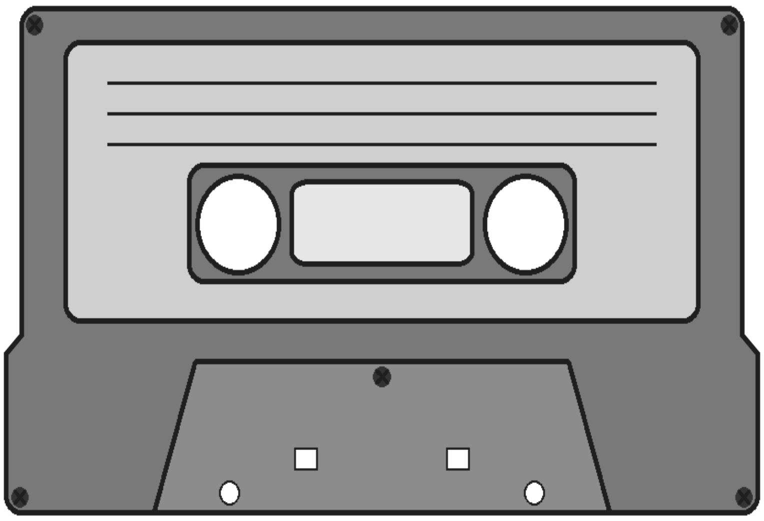
Technical University of Denmark - December 2014

DAP

Distributed Audiobook Player



By Ronnie Dalsgaard (s093487)

Abstract:

DAP is a player for audio books. It isn't the only audio book player on the market, but it solves a series of problems which you might encounter with other apps. I, my self, am an avid listener of audio books and DAP is an attempt to solve all the problems that I have come across. The main problems are; a user interface unsuited for night time use, a sleep timer buried in sub-menus and, most notable, an inability to continue listening on another device without having to fast forward to a specific time.

Table of Contents

Introduction

I have tried a variety of different audio book players, but none of them fully covered my needs. A lot of music players will allow me to synchronize a playlist between devices, but never my current position.

A typical scenario where I listen to audio books is in the bus on my way home from school or work. I will have placed my audio book on my Android phone and downloaded a player. The bus ride takes about 30 minutes. When I come home my phone often have very little battery left. I will open my tablet or computer to continue listening, but it is a nuisance that I will have to find the exact time I have reached on my phone.

Then, when I go to bed I will plug in my phone an listen to an audio book as I fall asleep. I will start a sleep timer, which stops the playback after 15 minutes. That way I can easily find the right time the next day.

On rare occasions I will wake in the middle of the night and listen a bit more.

The overall solution

The problem of listening to audio books while riding the bus can be solved with a mobile app. Preferably I would make both an Android app and an IPhone app, but since I have an Android phone, I will begin with an Android app. An Android app would also work on an Android tablet. But in order to play audio books on a computer I will need to make a Desktop version. I have decided to make the Desktop version in Java in order to reuse as much code as possible.

Due to time constraints I will have to settle for a limited version of both the Android app and the Desktop app. Especially the Desktop app will be just proof of concept.

In order to get a decent user interface I will do minor tests using friends and family as test subjects.

In stead of setting up a server, I am considering storing a file in the users own Dropbox. This is something which requires some research.

I have decided to adopt a some what loose interpretation of UP (Unified Process). I don't need a very rigid work structure. Agreeing on the best way of doing things is fairly easy, since this is a one man project.

Requirements

From the scenario I have deduced the following requirement specification.

Functional requirements

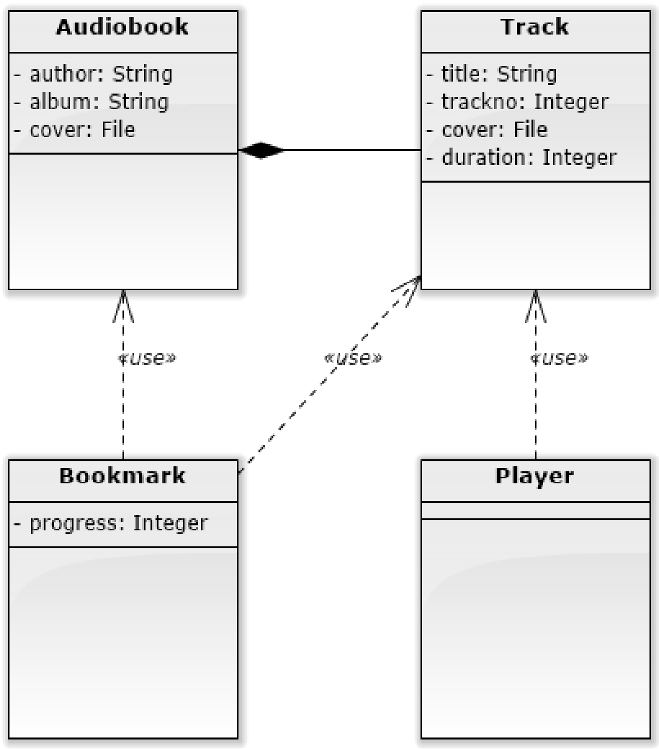
1. Play audio books
    1. Play
    2. Pause
    3. Next / Previous track
    4. Forward/ Rewind (1 minute or user specified)
2. Select audio book
3. Sleep Timer (15 minutes or user specified)
    1. Start Sleep timer
    2. Stop Sleep timer
4. Synchronize progress
    1. Upload
    2. Download
    3. Merge / Solve conflicts

Non-functional requirements

1. Primary features should be readily available - 1 click.
    1. Play / Pause
    2. Start Sleep timer
2. The user interface should be comfortable to look at - even at night
   - This goal is reached if the background color is black
3. The user interface should be responsive
   - No load time longer than 10 seconds
   - No load time longer than 1 second without a progress bar or similar
4. The user interface should be simple and easily navigable
   - Test subjects can use basic features without help

The domain

Based on the case and requirements, I have come up with this domain model:



Entity descriptions:

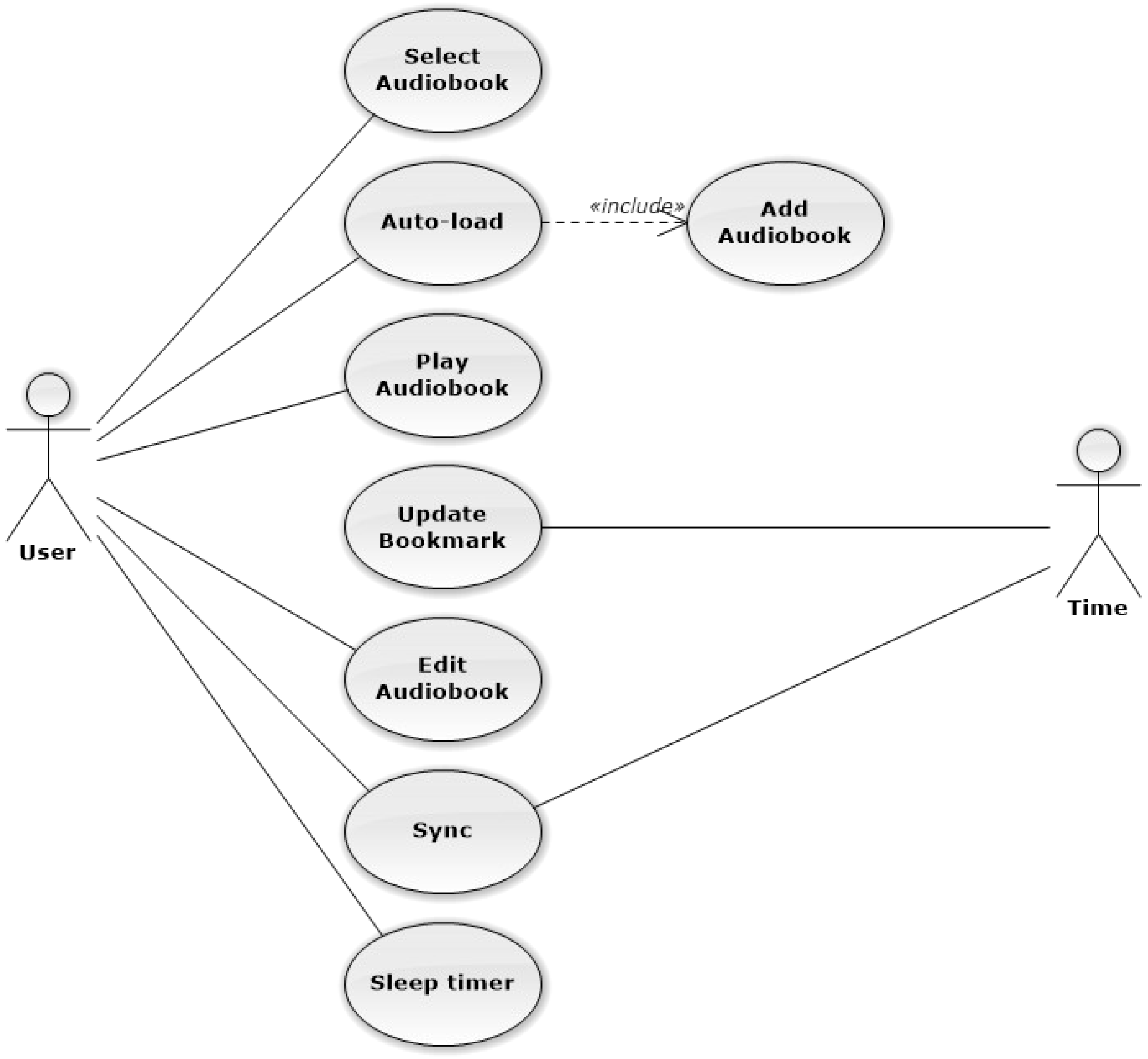
- **Player** (MediaPlayer)**:**
  The App is referred to as DAP. The actual Player is a part of the Android SDK (and/or JavaFX). The player can be loaded with a media file (in DAP only mp3-files are allowed), which can then be played.
- **Audiobook**[1]**:**
  An audiobook is a collection of tracks, which is given a title and an author, both of which are just text. The combination of author and title is used as unique ID. Furthermore an audiobook can hold a reference to an image - the cover.

