**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 and disk permissions for caching results.

**2.4 Implementation Outline**

This project will primarily be developed in Java using the Android SDK with network requests formatted in JSON.

**2.5 Operational Assumptions**

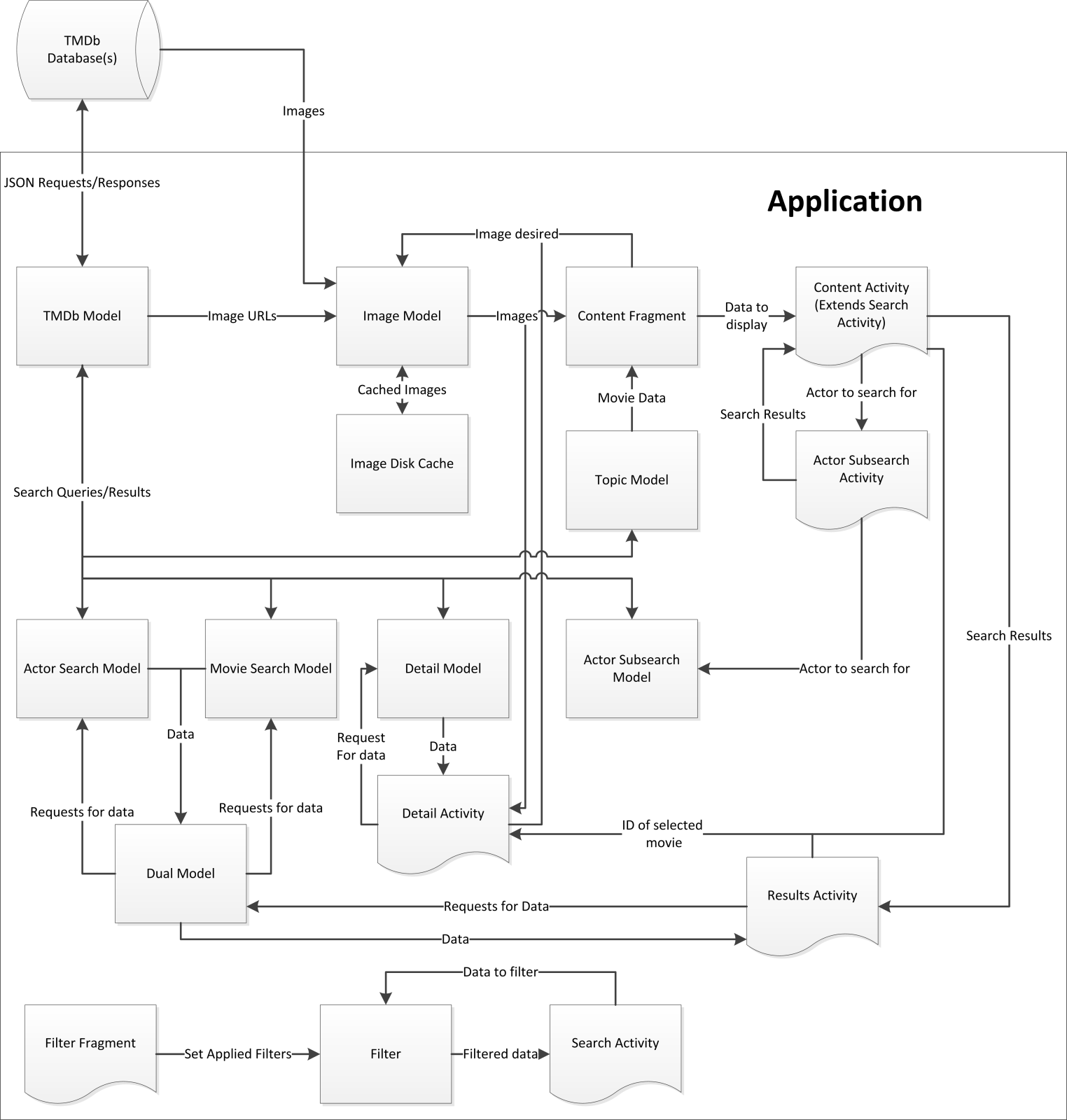
It is assumed that the TMDb servers are operational when the application is in use. Included in this is that the servers are assumed to return good and accurate responses to queries within a reasonable amount of time.

