**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

**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. There are existing APIs for use with TMDb, which require that our project not be for commercial use.

**2.2 Essential Solution**

The core functionality of this system can be broken down into three parts: the user ranking favorable actors, the user ranking unfavorable actors, and displaying the results of the query to the user.

**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, preferably a recent version, with an internet connection.

**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.

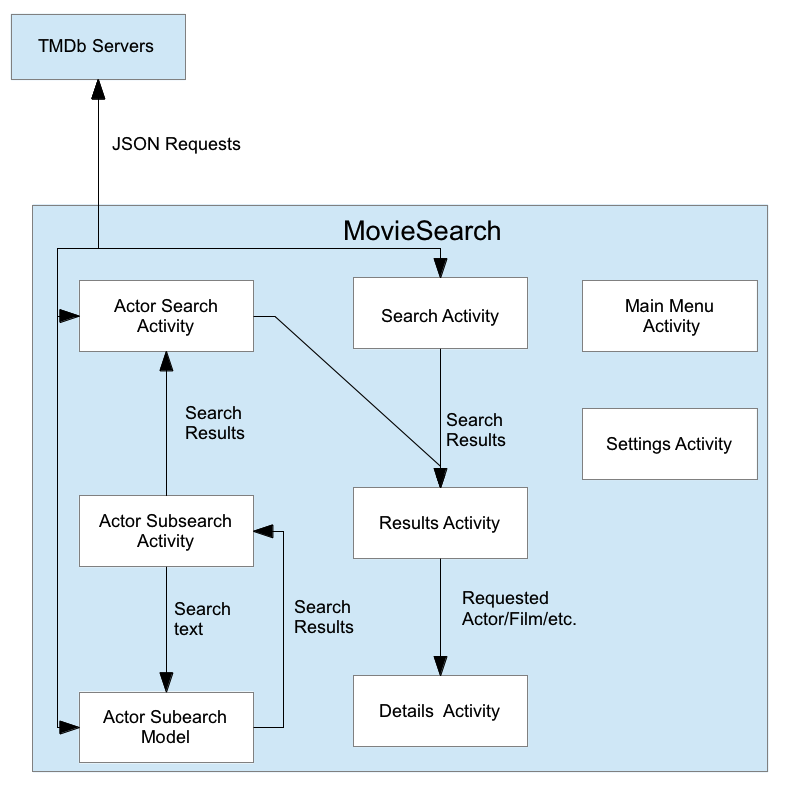
**3 Technical Specification**

**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**

****

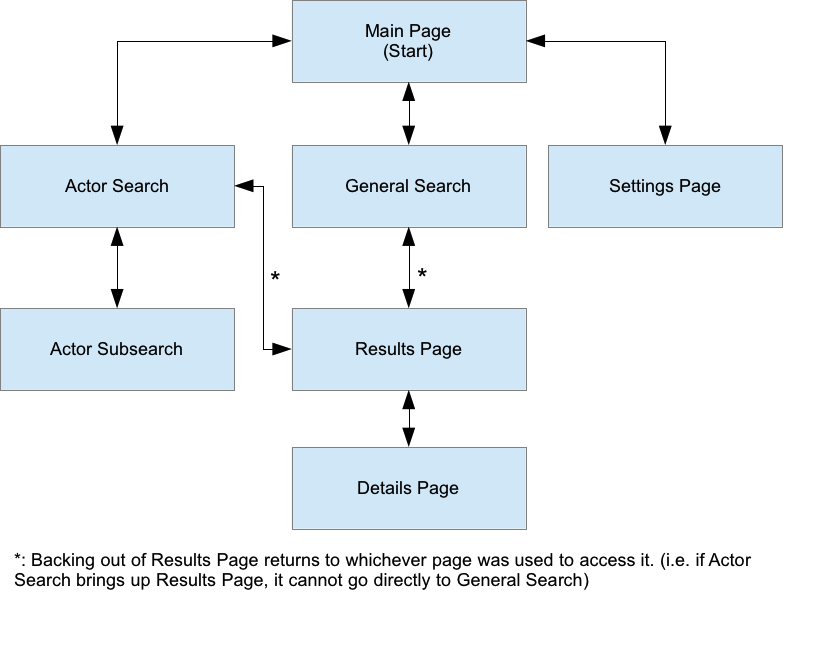
**3.2.1 Searching**

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.

**Figure 2: Page Navigation Diagram**



**3.3 User Interface**

**4 Standards**

The page activities were written in Java and should follow the Oracle Code Conventions for the Java Programming Language. The code segments written in MySQL, JSON, and XML should follow the W3C standards published for their respective language.

**5 Test Plan**

### Scenario: Actor Search

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | View Default Page | Launch Application | activity\_main\_menu.xml should display |  |
| 2 | Go to Actor Search Page | Click the “actor search” button from activity\_main\_menu.xml | activity\_actor\_search.xml should display |  |
| 3 | Go To Include Actors Page | Click the first “add new actor” button from activity\_actor\_search.xml | activity\_actor\_subsearch.xml should launch |  |
| 4 | Add Actor To Subsearch List | Type actor name into search bar of activity\_actor\_subsearch.xml | Actor names should appear in subsearch list |  |
| 5 | Confirm Actor | Click actor from subsearch list | A window should display asking to confirm your selection |  |
| 6 | Go To Exclude Actors Page | Click the “add new actor” button under “Exclude” from activity\_actor\_search.xml | activity\_actor\_subsearch.xml should launch |  |
| 7 | Passing NULL to subsearch model | JSONObject synchronousActorSearch receives NULL nameSubstring | Omits page from query |  |
| 8 |  |  |  |  |

### Scenario: Actor Search

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | View Default Page | Launch Application | activity\_main\_menu.xml should display |  |
| 2 | Go to Actor Search Page | Click the “actor search” button from activity\_main\_menu.xml | activity\_actor\_search.xml should display |  |
| 3 | Go To Include Actors Page | Click the first “add new actor” button from activity\_actor\_search.xml | activity\_actor\_subsearch.xml should launch |  |
| 4 | Add Actor To Subsearch List | Type actor name into search bar of activity\_actor\_subsearch.xml | Actor names should appear in subsearch list |  |
| 5 | Confirm Actor | Click actor from subsearch list | A window should display asking to confirm your selection |  |
| 6 | Go To Exclude Actors Page | Click the “add new actor” button under “Exclude” from activity\_actor\_search.xml | activity\_actor\_subsearch.xml should launch |  |
| 7 | Passing NULL to subsearch model | JSONObject synchronousActorSearch receives NULL nameSubstring | Omits page from query |  |
| 8 |  |  |  |  |