RSSin – clever RSS reader for Android

Jos Craaijo, Camil Staps, Joep Bernards, Randy Wanga

June 26, 2015

Abstract

In the past decades, the amount of information publicly available has grown dramatically, which is popularly considered to be the cause of more stress, burnouts, and Alzheimer's disease. It is necessary to develop methods to process information automatically, such that someone only has to read the information relevant for him. In this article we're proposing a way to do this with RSS feeds, based on artificial intelligence.

1 Introduction

News can be found in different sources, mostly organised by category. As an example, CNN [3] has different feeds for different regions of the world and different news categories (sports, tech, environment, ...). The first hurdle one has to take when designing a system that intelligently filters news, is to gather news from different sources. After that, the huge amount of news available has to be filtered by relevance for the reader, and displayed in an appropriate manner.

Besides filtering or ordering article lists, there also needs to be a system in place to filter article contents. Even when a reader gets an article list sorted by relevance, he doesn't have the time or the will to read all the relevant articles: he wants to read the gist of the article in at most a few sentences, and decide for himself whether he's going to continue reading or not. A second requirement is thus a system that can intelligently *shorten* article texts.

In this report we present RSSin [2], a clever RSS reader for Android. It combines the aforementioned features in order to combine article lists from different RSS feeds [1] into one, to order that one list by relevance from the user, and finally to summarise article texts to give a compact overview.

RSSin started as a university project at the Radboud University [13] but is now an open source community project. This document describes the current state of the project, along with the design rationale.

1.1 Report organisation

Section 1.2 will give an overview of older projects that are somehow related to what we have done with RSSin, and a brief discussion as to why the tools currently available are not good enough. In section 2 we will walk through the goals we wanted to accomplish in general with our RSS reader, while section 2.1 looks at the additional requirements when developing for Android. In section 3 we will describe the user interface and argue why this is a good one. Section 4 gives a brief overview of the different packages in the source code and is the absolute minimum one should read before forking our repository [2] and continuing development. During the development we made some discoveries that may be useful to others, we will describe these in section 5. Section 6 gives some suggestions for other developers to continue the work, and in section 7 we look back at our project.

1.2 Related work

Many, many RSS readers have been made, for Android as well as for other platforms. As for Android apps, we didn't find any that does something more intelligent than sorting articles in chronological order. For other platforms there is Yahoo Pipes [5], but using it is not intuitive, and building useful filters is time-consuming. Researchers at the University of Pisa have discussed algorithms for ranking news streams [4], but does not take into account the reader's preferences. Still, this latter paper is an interesting one and as far as we know the first to formalise ranking news streams.

Of course, several search engines are using artificial intelligence to predict how interesting a searcher is going to find a certain page – a notable example is the extended version of PageRank [6] Google is using. However, it can be assumed that these algorithms are using machine learning over the whole set of their clients, and are building large user profiles that go together with privacy issues. There should be a solution that does not immediately raise privacy concerns.

2 Specification

Having seen and used several of the applications mentioned above, we set out to build a new kind of RSS reader. One that does not simply display articles, but sorts them for you. One that gives you a compact overview of the most important news rather than requiring you to click through to read the article contents, or displays all contents in the overview such that the list isn't compact enough. And finally, one that does all this without requiring external resources besides the news sources, and uses only your data to order article lists. Concretely, we wanted to meet the following requirements:

- Fetches news from different sources
- Uses artificial intelligence to order articles based on relevance for the reader
- Only uses privacy-sensitive information of the user himself
- Is intuitive to use
- Summarises article contents for a compact overview
- Can do all this within reasonable time on a regular Android 4.0+ phone in terms of performance

2.1 Android frontend

With the current state of artficial intelligence and the platform restrictions found on a regular smartphone, we cannot rely solely on artificial intelligence. In particular, it wouldn't be sufficient to put articles from all sources altogether in one list and rely on the AI to order this properly. Users may suddenly want to only read sport news, and ignore other categories, or may decide that that moment they only care about news published by Reuters. Even if AI is already able to recognise this, it is certainly not something that can be done using the resources available on a regular smartphone. We thus decided to give the user the following views:

- An overview of all articles from all feeds
- An overview of the articles from a single feed
- Several customisable views ("filters") combining articles from several feeds, as configurated by the user

Furthermore, there are views for adding and removing feeds and filters, and for reading a full article.

