**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
   3. User Interface
4. Standards
5. Test Plan
6. QA Plan
7. Deployment Plan
8. Maintenance Plan
9. Team Organization
10. Schedule

**1 Abstract**

Movie Search is designed to provide the user with a fast and simple way to search for a movie to watch. The application makes use of the TMDb online movie databases extensive collection of films, and translates it into a user friendly Android application. 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.

**Case: Movie Search**

Once ‘Movie’ is selected from the Landing 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: Cast Search**

Once ‘Cast’ is selected from the Landing Page spinner, and an actor name is entered into the search bar, the user will be taken to a subsearch menu. Once the user selects the appropriate actor, the user will be taken to a results page listing movies relevant to their search.

**Case: Displaying the Results**

Once the user has completed their search, TMDb will be queried appropriately and the application will display a list of relevant movies. If a movie is selected, a page with detailed information about the movie will display.

**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**

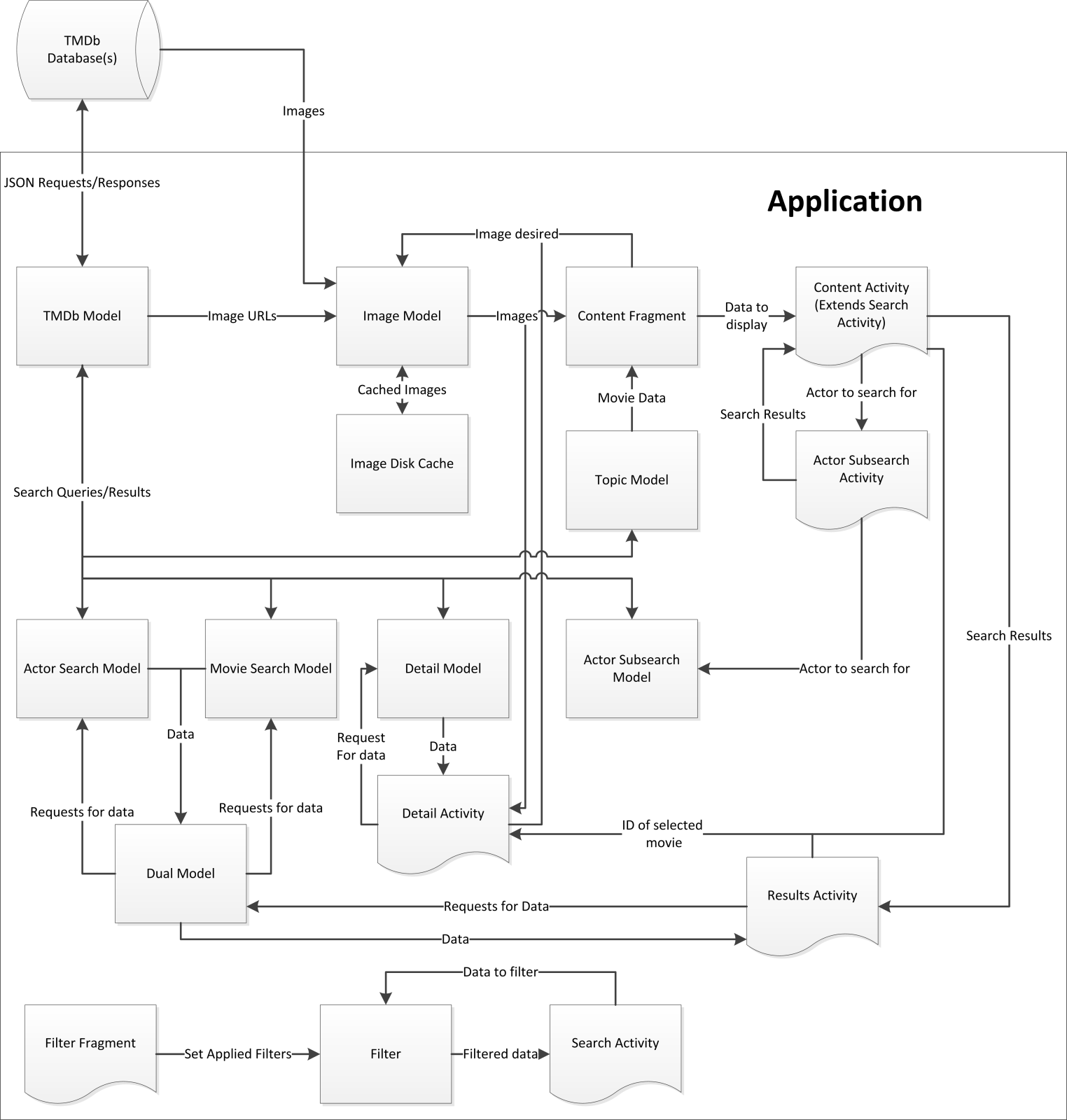
It is assumed that the TMDb servers are operational when the application is in use. Included in this is that the servers are assumed to return good and accurate responses to queries.

