Design and Implementation of Mobile Applications
2015-2016

MediaTracker

836897 Simone Graziussi

11 July 2016

Abstract

MediaTracker is an Android mobile application that allows to keep track of books, movies, TV shows and videogames. It provides the means to add media items (possibly retrieving their data automatically from the internet) and to view their lists (tracked media items and completed media items). It also allows to receive random suggestions from the lists, to be notified when upcoming media items become available and to backup/restore its contents with a JSON file.

Source Code: https://github.com/Simone3/MediaTracker
JavaDoc: https://simone3.github.io/MediaTracker/

Contents

Part I
Introduction

1 Purpose

MediaTracker is an Android mobile application that allows to keep track of books, movies, TV shows and videogames.

The application provides the means to insert a media item to the tracked list with the possibility of setting properties like title, release date, etc. This information can be retrieved from the internet: while typing the title of the media item the user will be able to select one of the given suggestions to automatically load its properties in the form.

After inserting the media items, the user can view their list, divided in sections (e.g. importance level selected by the user). From there the user will be able to click on a media item to view its details, set it as completed, move it in the list order, etc. The user will also be able to access the list of all completed media items, ordered by completion date.

Another screen of the application allows to receive random suggestions. For example, the user will set some constraints (e.g. duration, genre, etc.) and the application will suggest a media item taken from the list.

Other features of the application are notifications when "upcoming" media items that the user is tracking become available and the possibility to save (and then restore) all the media items information in a JSON file.

2 Definitions

- **Category**: created by the user, has one media type and contains several media items.
- **Media Type**: one among "Book", "Movie", "TV Show", "Videogame".
- **Media Item**: a generic item, it has a linked category that also represents its media type (e.g. "The Picture of Dorian Gray" is a media item that belongs to the category "My Books" that in turn has media type "Book").

3 Objectives

- Access the list of the categories.
- Add, update and delete categories.

- Access the list of tracked media items for a given category, divided by importance level.
- Add, update and delete media items.
- Retrieve media item information (e.g. director for a movie) from the Internet, given the media item name.
- Read media item details after clicking on an element in the media items list.
- Access the list of completed media items for a given category, ordered by completion date.
- Set media item as completed (i.e. move it from the tracked list to the completed list).
- Set media item as "redoing" (i.e. move it from the completed list to the tracked list).
- Access the suggestions page for a given category (i.e. specify a set of constraints and receive a random media item as a result).
- Receive notifications about tracked media items that become available.
- Import/export the database as a JSON file.

4 Features

For each of the previously mentioned objectives, *MediaTracker* should provide the following features:

- Access the list of the categories:
  - The system should retrieve all categories from the database.
  - The system should display all categories in a list or table.
- Add, update and delete categories:
  - The system should provide a dedicated button to add new categories.
  - When the user clicks on a category, the system should provide the buttons to update or delete that category.
  - In the category form, the system should provide all necessary inputs and a save button.
  - When the user clicks on the save button, the system should save all the category data in the database.

- Access the list of tracked media items for a given category, divided by importance level:
  - The system should provide a way to access the list (i.e. click on a category in the categories list).
  - The system should retrieve all tracked (i.e. not yet completed) media items in that category from the database.
  - The system should display the list, grouping the elements by section (i.e. status and importance level).
- Add, update and delete media :
  - The system should provide a dedicated button to add new media items.
  - When the user clicks on a media item, the system should provide the buttons to update or delete that media item.
  - In the media item form, the system should provide all necessary inputs and a save button.
  - When the user clicks on the save button, the system should save all the media item data in the database.
- Retrieve media item information from the Internet, given the media item name:
  - In the media item form the system should provide in the title input an auto-complete feature to search an external database.
  - If the user selects an auto-complete option, the system should load the information of that item in the form inputs.
- Read media item details after clicking on an element in the media items list:
  - When the user clicks on a media item, the system should redirect him/her to the details page of that media item.
  - The system should retrieve from the database all data set by the user during the media item creation.
- Access the list of completed media items for a given category, ordered by completion date:
  - The system should provide a way to access the list (i.e. click on a dedicated button in the navigation menu).
  - The system should retrieve all completed media items in that category from the database.

  - The system should display the list, ordering the elements by completion date.

- Set media item as completed:
  - When the user clicks on a media item in the tracked list, the system should provide a button to set it as completed.
  - When the user clicks on that option, the system should set the completion date of that media item and move it to the completed list.

- Set media item as "redoing":
  - When the user clicks on a media item in the completed list, the system should provide a button to set it as redoing.
  - When the user clicks on that option, the system should reset the completion date and move the media item to the tracked list.

- Access the suggestions page for a given category:
  - The system should provide a way to access the suggestions (i.e. click on a dedicated button in the navigation menu).
  - The system should allow the user to select a set of constraints on the suggestion.
  - The system should retrieve from the database a random media item that matches the constraints.
  - If no media item matches the constraints, the system should inform the user.

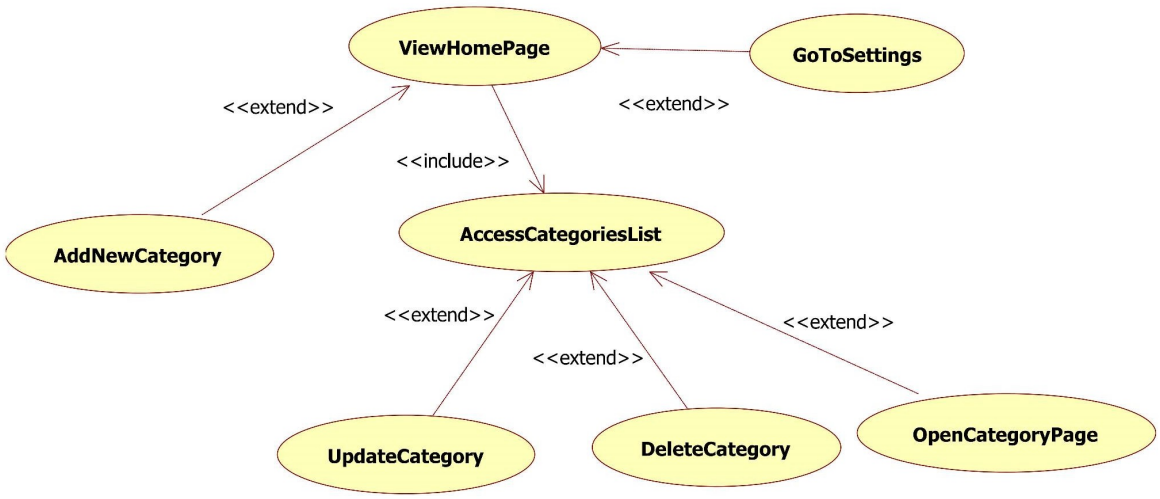
- Receive notifications about tracked media items that become available:
  - The system should allow a user to set the notification preferences (whether to receive them at all, time, ringtone, vibration).
  - If one or more media items were released on a given day, the system should send a notification that day at the time chosen by the user.

- Import/export the database as a JSON file:
  - The system should allow to save the database in a JSON file and then allow the user to save it wherever he/she wants.
  - The system should allow to import a JSON file into the application and rebuild the database starting from it.
  - If some error occurs during import/export, the system should notify the user.

