CS153A - Mobile Application Development
Brandeis University
Fall 2015

MIT CogNet App

Dimokritos Stamatakis, Xiaodong Qu and Namho An
December 3, 2015

1 Introduction

Our project is an Android app for the cognet.mit.edu website. Our app can get the secured data of books, journals and reference works from the website, the end users can perform actions like login/logout, view content list, search, scroll down,view content details and pdf, and add to favorite etc. We are using Agile Scrum to guide our development work. The website have development site, staging site, and live site, our development work is based on staging2 website, and our code and communication is saved in Bitbucket. We are trying to apply what we learned from this class to a live project with a lot of content and users. Also we learned how the data can be pass along between a website and an Android app.

2 Scope of Work

Within this semester, we planed to working on the following functions: User Authentication: Login with username and password; Access control: only users with right permissions or from the right IP-range can access pdf files; Get content data from the website to the app using RSS feed, setting up RSS feed at the website, Analyzing the XML data and displaying the content on the device; Add end user actions like Search and Scroll down; Add - to - Favorite function, so users can bookmark their favorite books and journals. By 12/02/2015, we had successfully finished the functions above, and conduct tests to confirm they are working.

2.1 User authentication

In order for users to access the PDF files from CogNet they must either have an account, or connect through a subscribed network. Brandeis has subscribed to CogNet, so anyone with a Brandeis IP address can access the PDF files. Our user authentication functions are using `RESTful API`, and perform `HTTP POST` requests with the *apache HTTP client* library. For example, a user can login by sending a `POST` request to the CogNet server in the appropriate `URL` and adding the username and password as `HTTP` headers. We then analyze the response from the server and if the login is successful we extract the session ID and session name, which both form the session cookie. The session cookie is stored in the Main Activity and is passed to the child activities so that to use it when viewing a PDF is required.

We automatically log out the user's account when he terminates the application, by adding the log out function call within the `onDestroy()` function. The process of logging out consists of a `HTTP POST` request and the cookie as a header.

2.2 Getting data from the server

We enabled the `RSS feed` in the CogNet server through the drupal admin panel. This means someone can ask for a page and get a response in `XML` format, instead of `HTML`. This is very useful for our application, since it will only have to parse `XML` and extract the required tags, instead of analyzing the entire `HMTL`, finding out what is required and discarding the rest. We perform `HTTP GET` requests to get the `XML` with information about books, journals, etc.

2.3 Displaying data

We are using a `ListView` to display the list of items on the device. After we get the response from the server, we store this information in an `ArrayList` and set a `SimpleAdapter` to bind the `ArrayList` ot the `ListView`. The `ListView` consists of three lines, showing the title, author and publication date of each book, journal, or reference work.

Since many items may be loaded in the `ListView` we made it to display only the first 20 items and added a scroll event for when the scroll reaches the bottom, also known as "endless ListView". If the scroll reaches the bottom, the next 20 items are loaded and appended at the end of the `ListView`. This was a tricky part, since Android does not allow background threads to update the data of a View's Adapter. If we change something in the Adapter's `ArrayList` in a background thread, an Exception may rise, indicating that Adapter's data has been changed and the Adapter is not notified.

We used the AsyncTask class to perform asynchronous tasks by spawning background threads. A background thread is spawned to add the next 20 items in the `ArrayList`, and when it is done an event is raised to indicate that the background thread is done. The event notifies the Adapter that the data has changed, but Android raised an Exception about not notifying the Adapter for the changed data. We tried to notify the Adapter right after we add each item in the `ArrayList` but this also had some issues. So we came up with the approach of having a copy of the `ArrayList` in the background thread and have it add the new items in the copy of the `ArrayList`, instead of the original `ArrayList`. When the background thread is done adding new items, the event will raise and set the contents of the `ArrayList` as the `ArrayList` copy, as got from the background thread. Finally, it will notify the Adapter that the data has changed, and everything works perfectly.

We enabled automatic search, where the user can search for a title or author by just typing, and see the results changing while typing. To support that, we had to add a key Listener in the search box and implement the `onTextChanged()` function so that to perform a search operation when the search text is changing. The search is performed locally, in the ArrayList with all the items, not only in the displayed items. Search operation finds the matches in the title and author and then starts a new background thread by using the `AsyncTask`, similarly to the scroll event, to update the ListView with the result set. The event is raised when the background task is done and it notifies the Adapter that the data has changed.

Object oriented techniques and inheritance came handy, by allowing us to have a super class for all the types of ListViews, called ItemsListActivity and then have one subclass for each item category, for example: BooksListActivity, JournalsListActivity, etc. that will extend the super class and inherit all the common functionality.

3 The item description Activity

This Activity shows description for a selected item from the ListView, including the title, author, publication date, image, description text and link to the PDF. We added a Click Listener in the `ListView`, where a new description Activity will be started when an item is selected. We also pass the information we got from the `RSS feed` to the description activity encapsulated in an Intent. At this point we can set the text of the `TextViews` according to the Intent's data, and also retrieve the image from the server and add it to an `ImageView`. We also have the ISBN of the book, from which we can construct the PDF URL.

4 viewing PDF files

For this task, we want to get a PDF file from CogNet server and display it on the device. As we said before, we store the ISBN of the selected book and from this we can construct the PDF URL.

Initially, we thought of using a `WebView` with the Google Docs plugin and the URL of the PDF, as

`https://docs.google.com/gview?embedded=true&url=cognet/pdfURL.pdf`

The initial request will first go to the Google Docs server and the Google Docs server will redirect it to the CogNet server. The problem is that the Google Docs server does not

forward the cookie to the CogNet server, so the request cannot be authenticated. Thus, our alternative approach is to download the PDF on the device and then start an intent to display it. We did it by storing the PDF in a file on the device through a `byte[]` buffer. Then, we start an intent for viewing the PDF by providing the PDF URL and its type, as:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setDataAndType(pdfPath, "application/pdf");
```

4.1 "Add-to-Favorite" function

We support an "Add-to-Favorite" function, which will allow the users to save the books they find interesting in a list and access them anytime. This list extends the ItemListActivity to inherit the common functionality, with the difference that it accesses the contents of the ViewList from the SQLite database. Thus, the favorites list will remain even if the application exits. We check whether an item already exists in the Database and if so, we make the "Add Favorite" button "Delete Favorite", which deletes the item from the Database.The "Add-to-Favorite" does not require login and will be stored locally in the device, since the login is only for the CogNet server authentication. The favorites list is also useful for someone who does not have an account in CogNet, but can access the PDFs through a subscribed network, ie. Brandeis. If he is not at the Brandeis network, he can add the books he finds interesting in the favorites list and view the PDFs later when he connects to the Brandeis network.