We will now proceed by walking through the design of both the user interface and the backend.

3 User Interface

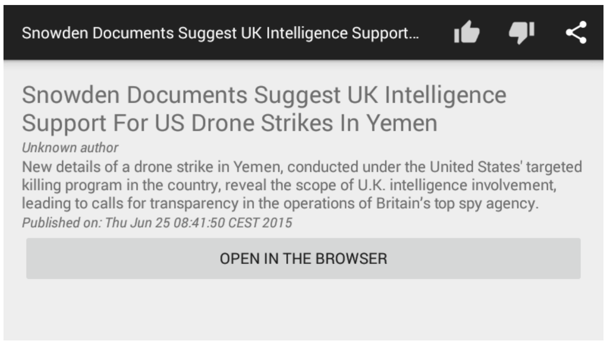
Since the amount of feeds and filters and thus the amount of overviews can be large, and all these views should be easily accessible, we have decided to use a hamburger menu (figure 1a). Almost all views are accessible from the hamburger menu. The "All feeds" item gives a sorted overview of articles from all feeds. Below, there are the customisable filters. Tapping on a filter itself gives you an article list with articles from the feeds that are linked to that filter. Tapping on the menu heading "Filters – EDIT" brings you to a view where you can add, edit and remove filters. Further down the menu we find the list of all feeds. Similar to the filters list, tapping on a feed itself gives an article list with only articles from that feed, and tapping on the menu heading brings you to a view where you can add and remove feeds.

As you can see, all menu items give you the same kind of view, which makes this an easy to use solution. Only the two menu headings for filters and feeds give a different view, but we gave these items a different layout (a lighter background and the text "EDIT" on the right) so that the distinction is clear.

All article lists (figure 1b) are simple lists, leaving room for the content. We used CardViews to give the lists an Android look and feel. The lists show the user the title and publication date of an article, as well as a short summary.

Figure 1c and 1d show parts of the configuration interface. There is nothing special there, but it is simple and intuitive to use.

Tapping on an article brings the user to a view with only that article (figure 2). Here, he may see the full text (that is, the text in the RSS feed – this may not be the full article, depending on the feed). There is a button to go to the website of the article. More importantly, in the action bar there are buttons for giving feedback to the artificial intelligence, using simple and intuitive "likes" and "dislikes". Lastly, the action bar has a button with which the article can be shared using other apps on the device.


Snowden Documents Suggest UK Intelligence Support...

Snowden Documents Suggest UK Intelligence Support For US Drone Strikes In Yemen

Unknown author

New details of a drone strike in Yemen, conducted under the United States' targeted killing program in the country, reveal the scope of U.K. intelligence involvement, leading to calls for transparency in the operations of Britain's top spy agency.

Published on: Thu Jun 25 08:41:50 CEST 2015

OPEN IN THE BROWSER

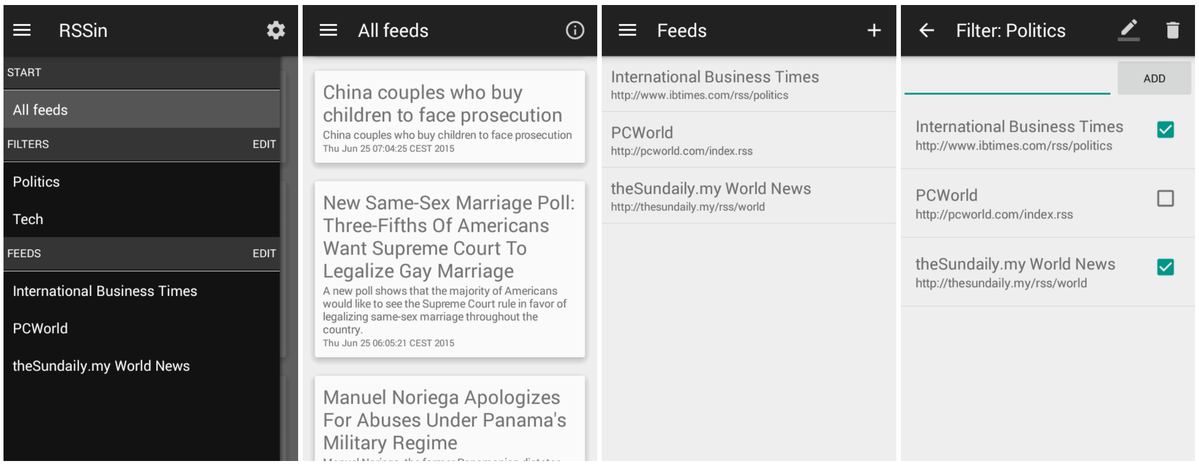
Figure 2: An article, with (dis)like and share buttons