Part II
Use Cases

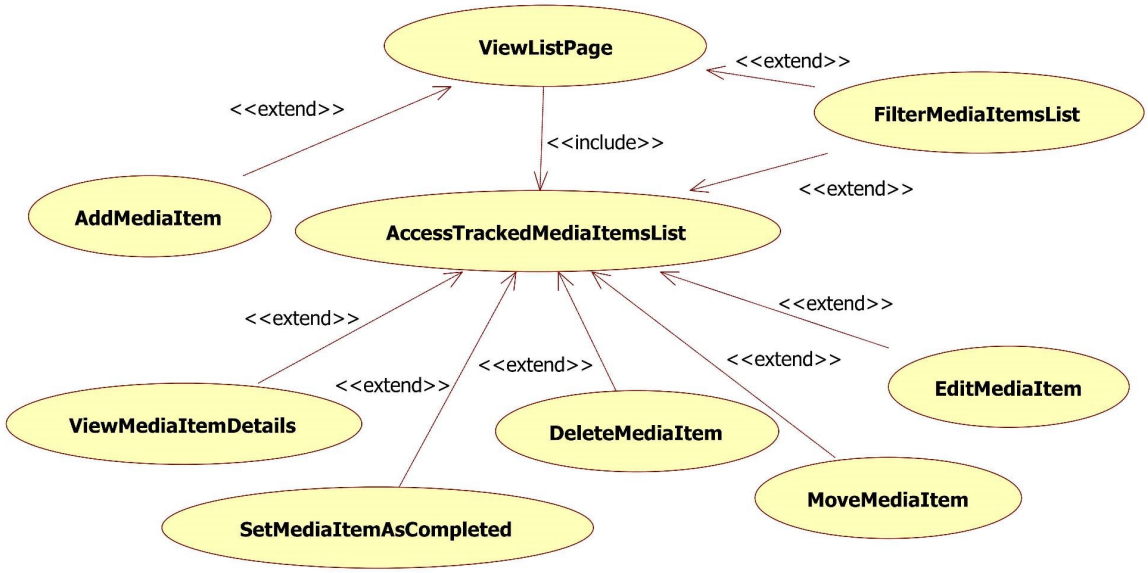
5 Home Page

Actors	User
Entry Conditions	The app has been opened by the user
Flow of events	The user in the home page can: • view the categories he/she created • add, update and delete a category • click on a category to open its page • navigate the application using the navigation drawer or the toolbar (e.g. access the settings page)
Exceptions	• while adding a new category, if it has an empty name the user is notified of the error • while adding a new category, if it has the same name as an existing category the user is notified of the error



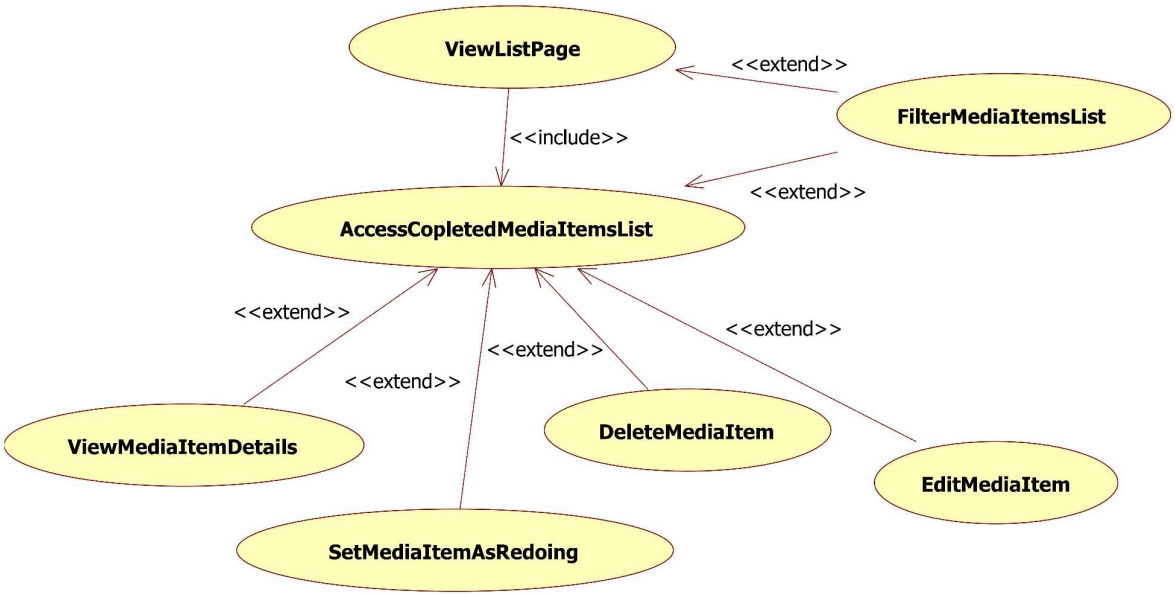
6 Tracked Media Items List

Actors	User
Entry Conditions	The user clicked on a category (either in the home list or in the navigation menu)
Flow of events	The user in this page can: • view the list of all tracked media items he/she previously added, divided by status and importance level • click on a media item to view/edit its details • set a media item as completed (e.g. set a videogame as played) • delete a media item • access the form for adding a new media item • reorder the media items in the list • search media items in the list (e.g. by name)
Exceptions	None



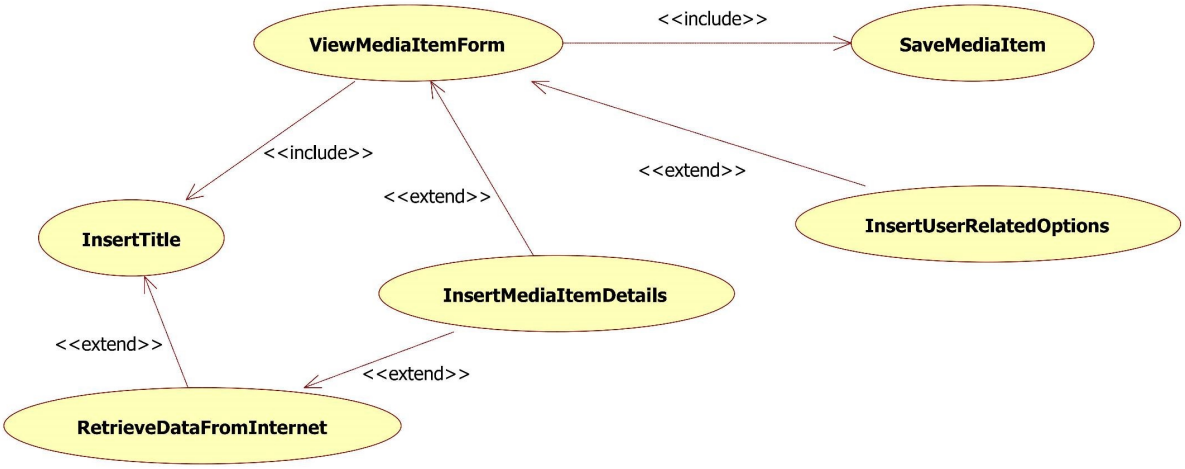
7 Completed Media Items List

Actors	User
Entry Conditions	The user selected the "completed" sub-page in the navigation menu for a given category
Flow of events	The user in this page can: • view the list of all completed media items ordered by completion date • click on a media item to view/edit its details • set a media item as "redoing" (e.g. set a book as re-reading) • delete a media item • search media items in the list (e.g. by name)
Exceptions	None



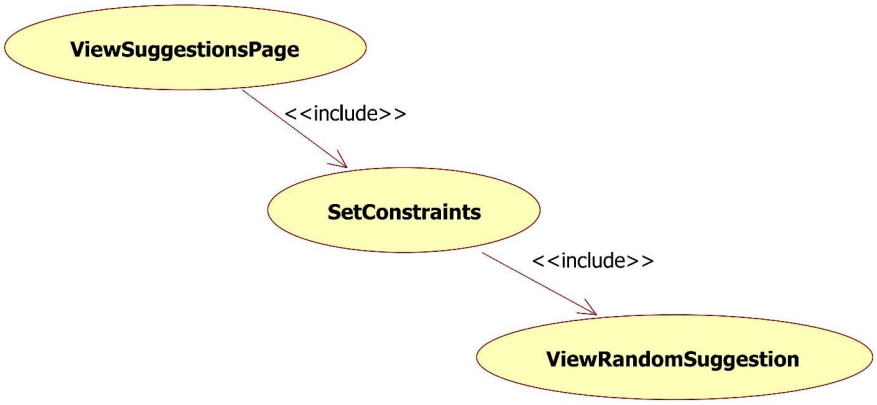
8 Add/Update Media Item

Actors	User
Entry Conditions	To add a new media item, the user clicked on the linked button in the tracked media items list; to update a media item, the user clicked on that item in the tracked media items list
Flow of events	The user in this page can: • insert a title for the media item. The system automatically suggests some results: if the user clicks on one of them the media item details (e.g. release date, description, etc.) are automatically set in the other inputs. • optionally insert/update details for the media item in the other form inputs • save the media item
Exceptions	• if the title is empty an error is shown • if there already is a media item with the same name an error is shown