**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**

Note 1: Square modules involve no direct interaction with the user, while objects with a curved base do.

Note 2: Data is transferred using several data structures not shown on the above diagram.

**3.2.1 Description of Significant Modules**

**3.2.1.1 TMDb Model**

The TMDb Model is the only module that interfaces with the TMDb databases using JSON. All other modules that wish to perform searches or get data from their databases make requests through this module. The results to queries are returned as JSON objects, and the module making the request must interpret this result itself.

**3.2.1.2 Image Model**

The Image Model handles images for other modules. Modules pass in the URL of the image (relative to TMDb’s base URL) that they wish to download, and the Image Model fetches this image, caches it, and returns it.

**3.2.1.3 Search Modules**

This is a category of modules consisting of the following: Actor Search Model, Movie Search Model, and Actor Subsearch Model. These modules perform searches of TMDb’s databases, using the TMDb Model. The Movie Search Model searches for films, as the name implies. The Actor Search and Actor Subsearch sound similar, but do different things. The Actor Search Model gets all information about actors, while the Actor Subsearch Model only searches for a list of names.

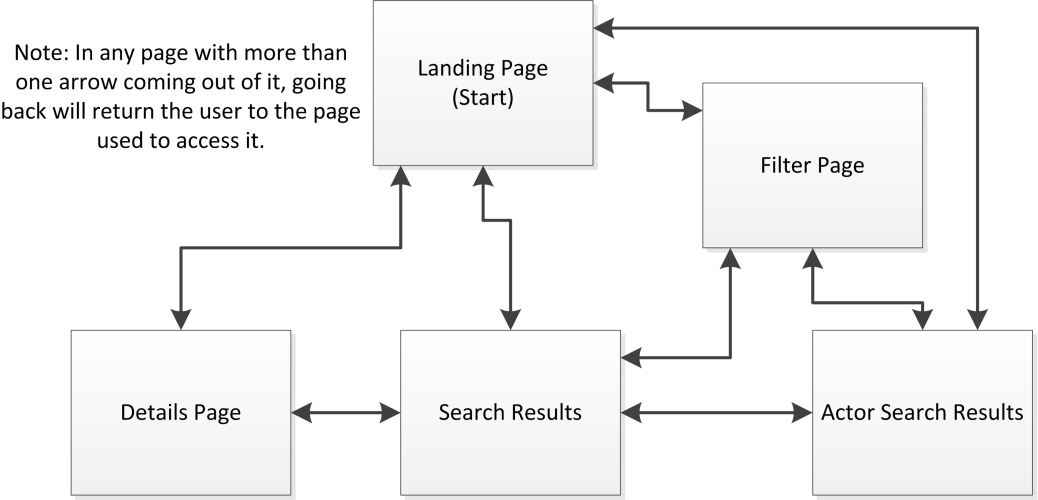
**3.2.1.4 Filter**

The Filter module stores the user’s current filter settings and filters data. The Filter Fragment (aka Filter Page, section 3.3.4) sets the filters to use. This object is shared through both the Search Activity module (and, by extension, Content Activity module) and the Results Activity. Only the former interaction is shown as a sample, with the other interaction being identical.