4 Code organisation

The source code may be found on GitHub [2]. In this section we will give a brief overview of the different packages and most important classes using a bottom-up approach. This is not intended to be a complete overview. For more information, the javadoc should be consulted.

The `org.rssin.rss` package provides basic functionality for storing an RSS feed in an easily accessible manner. The `Feed` class provides an easily accessible representation of an RSS feed. It has attributes for metadata, such a description of the feed, and a list of `FeedItems`, which corresponds to a `item` RSS tag. The `FeedLoader` can build such a `Feed` from an `XmlPullParser`. This `FeedLoader` class can also be used to fetch a feed from the internet. This is done using a `Fetcher` (a simple interface from the `http` package for something that can fetch data using HTTP requests) and an asynchronous `listener.FallibleListener` (an object with `onReceive` and `onError` methods).

These interfaces are, obviously, used to be able to easily



China couples who buy children to face prosecution
China couples who buy children to face prosecution
Thu Jun 25 07:04:25 CEST 2015

New Same-Sex Marriage Poll: Three-Fifths Of Americans Want Supreme Court To Legalize Gay Marriage
A new poll shows that the majority of Americans would like to see the Supreme Court rule in favor of legalizing same-sex marriage throughout the country.
Thu Jun 25 06:05:21 CEST 2015

Manuel Noriega Apologizes For Abuses Under Panama's Military Regime

International Business Times
http://www.ibtimes.com/rss/politics

PCWorld
http://pcworld.com/index.rss

theSundaily.my World News
http://thesundaily.my/rss/world

International Business Times ✓
http://www.ibtimes.com/rss/politics

PCWorld ☐
http://pcworld.com/index.rss

theSundaily.my World News ✓
http://thesundaily.my/rss/world

(a) The hamburger menu (b) An article list, here for all feeds (c) The list of feeds (d) Settings for a filter

Figure 1: RSSin screenshots

switch to different methods later, if necessary. As an example, the VolleyFetcher from the android package implements the Fetcher interface using the Volley library [8]. More importantly, it allows anyone to create a different frontend later on. One could even choose to not use Android. Anything outside the android package does not rely on Android whatsoever. One may choose to build a different frontend, as a CLI application, using Swing, or otherwise.

The org.rssin.rssin package consists of some very basic classes that are needed to store the user's preferences. This is basically his feeds and filters, and a FeedLoaderAndSorter which uses a neural network (which we will get to in a minute) to order FeedItems.

To provide different ways to store these preferences and the FeedLoaderAndSorter, we implement interfaces from the storage package. In the android package two such implementations can be found: an InternalStorageProvider for storing objects in the internal storage, and a SharedPreferencesStorageProvider that uses Android's shared preferences. In addition to this, there is a DefaultStorageProvider which is simply an extension of the SharedPreferencesStorageProvider, which allows you to change the StorageProvider later on even more easily. Currently, we're using the shared preferences everywhere since the internal storage seems to have some performance issues.