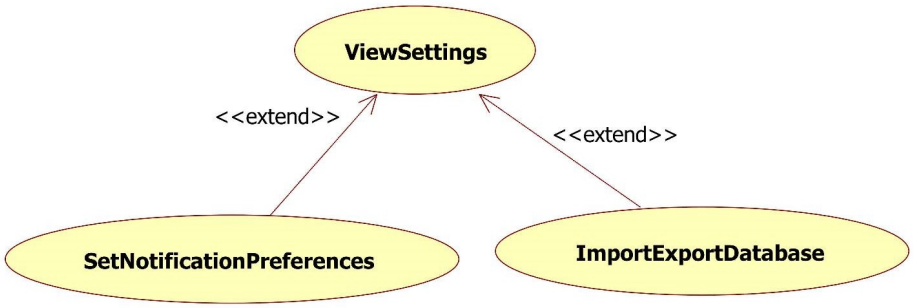
9 Media Item Suggestions

Actors	User
Entry Conditions	The user selected the "suggestions" sub-page in the navigation menu for a given category
Flow of events	The user in this page can: • set some constraints, e.g. status (tracked or already completed), genre, etc. • view a media item that satisfies the constrains randomly selected by the application
Exceptions	• if no media item matches the constraints a message is displayed



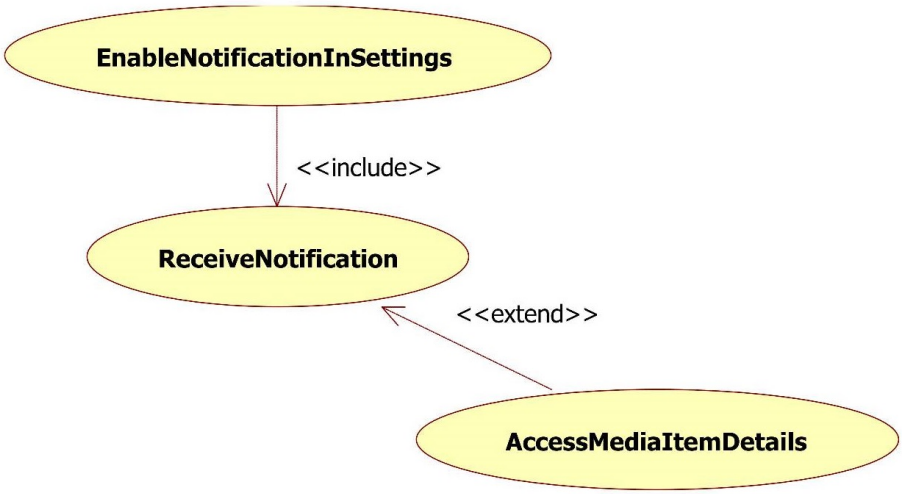
10 Settings

Actors	User
Entry Conditions	The user selected the "settings" option in the toolbar
Flow of events	The user in this page can: • set the preferences about notifications (whether to receive them at all, time, ringtone, etc.) • import/export the database (list of categories and media items) into a file
Exceptions	• if the file is unreadable or has unrecognized data an error is displayed



11 Notifications

Actors	User
Entry Conditions	The user received a "new release" notification (provided that the settings allow notifications to be sent)
Flow of events	The user receives a notification about one or more media items that were released today. Clicking on the notification the user can reach the details page for those media items.
Exceptions	None



Part III
Scenarios

12 Home and Settings

Jack opens the application and sees the default categories: "Books", "Movies", "TV Shows" and "Videogames". He deletes "Videogames" since he's not interested, and adds "Comics" category with "book" as media type. He then goes to the settings, where he sets the "new releases" notifications time at 8 PM.

13 Tracked Movies

He goes back to the Home and then clicks on "Movies" category. He is shown an empty list. He clicks on the button to add a new media item: he starts typing "Godf" and the application shows some suggestions. He clicks on "The Godfather (1972)" and the movie data (description, release date, runtime, etc.) is automatically set. He then selects the "Important" importance level, sets the movie as owned and saves it. He repeats the same operation for some other movies.

14 Completed Movies

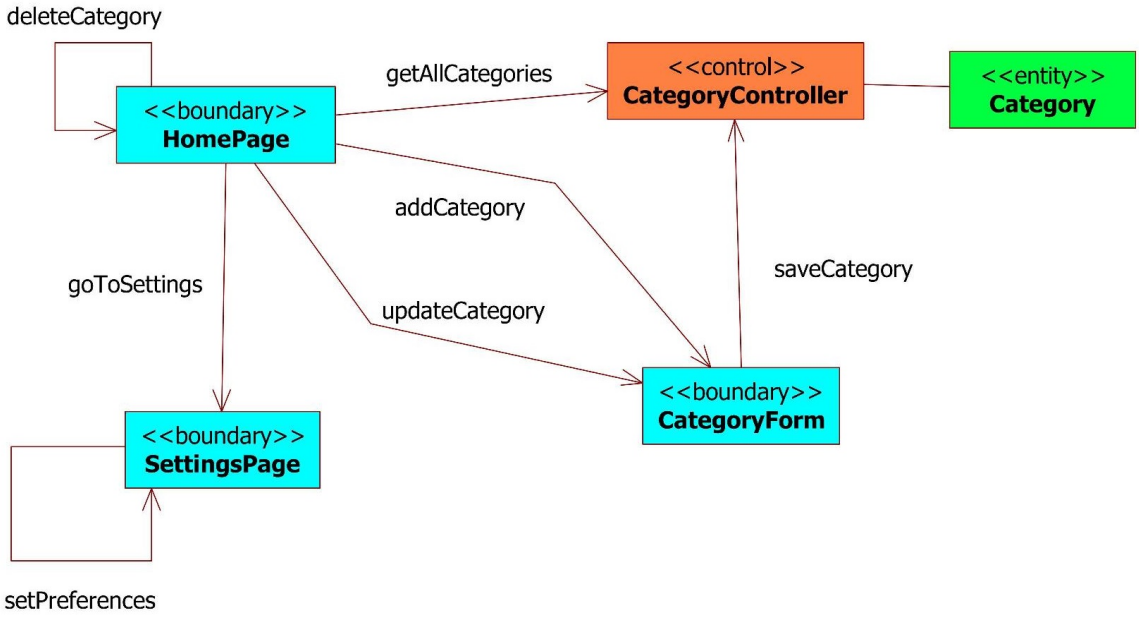
An evening he goes to the movies list. After reading some details, he decides to watch "Avatar" and, after doing so, he sets it as completed. A few years later, he decides to go to the completed movies list. In this list, ordered by date, sees that he watched "Avatar" a long time ago and decides to click on "I could rewatch this" to move it back to the tracked list.

15 Book Suggestions

Another day, after he finishes reading a book, Jack doesn't know what to read next. His tracked books list contains a lot of items, so he decides to open the suggestions page. He selects "historical" as genre and a "long" duration: the application selects "War and Peace". Jack, after reading some details, decides to start reading it.

Part IV
Design: Boundary-Control-Entity Diagrams

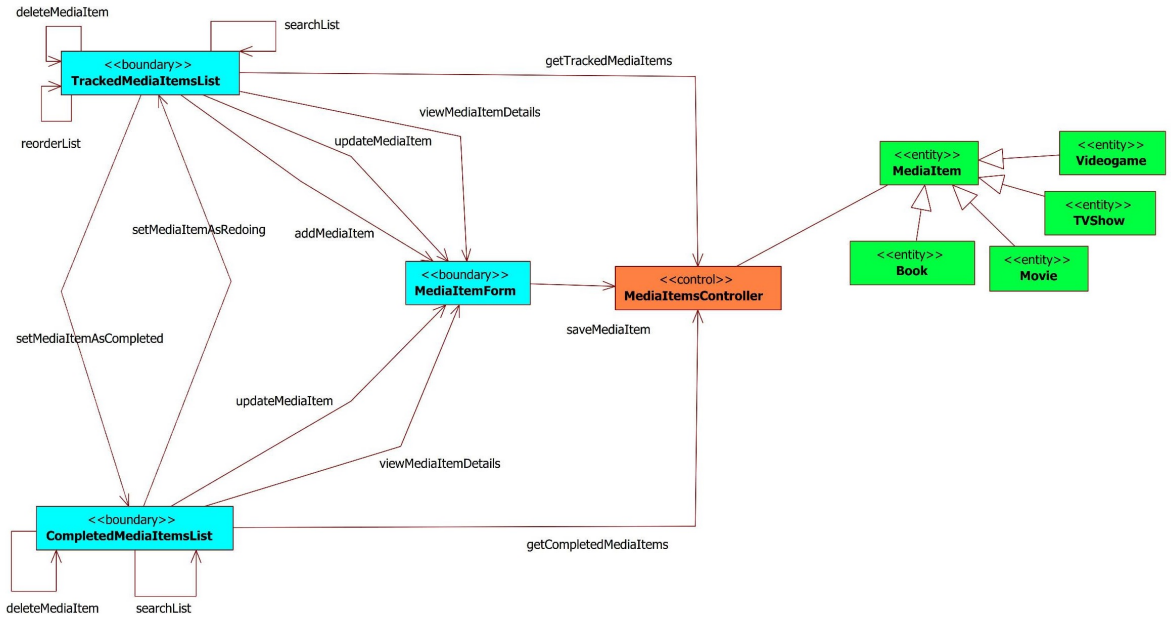
16 Home and Settings



The boundary **HomePage** allows to view the list of all categories. From there, the user can access the **SettingsPage**, where he/she will be able to manage the application preferences, and the **CategoryForm**, where he/she will be able to add or update a category. The **CategoriesController** manages both the home (retrieve all saved categories) and the form (add/update a category in the database), using the model class **Category** that defines all fields linked with that entity.