**3.2.1.5 Content Activity**

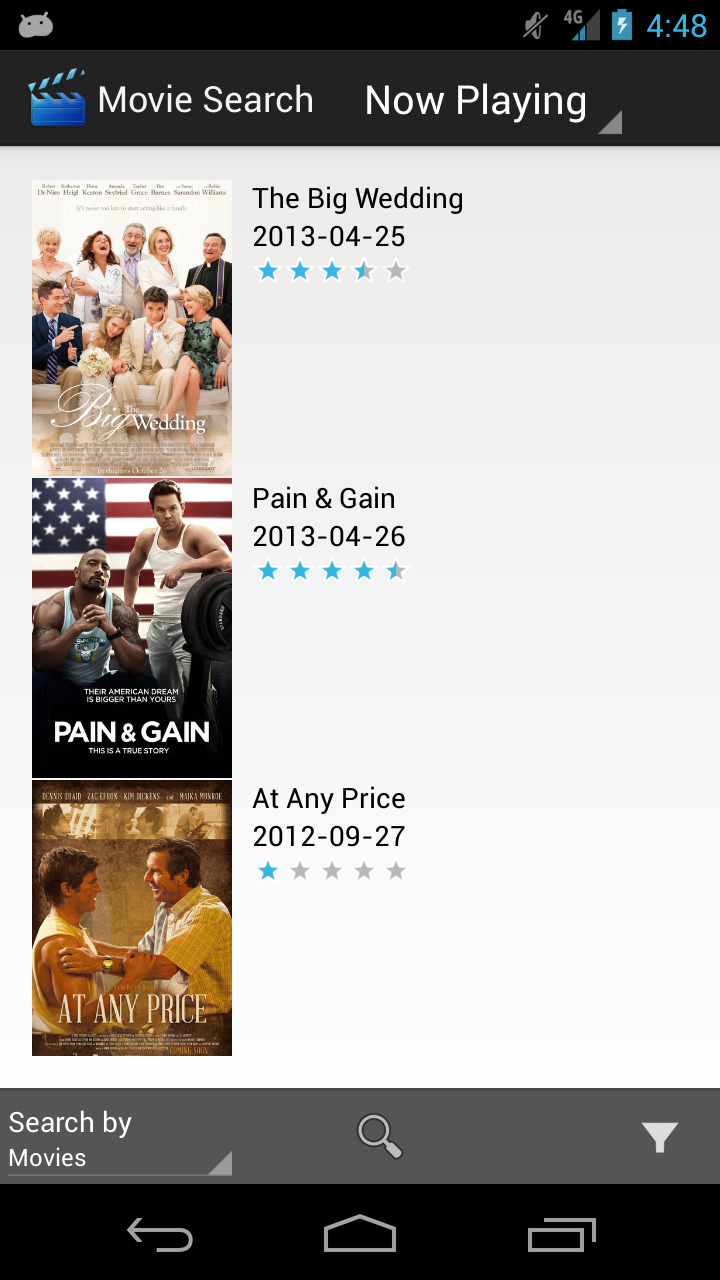
This is the internal name for the Landing Page (Section 3.3.1).