Such a StorageProvider can store Storables, for which the default writeObject and readObject methods are used. Classes may override these if wanted. Of course, the default Java implementation of these methods is undesirable, as it doesn't allow for flexible redesign of the stored classes. A better way to store data would be using JSON or XML. To this end we created a serialization package, with some tools to serialise objects using JSON. Actually using these classes gave us some problems with data getting lost, so they cannot be considered production-ready, and are just there as a starting point for development continuation.

All this already provides basic functionality for most of the RSS readers available in the Play Store: with this one could sort article lists chronologically, add and remove feeds, and use filters. We won't discuss the android package which contains the user interface in depth, as the user interface is fairly straightforward. In addition to what any simple RSS reader can, we want to sort on relevance and generate summaries.

The latter is done in the summaries package, which is a Java port of the Babluki summary tool [9], with extended functionality for different summary lengths and the like.

Finally, the neurons package is a set classes for simulating a neural network. The artificial neural network is an implementation of a relatively simple, two-layer feed-forward network. The Neuron class represents one single node, which the NeuralNetwork class uses to create a network. The neural network returns predications as a PredictionInterface. This interface can be used later on to provide feedback to the neural network using the learn() method, which uses backpropagation to train the network.

In some cases the network may get stuck in a local minimum, which means it is unable to correctly learn a specific pattern. To migitate this problem, a MultiNeuralNetwork class has been added. This class averages the predictions of multiple NeuralNetworks. Lastly, the FeedSorter class is used to sort a list of FeedItems. The class exposes most of the functionality of the neural network to other packages. Additionally, FeedSorter implements methods for sorting a list of feeds and keeping track of feedback, using the Feedback enum.

5 General recommendations

While working on RSSin, we've come across some interesting pecularities that we'd like to mention here. They may be interesting for other developers.

5.1 Storage methods

One interesting thing we discovered when developing different StorageProviders and were testing which one performed better, was that the internal storage has some serious issues compared to shared preferences. The scenario was that we had to store Java objects (from the rssin package).

We wrote a working implementation of StorageProvider using Android's internal storage, and one using the shared preferences. Shared preferences are XML files stored in the internal storage, so one would expect this would be slower due to some overhead. However, it turned out that the shared preferences were faster. Shared preferences can be applied asynchronously, but even when using the synchronous commit method, this StorageProvider was faster. It is unclear to us why.

It should be noted that we can't generalise this until we found the exact cause of the performance difference. We don't know if the shared preferences are faster in any case, or only in cases of serialisation of large objects, or only in very specific cases. Naturally, there are some cases where the internal storage is preferred in any case – one could think of situations where it's necessary to append data, such as when logging to a file.

5.2 RSS ≠ RSS

The RSS specification [1] gives a lot of freedom for implementers. That is a feature, but it also makes relying on feeds being consequent impossible. Every RSS feed supplier has different ideas of what should go in what field, how long the description should be, whether or not HTML tags should be stripped, if unstripped HTML tags should go in CDATA or not, etc. Concretely, we found the following inconsequences:

- The length of the description may vary between one sentence and several paragraphs.
- The description field may contain a) no HTML, b) HTML tags as if they were XML tags, or c) HTML tags in CDATA (as they're supposed to be)
- Different suppliers have different ideas about where images should go (as img tag in the description, in the enclosure tag, or in a separate media:content (or similar) tag).
- The RSS specification states that the author field should be an email, but plenty of feeds give a simple name.
- Some feeds may include tracking pixels, which is completely ridiculous considering the application.

Ideally, RSS readers would simply not support feeds with strange 'features'. However, there are so many RSS readers that the developers have to find *something* to be better than others, and now readers are supporting the most ridiculous inconsistencies. Unfortunately, RSSin doesn't have that many downloads yet to be a trend setter in this matter.

6 Future ideas

Currently, the latest release of the RSSin app has tag 0.1c, indicating that still a lot has to be done before being really production-ready, in particular in the user experience area. A list with some ideas, in no particular order, is provided below. The whole app is open source [2], and anyone is invited to participate in making the following list shorter.

