**TMDB Database Search**

CS483W Team Project

Trevor Chipley, Michael Costa, Alex Good

Spring 2013

The Pennsylvania State University

**Table of Contents**

1. Abstract
2. Requirements
   1. Background
   2. Essential Solution
   3. Environment
   4. Implementation Outline
   5. Operational Assumptions
3. Technical Specification
   1. Development Tools and Languages
   2. Design
      1. Searching
      2. Settings
      3. Diagrams
4. Standards
5. Test Plan
6. QA Plan
7. Deployment Plan
8. Maintenance Plan
9. Team Organization
10. Schedule

**1 Abstract**

The web movie database search Android application is designed to provide the user unique options when searching for a movie to watch. This application provides normal search criteria such as genre, director, and rating, however it provides a unique weighted actor search. The actor search provides the ability for the user to rank actors that the user wants to see in the movie and actors the user does not want to see in the movie. The application considers the user rankings of favorable and unfavorable actors, along with a combination of other possible search options, and returns movie results that best fit the search criteria. The TMDb on-line movie database is queried and searched in a timely manner. The use of an on-line movie database also provides automatic updates on the current list of movies.

In order to make this application widely usable, it is important that the graphical user interface is clean, simple, and intuitive. Since the application is geared toward the general public, the interface is self-explanatory but also provides easily accessible instructions upon request. The goal is to hide the details and make the application as user-friendly as possible.

Our goal is to develop a unique, easy, and fast Android application that will search a movie database in a way that the general public would prefer this application over the existing movie database search applications on the market. We plan to thoroughly test the application within our development group, as well as with potential users, to provide a smoother experience for the user and reduce any bugs that may be associated with the application.

**2 Requirements**

**2.1 Background**

This project is not being developed for a company, and the system requirements are being created by us. We will be using the movie data, actor, and image API given by themoviedb.org – a free and community based movie database[1].

**2.2 Essential Solution**

The core functionality of this system can be broken down into two parts. Cast search and movie search. The cast search consists of two sub searches for choosing favorable and unfavorable actors, and returning results based on your preferences.

**Case: Movie Search**

Once ‘Movie’ is selected from the main page spinner, and a movie is entered into the search bar, the user will be taken to a results page listing movies relevant to their search.

**Case: Ranking Favorable Actors**

Users will be able to begin typing the names of up to five actors in different search fields. They will only be able to search for actors that currently exist in the TMDb system. The system will begin to suggest names in a drop down menu as the user begins to type names. For example, if the user begins typing ‘Will’, the system will suggest ‘Will Ferrell’, ‘Will Smith’, ‘William Shatner’, etc. Once the user finishes selecting their favorite actors, a submit button will trigger an algorithm that creates search queries based on the users selections.

**Case: Ranking Unfavorable Actors**

The interface and behavior of this system will be nearly identical to the system for ranking favorable actors. The only thing this system will implement differently is that it the actors entered by the user will be entered into the where clause of the query with the not equal operator.

**Case: Displaying the Results**

Once the user has selected their favorite and least favorite actors, the TMDb will be queried appropriately and will return results in the form of movie titles. The movie titles returned by the query will link to the movie’s page on the TMDb website, so the user can find out more information on the movie.

**2.3 Environment**

The requirements for using this product are minimal:

* A device running the Android OS from SDK version 11 (Honeycomb[4]) to SDK version 17 (Jelly Bean[4]).
* An internet connection with relevant network permissions granted.

**2.4 Implementation Outline**

This project will primarily be developed in Java with JSON.