**3.3 User Interface**



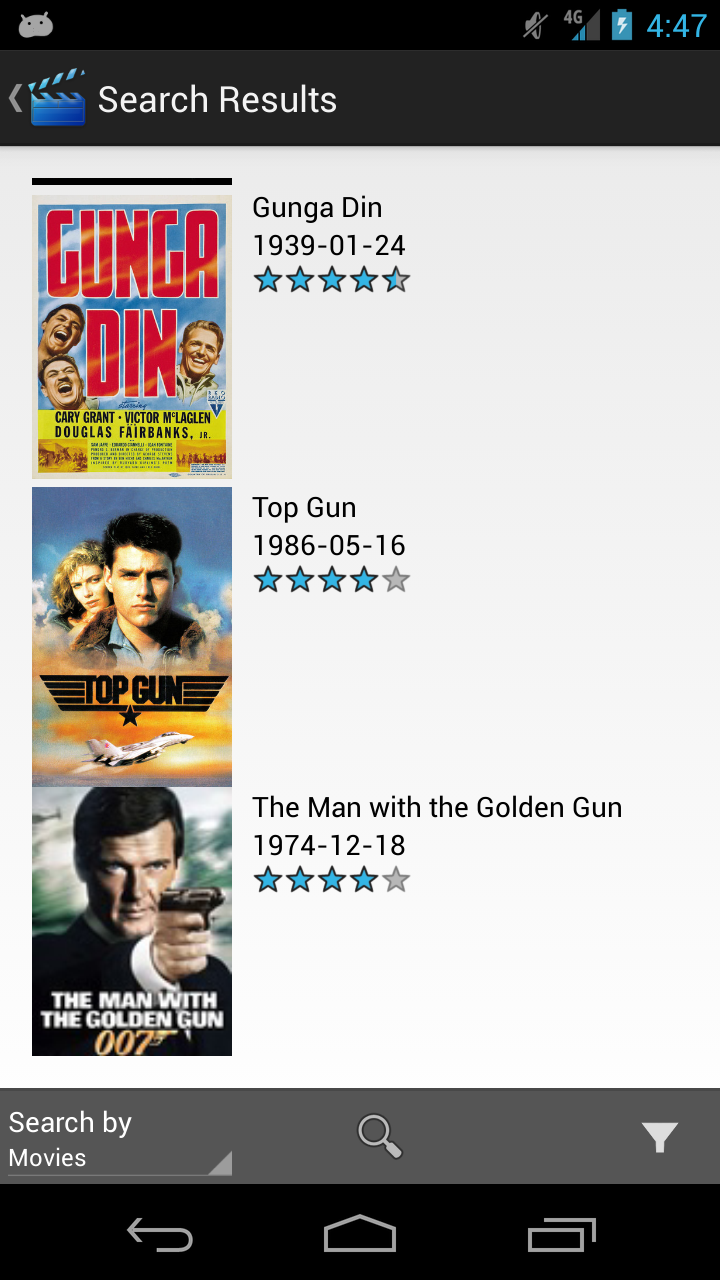
**Figure 2: Page Navigation Diagram**

**3.3.1 Landing Page**



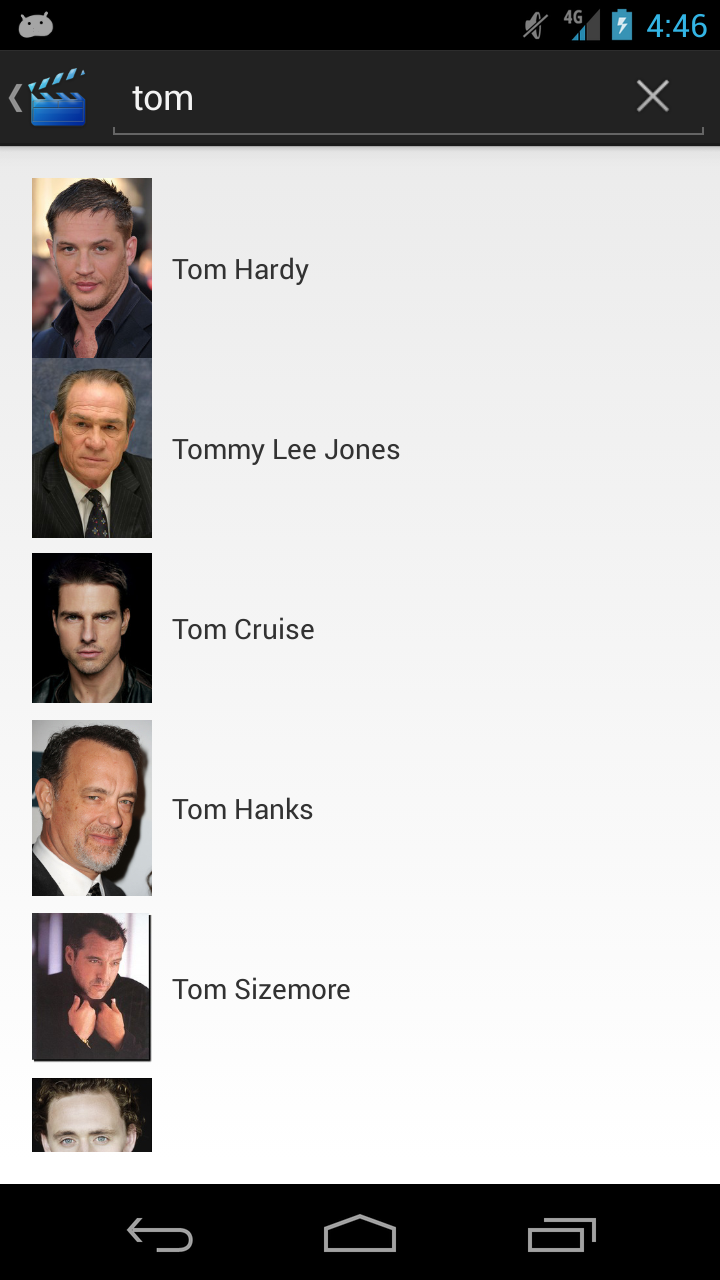
The Landing Page is the starting page of this application. By default, it displays a list of films that are currently in theaters. Also viewable on this page is a list of highly rated films, popular films and upcoming films, which can be switched between using a drop-down menu at the top of the page. These lists are provided by TMDb and are not altered. Also on this page is a button to access the Filter Page (Section 3.3.4) on the bottom left, a drop-down menu to switch between searching for movies and actors on the bottom right, and a button to initiate a search in the bottom center. This page can take the user to the Details Page (Section 3.3.5) if a film is tapped, Filter Page (Section 3.3.4) if the filter button is pressed, the Actor Search Results (Section 3.3.3) if an actor is searched for, or Search Results (Section 3.3.2).