Clicking on a category in the HomePage list, the user will be redirected to the media items list in that category, explained in detail in the next section.

17 Media Items Lists

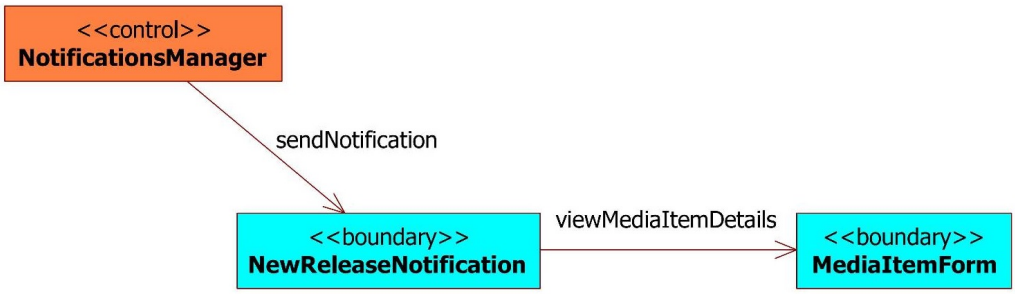


The application contains two media items lists, the **TrackedMediaItemsList** and **CompletedMediaItemsList**: the former contains the tracked media items, the latter the media items that have already been completed (e.g. watched, played, etc.) by the user. Both lists allow to search media items (e.g. by name) and to delete them; the tracked list also allows to reorder the media items. To move a media item from one list to the other the user can either set an item as completed or set it as redoing. The **MediaItemForm** allows to add (only coming from TrackedMediaItemsList) or update (coming from both lists) a media item. Both the lists and the form are managed by the **MediaItemsController** that allows to query the database and to update/delete media items from the persistent data. The controller interfaces with the (abstract) entity **MediaItem**, implemented by **Book**, **Movie**, **TVShow** and **Videogame**.

18 Media Item Suggestions

The **SuggestionsPage** allows to set some search constrains and to view a random media item suggestion. Like the lists seen in the previous section, it interfaces with the **MediaItemsController** to retrieve the proper suggestion from the database.

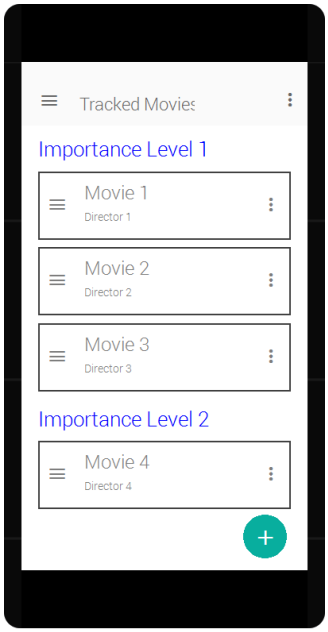
19 Notifications



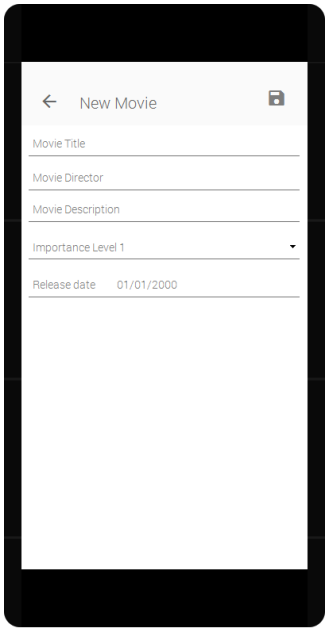
The **NotificationsManager** sends a notification when a media item was just released. The user is then shown a notification containing the title of the new media item and, if the user wants to, he/she can click on the notification to reach the details of that element, as seen in the previous sections.

Part V
Design: UI Prototypes

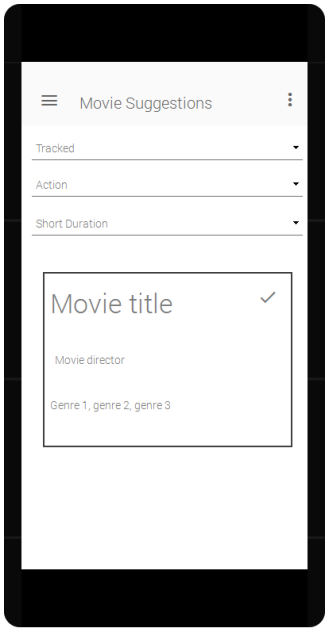
Home



Media Items List



Media Items Form



Suggestions

Part VI
Implementation: Architecture

20 Android

The application is built on top of the Android operative system, with support from API 19 (Android 4.4 KITKAT) to the current version, API 23 (Android 6.0 Marshmallow). It supports both phones and tablets.

21 External libraries

The application uses the following external libraries:

- **Gson** to manage JSON serialization/deserialization.
- **Retrofit** to connect to the external APIs to retrieve media item .
- **Sugar ORM** to manage the SQLite database.

Part VII
Implementation: Persistent Data

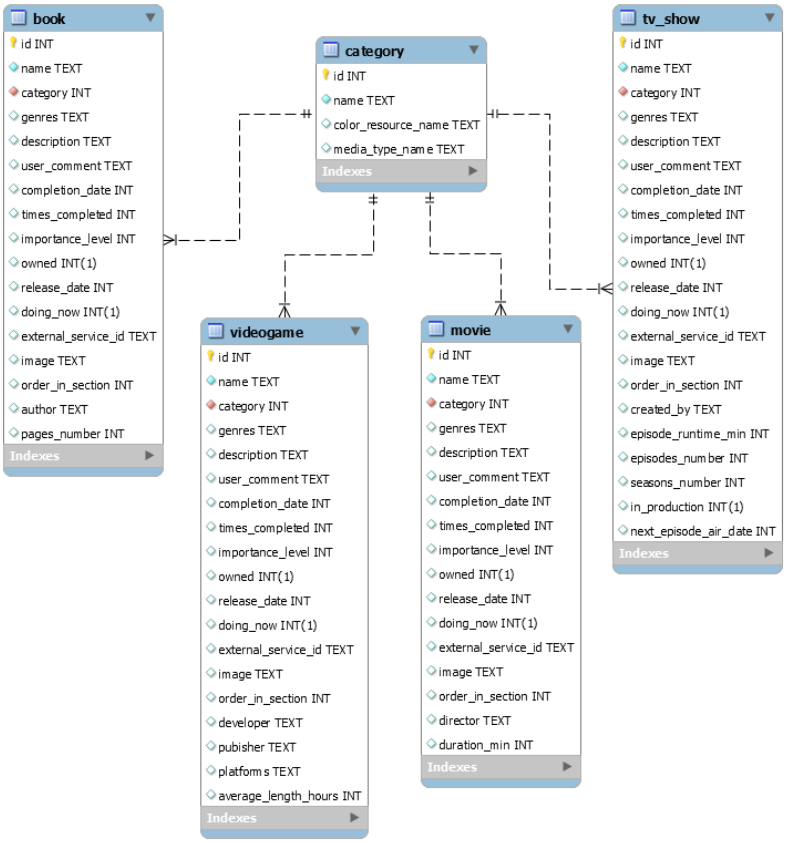
22 General Description

The application stores on the device the following data:

- Settings are stored using the SharedPreferences function provided by Android via the SettingsFragment implementation.
- The categories and media items information is stored in a SQLite database.

23 SQLite Game Database

The SQLite database is managed by the external library Sugar ORM that automatically creates the tables starting from the model classes defined in Java.



The database contains data about categories and media items (books, movies, TV shows and videogames).

