Ⅱ. 개선 아이디어**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 개선 아이디어 | 난이도점수 | 종합점수 |
| 1 | 사용자 개인정보 관리 | 2.5 | 2.5 |
| 2 | BGM 삽입 | 0.5 | 3 |
| 3 | 도서구매 사이트 연결 | 0.5 | 3.5 |
| 4 | 관심도서등록 | 2.5 | 6 |
| 5 | 좋아요, 싫어요 추가 (firebase) | 3 | 9 |
| 6 | 보고 있는 책이 전북대 중앙도서관에 있는지 검색 | 2.5 | 11.5 |

**Ⅲ. 코드설명**

0. 메인페이지

**public class** MainActivity **extends** Activity {  
 **private** MediaPlayer **mp**;  
 **private** Button **favoriteBookButton**;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
메인 페이지를 구현하는 액티비티입니다. 메인페이지에서는 로그인, 즐겨찾기를 확인 할 수 있도록 하는 역할을 맡고 있습니다.

<**activity  
 android:name=".activities.MainActivity"  
 android:label="@string/app\_name"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
</**activity**>

메니페스트에 선언된 액티비티 내용입니다.

1. 사용자 개인정보 관리

1. *회원가입(RegisterActivity.java)*

**public void** registerAct(String email, String password) {  
  
 ArrayList<String> Paramname = **new** ArrayList<String>();  
 Paramname.add(**"a"**);  
 Paramname.add(**"email"**);  
 Paramname.add(**"password"**);  
  
 ArrayList<String> Paramvalue = **new** ArrayList<String>();  
 Paramvalue.add(**"account\_sign\_up"**);  
 Paramvalue.add(email);  
 Paramvalue.add(password);  
   
  
  
  
 **new** AsyncHttpTask(RegisterActivity.**this**, **"https://unopenedbox.com/develop/bookamt/api.php"**, **mHandler**, Paramname,  
 Paramvalue, **null**, 1, 0);  
}

**public** Handler **mHandler** = **new** Handler() {  
 **public void** handleMessage(Message msg) {  
 *// IF Sucessfull no timeout* **if** (msg.**what** == 1) {  
  
 String result = msg.**obj**.toString();  
  
 Toast.*makeText*(RegisterActivity.**this**, **"회원가입 되었습니다."**, Toast.***LENGTH\_LONG***).show();  
 finish();  
  
 }  
  
  
 }  
 };  
}

editText에 가입할 이메일과 비밀번호를 입력하고 회원가입 버튼을 누르면, registerAct 메소드를 호출합니다. 매개변수는 이메일과 비밀번호를 가져옵니다.

Http 요청을 맡은 AsyncTask를 생성하여 POST 형식으로 회원가입 요청을 합니다.

핸들러로 요청 결과를 받아와 회원가입이 되었다는 메세지를 토스트를 통해 사용자에게 보여주고 회원가입 엑티비티를 닫습니다.

1. *로그인(LoginAcitivity.java)*

**public void** loginAct(String email, String password) {  
  
 ArrayList<String> Paramname = **new** ArrayList<String>();  
 Paramname.add(**"a"**);  
 Paramname.add(**"email"**);  
 Paramname.add(**"password"**);  
  
 ArrayList<String> Paramvalue = **new** ArrayList<String>();  
 Paramvalue.add(**"account\_auth"**);  
 Paramvalue.add(email);  
 Paramvalue.add(password);  
  
   
 **new** AsyncHttpTask(LoginActivity.**this**, **"https://unopenedbox.com/develop/bookamt/api.php"**, **mHandler**, Paramname,  
 Paramvalue, **null**, 1, 0);  
}

**public** Handler **mHandler** = **new** Handler() {  
 **public void** handleMessage(Message msg) {  
 *// IF Sucessfull no timeout* **if** (msg.**what** == 1) {  
  
 String result = msg.**obj**.toString();  
  
  
  
 Toast.*makeText*(LoginActivity.**this**, result, Toast.***LENGTH\_LONG***).show();  
  
  
 }  
  
  
 }  
};

editText에 로그인할 이메일과 비밀번호를 입력하고 로그인 버튼을 누르면, loginAct 메소드를 호출합니다. 매개변수는 이메일과 비밀번호를 가져옵니다.

Http 요청을 맡은 AsyncTask를 생성하여 POST 형식으로 로그인 요청을 합니다.

핸들러로 요청 결과를 받아와 회원가입이 되었다는 메세지를 토스트를 통해 로그인 결과를 사용자에게 보여주고 로그인 엑티비티를 닫습니다.