**3.3.2 Search Results**



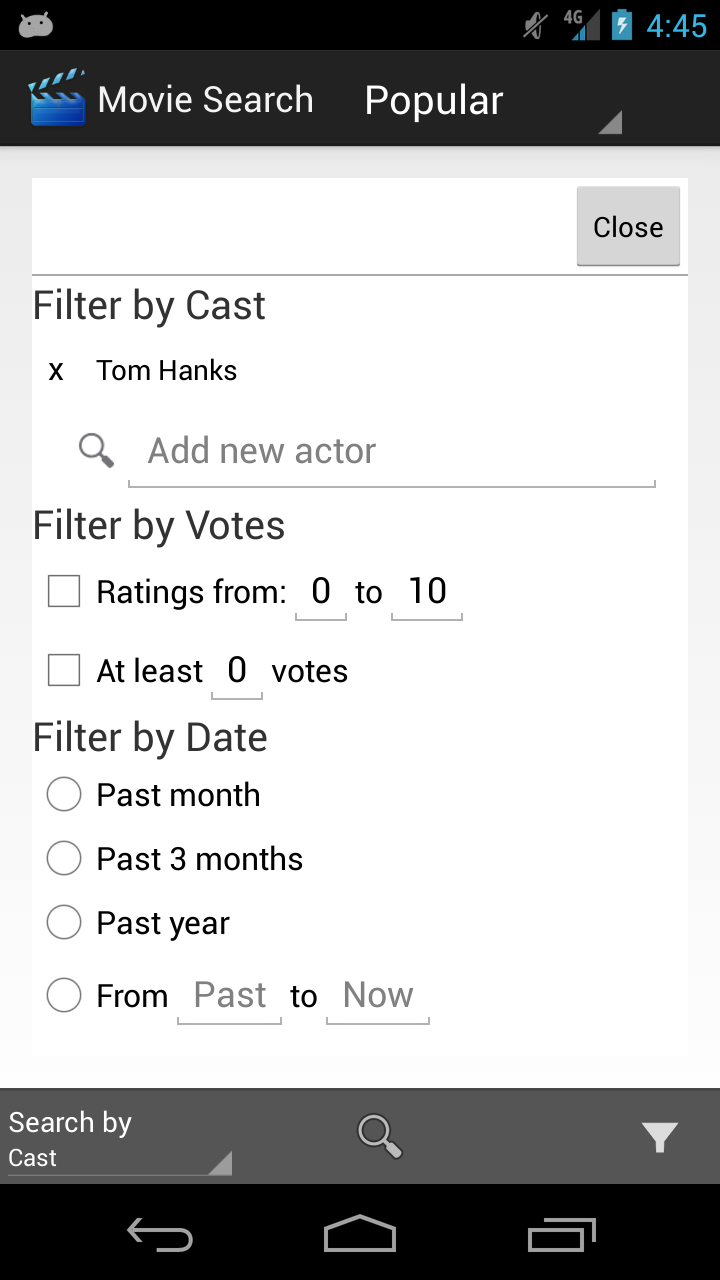
The Search Results page is a list of films received from TMDb based on a search term or actor and filtered based on the users selections on the Filter Page. For each film found, a movie poster is displayed if available and beside it the title and release date, if available, of the film. Tapping any entry on this page will take the user to the Details Page (Section 3.3.5). Filters can be applied on this page using the button on the bottom left (See section 3.3.4), and a selector for actor or movie searching is present, as well as a button to initiate a new search. Performing a new search with an actor will take the user to the Actor Results (Section 3.3.3).