The **category** table contains the description of a user-defined category of media items, i.e.:

- *id*
- *name*
- *color_resource_name*: the name of the resource representing the category color
- *media_type_name*: the name of the media type linked with this category (each category has one among "Book", "Movie", "TV Show" or "Videogame" as a media type)

Each media type has a different table to store the items the user is tracking, so we have **book**, **movie**, **tv_show** and **videogame** tables. These tables all contain a set of common attributes:

- *id*

- *name*
- *category*: id of the category the media item belongs to
- *genres*: string with a list of genres
- *description*: the summary
- *user_comment*: a comment the user may write
- *completion_date*: null if the user is tracking the media item, the last completion date if the user already "did" (e.g. watched, played, etc.) the media item
- *times_completed*: the number of times this media item was completed (e.g. watched, played, etc.) in the past
- *importance_level*: a value representing the user-defined importance
- *order_in_importance_level*: a value used to order items inside the same importance level
- *owned*: true if the user owns the media item
- *release_date*: the date in which the media item was (or will be) released
- *doing_now*: true if the user is "doing" (e.g. watching, playing, etc.) the media item now
- *external_service_id*: the external API ID for future updates
- *image*: the url of the media item image
- *order_in_section*: an integer value stored to keep the correct order in the media items list

In addition to these common fields, each media type table has specific columns:

- **book**
  - *author*
  - *pages_number*
- **movie**
  - *director*
  - *duration_min*: runtime in minutes
- **tv_show**

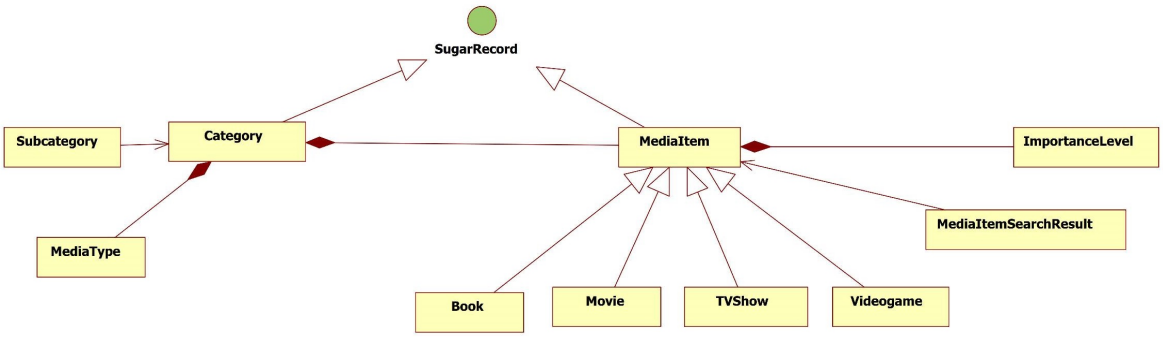
- *created_by*: the authors
- *episode_runtime_min*: runtime in minutes of each episode
- *episodes_number*: the number of all episodes
- *seasons_number*
- *in_production*: true if the TV show is still being produced
- *next_episode_air_date*: the date of the next episode (meaningful only if in_production is true)

- **videogame**
  - *developer*
  - *publisher*
  - *platforms*: string with the list of the game platforms
  - *average_length_hours*: number of hours that an average user takes to complete the game

Part VIII
Implementation: Class Diagrams

This part contains a description of all implementation classes. In order to avoid a unique large diagram, the explanation has been divided in sections. In the UML diagrams, green circles represent built-in Android classes to better understand the inheritance properties.

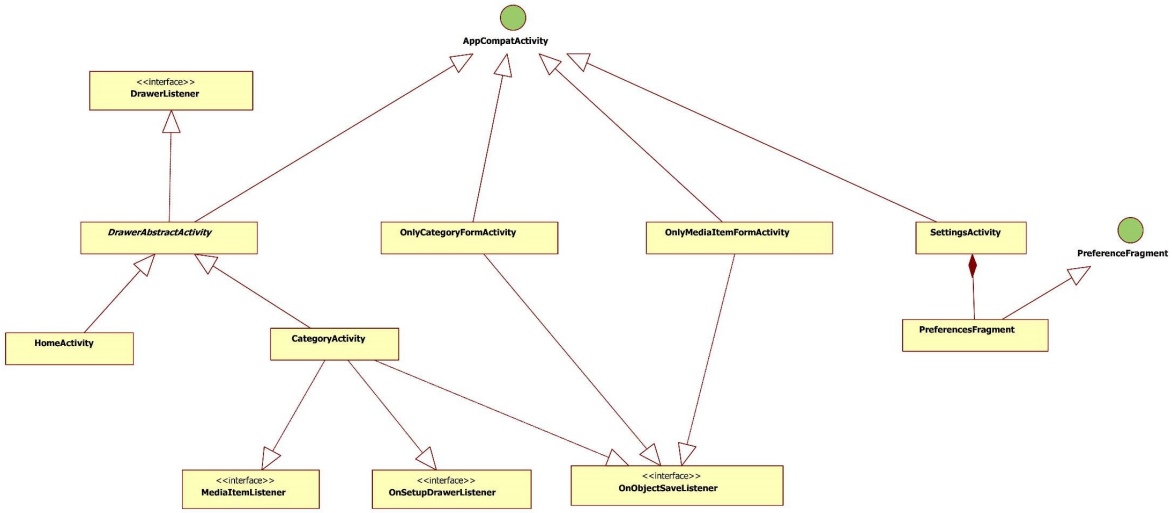
24 Model



- The main model classes are **Category** and **MediaItem** (with its implementations **Book**, **Movie**, **TVShow** and **Videogame**). As explained in the Persistent Data part, they represent the core data of the application and, since they extend the SugarRecord class, they are automatically translated into database tables.

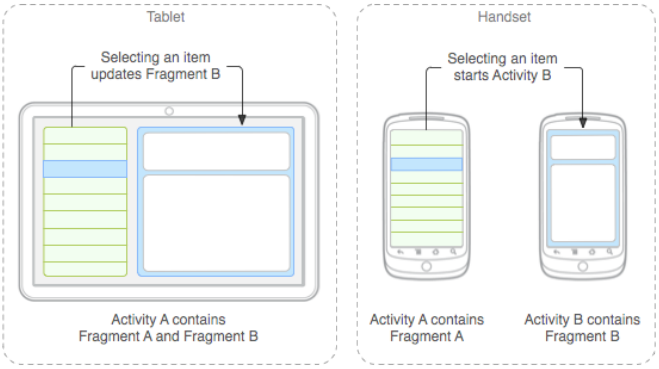
- The enum **MediaType** lists all available types of media items (books, movies, TV shows and videogames), one of which is linked to every category.
- The enum **ImportanceLevel** lists all available media items importance levels, which are then stored as integer values in the database.
- The enum **Subcategory** lists all the application subcategories, i.e. "Completed media items" and "Suggestions", used by CategoryActivity.
- **MediaItemSearchResult** class represents a search result returned by the MediaItemService.

25 Activities



The main activity of the application is **HomeActivity**: it contains a grid of all categories and allows to access the form to add/edit the categories. This activity, together with CategoryActivity, is a subclass of the abstract activity **DrawerAbstractActivity** that allows to manage the navigation drawer.

CategoryActivity is the core of the application. It's the "Activity A" of the Android multi-pane design pattern:



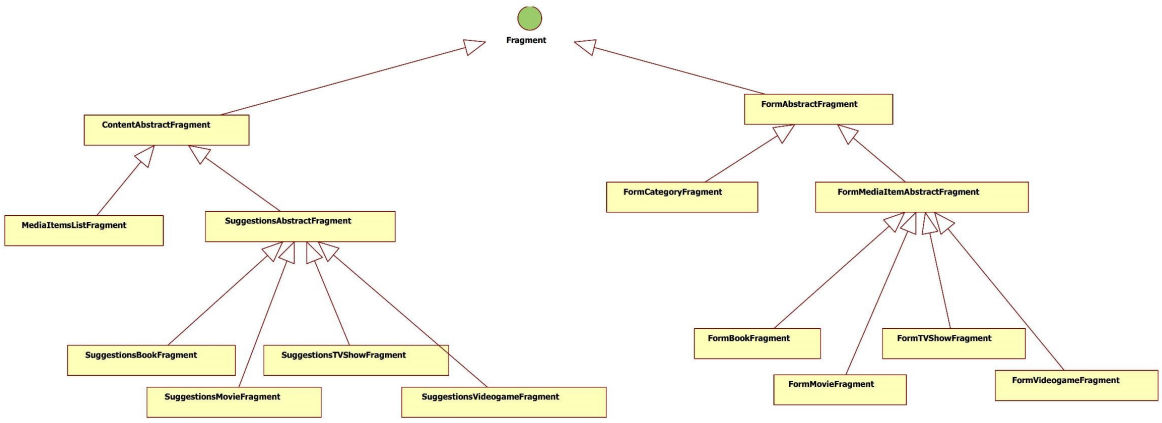
For this reason, on tablets it contains both a ContentAbstractFragment implementation and a FormAbstractFragment implementation while on phones it contains only a ContentAbstractFragment implementation. Its main parameters are Category and Subcategory: in particular

- if the subcategory is null the activity displays the list of tracked items in the category (see MediaItemsListFragment)
- if the subcategory is "Completed" it shows the list of all completed media items in the category (see MediaItemsListFragment)
- if the subcategory is "Suggestions" it shows the page for the suggestions in the category (see SuggestionsAbstractFragment)

OnlyCategoryFormActivity and **OnlyMediaItemFormActivity** are the "Activity B" of the above image: they contain a FormAbstractFragment implementation on phones (they are unused on tablets).