1 I have opted to use the term Audiobook instead of Audio book, even though it is misspelled, because it simplifies the concept.

- **Track**
  A track is a reference to a mp3-file and a title. A track also has a duration - when a track is loaded into the player the duration is saved. Loading a track is fairly time consuming, so the duration isn’t requested lightly. Unfortunately this means that the total duration of the audiobook can’t be determined until the last track is loaded.
  (Tracks also have a cover. The title and especially the cover is primarily to be used in case I later decide to read ID3 tags)
- **Bookmark**
  A bookmark is an author and a title to identify an audiobook. It also holds a track number and a progress, both are integers. A bookmark is worthless without a corresponding audiobook.

Use case

With the functional requirement as a basis I have deduced the following use cases



Brief use case descriptions:

- **Select audiobook / bookmark:**
  Selecting an audiobook creates a bookmark with the track number and progress set to 0. When this bookmark is activated the referenced track is loaded into the player. Also, data from the audiobook is displayed.
- **Play audiobook / bookmark**
  Once an audio file is loaded into the player, it can easily be played (and then paused).
- **Update a bookmark**
  When an audiobook is playing, the corresponding bookmark is automatically updated (or created if it doesn't already exist). The frequency with which bookmarks are updated is variable, but when the player stops playing it is only updated once more.
- **Sync (Importing and exporting bookmarks to the cloud)**
  The one special feature that makes DAP stand out, is the fact that bookmarks can be uploaded and downloaded on other devices. Only bookmarks are sync'ed - if a downloaded bookmark refers to an audiobook which is not on the device, that bookmark is discarded.
- **Sleep timer (start + stop)**
  I listen to audiobooks in bed - it helps me fall asleep. The Sleep timer makes it a lot easier to find how far along I have heard.
- **Edit audiobooks**
  On rare occasions audiobooks have errors. Maybe the Auto-load got the wrong author name or title, maybe there is an error with the charset and some of the special characters are read wrong.
- **Add audiobooks:**
  It is necessary to add an audiobook before it can be played. Adding an audiobook is the task of collecting a number of audio files, giving them a title and an author and maybe a cover image.
  DAP has a function to automate this task if certain parameters are met.
  (For further details, see the paragraph about the feature Auto-load.)

- **Auto-load**

  This feature creates an audiobook or audiobooks from a folder. It locates a folder which contains audio files. These files will be added to the playlist of a new audiobook, The title of the audiobook will be the folder name, and the parent folder name is used as author. If the folder also contains an image file (.jpg or .png) which is called AlbumArt, ignoring capital letters, this file is used as cover.

I have decided not to elaborate further on the use cases. Some of the use cases will depend on the UI and others will depend on external systems and libraries.
I would rather keep the definitions a bit vague and stay with them through out the process, rather than change them all together. This is a way of doing things that I have found to be successful in other recent projects.

This list of use cases does not fully cover the requirements, but when these use cases have been addressed, I believe the remaining functionality will be trivial to implement.

Risks

I have made an overall risk analysis, to avoid major caveats. I have then tried to counteract the worst cases. Further details can be seen in appendix A.

Tools and Techniques

JavaFX preferred over GWT and Swing

First of all, I opted for writing the Desktop version in Java as well, since I could save a lot of time by reusing as much code as possible from the Android version. After this decision was made I had to choose a framework for the application. I know Swing and have used it before, but I always found it lacking in main design issues - Especially when it comes to placing one component relative to another. Setting a specified size to a component, is also quite difficult.

JavaFX and GWT I have heard of, but never used. I decided to do some research before choosing.

I find Swing to be terrible and error prone. GWT can only be displayed in a browser. Although I could create a swing application which is just a plain browser, this doesn't seem like a good solution. Instead I opted for JavaFX. It is oddly similar to android and very much code can be reused. Besides JavaFX also has the ability to play media files and no further plug-ins are required.

JavaFX Scene Builder is a tool to make user interfaces.

JavaFX works pretty much the same way as Android. This example shows how the main class sets a click listener to a button in both Android and JavaFX:

Android

```
Image Button btn_next = (ImageButton) findViewById(R.id.seeker_btn_next);
btn_next.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {

      }
});
```

JavaFX

```
Button btn_next = (Button) loader.getNamespace().get("btn_next");
btn_next.setOnAction(new EventHandler<ActionEvent>() {

      @Override
      public void handle(ActionEvent event) {

      }
});
```

Eclipse preferred over Android studio

Eclipse will be my main tool. I know my way around Eclipse - I have even tried out ADT (Android Developer Tools). Eclipse is far from error free, but "better the devil you know".

It is also worth mentioning that I also can use Eclipse to create JavaFX projects. However I will use a separate copy of Eclipse for each of the projects. This is just a prudent safety precaution when working with Eclipse.

The alternative was Android studio which is based on IntelliJ. I have heard that it should be quite good. It is as of September 2014 still in beta test and not yet considered stable.[2] It is rumored that Google will release a stable version in December 2014 - That is a bit late for me.

2 Ref.: https://developer.android.com/tools/revisions/studio.html

Git

As explained in the previous chapter I have decided to use Git and Github. I could just as easy have chosen another repository server, but I already have a Github account.

3 layer model

Both Android and JavaFX allows me to write the UI in XML and keep it separate from the rest of the code. I will also isolate the base entities, but the remaining classes will also be divided into packages.

I intend to rely heavily on the design pattern "Protected variations" which is a part of GRASP. The best analogy of this pattern is a terror cell. A single cell works independently towards a specific goal. No matter what anyone else does it will keep going.

An example from this project could be the class Time. Time is actually an idea I had for another project, but found to fit DAP as well.

Design

Sync

As I see it, the most difficult part of this project is that I need to store information in the cloud. My first idea was that; everybody has Dropbox (or can get it for free) , so I tried it out.

Dropbox

Dropbox on a Desktop would be simple file handling, so the difficulties would be on Android.

Dropbox has an Android API which I downloaded. It was really easy to use, most of the code that I needed could be found in the Dropbox API tutorial. It worked and it is easy to use. It had just one error: I create a new file in Dropbox, but only locally. Dropbox decides when to upload. LIke the desktop version of Dropbox, the API will sync when the system is idle. This is generally speaking a great idea, but it doesn't fit my case.

Since the API will only upload if the app is running, I cannot upload a file and then immediately kill the app. If I do so, the file will not be uploaded, because the system never had time to become idle. I need instant upload. - This is a deal breaker.

Google Drive

I then turned to Google Drive. Anyone who has an Android also has a Google Drive. I found that Drive has a Web API, so worst case I could make a HTTP request to up- and download files.

Google Drive also has an Android API. In order to use this, a couple of conditions will have to be met:

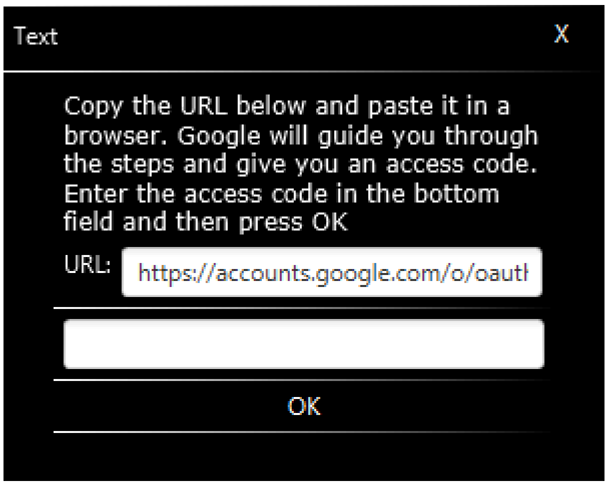
1. The user must have Google Play Services installed on the device - everyone has.
2. Google Play Services must be up to date. If it is not, the API itself will guide the user to update - I don't have to do anything.

(It is a fairly recent development from Google that places Google Drive along with other Google apps in the Player Services. Play Services used to be just Play store.)

Drive has the same exact problem as Dropbox - it will only upload when idle and while the app is running, but with one important difference. The app that needs to be running isn't DAP, it's Play Service, which always (almost) runs in the background.