2. BGM삽입

**private** MediaPlayer **mp**;

**mp** = MediaPlayer.*create*(**this**, R.raw.***bgm1***);  
**mp**.setLooping(**true**);  
**mp**.start();*//음악재생*

위 코드는 MainActivity 클래스에 onCreate 메소드에서 정의되었습니다.

MediaPlayer라는 안드로이드에 기본적으로 탑재되어 있는 기능을 mp객체를 통해 구현 시켰습니다. 앱이 종료되면 음악도 같이 꺼집니다.

3. 도서구매 사이트 연결

*3-1. activity\_book\_detail.xml 파일*

<**Button  
 android:id="@+id/purchaseButton"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="도서구매"** />

BookDetailActivity에서 현재 보고있는 도서를 구매할 수 있도록 하기위해 우선 BookDetailActivity의 xml파일에 버튼을 추가시켜주었습니다.

*3-2. BookDetailActivity 클래스*

**purchaseButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent intent = new Intent(Intent.*ACTION\_VIEW*,Uri.*parse*("https://www.aladin.co.kr/m/msearch.aspx?SearchWord="+tvTitle.getText().toString()+"&SearchTarget=All"));  
 startActivity(intent);  
 }  
});**

사이트 연결 기능을 앞서 만들어둔 purchaseButton을 누르면 실행 시킬수 있도록 onClickListener를 연결시켜 주었습니다. 해당 버튼을 클릭시 인텐트를 통해 웹브라우져로 이동하여 작성해둔 도서구매사이트( 알라딘 ) 주소와 현재 보고있는 책의 제목을 합쳐 도서구매사이트내에서 해당 도서를 검색할 수 있도록 하였습니다.

4. 관심도서 담아두기

*4-1. BookDetailActivity 클래스*

**bookMarkButton**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **if**(**isBookMark**)  
 removeBookMark();  
 **else** addBookMark();  
 }  
 });  
}

BookDetailActivity에서 현재 보고있는 책을 관심도서 목록에 추가시키기위해 내부저장소에 파일로 해당 도서에 관한 정보를 담아두는 것으로 구현 방향을 설정했습니다. 우선 즐겨찾기 버튼을 누르면 해당 도서에 관한 정보가 없으면 저장하고 이미 저장된 도서면 즐겨찾기에서 제거시키는 버튼 리스너를 작성해 주었습니다.

**private void** checkBookMark(Book book){  
 **favoriteBook** = book;  
 **filename** = book.getTitle();  
 String path = getFilesDir().getAbsolutePath()+**"/"**+**filename**+**".txt"**;  
 **file** = **new** File(path);  
 FileInputStream fins;  
 **try** {  
 fins = **new** FileInputStream(path);  
 **isBookMark** = **true**;  
 fins.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 **isBookMark** = **false**;  
 }  
}

즐겨찾기를 하기에 앞서 책 상세정보에 들어왔을 때 해당도서가 즐겨찾기에 추가되어있는 도서인지를 확인 하기위해 checkBookMark() 메소드를 만들어 주었습니다. 해당 메소드는 현재 도서의 정보를 담고있는 Book객체를 받아와 객체의 title값이 가지고있는 책의 제목과 일치하는 즐겨찾기 파일이 존재하는가를 체크하는 메소드입니다. 즐겨찾기 파일은 도서제목으로 파일 이름을 지정하기에 도서 제목에 해당하는 파일이 있으면 즐겨찾기가 있으므로 isBookMark 값을 true로 변환하고 파일이 없으면 예외처리하여 isBookMark 값을 false로 설정하도록 해주었습니다.

**private void** addBookMark(){  
 String title = **favoriteBook**.getTitle();  
 String athor = **favoriteBook**.getAuthor();  
 String coverUrl = **favoriteBook**.getCoverUrl();  
 String openLibraryId = **favoriteBook**.getOpenLibraryId();  
 **try** {  
 FileOutputStream fos = **new** FileOutputStream(**file**);  
 BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(fos));  
 bw.write(title+**"@@"**+athor+**"@@"**+coverUrl+**"@@"**+openLibraryId);  
 Toast.*makeText*(getApplicationContext(),**"즐겨찾기 추가되었습니다"**,0).show();  
 bw.close();  
 fos.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 Toast.*makeText*(getApplicationContext(),**"즐겨찾기 오류"**,0).show();  
 }  
 **isBookMark** = **true**;  
}