- Allow the user to swipe left and right an article from the article lists to dislike and like it, respectively. This will make giving feedback easier.
- Use a `SwipeRefreshLayout` to allow the user to refresh article lists (this is currently not implemented due to incompatibilities with the `RecyclerView`).
- A cache, to not fetch all feeds again when changing the view.
- Better error handling. For example, the "Loading" bar is still there when there is no internet connection.
- Enhanced progress indicator, to show how many feeds of the total have been loaded.
- Background synchronisation in combination with notifications for the most important news.
- Widgets with most important news, from all feeds, one feed, or a filter.
- Remove disliked articles from the article list.
- Sorting can be quite slow, especially on low-end devices. Either optimize the neural network, or use a simpler network (ideally the complexity of the network would depend on the device, and there could be some kind of calibration on first start-up)
- Add article images to the article lists.
- Add feed icons to the hamburger menu.

7 Evaluation

In section 2 we mentioned the goals we had before building RSSin. Looking back, we can be content with the goals we reached:

We **succeeded** in fetching news from different sources. RSSin uses **good** artifical intelligence to order articles by relevance. We **do not need privacy-sensitive information** from the user. The app produces **concise and complete summaries**. The app is **relatively** intuitive to use, even though there are still some things to improve (see section 6). RSSin is **fast enough** on a regular smartphone, but performance could be improved (see section 6). Lastly, there is a clean and simple interface with all the views we intended to use in section 2.1.

We used Git [10] to keep track of different versions. We found this to have different notable advantages (over other version control systems or using no such system at all):

- Working offline is possible
- It is faster than other systems
- It is possible to commit code without breaking the build, on a different branch

We are glad that through using it, we learned the basics of branching, merging and tagging in Git. We did find though that the GitHub client for windows [11] has very limited functionality. In the end, we preferred working with the CLI, even though it is limited on Windows, or using Git in Cygwin [12].

Similarly, we learned a lot about Android development. The Volley networking library [8] and the differences between different storage providers (see section 5.1) are just some examples. While we're happy to have gained experience with Android, we found it a pity to have to spend quite some time finding out the specifics of XML layouts. The ideas behind Android layout files are completely different from those that are at the base of HTML, with which we are more familiar. Furthermore, the documentation [7] – though complete – often lacks examples.

When preparing for the first release, we noticed we weren't writing enough comments. Many code snippets (like for `Activities` or layout files) can be generated automatically using Android Studio. After doing that, we stripped them down to the minimal functionality we needed and continued with that. This lead in the end to the lack of comments in many of these generated files. A next time we would take more care about this, to make sure development can continue later.

Starting the project, we made a highly simplified UML diagram and distributed tasks over different Java packages and developers. Having the interaction between different classes fixed, everyone could write a first version of his package(s) without having to bother about the others. This helped us to get a first version working very quickly. We only needed to communicate when making this UML diagram and when putting everything together once we had written our parts. A next time we would certainly take a similar approach. Depending on the size of the project, we would also write more automated tests to test the different functionalities separately.

References

[1] RSS 2.0 specification, `http://cyber.law.harvard.edu/rss/rss.html`

[2] RSSin: clever RSS reader for Android, `https://rssin.org/`, `https://github.com/camilstaps/RSSin`

[3] CNN RSS feeds, `http://edition.cnn.com/services/rss/`

[4] Del Corso, Gullí, Romani. Ranking a Stream of News, `http://www.di.unipi.it/~delcorso/papers/www.pdf`

[5] Yahoo Pipes, `https://pipes.yahoo.com/pipes/`

[6] PageRank, Method for node ranking in a linked database, `http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=6,285,999`

[7] Android Developer's Reference, `http://developer.android.com/reference/packages.html`

[8] Volley networking library for Android, `https://developer.android.com/training/volley/index.html`

[9] Babluki summary tool in Python, `http://thetokenizer.com/2013/04/28/build-your-own-summary-tool/`

[10] Git - the stupid content tracker, `https://git-scm.com/`

[11] GitHub for Windows, `https://windows.github.com/`

[12] Cygwin - Linux-like environment for Windows, `https://www.cygwin.com/`

[13] Radboud University Nijmegen, the Netherlands, `http://ru.nl`