**3.3.3 Actor Search Results**



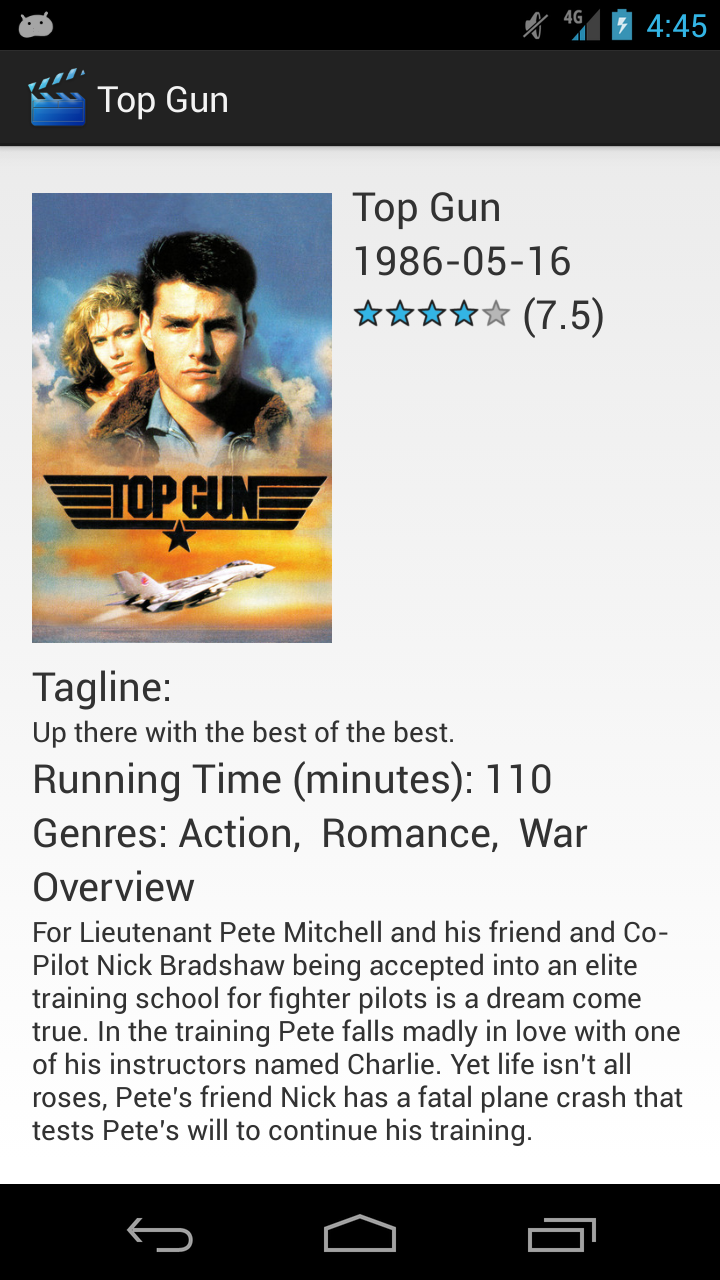
The Actor Search Results page is a list of actors found for a search term. Displayed is only the actor’s name. Tapping any actor will take the user to Search Results (Section 3.3.2) with results of films the selected actor has been in. Otherwise, this page is identical to the Search Results page (3.3.2).

**3.3.4 Filter Page**



On the Filter Page, the user can set up what filters they wish to use when searching. There are generally three types of filter that can be applied: the cast filter, vote filters or date filters. The cast filter allows the user to exclude films that a selected actor has appeared in. This selection is made using the Actor Search Results (Section 3.3.3, this is a feature not described there). The vote filters allow the user to only see a film that has a rating in a range that they specify or that has at least a specified number of votes. Date filters allow the user to select to only see films released in the past month, three months or year. The user can also specify a range of dates to see films from. Note that this is an overlay over the other pages, so the user can still do some other page navigation while accessing this page.

**3.3.5 Details Page**



The Details Page displays information about a film. The displayed information is as follows: movie poster, title, release date, rating, tagline, running time, genres, an overview, and the cast list. Any of these that are not available will not be displayed or a placeholder is displayed in its place.