**private void** removeBookMark(){  
 **file**.delete();  
 Toast.*makeText*(getApplicationContext(),**"즐겨찾기 삭제되었습니다"**,0).show();  
 **isBookMark** = **false**;  
}

addBookMark() 메소드와 removeBookMark() 메소드는 현재 보고있는 도서의 정보들을 파일로 저장하고 삭제하는 메소드입니다. addBookMark() 메소드에서는 도서 객체의 각 속성값을 get()메소드로 받아와서 이들을 “@@” 문자로 구분하여 파일에 일렬로 입력시키고 입력이 성공하면 즐겨찾기에 추가되었다는 메시지를 띄우게 하였습니다. 또한 즐겨찾기에 추가되었으므로 즐겨찾기 상태를 나타내는 isBookMark 변수 값을 true로 설정하였다. removeBookMark() 메소드는 반대로 즐겨찾기 제거시 파일 제거와 isBookMark false값 설정 역할을 하고있습니다.

*4-2. FavoriteBookActivity 클래스*

**public void** readFavoriteBookFile(){  
 String path = getFilesDir().getAbsolutePath();  
 File file = **new** File(path);  
 File[] filelist = file.listFiles();  
  
 String[] filename = **new** String[filelist.**length**];  
 String title = **new** String();  
 String author = **new** String();  
 String coverUrl = **new** String();  
 String openLibery = **new** String();  
 String[] tmp = **new** String[4];  
  
 **for**(**int** i=0; i<filelist.**length**; i++){  
 Book book = **new** Book();  
 filename[i] = filelist[i].getName();  
 **try** {  
 FileInputStream fis = **new** FileInputStream(file+**"/"**+filename[i]);  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(fis));  
  
 tmp = br.readLine().split(**"@@"**);  
 title = tmp[0];  
 author = tmp[1];  
 coverUrl = tmp[2];  
 openLibery = tmp[3];  
  
 br.close();  
 fis.close();  
  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 Toast.*makeText*(getApplicationContext(),**"불러오기 오류"**,0).show();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 book.setTitle(title);  
 book.setAuthor(author);  
 book.setCoverUrl(coverUrl);  
 book.setOpenLibraryId(openLibery);  
 Log.*d*(**"tag"**, **"result:"** + book.getTitle());  
 **books**.add(book);  
  
 Log.*d*(**"tag"**, **"result:"** + **books**.toString());  
 }  
  
}

**public void** adaptFavoriteBook(){  
 **bookAdapter**.clear();  
 readFavoriteBookFile();  
 **if**(**books**.isEmpty()){  
 Toast.*makeText*(getApplicationContext(),**"즐겨찾기된 책이 없습니다"**,0).show();  
 }  
 **for**(Book book : **books**){  
 **bookAdapter**.add(book);  
 }  
  
}

FavoriteBookActivity클래스는 BookDetailActivity에서 저장한 즐겨찾기 파일들을 listview를 통해 목록을 확인하고 각 도서의 상세페이지를 확인할 수있도록 해주는 클래스입니다. 즐겨찾기 파일을 불러오는 주기능을 하는 readFavoriteBookFile() 메소드는 내부저장소의 경로에 저장되어있는 즐겨찾기 도서 목록들을 배열에 담아주어 각 배열에 있는 파일들을 하나씩 읽어 이에들어있는 값들을 Book 객체에 매칭시키는 기능을 담당합니다. 또한 파일 저장시 구분 문자열로 설정한 “@@”를 통해 책의 저자와 커버Url주소 들을 읽어와 임시 Book 객체에 저장하고 속성값을 가지게된 Book 객체들을 ArrayList에 추가하도록 하였습니다. 이렇게 ArrayList에 즐겨찾기된 Book객체들을 저장하고 adaptFavoriteBook메소드를 통해 listview의 역할 수행에 필요한 adapter클래스로 파일들을 추가시켜주었습니다.

*4-3. AndroidManifest.xml 파일*

<**activity  
 android:name=".activities.FavoriteBookActivity"  
 android:label="@string/activity\_favorite\_book"**>  
</**activity**>

마지막으로 Manifest파일에 FavoriteBookActivity를 선언하여 해당 Activity를 사용할 수있도록 설정했습니다.

5. Firebase 이용한 책 (좋아요/싫어요) 기능 추가

DatabaseReference **myRef**;

**private** FirebaseDatabase **firebaseDatabase** = FirebaseDatabase.*getInstance*();

**private** DatabaseReference **databaseReference** = **firebaseDatabase**.getReference();

데이터베이스에서 데이터를 읽거나 쓰려면 DatabaseReference의 인스턴스가 필요합니다

**myRef**= FirebaseDatabase.*getInstance*().getReference().child(**"msg"**);

**myRef**=FirebaseDatabase.*getInstance*().getReference().child(**"dislike"**);

데이터베이스 안에 msg또는 dislike라는 child의 인스턴스값을 가져오는 코드입니다.