SettingsActivity, together with its **PreferencesFragment**, manages the application settings. It is linked with the SettingsManager to get/set the data and calls the DatabaseManager for the db import/export options

26 Fragments

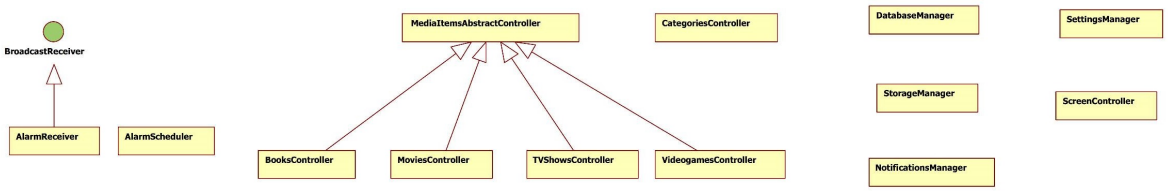


The application uses two main "types" of fragments ContentAbstractFragment implementations and FormAbstractFragment implementations:

- **ContentAbstractFragment** implementations are placed inside CategoryActivity and on tablets are shown in the left side of the multi-pane layout.
  - **MediaItemsListFragment** manages the tracked and completed media items lists. Thanks to the MediaItemsAbstractAdapter the list is divided in sections and thanks to the **EndlessRecyclerViewScrollListener** it is able to partially load the data if there are many elements.
  - **SuggestionsAbstractFragment** implementations (**SuggestionsBookFragment**, **SuggestionsMovieFragment**, **SuggestionsTVShowFragment**, **SuggestionsVideogameFragment**) manage the suggestions page.
- **FormAbstractFragment** implementations, placed either on the right side of CategoryActivity (tablets) or inside OnlyCategoryFormActivity/OnlyMediaItemFormActivity (phones), manage the media items or category forms (both for inserting and updating).
  - **FormCategoryFragment** manages the form for categories.
  - **FormMediaItemAbstractFragment** (in turn extended by **FormBookFragment**, **FormMovieFragment**, **FormTVShowFragment** and **FormVideogameFragment**) manages the form for all media items. In particular, FormMediaItemAbstractFragment contains an AutoCompleteTextViewWithDelay input for the title that,

when the user starts typing something, it queries the external service APIs for matches: if the user selects one of the options, the media item data (e.g. release date, description, etc.) are automatically inserted in the fields below.

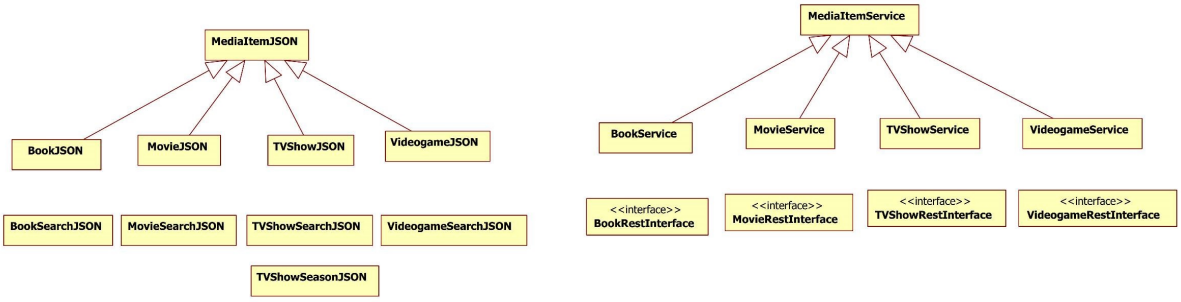
27 Controllers



- **MediaItemsAbstractController** (with its implementations **BooksController**, **MoviesController**, **TVShowsController**, **VideogamesController**) manages each media type. In particular they allow to query the database for data (e.g. list of media items of that type in a category), as well as inserting/updating/deleting media items. They also provide utility methods to retrieve the correct fragments (e.g. MoviesController will return SuggestionsMovieFragment when a suggestions page needs to be opened for a given category).
- **CategoryController** allows to manage the category database.
- **ScreenController** is a utility class to build the correct intents to navigate between activities.
- **SettingsManager** is just a "wrapper" for the built-in Shared Preferences, to offer getters and setters for the application settings.
- **NotificationManager** takes care of creating and displaying the app notifications. It is called by the AlarmReceiver when the notifications alarm is fired.
- The application alarms are managed by **AlarmScheduler** and by **AlarmReceiver**. The former is in charge of adding or removing the alarms, the latter is a BroadcastReceiver to actually receive them. In particular, AlarmReceiver receives:
  - the Android boot alarm in order to call the AlarmScheduler to reset all alarms after a device restart
  - the notifications alarm to call the NotificationManager when needed

- The **DatabaseManager** takes care of the database "as a whole", i.e. since queries to get/set data are managed by MediaItemsAbstractController and CategoryController this class only takes care of operations that involve the whole database. In particular it allows to backup/restore the database (categories and media items) as a JSON file. During import, data is validated: if something is wrong the controller tries to solve the problem (e.g. if importance level is missing it sets the default one) or, if that's not possible (e.g. category name is missing), shows an error and rolls back the database to the original configuration.
- **StorageManager** is called by the DatabaseManager to save/read the JSON backup files.

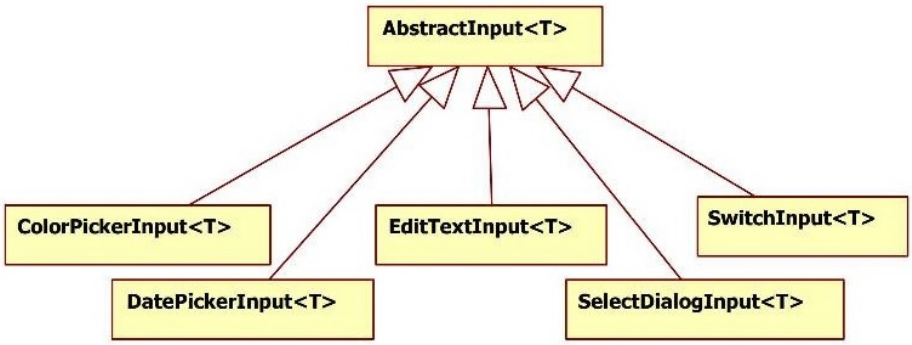
28 External Services



- **MediaItemService** and its implementations **BookService**, **MovieService**, **TVShowService** and **VideogameService** take care of the connection to the external APIs to retrieve the media item data, internally using a Retrofit object. They offer methods (with callbacks) to retrieve a list of MediaItemSearchResult given a search query or a MediaItem detailed information given an ID.
- To define the methods used to query the external APIs (REST interface) the MediaItemService implementations use one of **BookRestInterface**, **MovieRestInterface**, **TVShowRestInterface**, **VideogameRestInterface**.
- To interpret the JSON responses coming from the external APIs the services use model classes **MediaItemJSON** (with implementations **BookJSON**, **MovieJSON**, **TVShowJSON**, **VideogameJSON**), **BookSearchJSON**, **MovieSearchJSON**, **TVShowSearchJSON**, **VideogameSearchJSON**. These classes basically work as "translators" between the JSON response and the actual model classes like MediaItemSearchResult and MediaItem actually employed by the

services: for example "MovieJSON" exposes fields like "title", "release_date", etc. that get mapped into the class by Gson and then offers a method to get the actual "Movie" model object with the retrieved fields set. For TV Shows, there's also a special class called **TVShowSeasonJSON** that represents information about a specific season of the show.