**3 Technical Specifications**

**3.1 Development Tools and Languages**

We will be using Java with the Android SDK, using JSON for querying external databases, and XML for generating the user interface. The development will be done in the Eclipse IDE with Android ADT plugin.

**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) along with a set of preferred dimensions that they wish to download. The Image Model determines the optimal size, fetches the image, caches it in memory and on disk, 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. There is also a DualModel which consists of a Movie Search Model and an Actor Search Model, allowing for ease of switching minimizing overhead for activities that use it.

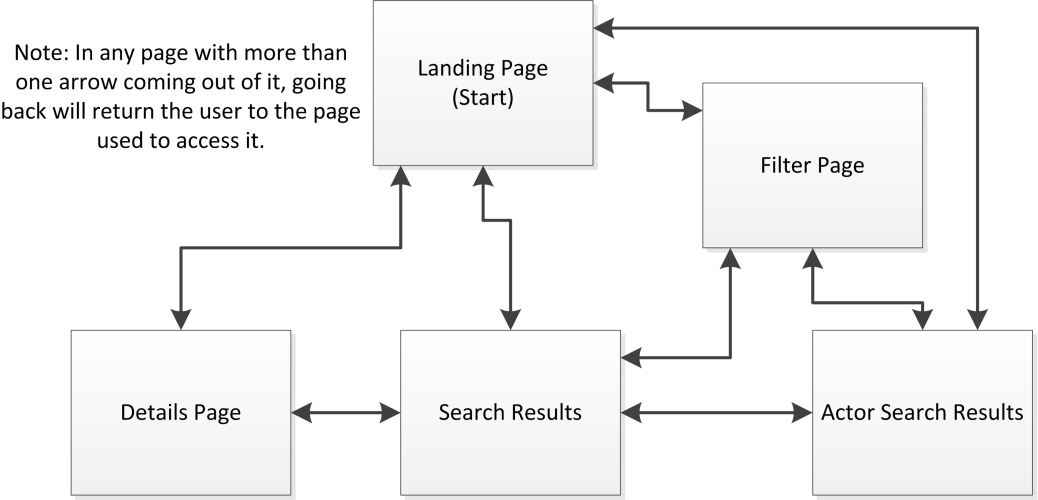
**3.2.1.4 Filter**

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**

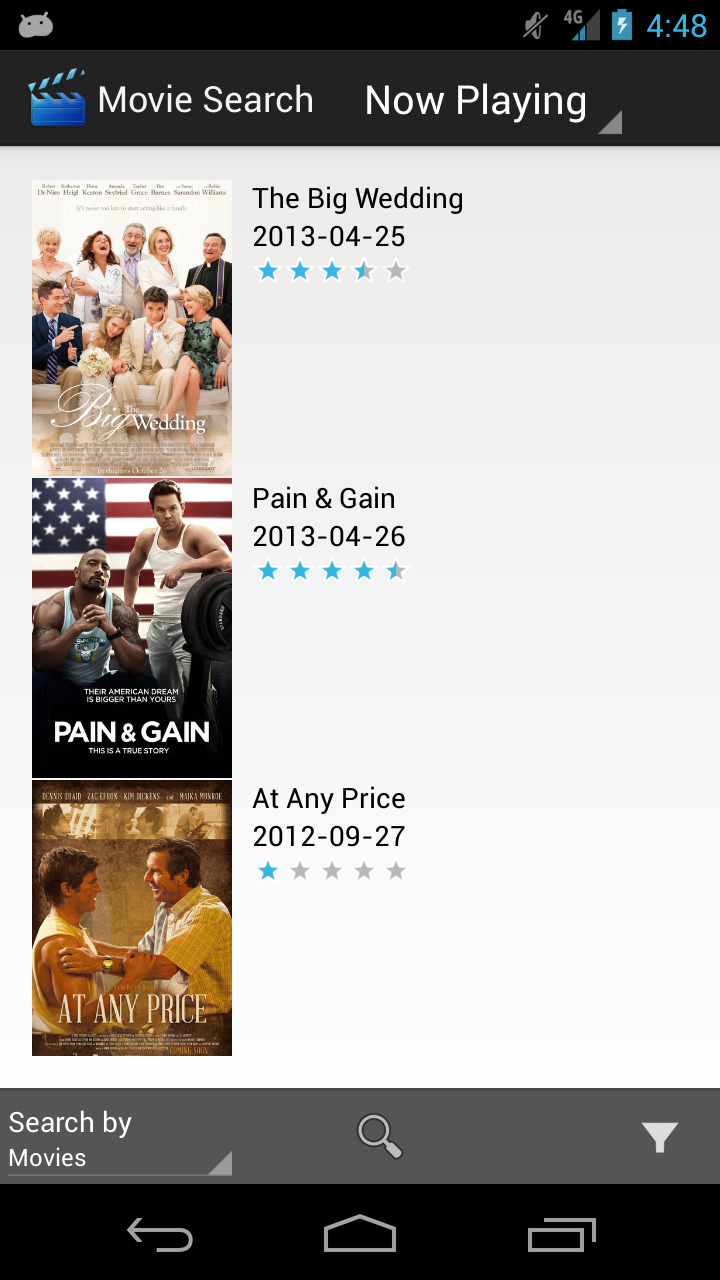
This is the internal name for the Landing Page (Section 3.3.1).