**myRef**.addListenerForSingleValueEvent(**new** ValueEventListener() {

@Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {

Long numberOfLike = (Long) dataSnapshot.getValue();

**likeText**.setText(String.*valueOf*(numberOfLike));  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {

}  
});

**myRef**.addListenerForSingleValueEvent(**new** ValueEventListener() {

@Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {

Long numberOfdisLike = (Long) dataSnapshot.getValue();

**dislikeText**.setText(String.*valueOf*(numberOfdisLike));  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {

}  
});

AddListenerForSingleValueEvent 메소드를 사용하면 한번 호출된후 다시 호출되지않습니다 이 방법은 한 번 로드된 후 자주 변경되지 않거나 능동적으로 수신 대기할 필요가 없는 데이터에 유용합니다. 예를 들어 위 예시의 북서치 앱에서는 사용자가 텍스트를 수정하기 시작할때 이메소드로 사용자에게 보여지는 dilike,likeText의 보여지는 값을 변경해줍니다

**likebutton**.setOnClickListener(**new** View.OnClickListener() {

@Override  
 **public void** onClick(View view) {

**myRef** = FirebaseDatabase.*getInstance*().getReference().child(**"msg"**);

**myRef**.addListenerForSingleValueEvent(**new** ValueEventListener() {

@Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {

Long numberOfLike = (Long) dataSnapshot.getValue();

numberOfLike++;  
 **myRef**.setValue(numberOfLike);  
 **likeText**.setText(String.*valueOf*(numberOfLike));  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {

}  
 });  
 }  
});  
  
  
**dislikebutton**.setOnClickListener(**new** View.OnClickListener() {

@Override  
 **public void** onClick(View view) {

**myRef** = FirebaseDatabase.*getInstance*().getReference().child(**"dislike"**);

**myRef**.addListenerForSingleValueEvent(**new** ValueEventListener() {

@Override  
 **public void** onDataChange(DataSnapshot dataSnapshot) {

Long numberOfdisLike = (Long) dataSnapshot.getValue();

numberOfdisLike++;  
 **myRef**.setValue(numberOfdisLike);  
 **dislikeText**.setText(String.*valueOf*(numberOfdisLike));  
 }  
  
 @Override  
 **public void** onCancelled(DatabaseError databaseError) {

}  
 });  
 }  
});

Setonclicklistener를 통해 like button이 클릭될때 마다 인스턴스로 값을 받아오고

또 onDatachange { onDataChange() 메소드는 하위 위치에서의 변경을 포함하여 지정된 데이터베이스 참조가 변경될 때마다 호출됩니다}

메소드를 이용해. Numberoflike 의 값이 이벤트 발생시점에 데이터베이스에서

지정된 위치에 있던 데이터를 포함해 DataSnapshot을 수신합니다.

그리고 수신하고난후 numberOfdisLike을 1증가시켜줍니다

그후에 dislikeText의 값을 1증가된 numberOfdisLIke 으로 가져와줍니다

6. 전북대 도서관에 대출 가능한지 여부 확인(BookDetailActivity.java)

**public void** requestLibraryBook(Context cx, String title) {

**new** AsyncHttpTask(cx, **"https://dl.jbnu.ac.kr/eds/brief/integrationResult?x=0&y=0&st=KWRD&si=TOTAL&lmtst=OR&lmt0=TOTAL&q="** + title, **mHandler**, **null**,

**null**, **null**, 1, 0);

}

**public** Handler **mHandler** = **new** Handler() {  
 **public void** handleMessage(Message msg) {  
 *// IF Sucessfull no timeout* **if** (msg.**what** == 1) {  
   
**try** {  
 Document doc = Jsoup.*parseBodyFragment*(msg.**obj**.toString());  
 Element body = doc.body();  
 Element lib = doc.getElementsByClass(**"briefDeFont"**).first();  
  
  
 **tvDescription**.setText(**"전북대 도서관\n"** + lib.text());  
} **catch** (Exception e){  
 **tvDescription**.setText(**"전북대 도서관\n"** + **"전북대 도서관에 없는 책"**);  
}  
 }  
  
  
 }  
 };

BookDetailActivity에 들어가 책을 로드하게 되면 전북대 도서관에도 책을 빌릴 수 있는지 확인하기 위해 requestLibraryBook 메소드를 호출하게 됩니다. 매개변수는 현재 Activity의 Context와 책 제목입니다.