Authorizing with Google Drive

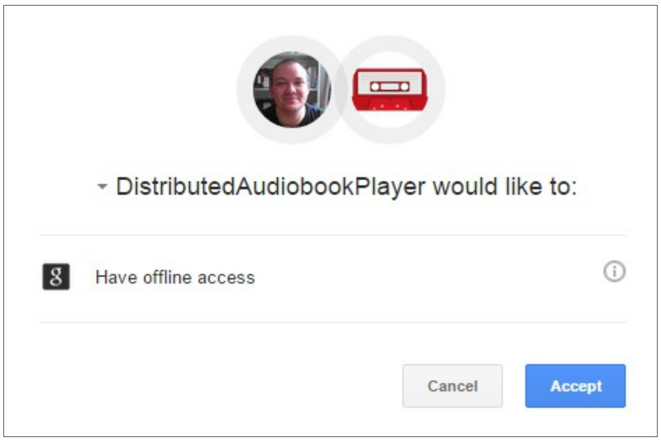
The user has to authorize DAP with OAuth to have access to Drive, but the API handles this as well, but on the desktop app it is far more cumbersome: There is no java API for Drive. I have to use the Web API. When the user has to authorize, all the API gives me (the developer) is an URL. I then have to get the user to start a browser and visit this page, authorize and get an access key. This access key, the user then has to enter into the app, before I can access Drive and get an access token.


Text X

Copy the URL below and paste it in a browser. Google will guide you through the steps and give you an access code. Enter the access code in the bottom field and then press OK

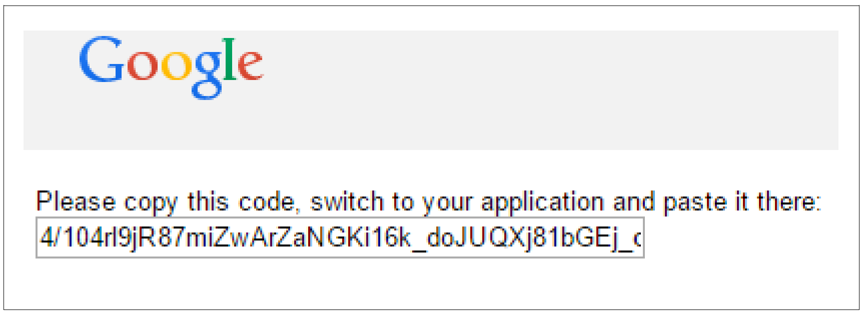
URL: https://accounts.google.com/o/oautl

OK


DistributedAudiobookPlayer would like to:

Have offline access

Cancel Accept



Please copy this code, switch to your application and paste it there:
4/104rl9jR87miZwArZaNGKi16k_doJUQXj81bGEj_c

This only has to be done once. DAP will save the access token and use it for all future communication with Google Drive. I can only hope that Google will change this way of authorizing in the future, as it is not ideal for this kind of app.

Using files in Google Drive

Early on I decided that audiobook, track and bookmark should only consist of attributes that can be seen as text. That is:

- Strings: Obviously - author, album and track title
- Numbers: integers like track number and progress can be easily stringified.
- Files: are represented by their path - audio files and covers

(This changes the entities a bit compared to the domain model)

I then use Gson (Google's json encoder) to encode these entities when saving them to files.

Each bookmark or audiobook is parsed to json and appended to the file. That means that the file content is not one big json string, but a collection of json strings, which must be parsed separately. This makes the file much more readable to humans, but it also means that line-breaks are used as a split token. However, since line-breaks are converted when up- or downloaded to Google Drive, they become unreliable. I had to ignore line breaks and end each line with my own token (/END), and use that to split the bookmarks from each other.

Accessing Google Drive Files from separate applications

After having created a bookmark file from the Android app, I wanted to access, read and update this file from the desktop app.
I have successfully gained access to Google Drive, but when requesting a list of files a lot of files are missing. Even though I can read "bookmarks.dap" I don’t have access to the folder "DAP" (I don't actually use this folder). This might be a problem (it might not). The biggest part of the problem is that I don’t know why I can’t see all

the files I expected.
The scope I used to gain access should give me global access - or so I thought. Apparently it doesn’t.
I found that using the exact same (limited) scope as the Android app gave me access to the exact same files. Even though this is more avoiding the problem than actually solving it, I consider the matter resolved.

The app is identified by a project number which can be found when the project is created on Google's Developer Console. That means that even though DAP for Android and DAP for desktop is in fact 2 different apps, Google accepts that they are the same, because they have the same project number.

Name clash **in DAP for Desktop**

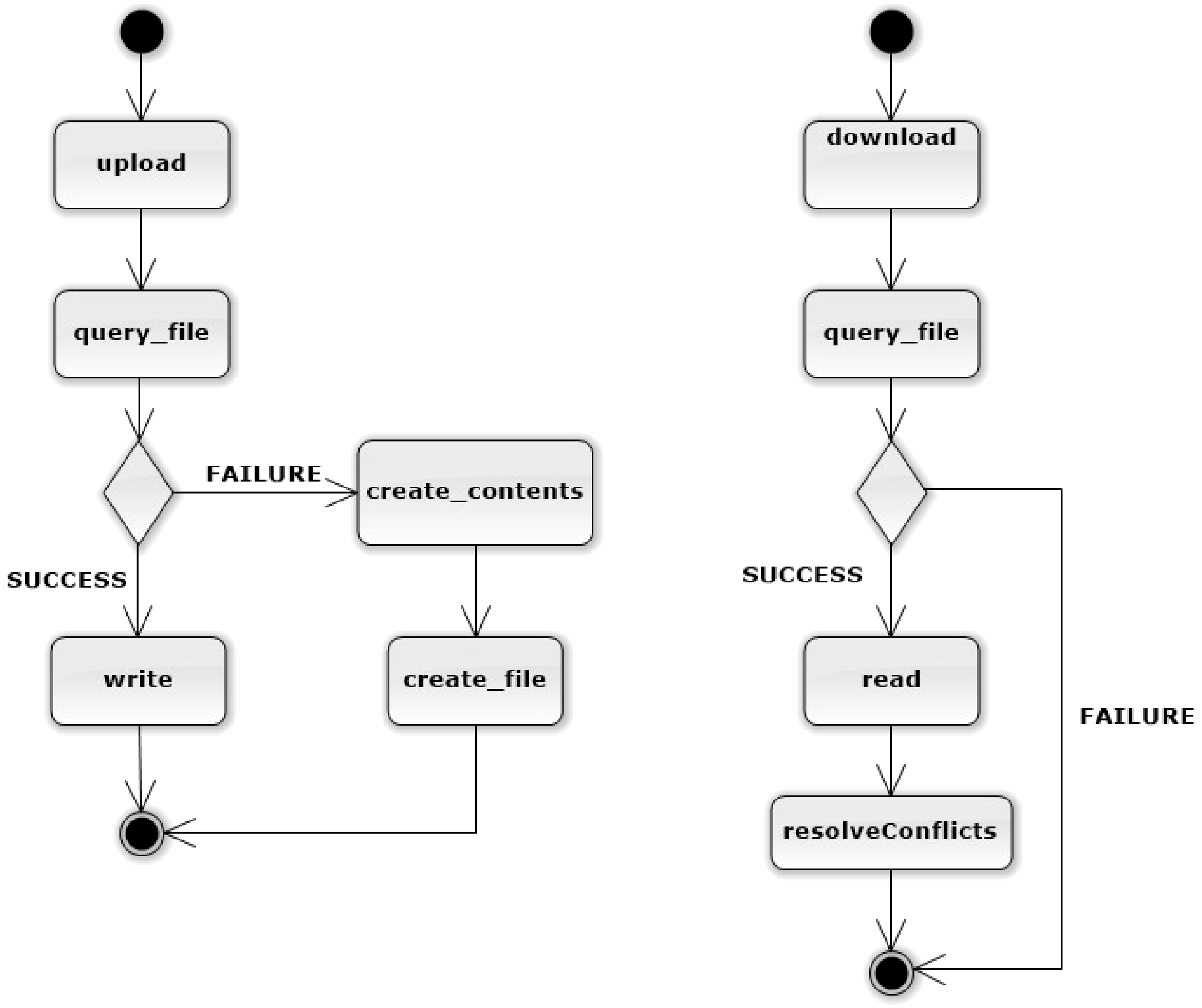
A file retrieved from Google drive is of type "com.google.api.services.drive.model.File" and the content is wrapped in a "java.io.File". If you don't define at least one of the types explicitly every time, this causes a name clash. This might not actually be a real problem, but it is certainly a nuisance. I have wrapped the File from Google Drive in my own type DriveFile. DriveFile is a composite - it contains a com.google.api.services.drive.model.File and all calls to DriveFile is redirected to the contained File. The composite pattern is not completed - as I only need a few methods, I have only implemented those.

Likewise with the DriveFileList.

In the Android API, files and file lists are called DriveFile and DriveFileList, I didn't invent the names.

Upload and Download

The process is a little different from Desktop to Android, but the basic principle is the same:



Conflict resolution

Note that when uploading there is no conflict resolution. The content of the file is simply replaced.