29 Inputs



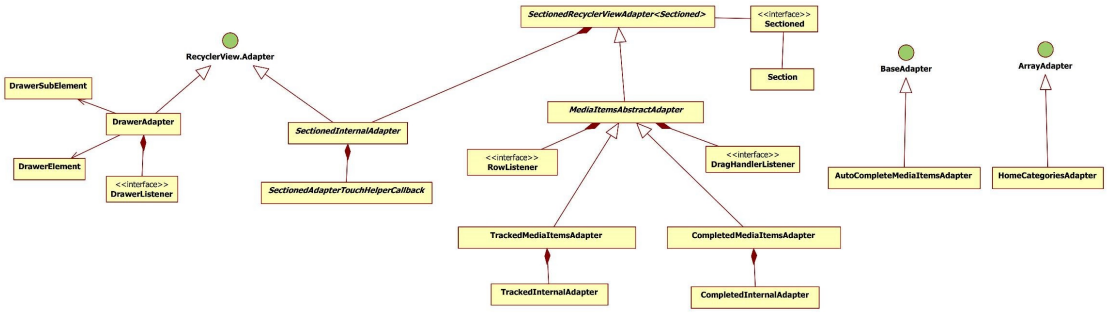
AbstractInput<T> and its implementations **ColorPickerInput<T>**, **DatePickerInput<T>**, **EditTextInput<T>**, **SelectDialogInput<T>**, **SwitchInput<T>** are wrappers for the built-in input Views that allow to translate between the field value and a model object (parameter T, e.g. media item or category) attribute and vice versa, thanks to the Callback nested class each one has. They also take care of saving their instance and restore it after a configuration change (e.g. device orientation). This way, in screens like FormAbstractFragment and SuggestionsAbstractFragment it is enough to build a list of inputs with linked callbacks to manage the whole lifecycle of the page (set initial values, set fields in the model object from the input values set by the user, etc.), for example:

```
AbstractInput<MediaItem> descriptionInput = new
    EditTextInput<>(true, view, R.id.form_description_input, new
    EditTextInput.Callback<MediaItem>()
{
   @Override
   public void setModelObjectValue(MediaItem mediaItem, String text)
   {
      mediaItem.setDescription(text);
   }

   @Override
   public String getModelObjectValue(MediaItem mediaItem)
   {
```

```
        return mediaItem.getDescription();
    }
});
```

30 Adapters



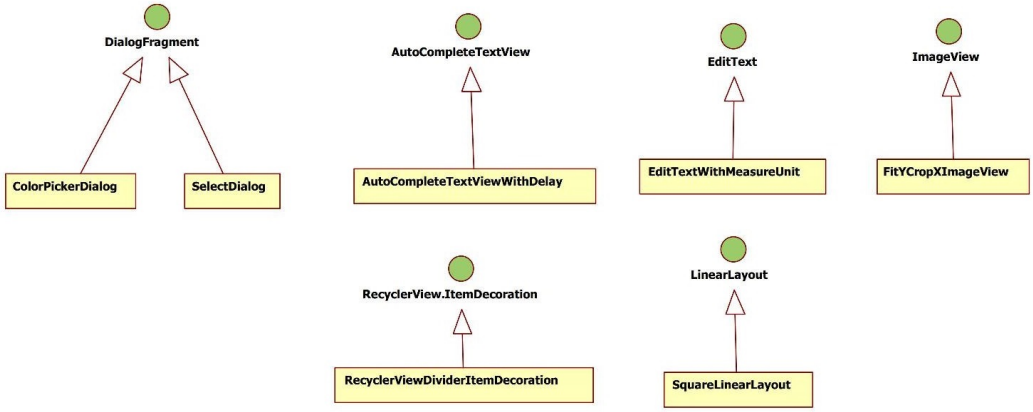
- **DrawerAdapter** is the adapter used for the navigation drawer. It contains DrawerElement objects (who in turn may contain DrawerSubElement objects) and displays them with the linked name and icon. Thanks to the **DrawerListener** interface the DrawerAbstractActivity can receive the selection callbacks.

- **SectionedRecyclerViewAdapter** and its nested class **SectionedInternalAdapter** allow to split a RecyclerView into sections, with support to swipe and drag&drop gestures thanks to the **SectionedAdapterTouchHelperCallback**. The elements passed to the adapter must implement the **Sectioned** interface in order to provide a **Section** for each of them. Its implementation **MediaItemsAbstractAdapter** is specific for media items, providing the general information to display them in the MediaItemsListFragment (e.g. split by importance level). The subclasses **TrackedMediaItemsAdapter** + **TrackedInternalAdapter** and **CompletedMediaItemsAdapter** + **CompletedInternalAdapter** allow to distinguish between the tracked media items list and the completed media items list.

- **AutoCompleteMediaItemsAdapter** is a simple adapter used in the AutoCompleteTextViewWithDelay.

- **HomeCategoriesAdapter** is a simple array adapter used in the HomeActivity.

31 Dialogs and Views



- **ColorPickerDialog** is a dialog that allows to choose a color among the given choices.
- **SelectDialog** is a dialog that allows to choose a (textual) option among the given choices.
- **AutoCompleteTextViewWithDelay** is an AutoCompleteTextView that starts the auto-complete procedure only after the user stops typing for a few milliseconds (useful to avoid too many external API calls).
- **EditTextWithMeasureUnit** is an EditText that shows a non-editable part at the end of the input.
- **RecyclerViewDividerItemDecoration** is a custom decoration for the line separating two items in a RecyclerView.
- **SquareLinearLayout** is a LinearLayout that has height equal to width.
- **FitYCropXImageView** is an ImageView that, given an height value, it scales the image on Y (i.e. the image is completely shown in height) and, if there's no space available, crops it on X (i.e. the image may not be completely shown in width), without losing the aspect ratio.

Part IX
Implementation: Testing Report

32 Devices

For the tests that require an Android device the following were used:

- Sony Xperia Z3 Compact (phone, real device), Android 5.1
- Sony Xperia Z3 Compact (phone, real device), Android 6.0
- Nexus 5 (phone, emulator), Android 6.0
- Nexus 4 (phone, emulator), Android 4.4
- Nexus 10 (tablet, emulator), Android 5.1

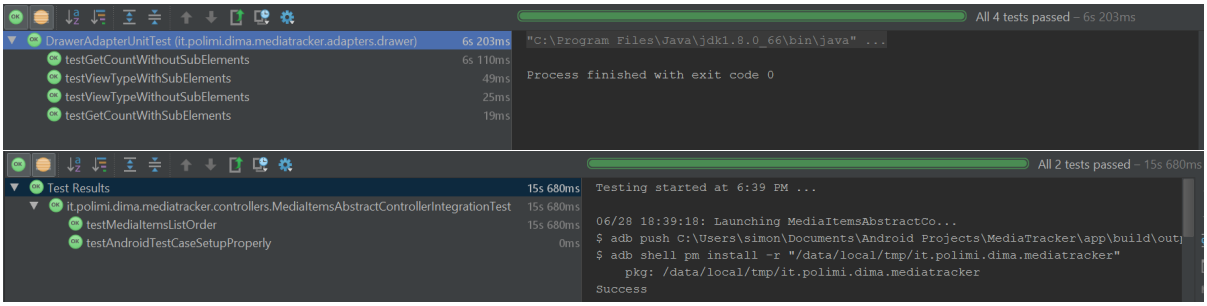
33 Unit and Integration Testing

For unit testing the Robolectric was used. It allows to perform tests directly on the development machine, without the need of a mobile device (real or emulated).
For integration testing the built-in AndroidTestCase was used. It allows to perform tests that can access Resources or other things that depend on Activity Context.

Tests:

- **DrawerAdapterUnitTest**: tests the navigation drawer adapter, which allows to display elements with, optionally, sub-elements
  - *testGetCountWithSubElements()*: tests the number of displayed elements when the currently selected one has sub-elements
  - *testGetCountWithoutSubElements()*: tests the number of displayed elements when the currently selected one has no sub-element
  - *testViewTypeWithSubElements()*: tests the displayed elements type (element or sub-element) when the currently selected one has sub-elements
  - *testViewTypeWithoutSubElements()*: tests the displayed elements type (element or sub-element) when the currently selected one has no sub-element
- **MediaItemsAbstractControllerIntegrationTest**: tests the controller for media items

- *testMediaItemsListOrder()*: performs thousands of random operations (add, delete, move and update importance level) on a list of media items to check if the persisted order in the database is consistent



34 UI Testing

For UI testing the Android Espresso library was used. It allows to test the user interface by specifying what gestures (taps, swipes, etc.) to perform on the available view elements.