**3.3 User Interface**



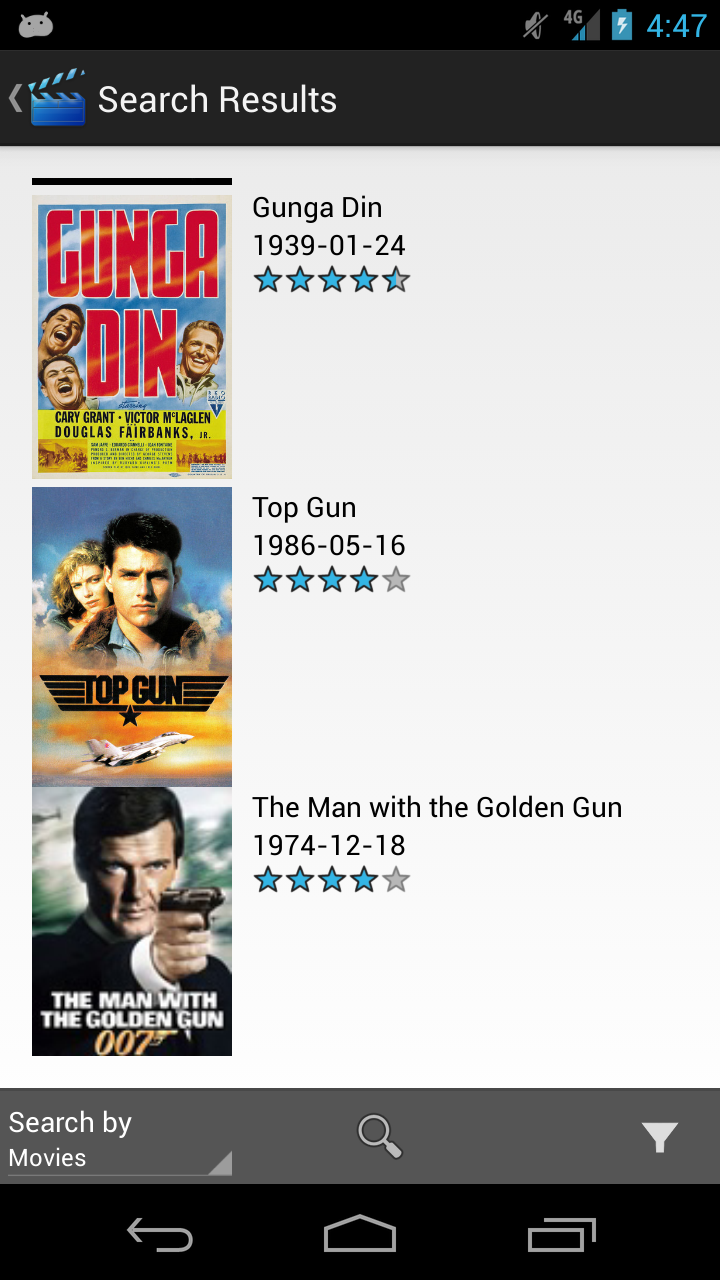
**Figure 2: Page Navigation Diagram**

**3.3.1 Landing Page**



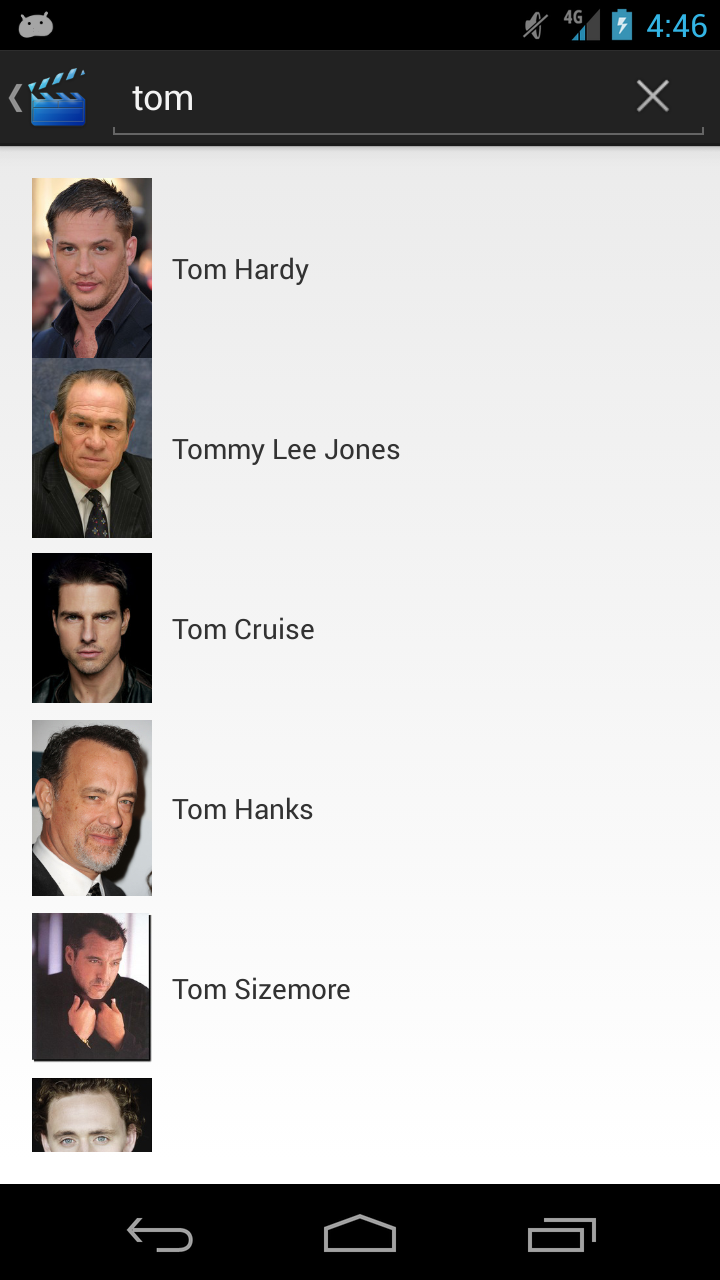
The Landing Page is the starting page of this application. By default, it displays a list of films that are currently in theaters. Also viewable on this page is a list of highly rated films, popular films and upcoming films, which can be switched between using a drop-down menu at the top of the page. 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**