**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 Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | View Default Page | Launch Application | The Landing Page (see 3.3.1) should display |  |
| 2 | View Details | Click on any listed film on the Landing 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 Landing 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 Landing 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. |  |
| 7 | View Popular Titles | Select “Popular” from the spinner on the Landing Page | The Landing page’s default results list should repopulate with a list of popular films |  |
| 8 | Test Rotation | Change orientation of device from vertical to horizontal and vice versa. | All information on the current page should be retained. |  |
| 9 | Filter Results | Select the filter for “past year” and preform a search for “Bourne” under movies. | The only result displayed should be “The Bourne Legacy”. |  |
| 10 | Continuous Scrolling | From the “Popular” results page, continue scrolling downwards. | New results should continually load as you reach the bottom of the page. |  |

**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. |  |
|  | 6 | MovieListingData constructor | Call the 7-variable constructor with the arguments adult = false, title = “Top Gun”, releaseDate = “1986-5-15”, id = 500, poster = “poster.jpeg”, rating = 5, and voteCount = 5. | Retrieve the adult, title, releaseDate, poster, rating and voteCount variables using their appropriate accessor methods and test that all of the following relations are equal: adult = false, title = “Top Gun”, releaseDate = “1986-5-15”, id = 500, poster = “poster.jpeg”, rating = 5, and voteCount = 5. |  |
|  | 7 | MovieListingData write parcel | Given an initialized MovieListingData object, using the values from the MovieListingData 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() = “Top Gun”, result.readString() = “1986-5-15”, result.readInt() = 500, result.readString() =”poster.jpeg”, result.readDouble = 75, and result.readInt = 5. |  |
|  | 8 | Test execute query results | Call synchronousMovieSearch with arguments movieSubstring = “Top Gun” and page = 1. Replace TMDb method with a stub that returns a JSON object with the values page = 1, total\_pages = 1, total\_results = 1, results = [adult = false, title = “Top Gun”, id = 500, release\_date = 1986-5-15, poster\_path = poster.jpeg, vote\_average = 5, and vote\_count = 5] | synchronousMovieSearch returns movies = “Top Gun” and numResults = 1.The JSON object shouldhave the following relations: page = 1, total\_pages = 1, total\_results = 1, results = [adult = false, title = “Top Gun”, id = 500, release\_date = 1986-5-15, poster\_path = poster.jpeg, vote\_average = 5, and vote\_count = 5] |  |
|  | 9 |  |  |  |  |
|  | 10 |  |  |  |  |

**Integration Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | Test database query (Detail model) | Call DetailModel’s static method synchronousDetailPrimaryRetrieve with movieId = 155 | synchronousDetailPrimaryRetrieve 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 |  |
| 2 | Test database query (Actor subsearch model) | Call ActorSubseachModel’s synchronousActorSubsearch with arguments nameSubstring = “Tom Cruise” and page = 1. | synchronousActorSubsearch should return an actorData object with the following relations: adult = 0, name = “Tom Cruise”, id = 500, popularity = 5 and profile = “profile.jpeg” |  |
| 3 | Test database query (Movie search model) | Call MovieSearchModel’s synchronousMovieSearch with arguments movieSubstring = “Top Gun” and page = 1. | synchronousMovieSearch should return an MovieListingData object with the following relations: adult = false, title = “Top Gun”, id = 500, release\_date = 1986-5-15, poster\_path = poster.jpeg, vote\_average = 5, and vote\_count = 5 |  |
| 4 | Test database query (Actor search model) | Pass ActorSearchModel’s synchronousCastSearch an actorData object with the relations adult = 0, name = “Tom Cruise”, id = 500, popularity = 5 and profile = “profile.jpeg” | synchronousCastSearch should return a MovieListingData object with the following relations: adult = false, title = “Top Gun”, id = 500, release\_date = 1986-5-15, poster\_path = poster.jpeg, vote\_average = 5, and vote\_count = 5 |  |
| 5 | Receive Data (Dual Model > Results Activity) | Enter “Top Gun” into the search bar as a movie search. | If the results page loads and the movie “Top Gun” appears, the results activity is successfully receiving data from the dual model. |  |
| 6 | Receive Data (Dual Model > Movie Search) | Enter “Top Gun” into the search bar as a movie search. | If the results page loads and the movie “Top Gun” appears, the results activity is successfully receiving data from the dual model. |  |
| 7 | Receive Data (Dual Model > Actor Search Activity) | Enter “Tom Cruise” into the search bar as a cast search | If the ActorSubsearch menu loads and the name Tom Cruise is there, the actor search is successfully receiving data from the dual model. |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
|  | 10 |  |  |  |  |

**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

**Team Member Assignments**Alex Good  
Michael Costa  
Trevor Chipley  
  
**Time Breakdown**

# 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>