The conflict resolution has it's own class, which means that it is easy to change or replace.

I have decided to keep the rules of the resolution as simple as possible - I don't even compare the old with the new, I just use the new regardless.

```
public Bookmark resolveConflicts(Bookmark oldData, Bookmark newData) {
      return newData;
}
```

Another solution would be to use the one which is furthest along. That is, the bookmark with the highest track number - or the highest progress if the track number is the same.

Player

Before worrying about the player, DAP needs something to play - it needs audiobooks.

- **Home folder vs single audiobook creation**
  In earlier versions audiobooks were created, one at a time, by the user, although heavily assisted by the system. This allowed the user to inspect every audiobook for errors during their creation, but preliminary user testing indicated that this is too much work for most users. Instead the user now sets a home folder and allows the system to auto-create all audiobooks at once. This takes a couple of seconds (the user is showed an infinite progress bar - to say that the system is working and is not crashed - also this is done in a secondary thread, so the system can be halted by the user). The amount of time this takes is off course dependent on the number, and sizes, of audiobooks in the home folder. This has to be done only once, after which the data is stored persistently. Only when new audiobooks are added will this procedure have to be repeated.

- **Managers**
  Internally bookmarks and audiobooks are stored in BookmarkManager and AudiobookManager. Both managers are singletons, but a few too many methods are static, and shouldn't be. They need a clean up.

- **Bind service vs Start service**

  I decided very early to make the player a service. That way it runs in the background even if DAP is closed.

  There are 2 ways of using a service: Bind to it or start it.
  Access to player service upsides and downsides:

	Upsides	**Downsides**
Bind to service	Service is easy to use	Service terminates when unbound
Start service	Service keeps running in the background until manually stopped	Manipulation is difficult

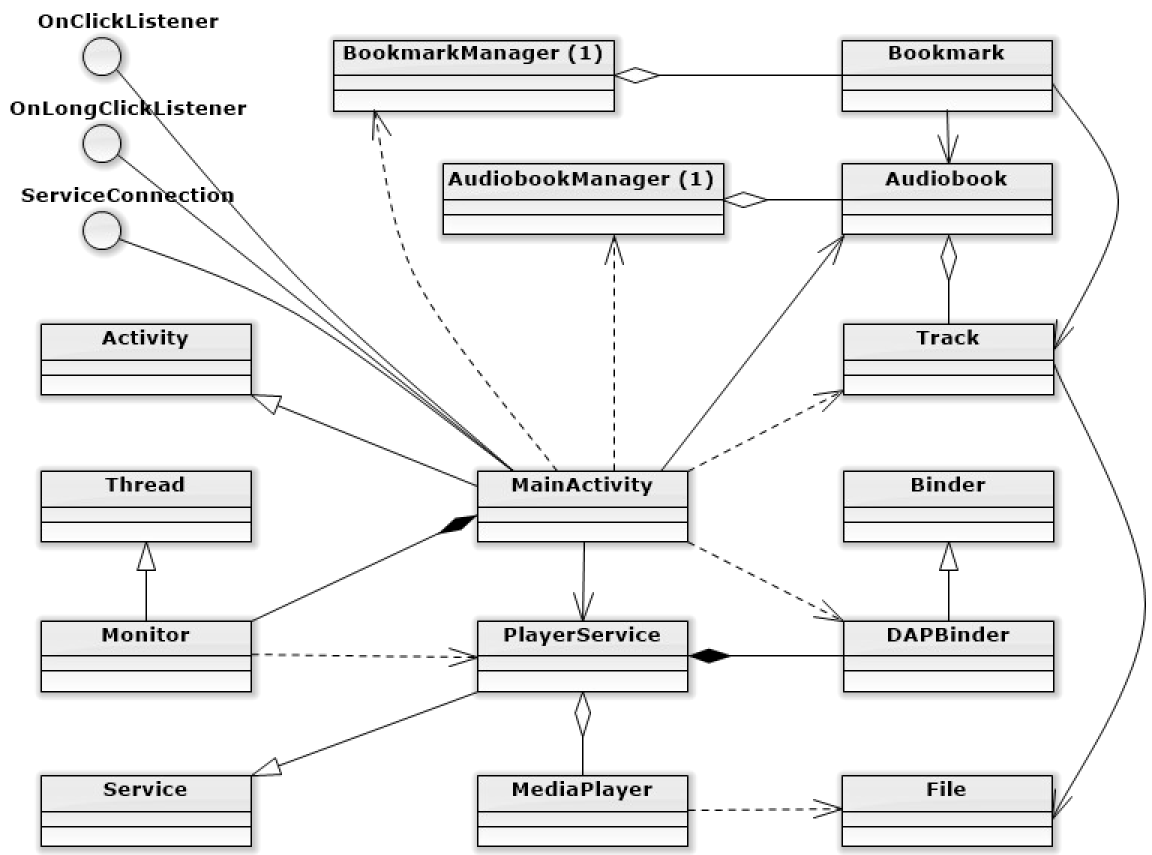
  When binding to a service it is easy to manipulate it. Through a binder object you get access to the service itself and can call methods on it. However when the bound Activity is destroyed the service is also terminated.
  When a service is started using startService() it will run until stopService() is called. But all commands have to be sent using the same method - startService. The attached Intent can contain data, such as action strings, but all data has to be parceled or serialized. The entities audiobook, track and bookmark can be serialized, so this is a viable solution, but tedious and cumbersome.
  I decided to do both: In the main activity the player service is started with startService(), but no action is requested. Then when needed, any activity can bind to the service and use it, without destroying the service when unbound.

- **PlayerService is observable**

  When the main activity binds to the playerservice it adds itself as an observer, using a standard oberserver pattern. That means that whenever the player changes state, the main activity is notified and can update the UI. It doesn't matter how the change of state was initiated.

- **Associations between MainActivity and PlayerService**
  This diagram shows an overview of the MainActivity, PlayerService and their surroundings. It illustrates the complexity of playing a track - it's not just an activity and a player. The diagram below isn't even complete (but almost) and it only show how the MainActivity communicates with the player - not the other way around (Observer pattern - see previous paragraph).

  Since the making of this artifact, the code has been subject to minor changes.



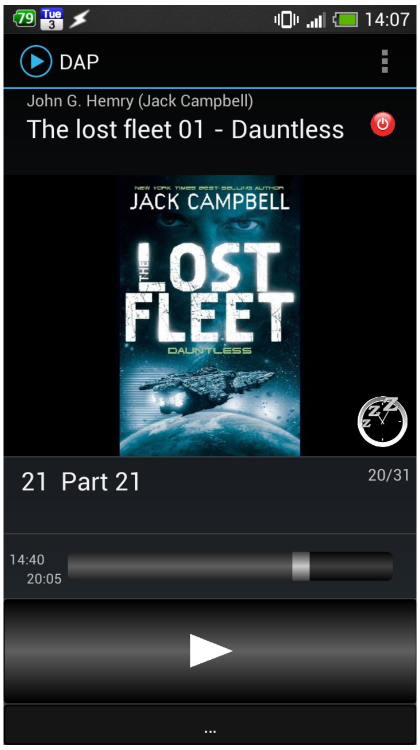
BookmarkManager and AudiobookManager are singletons as marked in the diagram with "(1)".

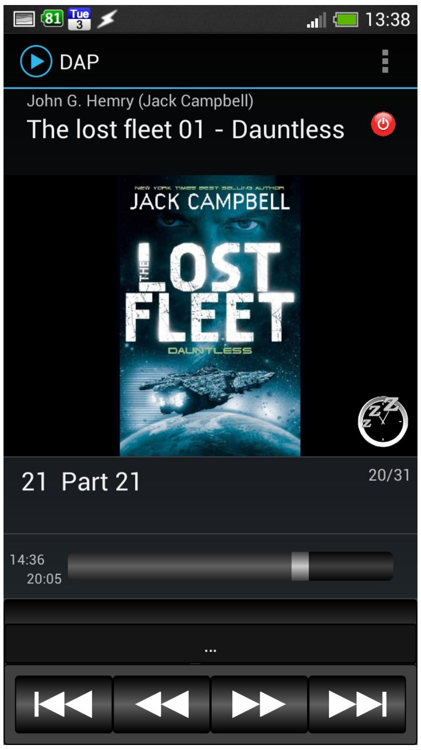
User interface evolution

The UI is kept dark, because it is used at night (and it conserves battery). This also fulfills one of the non-functional requirements

Version 0.1.0

A very early - first attempt, at a usable interface. This interface bears a remarkable resemblance to Mortplayer - an audiobook player I used before.





- **Icon**

  This version had no icon, so I used a default. - Not important.

- **Basic information and buttons**
  The screen consists of basic information; author, album, cover, track number and title, progress and a very big play button.
  In particular the cover is important as it is the way a user will recognize an audiobook at a glance.
  Play / Pause is the main feature, hence the big button. At the very bottom of the screen there is a drawer with more buttons (picture on the right). The drawer icon would eventually have been a proper overflow icon, but it was discontinued.