검색할 책 이름을 전북대 도서관 검색 결과 URL 쿼리 부분에다 추가한 후 HTTP 요청을 진행합니다.

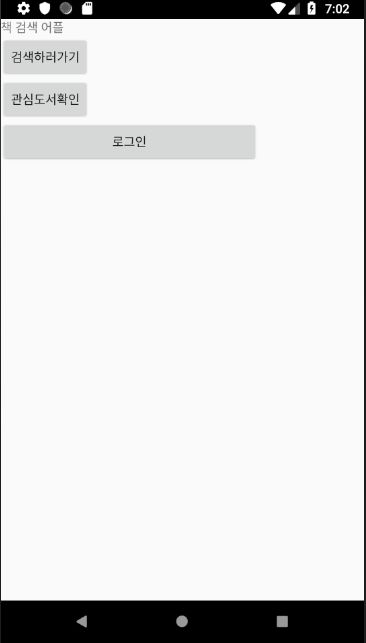
핸들러로 요청 결과를 처리해줍니다. 전북대 도서관에서 받은 html 결과 값을 JSoup을 통해 파싱해줍니다.

Html Elements 중에서 Class 이름이 briefDeFont인 것을 가져옵니다. 그 중에서 가장 높은 우선순위로 정렬된 첫번째 항목을 가져옵니다.

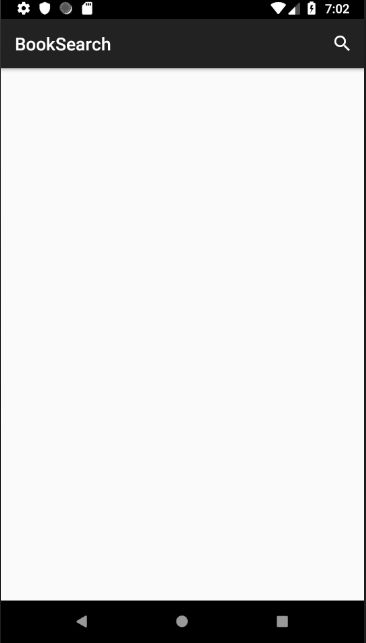
뽑아낸 값을 TextView인 tvDescription에 setText해줍니다. 없는 책 검색 결과라 오류가 발생한다면 예외 처리로 전북대 도서관에 없는 책이라 표시해줍니다.

**Ⅳ. 스크린샷**

1. 메인페이지



1. 검색하기 버튼 누른 후



1. 검색 후 항목 선택했을 때

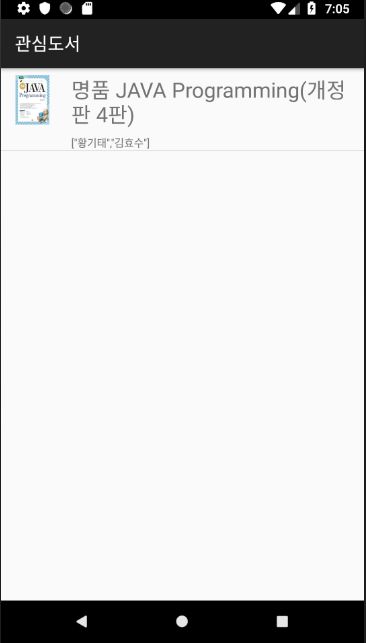




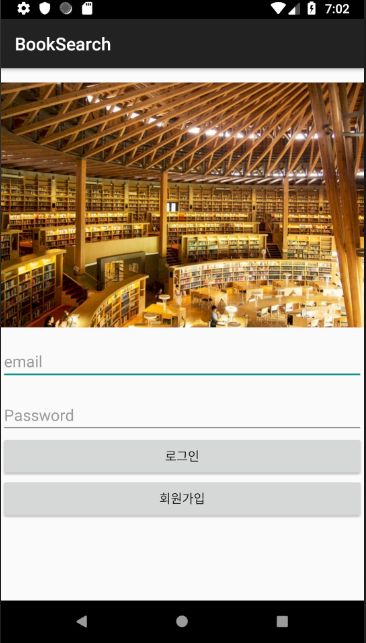
3-1. 좋아요 싫어요 작동 확인



1. 메인페이지에서 즐겨찾기 버튼 눌렀을 때



1. 로그인 화면



**Ⅴ. 향후 개선사항**

1. 책 정보를 가져올 때 Pagination기법 활용
2. 로그인한 계정과 즐겨찾기 정보를 연동해서 활용
3. 음성 검색 기능 추가
4. 최근 검색 기록 보여주기 기능 추가