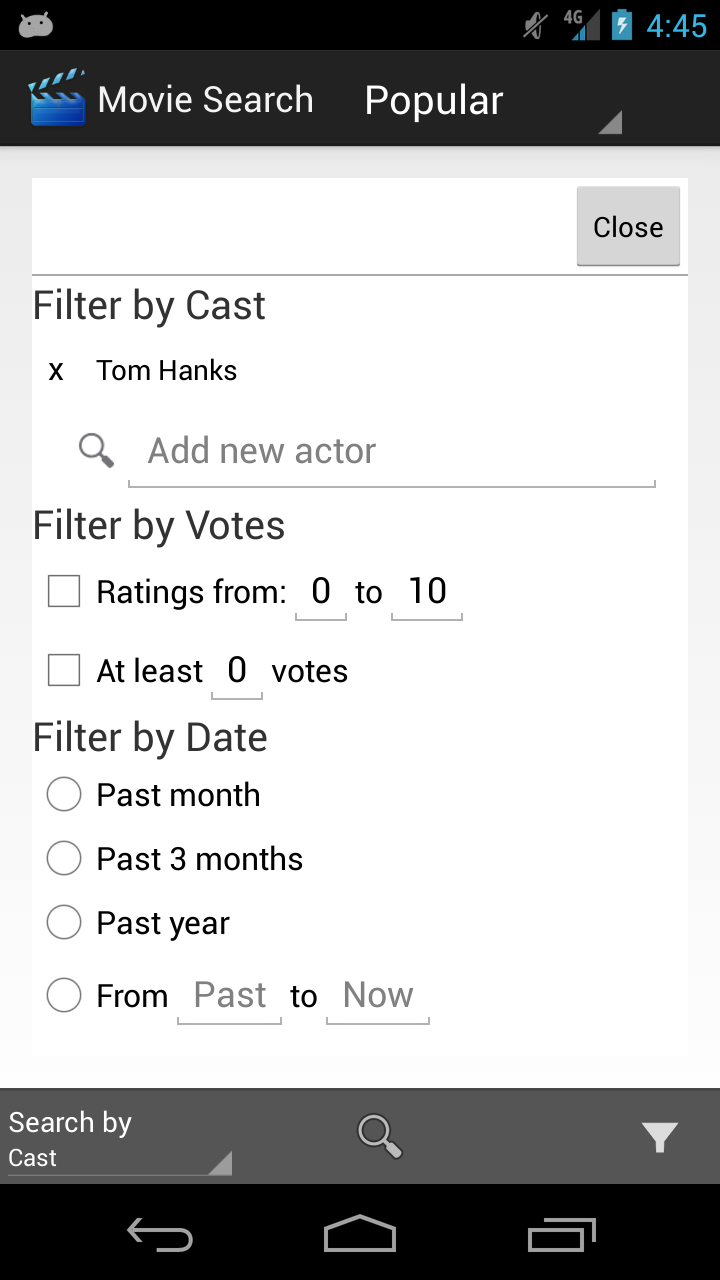
The Search Results page is a list of films received from TMDb based on a search term or actor and filtered based on the users selections on the Filter Page. For each film found, a movie poster is displayed if available and beside it the title, release date, if available, and average rating of the film. 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 right (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**