**2.5 Operational Assumptions**

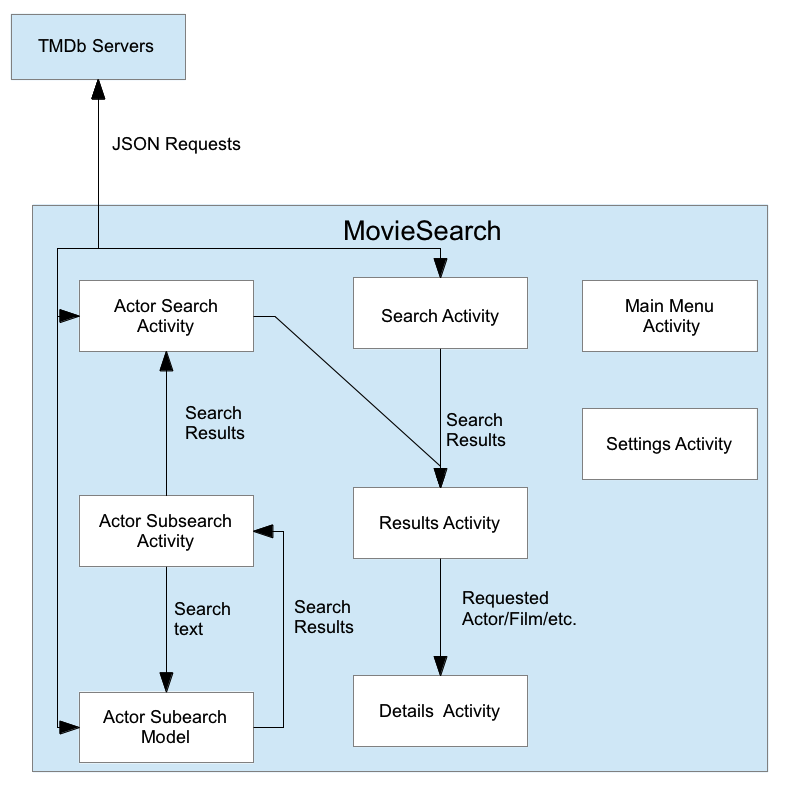
It is assumed that the TMDb servers are operational when the application is in use. Included in this is that the servers are assumed to return good and accurate responses to queries.

**3 Technical Specifications**

**3.1 Development Tools and Languages**

We will be using Java with JSON for querying external databases, XML for generating the UI, and MySQL for storing persistent user data in a local database. The development will be done in the Eclipse IDE with Android SDK.

**3.2 Design**

  
**Figure 1: Modular Decomposition Diagram**

**3.2.1 Searching**

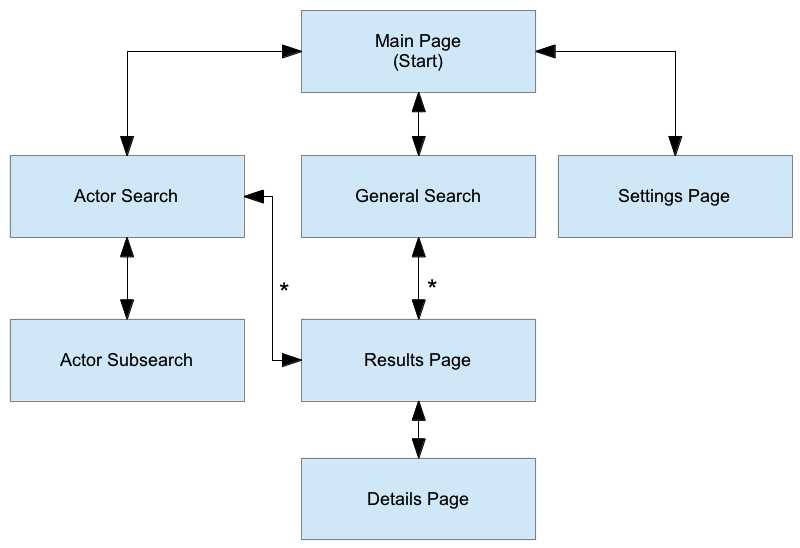
The information in this section is relevant to any Activity in Figure 1 that exchanges JSON requests with the TMDb servers.

Searches are done by sending a JSON request to TMDb for a user-provided keyword in a selected category (Films, People or Genres). Searches will be limited to 60 items per search (three pages), to reduce the potential load of our application on TMDb's servers. There is an additional concern about TMDb limiting API requests to ten requests every ten seconds, though this is not likely to come up often, and potential memory issues with very broad searches (the search “a” in People, for example). Search results are presented in the order they are received, as only one page (20 items) is displayed at a time and, if sorting differently than TMDb does, adding additional pages could result in new items appearing above old items.

**3.2.2 Settings**

The user has the ability to alter several settings. The user can choose to automatically exclude some genres of film, select what languages the search can return, to display upcoming and unreleased films, and to allow adult films to be allowed.

**3.3 User Interface**



**Figure 2: Page Navigation Diagram**

\*: See section 3.3.6

**3.3.1 Main Page**

[Screenshot]

The Main Page is a launching point to the other pages of this application. From the main page, tapping one of the available buttons takes the user to the associated page: Settings (section 3.3.2), Actor Search (section 3.3.3), or General Search (section 3.3.5).