  Buttons looks as if they are convex. This brings the thoughts back to cassette players which is how audiobooks used to be distributed. It is, however, not the Google way. It is not how it should be on an android app.

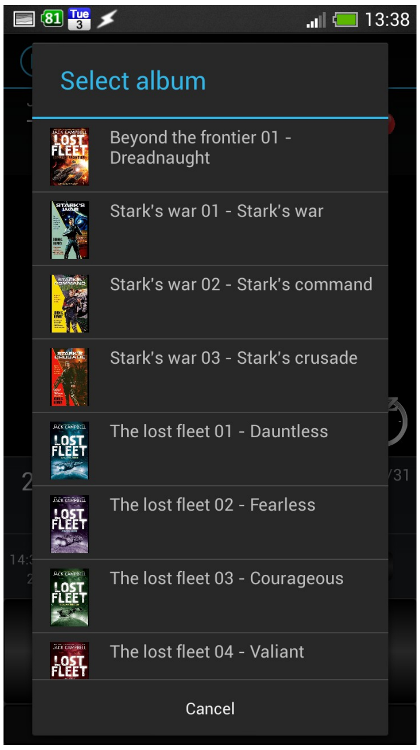
- **Track number**
  The track is labeled by number and a title, but further more a “track number / track count” indicates how far along the story has come. This however is not very visual and, unless you actually look for the information, you wont notice this detail.

- **Exit button**
  The player had an exit button - this was simply a mistake. Activities can not run in the background (that is why the player is implemented as a service), so when the user hits the home button it is closed. It is true that some resources are kept available, in case the user restarts the app, but these resources are freed as needed by the oprating system.

- **Sleep timer**
  The sleep timer is located on top of the cover in the lower right corner. Usually the cover doesn’t take up this space, but just in case, the sleep timer has a higher Z-coordinate - that is the button is visible on top of the cover. The button is semi-transparent.
  When the sleep timer is active and counting down the button glows red and an indicator appears on the progress bar.
  This was discontinued, but tests shows that it probably should have stayed.

- **The progress bar**
  The progress bar is big and bulky, but not very pretty. The progress (and duration) is also shown as text. This is more accurate, but takes some of the width of the progress bar, making that a little less detailed. In opposition to the track number, this is a very visual way of showing the progress, albeit only of the current track.

- **Audiobooks and bookmarks**



13:38

Select album

Beyond the frontier 01 - Dreadnaught

Stark's war 01 - Stark's war

Stark's war 02 - Stark's command

Stark's war 03 - Stark's crusade

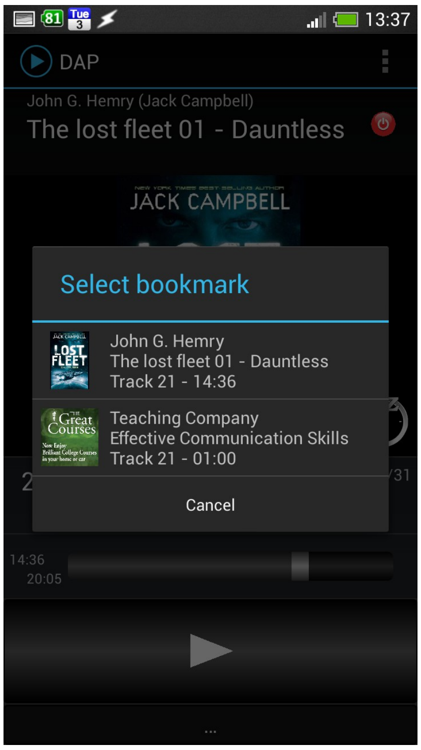
The lost fleet 01 - Dauntless

The lost fleet 02 - Fearless

The lost fleet 03 - Courageous

The lost fleet 04 - Valiant

Cancel



13:37

DAP

John G. Hemry (Jack Campbell)

The lost fleet 01 - Dauntless

Select bookmark

John G. Hemry
The lost fleet 01 - Dauntless
Track 21 - 14:36

Teaching Company
Effective Communication Skills
Track 21 - 01:00

Cancel

14:36
20:05

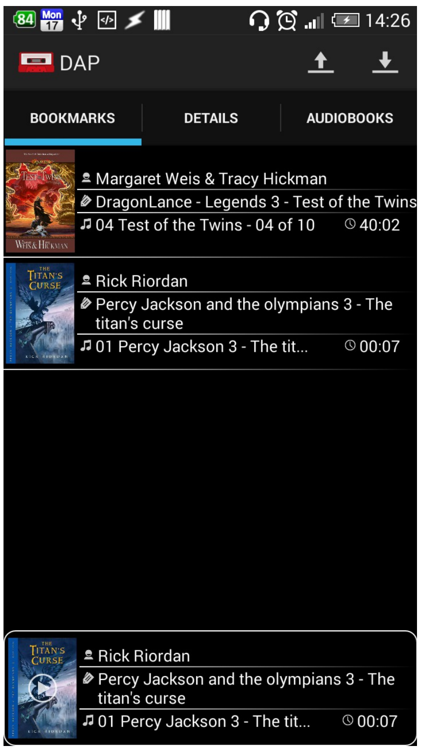
Both audiobooks and bookmarks are identified by author and album. Bookmarks also have a track number and a progress. Both are also displayed with the corresponding cover. Although this cover is too small to read the text, it is an easy visual cue for the user to find a specific item.
In this early version both audiobooks and bookmarks are selected using dialogues.

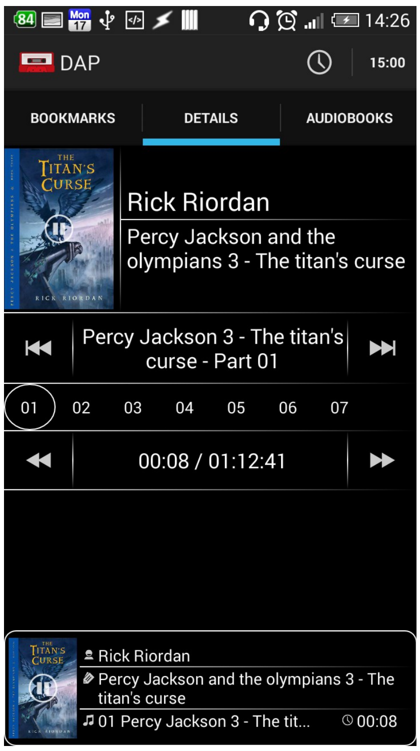
- **Cover**

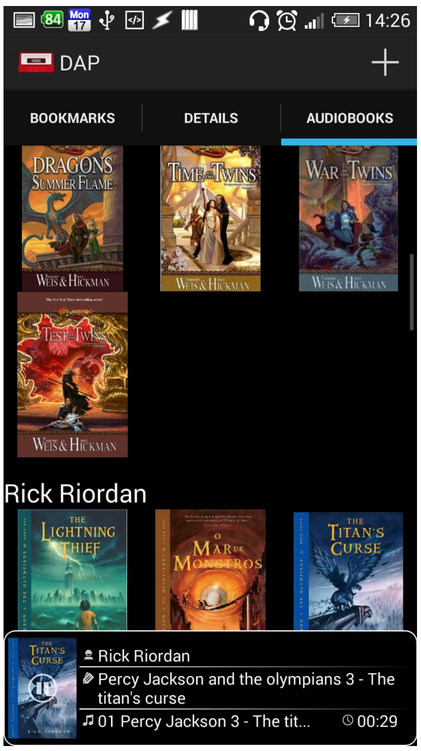
  Covers are the way users recognize an audiobook without having to read author and album, it enhances the experience for the user.
  Covers are in fact not required. If an audiobook has no cover, a default cover is shown. This however is not the main idea behind the system and it is also highly unusual that audiobooks have no cover image.

Version 0.2.0

Version 0.2.0 has 3 pages. The user can swipe between pages or use the tabs in the top of the screen. These tabs are also used as breadcrumbs, letting the user see where he/she is.