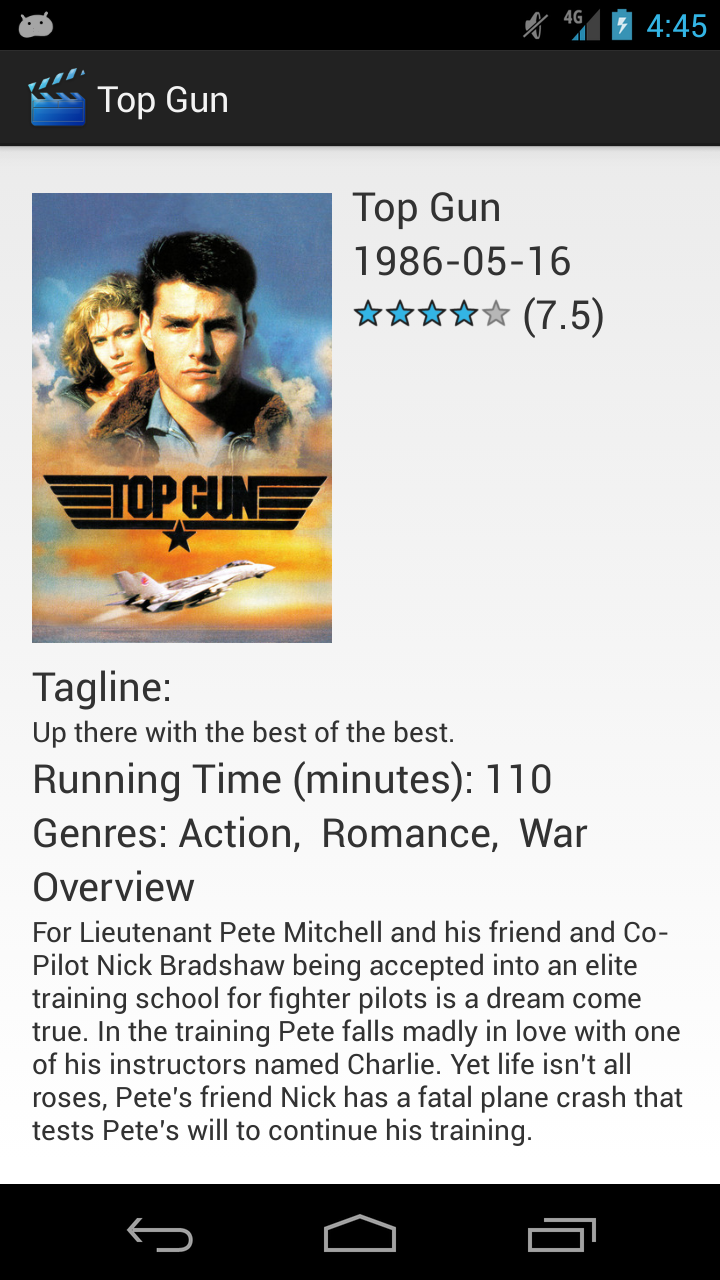
The Actor Search Results page is a list of actors found for a search term. Displayed is the actor’s name as well as a profile picture of the actor, if available. Tapping any actor will take the user to Search Results (Section 3.3.2) with results of films the selected actor has been in. Otherwise, this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