**3.3.2 Settings Page**

[Screenshots]

The Settings Page is where the user can customize their preferences, as detailed in section 3.2.2. The first screenshot is what the user sees when first accessing this page. Some settings may bring up a sub-page with a checklist, shown in the second screenshot. Pressing the physical back button on the device will return the user to the Main Page (section 3.3.1)

**3.3.3 Actor Search**

[Screenshot]

The Actor Search page is where the user can search for movies featuring a specific set of individuals (which must always be at least one person) and not featuring some other set of individuals. Tapping either button labeled “Add New Actor” brings up the Actor Subsearch (section 3.3.4), and when that page returns, the selected actor is added to the associated list. If no actor was selected, no changes will be made. Tapping the button labeled “Search” will bring up the Results Page (section 3.3.6). Pressing the physical back button on the device will return the user to the Main Page (section 3.3.1).

**3.3.4 Actor Subsearch**

[Screenshot]

The Actor Subsearch searches for actors whose name matches the string entered by the user in the text box at the top of the screen. The results are displayed as text only, as shown in the screenshot. The user can scroll up and down this list. Tapping on a name brings up a dialog box that asks for confirmation that the tapped actor is the actor the user wishes to select. If the user selects no, the dialog box disappears and the user may select again. If the user selects yes, they are returned to the Actor Search (section 3.3.3), and the selected actor is passed back to the Actor Search (section 3.3.3). Pressing the physical back button on the device will return the user to the Actor Search (section 3.3.3) with no changes to that page.

**3.3.5 General Search**

[Screenshot]

The General Search is unimplemented at this time. The General Search allows the user to search for films using the name of the film as a search key. The user enters this search in the text box at the top of the screen. When the user is done, they start the search and are taken to the Results Page (3.3.6). Pressing the physical back button on the device will return the user to the Main Page (section 3.3.1).

**3.3.6 Results Page**

[Screenshot]

The Results Page displays a list of films acquired from a search executed by the user. The results are displayed as text only, with the name of the film and release date displayed. Tapping any item on the list brings up the Details Page (section 3.3.7) with that films data filled in. Pressing the back button will return the user to either the General Search (3.3.5) or the Actor Search (3.3.3), depending on which page they reached the Results Page from. Otherwise, from the user's perspective, this page has the same behavior regardless of which page was used to reach this one.

**3.3.7 Details Page**

[Screenshot]

The Details Page displays the details of a selected film. The information displayed consists of the movie's poster, title, release date, rating, tagline, running time (in minutes), genres, an overview (usually a brief plot summary), and a cast list. This information is received directly from TMDb. Pressing the physical back button on the device will return the user to the Results Page (section 3.3.6).

**4 Standards**

The page activities were written in Java and should follow the Oracle Code Conventions [2]

for the Java Programming Language. The code segments written in MySQL, JSON, and XML should follow the W3C standards [3] published for their respective language.

Java Example

/\*\*

\* This function includes…

\*/

void codeExample() {

if (condition) {

int variable = 0; //this is a comment

}

}

MySQL Example

SELECT \* FROM Actors WHERE Name=’Tom Cruise’  
  
  
JSON Example

See TMDb API

XML Example  
  
<Example>  
 <name>Alex</name>

<description>A cool guy</description>  
</Example>

**5 Test Plan**

Acceptance Testing

### Scenario: General Searches

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | View Default Page | Launch Application | The Main page (see 3.3.1) should display |  |
| 2 | View Details | Click on any listed film on the Main page | The details page (see 3.3.7) should appear with details about the selected film. |  |
| 3 | Search Movies | Select “Movies” from the spinner on the main page, and search for “The Hangover” | The Results page (see 3.3.6) should appear with “The Hangover” and other related films listed. |  |
| 4 | Search Actor | Select “Cast” from the spinner on the main page, and search for “Tom Cruise”. “Tom Cruise” should appear in the subsearch list. Click on his name. | The Results page (see 3.3.6) should appear with a list of movies where Tom Cruise is a member of the cast. |  |
| 5 | View Movie Details | From the results page accessed through test #4, select the film “Top Gun”. | The details page (see 3.3.7) should appear with details for the film “Top Gun”. |  |
| 6 | Return to Results | Press the back button on the device | The Results page accessed through test #4 should appear, with the same results as before. |  |