14:26
DAP
BOOKMARKS
DETAILS
AUDIOBOOKS
Margaret Weis & Tracy Hickman
DragonLance - Legends 3 - Test of the Twins
04 Test of the Twins - 04 of 10
40:02
Rick Riordan
Percy Jackson and the olympians 3 - The titan's curse
01 Percy Jackson 3 - The tit...
00:07
Rick Riordan
Percy Jackson and the olympians 3 - The titan's curse
01 Percy Jackson 3 - The tit...
00:07


14:26
DAP
15:00
BOOKMARKS
DETAILS
AUDIOBOOKS
Rick Riordan
Percy Jackson and the olympians 3 - The titan's curse
Percy Jackson 3 - The titan's curse - Part 01
01
02
03
04
05
06
07
00:08 / 01:12:41
Rick Riordan
Percy Jackson and the olympians 3 - The titan's curse
01 Percy Jackson 3 - The tit...
00:08

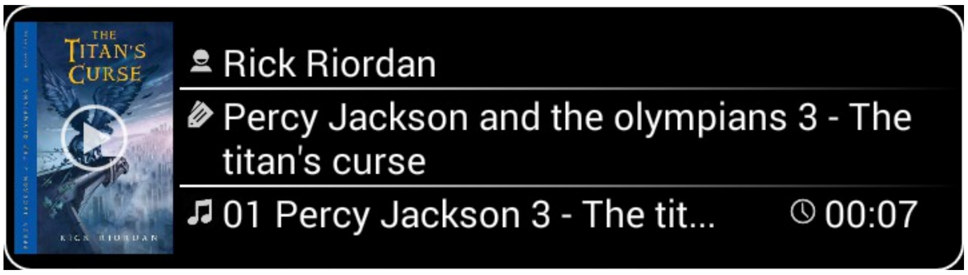

14:26
DAP
BOOKMARKS
DETAILS
AUDIOBOOKS
Rick Riordan
Rick Riordan
Percy Jackson and the olympians 3 - The titan's curse
01 Percy Jackson 3 - The tit...
00:29

- ***Swipe-Views and Fragments***
  To begin with quite a few components were made as Fragments. Although they weren't reused, they were self-contained. That means that all controller logic was built in, and the Activity was much smaller and more manageable. Some interaction between the fragment and the activity was necessary. Using a FragmentManager, the Activity could get a hold on the fragment. The other way around I used an Observer pattern.
  The application had, at this time, 3 main screens. In order to improve user experience I used a standard android navigational pattern - Swipe-Views. Swipe-Views is a ViewPager with a number of Fragments and SectionsPageAdapter. When the user swipes left or right on the screen, the adapter fills the ViewPage with one of the given Fragments.
  I also added tabs to switch between views, the tabs are not only navigation

buttons, they also work as breadcrumbs. The user can easily see which screen is active and how to get to another.
Using this navigational pattern I had to convert the 3 Activities to Fragments, which was easy, but whereas using Fragments in Activities is simple, using Fragments in Fragments is problematic. According to Google it is possible since android version 4.2. However, since it was problematic (although probably not impossible) and my only reason for using Fragments was to avoid code clutter, it felt easier to discontinue the use of fragments and instead clean up the code.

Actually I later found that Swipe-Views was overkill and reverted to a single main activity. See next chapter (Version 1.0.0).

- **Miniplayer**
  All pages have a "Miniplayer" at the bottom of the screen.



Rick Riordan
Percy Jackson and the olympians 3 - The titan's curse
01 Percy Jackson 3 - The tit... 00:07

- The cover on the Miniplayer has a play/pause icon on top, because the cover is clickable, in fact the entire Miniplayer is one big play/pause button. (There is also a cover/play button on the details screen.)

The miniplayer is a part of the 3 fragments. Although it is just one include statement, each fragment screen has its own miniplayer. This is definitely a usable solution, but it looks odd.
The miniplayer is just a UI for the player service which runs in the background. 3 miniplayers is no more resource demanding than 1, but obviously some resources are required to create a new miniplayer every time the user swipes to another screen. There is a small time delay from when a miniplayer is destroyed to the next miniplayer is created and has connected to the player service. This delay is not a big problem, but I would still like to get rid of it - it looks unprofessional.
With some effort I succeeded in moving the miniplayer out of the fragments and place a single miniplayer below the ViewPager, so that it is not affected by swiping to another screen. It required a major redesign of elements, putting a lot more responsibility on the main activity instead of in the fragments. The fragments also have to be able to call back to the main activity - I did this using an observer pattern.

- **Separators**
  Separators are just an attempt to make the UI more visually pleasing. They are used to separate elements, since elements can be multi-lined.
  I used gradient to make the separators less dominant.
  The vertical separator gradiates from black to white in the center and back to black.
  The horizontal separator changes from white on the left to black on the right.
  I could also have made the horizontal separator white in the center and black left and right, but I decided that that has been done too many times before and I wanted to do something else.

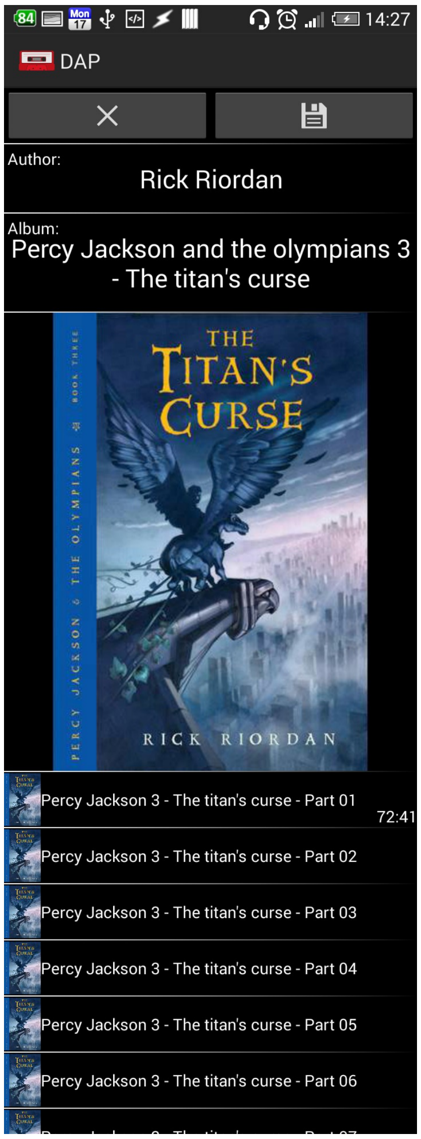
- **Bookmarks**
  Bookmarks are pretty much unchanged. A few icons have been added, but other than that changes are neglectable.

- **Audiobooks**
  Audiobooks are now shown in a grid instead of a list. The grid has headers with the author name, but the album title has been left out and the user will have to rely solely on the cover image.
  A major change in this version is that audiobooks aren’t hard-coded and loaded automatically. The user has to add audiobooks individually. This is something that the user tests showed to be a bad idea, and it will be changed. When adding a new audiobook the user selects a folder and the system will find all audio files in this folder or sub folders (See the feature "Auto-load"). Selecting a folder is done by using the “select file UI”

- **Controller/Details**
  The controller screen has been redesigned, but the functionality stays the same. The play button has been removed but the cover is a play button as it is on the Miniplayer. And as the Miniplayer the cover on the controller has an overlay with a play/pause icon, making it easy to identify as a button.

- **Tracks and progress**
  There is nothing worse than believing you are in the middle of a book and then suddenly it ends and you don’t have the sequel at hand. The user needs to know how much is left of the book.
  That is why I decided to show all the track numbers in a grid and circling the current track. This is much more visual than the simple “track number / track count”. I feel that the user will get an impression of how much is left of the audiobook with just a glance at the screen.
  Knowing how much time is left of the current track isn't as important as knowing how many tracks remain. For the time being the progress is shown as text: “progress / duration”. I plan on eventually using a progress bar of some kind, but it has a low priority.

The screen below is a detail view of a single audiobook containing the author name, album title, a cover image and a list of tracks represented by its file name. For the moment a track has a cover image. I intend to change this to a trackno. If a track has been loaded into the player, the duration of the track is saved and displayed on the right hand side.



14:27

DAP

Author:
Rick Riordan

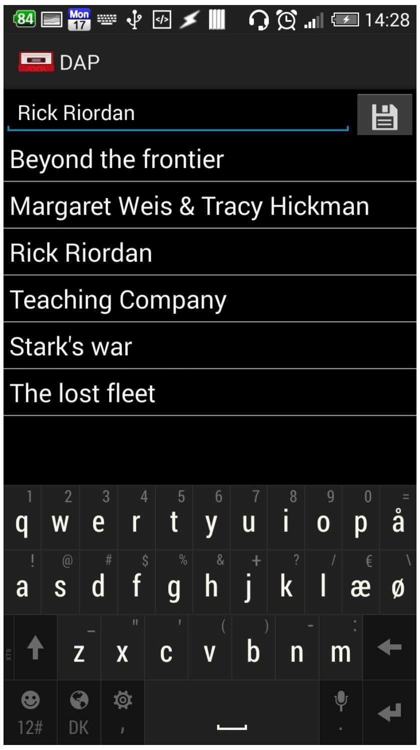
Album:
Percy Jackson and the olympians 3 - The titan's curse

Percy Jackson 3 - The titan's curse - Part 01 72:41
Percy Jackson 3 - The titan's curse - Part 02
Percy Jackson 3 - The titan's curse - Part 03
Percy Jackson 3 - The titan's curse - Part 04
Percy Jackson 3 - The titan's curse - Part 05
Percy Jackson 3 - The titan's curse - Part 06

This picture is modified to show more than what a usual user would see on a screen

Clicking on a field - author, album, cover or track, opens an edit menu.