}

}

JSON Example

See TMDb API

XML Example  
  
<Example>  
 <name type=”first\_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 | From the Landing Page, elect “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 on the Landing Page. | All information on the current page should be retained. |  |
| 9 | Filter Results | From the Landing page elect the filter for “past year” and preform a search for “Bourne” under movies from the Landing Page. | The only result displayed should be “The Bourne Legacy” and “Bette Bourne: It Goes With the Shoes” |  |
| 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 returned ActorData object should have the following relationships:  isAdult() = false, getName() = Tom Cruise, getId() = 500, getPopularity() = 1.96, getProfile() = profilepicture.jpeg |  |
| 4 | ActorSubsearchActivity life cycle initialization (starting) | Using instrumentation, create an Intent named “intent”, and call intent.putExtra(“query”, “Tom Cruise”) and call the onCreate method of ActorSubsearchActivity | The resulting ActorSubsearch data should have the following properties: the grid view with id gridview\_actor\_subsearch should be initialized, empty, and should be visible within the root view. The string query should be set to “Tom Cruise” and the variables subsearchModel and image should be empty. |  |
| 5 | ActorSubsearchActivity life cycle initialization (from previous instance) | Using instrumentation, call the onCreate method of ActorSearch, add “Tom Cruise” to the intent, call callActivityOnSaveInstanceState(), call onDestroy(), call onCreate() | The resulting ActorSubsearchActivity from the 2nd call to onCreate should have the grid view with id gridview\_actor\_subsearch initialized, empty, and visible within the root view. The string query should be set to “Tom Cruise” and the variables subsearchModel and image 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 MovieSearchModel JSON parsing | Call synchronousMovieSearch with arguments movieSubstring = “Top Gun” and page = 1. Replace TMDb executeQuery 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 should have 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] |  |
|  | 9 | Test Filter check time | Create a Filter object with the default constructor, call setTimeFilter(Filter.ONE\_YEAR). Create a MovieListingData object using the 5-variable constructor with the following values: adult = false, title = “Top Gun”, id= 500, release\_date = “1986-5-15”, and poster\_path = poster.jpeg. Call Filter.checkTimeFilter(data) using the MovieListingData as the argument data. | checkTimeFilter returns false. |  |
|  | 10 | Test calculating best poster size | Assign the static variable TmdbModel.posterSizes the array of strings [“w92”, “w154”, “w185”, “w342”, “w500”, “original”]. Call the static method TmdbModel.getBestProfileSize(150). | The returned value of TmdbModel.getBestProfileSize(150) should be the string “w154” (as 154 is the first number greater than or equal to 150). |  |

**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 | FilterFragment startActivity interface test (FilterFragment > ContentActivity > ActorSubsearchActivity) | Using instrumentation, create a ContentActivity using the onCreate() method, set the search type spinner to “cast” and create a FilterFragment from ContentActivity using the static method FilterFragment.newInstance(). Find the view with id search\_filter\_add\_exclude\_cast and set the text to “Tom Cruise”. Submit the query. | Verify that the current activity is ActorSubsearchActivity by calling getActivity() and testing if getActivity() instanceof ActorSubsearchActivity. |  |
| 6 | ActorSubsearch passing mechanism test (ActorSubsearchActivity > ContentActivity) | Using instrumentation, set the search type spinner to “cast” and enter “Tom Cruise” in the search widget and submit the query. Read the list items in the gridview with id gridview\_actor\_subsearch and look at the TextView with id textview\_actor\_subsearch\_grid\_item\_name. Search for “Tom Cruise” and select the first item that matches. | In the onActivityResult() method of ContentActivity, verify that resultCode = RESULT\_OK, requestcode = ACTOR\_SUBSEARCH\_REQUEST, and that result = data.getParcelableExtra(“result”) instanceof ActorData and that result.getName() = “Tom Cruise” |  |
| 7 | FilterFragment initialization from Filter (Filter > FilterFragment) | Create a Filter object with the default constructor, call filter.setTimeFilter(Filter.ONE\_MONTH, null, null) and using instrumentation create a FilterFragment by calling FilterFragment.newInstance() and call intent.putExtra(“filter”,filter) | Verify in the resulting FilterFragment object that the RadioButton with id “radio\_filter\_date\_past\_month” is checked (radio.isChecked()) |  |
| 8 | EndlessAdapter load additional data test (ContentActivity > EndlessAdapter) | Create a ContentActivity using instrumentation by calling onCreate(), switch the navigation type to “Popular” by finding the spinner on the menu. Find the gridview with id “gridview\_content” and call getCachedDataCount() and record it in a variable count1. Scroll to the bottom of the gridview with id “gridview\_content”. Wait for the asynchronous task to complete. | Call getCachedDataCount() again on the Gridview adapter and store the value in count2. Verify that count2 > count1 or (count2 == count1 && getCachedDataCount() == getDataCount()) |  |
| 9 | Image Disk cache test (Content Activity > Image Disk Cache) | Create a ContentActivity using instrumentation by calling the onCreate() method. Wait for results to load asynchronously and then call onSaveInstanceState() followed by onDestroy() then onCreate(). | Verify that the images have been loaded from disk (as the memory cache has been deleted) by finding the imageViews with id “imageview\_content\_grid\_item” and checking that the bitmap is not film\_reel.png |  |
|  | 10 | DualModel switching test (DualModel > ActorSearchModel – MovieSearchModel) | Using instrumentation, create a ContentActiviy by calling onCreate. Enter “Top Gun” into the search bar and submit the query, note the state of activeType in DualModel after performing the search by storing it into type1. Go back to ContentActivity and switch the searchType to cast and enter the query “Tom Cruise” and submit the query. Select “Tom Cruise” from the results of the ActorSubsearchModel and note the state of activeType in DualModel by storing it into type2. | Following this procedure, the variables type1 and type2 should not be equal. This shows that DualModel can switch between the two types of models. |  |