**Unit Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | ActorData data constructor | Call the 5-variable constructor with the arguments adult = false, name = Tom Cruise, ID = 500, popularity = 1.96, and profile = profilepicture.jpeg | Retrieve the adult, name, id, popularity, and profile variables using their appropriate accessor methods and test that all of the following relations are equal: isAdult() = false, getName() = Tom Cruise, getId() = 500, getPopularity() = 1.96, getProfile() = profilepicture.jpeg |  |
| 2 | ActorData write parcel | Given an initialized ActorData object, using the values from the ActorData data constructor unit test, call writeToParcel. | The resulting parcel “result” has the following relations when performed in the following order: result.ReadByte() = 0, result.readString() = Tom Cruise, result.readInt() = 500, result.readDouble = 1.96, result.readString() = profilepicture.jpeg |  |
| 3 | ActorData parcel constructor | Create a parcel “input” and perform the following operations on it: input.writeByte(0), input.writeString(Tom Cruise), input.writeInt(500), input.writeDouble(1.96), input.writeString(profilepicture.jpeg),  Pass input into the parcel constructor of ActorData | The resulting ActorData object should have the following relationships:  isAdult() = false, getName() = Tom Cruise, getId() = 500, getPopularity() = 1.96, getProfile() = profilepicture.jpeg |  |
| 4 | ActorSearch life cycle initialization (starting) | Using instrumentation, call the onCreate method of ActorSearch | The resulting ActorSearch data should have the following properties: the views with ids execute\_search\_button, search\_add\_actor, exclude\_add\_actor, actor\_search\_list, actor\_exclude\_list should be initialized and should be visible within the root view. actor\_search\_list and actor\_exclude\_list should both have no elements. |  |
| 5 | ActorSearch life cycle initialization (from previous instance) | Using instrumentation, call the onCreate method of ActorSearch, add “Tom Cruise” to the actor\_search\_list, call callActivityOnSaveInstanceState(), call onDestroy(), call onCreate() | The resulting ActorSearch activity from the 2nd call to onCreate should have all the ids from above initialized and visible within the root view. actor\_search\_list should have “Tom Cruise” in it and actor\_exclude\_list should be empty. |  |

**Integration Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | Retrieve detailed information (DetailModel-TMDB) | Call DetailModel’s static method synchronousDetailPrimaryRetrieve with movieId = 155 | DetailModel should return a DetailData object with the following relations: adult = false, genres != null, id = 155, overview != null, popularity >= 0, posterPath != null, releaseDate = “2008-07-18”, runtime = 152, tagline != null, voteAverage >= 0 |  |

**Systems Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | ActorSubsearch fetch actor | Using instrumentation, call onCreate of ActorSearch, fetch the view with id search\_add\_actor, call that view’s onClick method, populate the view with id “listView” with the name “Tom Cruise”, call the onClick method of the first item in the listView, click the positive button in the resulting dialog (with instrumentation) | The view actor\_search\_list in ActorSearchActivity is initialized and has a single item with text “Tom Cruise”. |  |

# QA Plan

Throughout the development phase of the project, code quality with be reviewed by members of the team to ensure it corresponds to the specified standards. Acceptance testing will ensure that the application meets the specified requirements.

# Deployment Plan

The application will be uploaded and distributed for free on Android devices through the Google Play store. Once we register as a Google Play Developer at http://play.google.com/apps/publish, we must upload our APK file, two in-app screenshots, and listing details including title, language, application description, recent changes, promo text, application type and category. Within an hour after upload, our application should be searchable and downloadable from the Play Store on all supported android devices.

# Maintenance Plan

Errors in the software will be handled as they are found. Users will have to report errors they encounter so that we can fix them. Updates will be distributed to users using the Google Play store.

# Team Organization

# Schedule







  
**11 Project Cost**

The team consists of Penn State University students, using resources provided the university. There is a one-time $25 fee to deploy applications to the Google Play Store, and that is the only cost associated with this project.

**12 Sources**

**[1] –** "The Movie Database." *The Movie Database*. N.p., n.d. Web. 02 Apr. 2013. <http://www.themoviedb.org/>.

**[2] –** "Code Conventions for the Java Programming Language: Contents." *Code Conventions for the Java Programming Language: Contents*. Sun Microsystems, 20 Apr. 1999. Web. 04 Apr. 2013. <http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html>.

**[3] –** "W3C" *Standards*. W3C, n.d. Web. 04 Apr. 2013. <http://www.w3.org/standards/>.

**[4]** – “<uses-sdk>” *Android Developer API Guides*. Google, Web 18 Apr. 2013. <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html>