- **Edit author / album**



14:28

DAP

Rick Riordan

Beyond the frontier

Margaret Weis & Tracy Hickman

Rick Riordan

Teaching Company

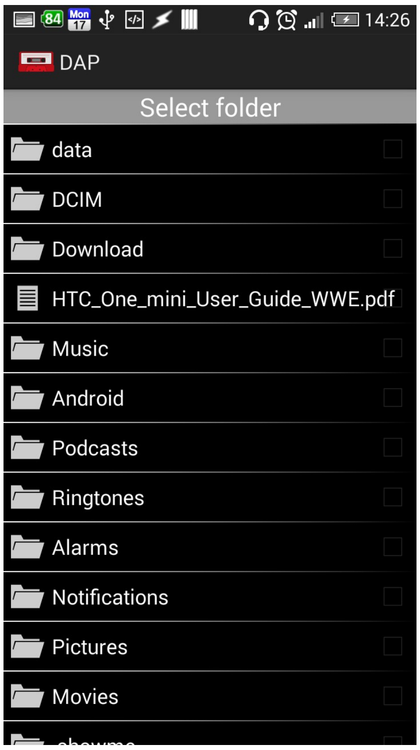
Stark's war

The lost fleet

The picture is an example where the user wants to edit the author name.
At the top there is an input field with a save button. The input field's default value is the current value. This makes it easier for the user to correct minor typos. The thought behind it is that it is easier to delete the current value if it is wrong than to type the current value if it is (almost) correct.
Below is a list of all the authors (also the wrong ones). Instead of typing a name the user can select an already existing name - It is common to have more than one audiobook written by an author.
When changing album title the list consists of all albums by the current author. *(This is not actually implemented yet)*

- **Edit cover**
  Changing a cover image is simply a matter of selecting a new image file. See “select file UI”.

- **Select file UI**


14:26

DAP

Select folder

data

DCIM

Download

HTC_One_mini_User_Guide_WWE.pdf

Music

Android

Podcasts

Ringtones

Alarms

Notifications

Pictures

Movies

The select file UI is very flexible and can be used to select either a specific file or a folder. At the moment there are only 4 types of files

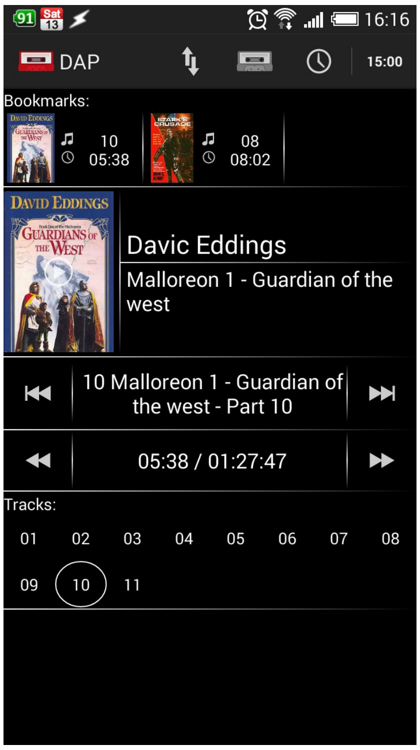
1. Audio files (.mp3)
2. Image files (.png or .jpg)
3. Folders
4. All others

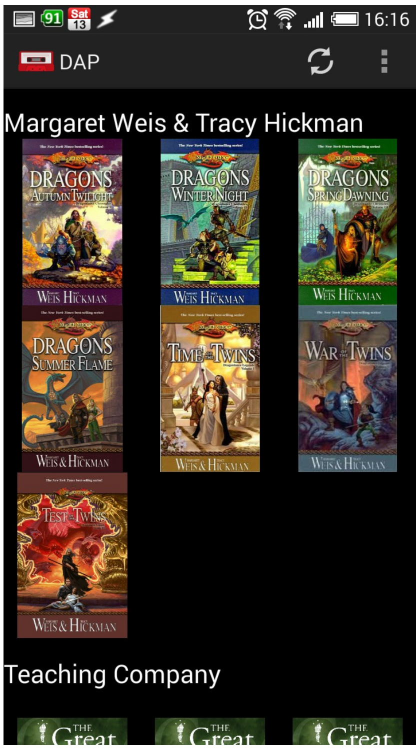
The UI displays a message at the top, telling the user what to do.

Clicking on a folder enters the folder, but if the user has to select a folder he/she will have to either long press the folder or use the checkbox on the right hand side. Preliminary tests showed that neither is obvious to the user. The checkbox is gray on a black background and is very hard to see. I intend to make something more visible than a gray checkbox, but haven't as of yet.

Version 1.0.0

In this the latest version the main screen has been altered:





- **Bookmark and audiobooks are not as important as controller**
  Usually no more than 1 maybe 2, 3 maximum, bookmarks are needed at the same time. Audiobooks are plentiful, but rarely used. Hence they ought not be equated with the controller which is used a lot and often. Hence the controller became the main activity, audiobooks is a secondary activity, and bookmarks are just displayed as a scrollpanel atop the controller. There is in fact room for an infinite number of bookmarks, but realistically, if more than 10 bookmark exist, it will become increasingly annoying to find any specific bookmark. (Technically the number of bookmarks will be limited by amount of available memory, since all bookmark-views are created in activity's onCreate() method - it is not a listview where views are created as needed (listview cannot be horizontal).

- **Action Bar**

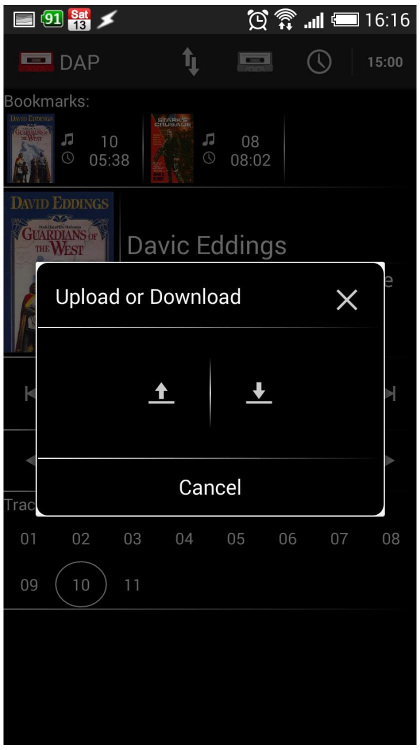
5 icons in the action bar is too much even though they are all valid, according to Google's FIT-scheme[3]

Up- and download are key features - without them DAP would be only AP (Audiobook Player), and there are more professional (and probably more reliable) apps “out there”.

The Audiobook grid is maybe not used often, but when it is needed it should be easily accessible, since it is the only navigation in DAP and otherwise it would might be overlooked.

The timer is used often and the countdown is worthless if not visible.

Up- and download can be joined to one icon (import-export) and a dialogue.



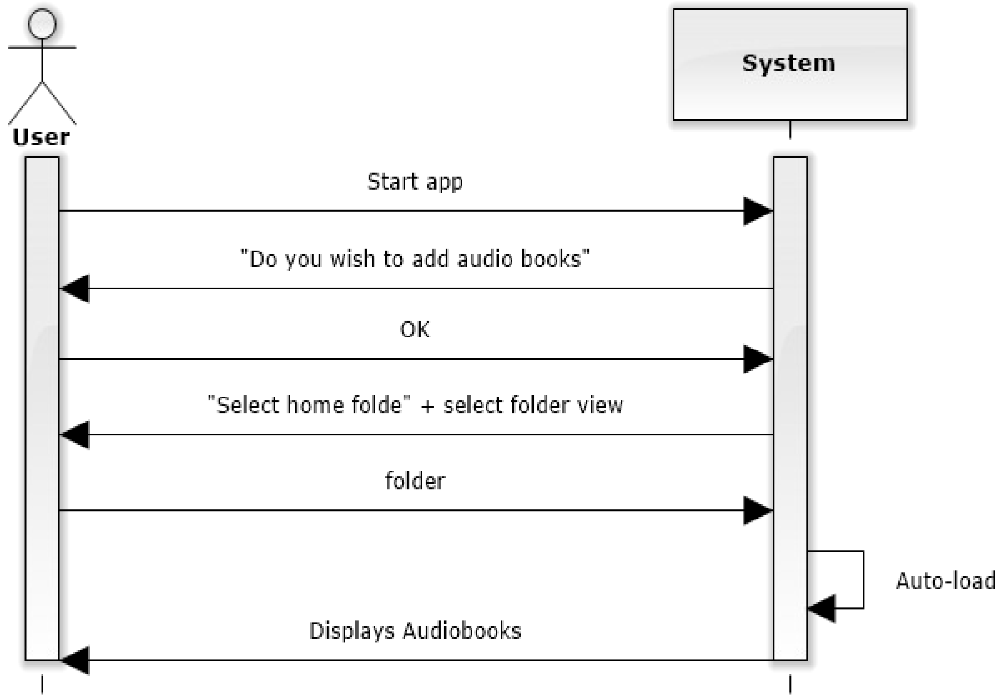
The dialogue for adjusting the sleep time can be found in appendix B

3 See appendix E

Test

I did a preliminary test, very early on. I was unable to finish the test, because the app was simply too difficult to use. My test subject was unable to add a new audiobook.

This wasn't a proper test, but the result was undeniable. That is why the system now behaves like this, the first time it is started.


User
System
Start app
"Do you wish to add audio books"
OK
"Select home folde" + select folder view
folder
Auto-load
Displays Audiobooks

The Auto-load feature works after some basic instructions which may lead to errors in the audiobooks[4] - The user will have to correct these errors manually.

The real tests were carried out at a point in time when both the apps were very close to completion.