Tests:

- **HomeActivityUITest**: tests the home page, with the list of all categories
  - *testCategoryClick()*: checks if a click on a category brings us to the media items list
  - *testHomeFAB()*: checks if clicking on the "+" button opens the form to add a new category
  - *testCategoryEdit()*: checks if editing a category the form is opened and the current data is correctly set
  - *testSettingsClick()*: checks if the "Settings" option in the action bar works
- **MediaItemsFormUITest**: tests the form to add/edit media items
  - *testInsertMediaItem()*: tests if the form works and if the media item is correctly stored in the database
  - *testBackButton()*: checks if the back button asks for confirmation if there are unsaved changes
  - *testPauseResume()*: tests if after onPause → onResume (e.g. activity loses focus because of dialog) the form is still displayed correctly and the inputs don't lose their data

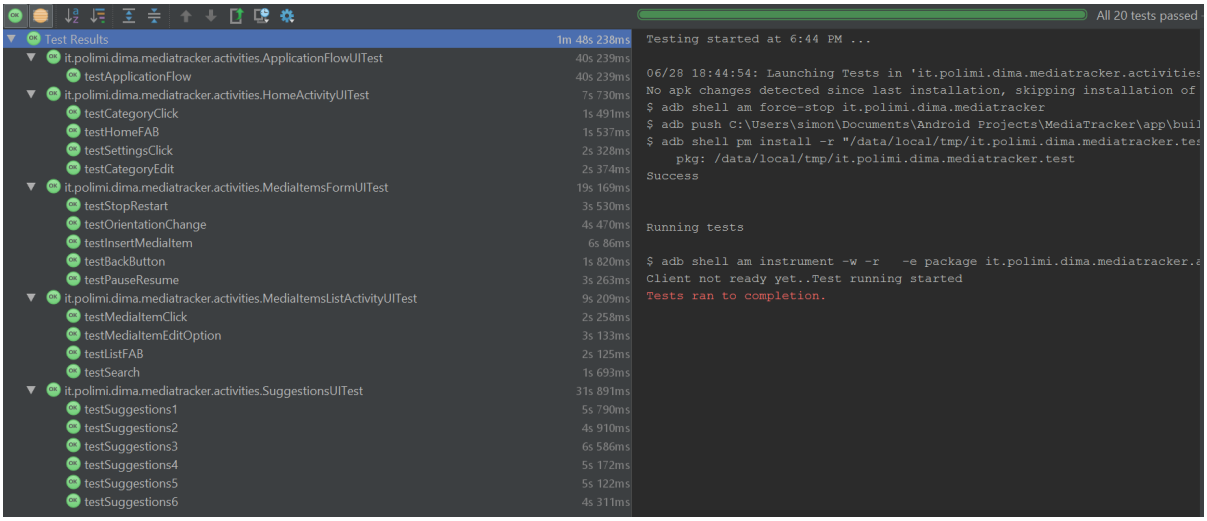
  - *testStopRestart()*: tests if after onPause → onStop → onRestart → onStart → onResume (e.g. activity is moved to the background and then restored) the form is still displayed correctly and the inputs don't lose their data
  - *testOrientationChange()*: tests if after a device orientation change the form is still displayed correctly and the inputs don't lose their data

- **MediaItemsListActivityUITest**: tests the tracked list of media items
  - *testMediaItemClick()*: checks if clicking on a media item brings us on the correct details page
  - *testMediaItemEditOption()*: checks if the context menu for each media item works and that clicking on "Edit" we go to the form with the current data correctly set
  - *testSearch()*: checks if the search function works
  - *testListFAB()*: checks if clicking on the "+" button opens the form to add a new media item

- **SuggestionsUITest**: tests the media items suggestions page
  - *testSuggestions1()*: asking for a tracked, very long, owned, "RPG" game should give only 1 possible result
  - *testSuggestions2()*: asking for a tracked, very long, owned, "NonExistingGenre" game should give no result
  - *testSuggestions3()*: asking for a tracked, non-owned, "Adventure" game should give 2 possible results
  - *testSuggestions4()*: asking for a tracked, long, "Adventure" game should give only 1 possible result
  - *testSuggestions5()*: asking for a completed (at least 3 years ago), "Action" game should give only 1 possible result
  - *testSuggestions6()*: asking for a completed (at least 1 year ago), "Action" game should give no result

- **ApplicationFlowUITest**: tests a "normal" application flow
  - *testApplicationFlow()*: the test performs these steps
    * create new category
    * try to create a category with the same name (error)
    * open that category
    * creates several media items with different importance levels

* checks that the list displays them all in the correct order and sections
* from the list, sets one media item as owned
* from the list, sets one media item as "doing now"
* in the list, swipes an element to set it as completed
* from the list, deletes a media item (also checking the confirm message)
* back button to go from the media items list to the home page
* delete a category (also checking the confirm message)

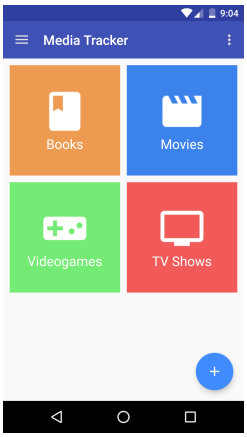


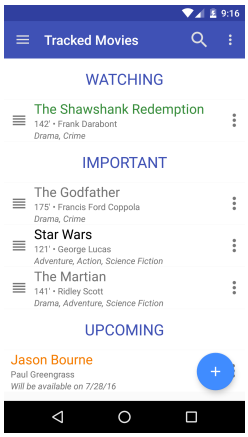
35 Other Testing

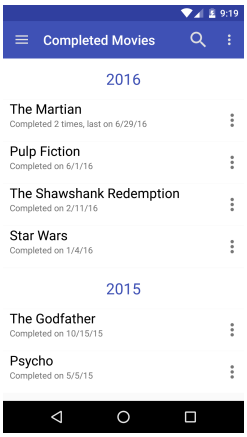
- For automated testing the built-in Monkey tool was used. It allows to inject a number of random UI events (taps, swipes, etc.) in the application.
- The tool Leak Canary was run to find some memory leaks.

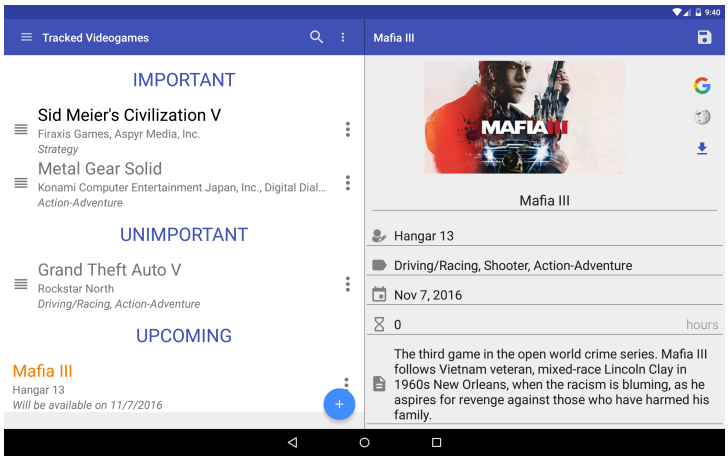
Part X
Conclusion

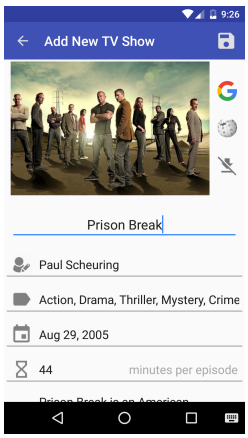
36 Image Gallery

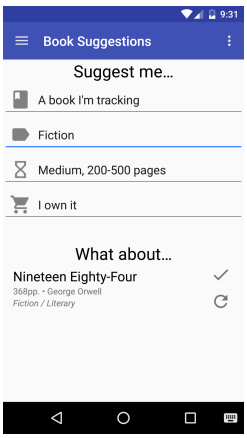












37 Used Tools

- **Android Studio** to develop the application
- **Genymotion** to run the emulators
- **Texmaker** to build this document using LaTeX
- **WhiteStarUML** for all UML diagrams in this document
- **Justinmind** for all UI prototype screens in this document