**Systems Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Step** | **Test Name** | **Action Description** | **Expected Outcome** | **Success?** |
| A | 1 | Receive Data (Dual Model > Results Activity) | Enter “Top Gun” into the search bar as a movie search and click the search button. | If the results page loads and the movie “Top Gun” appears, the results activity is successfully receiving data from the dual model. |  |
|  | 2 | Receive Data (Dual Model > Actor Subsearch 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. |  |
| 3 | Transfer data Detail Activity (ContentFragment > DetailActivity) | From the launch page (ContentFragment), click on any of the movies listed. | If the Detail Activity appears with detailed information about the movie loaded, then ContentFragment passes data to Detail Activity successfully. |  |
| 4 | Transfer filter settings FilterFragment (FilterFragment > SearchActivity > FilterFragment) | From the launch page, click on the filter action button in the action bar. Click on the checkbox next to “At least 0 votes”, click on the close button, then click on the filter action button again | If the checkbox is still checked, then the FilterFragment has successfully passed data to SearchActivity and SearchActivity has successfully passed back the data to FilterFragment. |  |
| 5 | Retrieve image from Tmdb (ResultsActivity > ImageModel > Tmdb) | Clear the cache for the application from the applications menu. From the launch page, enter the query “Top Gun” into the search bar with “Search By Movies” selected. Wait for the search results to load if necessary. | If the default film reel image has been replaced with another image, then the ResultsActivity has successfully downloaded an image. |  |
| 6 | Device rotation save state | Enter “Top Gun” into the search bar from the landing page (ContentActivity) and wait if necessary for results to appear. Rotate the screen from portrait to landscape orientation. | If the contents of the results list have remained then ResultsActivity is able to save the state of its list when the activity is destroyed. |  |
| 7 | Searchtype spinner text resize test | Starting from ContentActivity, rotate the screen from portrait to landscape orientation. | Verify that all of the text in the search type spinner has resized by asserting that the text fits within the bounds of the containing actionbar. |  |
| 8 | Filter by cast | Preform a cast search for “Tom Cruise”, and note that “War of the Worlds” is the first listed film. Return to the Landing Page, open the filter page and add Morgan Freeman to the “Filter By Cast” field. Close the filter page and search for “Tom Cruise” again. | The movie “War of the Worlds” should not appear on the results page as Morgan Freeman is a cast member. |  |
| 9 | Action bar rotation saved state – Search by Cast | Select “Search by Cast” from the spinner in the action bar on the main page. Change the screen orientation from vertical to horizontal and vice versa. | “Search by Cast” should remain the selected object on the spinner |  |
| 10 | Action bar rotation saved state – Popular Films | Change “Now Playing” to “Popular” on the action bar on the landing page. Change the screen orientation from vertical to horizontal and vice versa. | “Popular” and its associated results should remain on the page |  |

# 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 Alex Good Michael Costa Trevor Chipley

**Time Breakdown**

Movie Search iteration 1 – 40 hours total (various)

Movie Search iteration 2:

User interface design and research – 20 hours, Endless adapter design and cached data source – 15 hours, Back end and network code – 30 hours, User interface implementation with planning – 50 hours, Filtering –25 hours

# 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>