4 See the section "Home folder vs single audiobook creation" in the chapter "Player"

Descriptions of the test and test subjects can be found in appendix C and D.

The tests revealed a few minor bugs, which have been fixed and a few errors:

- It is too difficult to tell when the sleep timer is active. So far I have not implemented any solution to this. One solution could be for the button to glow red like in version 0.1.0.
- When an audiobook is selected, it is not enough to add a bookmark - it should also be loaded into the player. This has been corrected.

To my surprise nobody had trouble with case 3 (Authorizing DAP to use Google Drive).

Please note that the Android project still contains all the old and unused layouts (XML files), in case they are needed later.

Conclusion

I have succeeded in creating an Android app, which plays audiobooks to the desired specifications. Although it is not flawless, all major features are working properly. It might not have all the features of other audiobook players in Google Play Store, it does exactly what it is supposed to do and it is very easy to use.

Bookmarks are distributed using Google Drive in stead of a custom server. That means that it is as scalable as Google Drive.

The desktop app isn't quite as far along, but it is good enough to verify that the concept holds.

The apps still contains errors:

- Some features from ver. 0.1.0 doesn't work in ver. 1.0.0
- When adding a new bookmark, that bookmark is also selected.
- Up- and download buttons are reversed in the desktop app.
- Wrong border and missing texts in dialogues in desktop app.
- Display audiobooks side by side with the controller on tablets.
- Most texts are hard-coded, but should be loaded as resources.
- Desktop app is missing forward and rewind buttons.
- BookmarkManager and AudiobookManager needs a clean-up.

In the near future I intend to implement the following features:

- Upload settings and rename actions
  (That way if an audiobook, author or similar is renamed on one device it will automatically be renamed on other devices. Otherwise bookmarks won't refer to the right audiobook.)
- Automatic upload of bookmarks
- Progress as progress bar instead of "progress / duration"
- List of album titles to use when editing - just when editing an author name.

At some later time I might include an option to read ID3 tags from mp3-files.

Appendices

Appendix A: Risk

	Event	Probability (1-10)	Impact (1-10)	Risk
A	Major loss of code	5	9	45
B	Flawed design	4	7	28
C	Minor feature flawed	8	2	16
D	Hardware breakdown	2	9	18
E	No physical work place after December 5.	9	2	18
F	Minor loss of code	5	2	10
G	Major time loss due to illness or other	2	5	10
H	Minor time loss due to illness or other	8	1	8

A: Major loss of code

A major code loss would be a big problem. I intent to eliminate this by frequent backup. I have decided to use git and Github - this gives me both backup and the ability to create branches to test new ideas.

B: Flawed design

If the basic design doesn't work the project is doomed in advance. I have found other apps on Google play store which have dealt whit similar problems. If they can do it so can I.

C: Minor feature flawed

Dependent on when in the process this occurs, it might be necessary to revise the requirements specification, but the project, as a whole, is not affected.

D: Hardware breakdown

Without a computer I have no chance of finishing the project. However I have not 1 but 2 extra computers on which I could finish the project. It would take a bit of time to setup, but it is doable. The same goes for my phone, as I have access to another.

E: No physical work place after December 5.

I don't, at the moment, know if I can stay at my current location to the end of the project. I can move, but it would result in a more noisy environment, and the quality of the project might suffer.

F: Minor loss of code

This is of little importance. If I loose a key component I will still have a notion of the basic concept and I would only loose a bit of time recreating the component.

G: Major time loss due to illness or other

I can only hope this won't happen. Since there isn't anything more I can do to avoid it. I will remember to take my vitamins.

H: Minor time loss due to illness or other

I suffer from migraines, so this is highly likely to happen, but it is already included in my time schedule.

After these considerations I have revised the risk analysis.

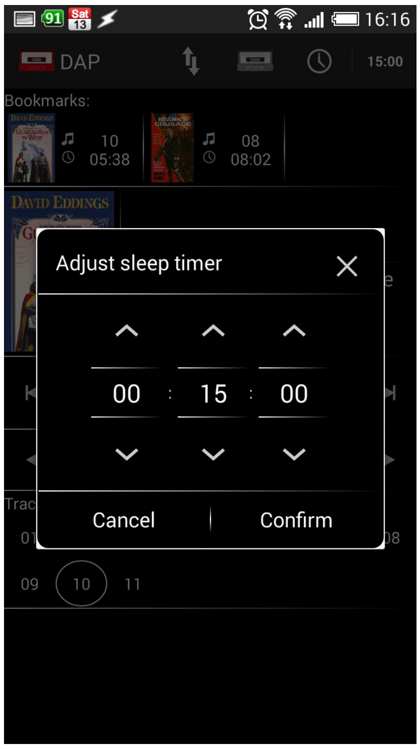
Event		Probability (1-10)	Impact (1-10)	Risk
E	No physical work place after December 5.	9	2	18
C	Minor feature flawed	8	2	16
A	Major loss of code	5	2	10
G	Major time loss due to illness or other	2	5	10
H	Minor time loss due to illness or other	8	1	8
F	Minor loss of code	5	2	10
D	Hardware breakdown	2	3	6
B	Flawed design	0	7	0

”A Major loss of code (late)” This now has a much lower impact as fairly recent copy always will be available on github.

”B Flawed design” I actually wrote a bit of code and solved this problem - the probability is reduced to 0.

”D Hardware breakdown” I now know that I can continue in spite of this.

Appendix B: Sleep time dialogue



Appendix C: Test cases

Test case 1

This assignment is simply to set the Sleep timer to a 20 minute countdown (the default is 15 min.). I will be watching for the following:

1. Can the tester find the sleep timer
2. Can the tester tell when the Sleep timer is counting down.
3. Can the tester find the Sleep time settings button.
4. Can the tester tell what is hours, minutes and seconds

Test case 2

In this assignment the tester will have to start a new audiobook - Not add a new audiobook, but simply to select another one. I will be watching for the following:

1. Can the tester find the audiobooks.
2. Does the tester know how to select an audiobook.

Test case 3

The assignment is to authorize DAP to upload bookmarks to Google Drive. On the Android app this is really easy, but it is a bit difficult to do in the desktop app. So the tester will have to do this assignment on a computer. I will be watching for the following:

1. Can the tester find the sync button
2. Can the tester tell the difference between up- and download.
3. Are the instructions enough to understand the process.

Appendix D: Test subjects

Tester 1

Gender: Male

Age: 34

Android expert

Never listens to audiobooks.

Result: No problems

Tester 2

Gender: Female

Age 31

Very familiar with Android

Listens to audiobooks very often

Result: Had trouble determining if the sleep timer was active

Tester 3

Gender: Female

Age 26

Unfamiliar with Android

Never listens to audiobooks.

Result: Minor trouble locating the sleep timer

Tester 4

Gender: Female

Age 60

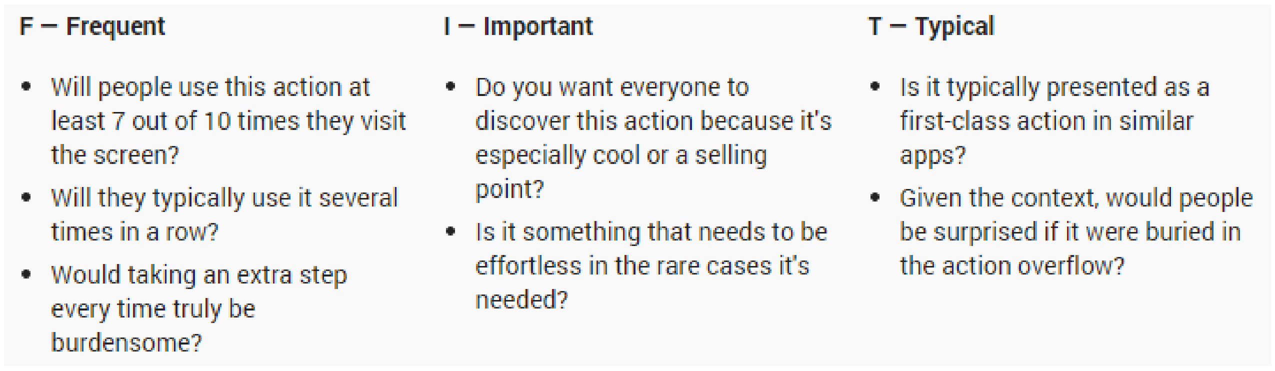
New to Android

Listens to audiobooks very often

Result: Had difficulties finding the audiobooks

Appendix E: FIT

This is a snippet of Google's Android documentation. It is a FIT-scheme which is used to determine if a feature should be present on the action bar or if it should be in the overflow.



F — Frequent	I — Important	T — Typical
• Will people use this action at least 7 out of 10 times they visit the screen? • Will they typically use it several times in a row? • Would taking an extra step every time truly be burdensome?	• Do you want everyone to discover this action because it's especially cool or a selling point? • Is it something that needs to be effortless in the rare cases it's needed?	• Is it typically presented as a first-class action in similar apps? • Given the context, would people be surprised if it were buried in the action overflow?

https://developer.android.com/design/patterns/actionbar.html